

CASSETTE BOOK
FOR THE
COMMODORE-64
AND VIC-20



A DATA-BECKER BOOK by Dirk Paulissen

YOU CAN COUNT ON
Abacus
Software




CASSETTE BOOK

for the Commodore 64 (and Vic-20)

By D. Paulissen

A DATA BECKER BOOK

Published by:

Abacus You Can Count On  **Software**

All commands, technical instructions and programs contained in this book have been carefully worked on by the authors, i.e., written and run under careful supervision. However, mistakes in printing are not totally impossible; beyond that, ABACUS Software can neither guarantee nor be held legally responsible for the reader's not adhering to the text which follows, or for faulty instructions received. The authors will always appreciate receiving notice of subsequent mistakes.

First Printing, May 1985

Printed in U.S.A.

Edited by Jim D'Haem
Drawings by Russ Taber

Copyright (C) 1984

Data Becker GmbH
Merowingerstr. 30
4000 Dusseldorf, W. Germany

Copyright (C) 1985

Abacus Software, Inc.
P.O. Box 7211
Grand Rapids, MI 49510

This book is copyrighted. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of ABACUS Software or Data Becker GmbH.

ISBN 0-916439-04-6

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
2.	INSTRUCTIONS FOR DATASSETTE HANDLING.....	3
2.1	SAVE.....	3
2.2	LOAD.....	8
2.3	VERIFY.....	9
2.4	OPEN.....	10
2.5	PRINT# and CLOSE.....	12
2.6	INPUT#.....	15
2.7	GET#.....	23
3.	SECONDARY ADDRESS.....	25
4.	STATUS VARIABLE.....	29
4.1	SAVING PROGRAMS AFTER 'LOAD ERROR'.....	31
5.	LOADING AND STORAGE UNDER PROGRAM CONTROL.....	37
5.1	OVERLAY TECHNIQUE.....	40
6.	STORAGE OF MACHINE LANGUAGE PROGRAMS.....	43
7.	CASSETTE BUFFER.....	49
7.1	CREATING A LIST OF CASSETTE CONTENTS.....	50
7.2	DISPLAY OF FILES FOUND.....	53
7.3	SELF-STARTING PROGRAMS AND PROGRAM PROTECTION.....	55
8.	STORAGE FORMAT IN CASSETTE STORAGE.....	63
9.	APPENDING BASIC PROGRAMS.....	67
10.	PROGRAM CONTROL OF DATASSETTE.....	73

11.	DATASSETTE HARDWARE.....	83
11.1	CARE OF DATASSETTE.....	83
11.2	CHOICE AND HANDLING OF CASSETTES.....	85
11.3	DATASSETTE OPERATION.....	87
11.4	LOUDSPEAKER FOR DATASSETTE.....	90
11.5	ADJUSTING THE HEAD.....	92
11.6	OTHER CASSETTE RECORDERS FOR DATA STORAGE.....	99
12.	NEW 'FASTTAPE' CASSETTE OPERATING SYSTEM.....	103
12.1	PROGRAM DESCRIPTION.....	126
13.	DATA PROCESSING WITH 'FASTTAPE'.....	137
13.1	PROGRAM DESCRIPTION.....	159
14.	CASSETTE CONTENTS & CATALOG OF FASTTAPE CASSETTES.....	169
15.	DISK BACK-UP FOR CASSETTE AND VICE VERSA.....	171
15.1	DISK BACK-UP FOR CASSETTE.....	171
15.2	PROGRAM DESCRIPTION OF DISK BACK-UP FOR CASSETTE.....	179
15.3	CASSETTE BACK-UP FOR DISKETTE.....	181
15.4	PROGRAM DESCRIPTION OF CASSETTE BACK-UP FOR DISK.....	208
APPENDIX	IMPORTANT ADDRESSES FOR DATASSETTE HANDLING.....	211

1. INTRODUCTION

Having gone to some trouble and expense to buy a computer, you will be unpleasantly surprised to find, the moment you switched it off, everything you have taken so much time and trouble to teach it, is immediately forgotten. In short, you need some means of storing everything it has learned, while the computer takes a well-earned rest.

Since everything the computer knows consists of nothing but zeros and ones, the idea of storing all this information magnetically, on a cassette for instance, seems like a good idea.

However, if the computer is to be able to read all the information which has been previously put into storage, it must store the data in a certain format and provide it with certain parameters. In order to ensure that this happens, the computer has a built-in program which takes care of all this.

When you read through the operating instructions, you soon realize very little is said about how the C-64 or VIC-20 with a Datassette functions, or how to operate them.

The purpose of this book is to help you get more out of the Datassette, and to understand how the computer and Datassette work with one another.

Using many sample programs, I will make it clear that you can get a great deal more out of the cassette than the manual suggests. The crowning glory, in the latter part of the book, is a completely new operating system for storing

programs and data on the Datassette. This system is considerably quicker than the built-in operating system, and is indeed even quicker than a disk drive. With the aid of this program package, I will show you that much may be done with a personal computer, if you have the right software.

And now a few tips on how to use this book:

All the programs published in this book will run on the C-64 and on the VIC 20. Since it is not always possible to write a program that will run on both computers, two versions are given.

The machine code programs consist of an Assembler listing, created using ABACUS Software's Assembler/Monitor-64. Following the assembly listings are a Basic loader for the C-64, and a Basic loader for the VIC 20, this will allow all Commodore users to use these programs. ABACUS Software has available an optional disk containing the programs listed in this book. Please refer to the order blank in the back of the book for ordering information.

In the Basic programs, all the control characters have been placed in brackets in order to make the programs easier to read. For example, when you come across [HOME RVSON}, in square brackets, you must press the CLR/HOME key for HOME, followed by the CTRL key with 9 for RVSON.

2. INSTRUCTIONS FOR DATASSETTE HANDLING

Commodore Basic provides 8 instructions for operating the Datassette:

- | | | |
|----|--------|----------------------------|
| 1. | SAVE | Store programs |
| 2. | LOAD | Load programs |
| 3. | VERIFY | Check for correct programs |
| 4. | OPEN | Open file |
| 5. | CLOSE | Close file |
| 6. | PRINT# | Write to a file |
| 7. | INPUT# | Read from a file |
| 8. | GET# | Read in a single byte |

I will describe each command in greater detail in the following sections.

2.1 SAVE

This instruction causes the computer to save the program currently in memory. To tell in which part of memory the program is held, your computer reserves a few memory locations where the start and end of each program is noted.

Immediately after being switched on, your computer sets up a sort of 'notebook', which consists of four pages (0 - 3) and is located at the beginning of memory. Under the control of the computer's operating system, 256 memory locations are grouped together to form each page, as each memory location can have exactly 256 (0 - 255) different values.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

On page 0, i.e. from 0-255, the computer makes a note of the values it uses most.

Memory locations 43, 44 always contain the current start of Basic, and 45, 46 contain the end of a Basic program.

To check this enter PRINT PEEK (43), PEEK (44), PEEK (45), PEEK (46). If you have not loaded a program, and you are using a CBM 64, then,

1 8 3 8

will be displayed on the screen. The meaning of the individual pointers can be seen from the diagram on the next page.

As the computer has 65,536 memory locations, addresses are always expressed in two bytes. The first is, the LSB, or Least Significant Byte, which shows the position within a page, while the second, the MSB, or Most Significant Byte, shows the page number.

Now, back to our example above. Our Basic area starts on page 8 at the first memory location, i.e. memory location $8*256+1 = 2049$.

"But how did the values 3 and 8 get into locations 45 and 46 when I haven't got a program in the computer?", you will say. Well, the reason for that is as follows: The computer must know where the end of a Basic program is, therefore it always adds a couple of zeros to the end of the program as a marker. At the moment, you have a program in the computer which consists of nothing more than a couple of zeros. You can easily check this for yourself.

BASIC RAM Division and Pointers

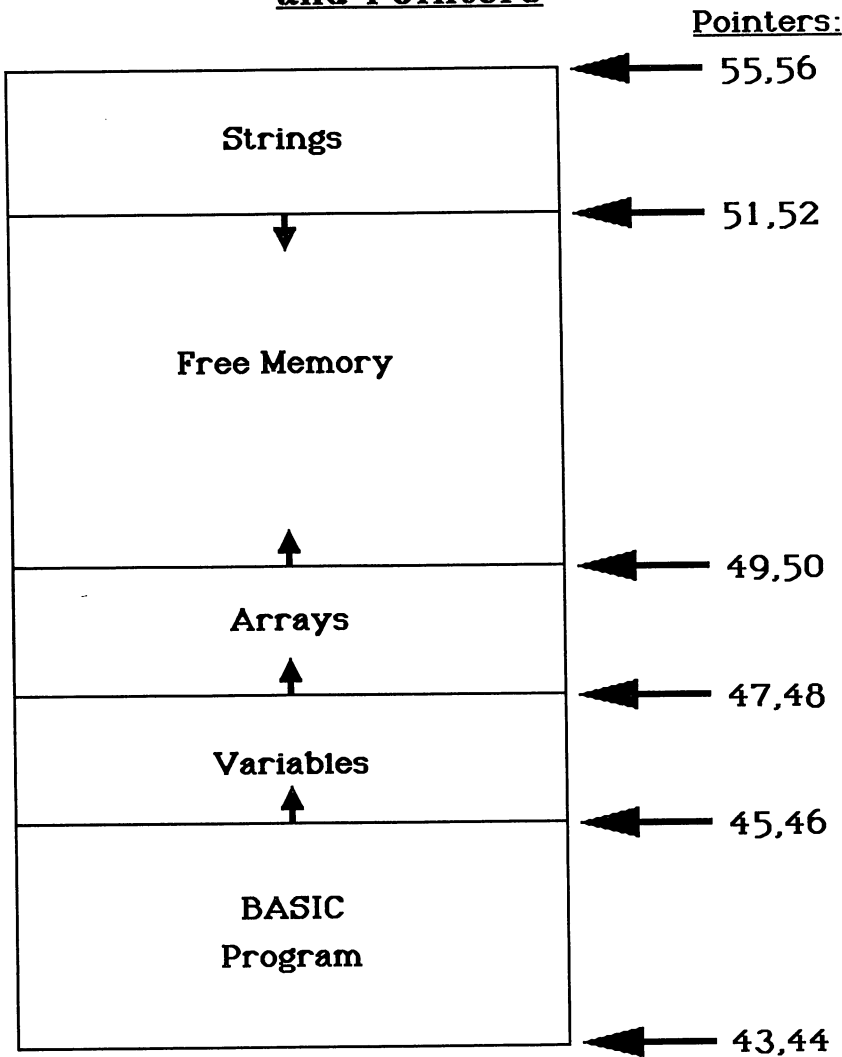


Fig. 1: General structure of BASIC RAM and pointers

If you enter;

```
PRINT PEEK (PEEK(45)+PEEK(46)*256-1)
```

you will get 0. If you enter a line of Basic, for example:

```
10 PRINT "HELLO"
```

and conclude the input with RETURN, and now PEEK the contents of memory locations 43, 44, 45, and 46, you will get;

```
1      8      16      8
```

The computer has changed the old entries in its notebook (zero page) to correspond with your program.

After this little excursion into memory management, let's return to our discussion of the SAVE instruction.

This instruction causes the computer to save everything between the pointers specified above, on tape. The computer doesn't know and doesn't care whether anything is already on the tape or not, as with a normal cassette recorder, if anything is there, it is overwritten.

In order for you find the program again when you have saved it on tape, you can give it a name. The syntax is as follows:

```
SAVE "NAME"
```

You can also enter the string "NAME" into a string variable, which enables you to store the program using the following syntax:

```
A$="NAME"  
SAVEA$
```

This facility can prove useful if you want to use the SAVE instruction in a program. But more on that in Chapter 5.

Having specified a name, you can also follow it with a device address and a secondary address, in each case separated by a comma. The precise significance of the secondary address will be described in Chapter 3.

The device address tells the computer which peripheral device to select, serving much the same purpose as the number on your own front door. The 'house number' of the datassette is 1, and that of a disk drive is 8 or 9.

To save you a bit of work, the CBM-64 has been programmed to select the datassette if no other device address is specified.

2.2 LOAD

This instruction enables you to load the program you have saved on cassette. If you don't specify a program name, the computer will load the first program it finds. However, just as with the SAVE instruction, you can specify a name, and follow it with a device address and secondary address. The rules are the same in both cases.

Following every LOAD instruction in direct mode, the computer automatically updates its notebook (zero page) to correspond to the program just loaded. This is important when you are loading machine language programs. For example, if you are loading a machine language program which is located at the end of Basic memory, then the Basic end vectors will be set accordingly, to the Basic RAM end. Any further input after that will draw the comment "?OUT OF MEMORY ERROR" from the computer. If you want to make any further entries after loading, you must either restore the old values in pointers 45 and 46, or you must enter a NEW instruction so that the program in the Basic area will be deleted, but not the machine language program.

One can prevent the pointers from being reset by loading the machine language program in program mode. You will find more about that in Chapter 5.

2.3 VERIFY

This instruction operates in much the same way as the LOAD instruction. The only difference is that, with this instruction, the program is not saved, but compared, byte for byte, with the program in memory. In this way, you can test whether a program has been successfully saved.

If the save was successful and if there are no errors on the cassette, then "OK" will appear on the screen. If any error is discovered during the comparison, then the message "?VERIFY ERROR" will appear.

2.4 OPEN

So far, I have covered the loading and storing of programs. You can also load and store data. To tell the computer that it is data which is to be loaded or stored, you must first open a file using the OPEN instruction.

With this instruction, you enter all the parameters the computer needs; first the logical file number, the device address, the secondary address, which indicates whether it is a loading or saving operation, and the file name. To open a file, the following instruction must be entered:

OPEN lf,DA,SA,"NAME"

As with SAVE and LOAD, a file can be given a name with "NAME".

SA stands for secondary address:

Using this number, you indicate to the computer whether you wish to send (= 1) or receive (= 0) data.

If you enter a 2, then an EOT (End Of Tape) block is written on completion of storage. Further details of this will be found in Chapter 3.

DA stands for device address:

This number tells the computer which device to work with.

The various device addresses used are as follows:

0	=	keyboard
1	=	cassette
3	=	screen
4,5	=	printer
8,9	=	diskette

lf stands for logical file number:

This number, which may be anywhere between 0 and 255, represents the index for this OPEN instruction. So you don't have to enter the parameters with every input and output instruction, the computer sets up a table of all the parameters, storing them under this logical file number.

Now that we have opened a file, we can take a look at the input and output instructions.

2.5 PRINT# and CLOSE

The PRINT# is used to write individual pieces of data onto tape.

To help make this more understandable, enter the following short program:

```
10 OPEN 1,1,1,"DATA FILE"
20 PRINT "FILE OPENED"
30 PRINT:PRINT"ENTER DATA"
40 INPUT"DATA";A$
50 PRINT#1,A$
60 A=A+1
70 PRINT "DATA ITEM "A" TO INTERMEDIATE STORAGE"
80 PRINT:PRINT"FURTHER DATA (Y/N)"
90 GET F$: IF F$ = "" THEN 90
100 IF F$ = "Y" THEN 40
110 CLOSE 1
120 PRINT:PRINT"FILE CLOSED"
130 END
```

Now insert a blank cassette in your recorder and start the program with RUN. The message "PRESS RECORD & PLAY ON TAPE" will appear at once. If you follow this instruction, a file header will be written onto the tape, in accordance with the OPEN instruction in line 10. This will continue until the name of the file is written. The recorder will then stop and you will be requested by line 40 to enter your data.

In line 50, the string entered is placed into intermediate storage by the PRINT# instruction. The number following

the PRINT# instruction, i.e. the logical file number, represents the connection to the OPEN instruction.

The string is not immediately written onto tape, notice that the tape has not moved. Only when you have entered an entire series of data will the tape move, and then everything entered will be written onto tape. I will come back to the intermediate storage facility, i.e. the cassette buffer, in Chapter 7 and explain it further.

If you answer the question in line 100 with "Y", you can enter more data. If the buffer is already full, then you will notice that there is a pause and the tape will move for a moment. The buffer contents are now written onto tape and the buffer is overwritten with "spaces", (CHR\$(32)), thus ready to receive more data.

When you conclude the input of data, the computer will branch to the CLOSE instruction in line 110. The number following this instruction is another logical file number which tells the computer to close the file opened under logical file number 1. This is done by deleting from the computer's notebook (zero page) all the entries made under the index 1. The computer now writes a 0 byte after the last entry, to signify end of data, and stores the contents of the buffer on tape. It is therefore very important to close a file correctly, otherwise all data will be lost.

So far, we have stored only strings, or the contents of string variables. The question is now: "How does the computer process numerical expressions and variables?".

Well, such expressions are stored as if they were strings, i.e. the computer executes a STR\$ instruction automatically

with each PRINT# instruction. It is the same whether you write:

PRINT#1,55 or A\$=STR\$(55):PRINT#1,A\$

For this reason, you can read any numerically stored value into a string variable at some later stage.

Another important matter is the format of a PRINT# instruction, there are many ways to store several expressions with a single instruction. But I will return to this particular problem after I have dealt with the INPUT# instruction, then it will be easier to understand.

2.6 INPUT#

The behavior of this instruction is completely analogous to that of the 'normal' INPUT instruction, except that, it is not only keyboard which can be read. If, for example, you define the keyboard as the input device (0=keyboard) by entering;

```
OPEN1,0
```

then INPUT#1,A\$ will have the same effect as INPUTA\$, except that no question mark will be printed out.

Just how INPUT# behaves in relation to the cassette recorder will best be seen with the aid of an example program. So rewind your tape back to the point where you stored the file "DATA FILE" and enter the following program:

```

10 OPEN 1,1,0,"DATA FILE"
20 PRINT "FILE OPENED"
30 PRINT:PRINT"DATA BEING READ IN"
40 INPUT#1,A$
50 PRINTA$
60 A=A+1
70 PRINT " DATA ITEM "A" READ IN"
80 PRINT:PRINT"FURTHER DATA (Y/N)"
90 GET F$: IF F$ = "" THEN 90
100 IF F$ = "Y" THEN 40
110 CLOSE1
120 PRINT:PRINT"FILE CLOSED"
130 END

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

If you start the program and press the PLAY key in accordance with the instructions, you will see that the tape runs for a while and then stops. On receiving the OPEN instruction, your computer looks for the file "DATA FILE" and then, on receiving the INPUT# command it transfers the first string to the variable A\$, which is displayed on the screen via line 50.

The individual pieces of data are stored in the buffer in exactly the same way as you wrote them. Due to the PRINT# instruction, each string is followed by a 'Carriage Return' (CHR\$(13)). The PRINT# instruction writes the data into the buffer in exactly the same way as a PRINT instruction would write data to the screen. The INPUT# instruction recognizes, from a comma, or a 'Carriage Return', that the expression is finished, just as any INPUT instruction would.

Byte- 0 1 2 3 4 5 6 7 8 9 10 11...27 28 29 30

Content-S T R 1 CR S T R 2 CR S T R 5 CR 00

Now answer the question whether there is any other data to be read. With our little program, you answer "Y". You can now read in string for string and display the data on the screen until the message "?STRING TOO LONG ERROR" appears.

You will get this message because you have read in the last string on tape, and are now trying to read in another. The computer looks for the end of the string, i.e. a comma or carriage return (CR for short), but does not find one in the following 80 memory locations (you can read a maximum of 80 characters with an INPUT or INPUT# instruction). The computer concludes from this that it has found a string of more than 80 characters and outputs an error message.

To prevent you getting this error message with every 'normal' file program, there is a way of recognizing the end of the file with the aid of the status variable (ST). You will find out how to program the status variable in Chapter 4.

You will get this error message, not only if you try to read past the end of the file, but also if the data was stored in the wrong format.

When discussing the PRINT# instruction, we stored only one expression per PRINT# instruction, with the result that a CR (carriage return) was automatically added to the end of each one. It is possible to store several expressions with a single PRINT# instruction. It is important in this case that the individual expressions are separated from each another. For example, if you write

```
PRINT#1,A$,B$
```

with A\$="STRING1" and B\$="STRING2", then the two expressions will be written into the buffer in exactly the same way as they would be displayed on the screen by a PRINT instruction;

```
Byte  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
ContentS T R I N G 1           S T R I N G 2 CR
```

If you give the following instruction:

```
T$=", "
PRINT#1,A$;T$;B$
```

you will get

Byte 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Contents T R I N G 1 , S T R I N G 2 CR

The strings are now clearly separated from one another and will be read in correctly by an INPUT#1,A\$,B\$ instruction.

I must draw your attention to one peculiarity. The comma is sufficient to guarantee that the data is clearly assigned to a variable by the INPUT instruction, but it is not sufficient to inform the computer that the entry is finished and that a new one is starting with the next byte. It may sound as if I'm saying the same thing, but in fact I'm not, so I shall go into more detail about it.

One could compare the computer with a company which has many different departments. Processing of data when reading and writing is handled by two different departments.

Department 1. - Processing of Variables

This department ensures that an expression anywhere in storage (cassette buffer, screen storage, program, etc.) is assigned to the correct variable in the correct format.

Department 2. - Memory Area Administration

This department always tells the first department where to find the expression it is interested in.

Each of these two departments has its own way of looking for the end of an expression, and thus the beginning of

another. The first department doesn't care whether it finds a comma or a CR. Department 1 knows that both signify the end of one expression and the beginning of another. The second department takes absolutely no notice of the comma at all. It recognizes only the CR as a separator, i.e. it always gives the location following a CR as the starting point of a new expression.

This will become much clearer if we have a program to work with. Enter the following statements, insert an empty tape in the recorder, and start the program with RUN.

```

10   OPEN1,1,1,"TEST"
20   T$=","
30   A1$="STRING1" : A2$="STRING2"
40   A3$="STRING3" : A4$="STRING4"
50   PRINT#1,A1$;T$;A2$;T$;A3$
60   PRINT#1,A4$
70   CLOSE1
80   END
100  OPEN1,1,0,"TEST"
110  INPUT#1,A$
120  PRINT A$
130  INPUT#1,A$
140  PRINT A$
150  CLOSE 1
160  END

```

When the file TEST has been stored, rewind your cassette to the start of the file and start the program with RUN 100. When the computer has read in the file, then the strings will be arranged in the buffer in the following way:

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
S T R I N G 1 , S T R I N G 2 , S T R
19 20 21 22 23 24 25 26 27 28 29 30 31 32
I N G 3 C R S T R I N G 4 C R 00
```

On the screen, you will see:

```
STRING1
STRING4
```

Department 2 set its pointer to the first string on the first INPUT# and Department 1 then assigned it to A\$. There are no other expressions to be read in with this INPUT# instruction, so control is passed back to Department 2. At the next INPUT# instruction, Department 2 looks for the next CR and finds it in front of STRING4. It passes this starting point back to Department 1, which then assigns the expression "STRING4" to A\$.

Now change lines 110 and 120 by adding ',B\$,C\$'. If you now read the file in again, you will see on the screen:

```
STRING1    STRING2    STRING3
STRING4
```

Department 1 now assigns the two following expressions to the variables B\$ and C\$, in line 110, before handing control back to Department 2.

In brief:

The pointer of Department 1 is set to the next separator (comma or CR) by every assignment to a variable.

The pointer of Department 2 is set to the next separator (CR) by every INPUT# instruction.

In conclusion, one can say that a comma is sufficient for separating expressions, so long as data is read in the same format as it was saved.

If you wish to free yourself from this format when reading in data, then the separation string should be defined with CR (Carriage Return).

i.e. 20 T\$=CHR\$(13)

Data stored in this way, will always give you the same result, regardless of whether you read in one or more variables with an INPUT# instruction.

So far, I have only discussed the separation of expressions. In many applications it makes sense not to separate the expressions. This is achieved by ending the PRINT# instruction with a semicolon, just as one does with the PRINT instruction for screen display. Let's try it out with the short test program we used in the previous example, with the following line changes:

```
10    OPEN1,1,1,"TEST2"  
50    PRINT#1,A1$;A2$;A3$  
100   OPEN1,1,0,"TEST2"
```

Delete lines 130 and 140. When you store file TEST2 with RUN and load it with RUN 100, you will get:

STRING1STRING2STRING3STRING4

If this leads to strings which are longer than 80 characters, then these cannot be read with the INPUT# instruction. They can only be read in, byte by byte, with the GET# instruction.

2.7 GET#

The manner in which this instruction operates is also analogous to the GET instruction. The GET# command is used to read in data byte by byte. Enter the following program:

```

10 OPEN1,1,0,"TEST"
20 GET#1,A$
30 PRINTA$;
40 GOTO 20

```

Now rewind to the start of file TEST and start the program. When the computer has found the file, the following character pattern will appear, letter by letter:

```

STRING1      STRING2      STRING3
STRING4

```

If any other characters are read in now, you will not notice any change on the screen, as the buffer is filled with "SPACES". The characters can be made visible with the following program change:

```

30 PRINT A$,ASC(A$+CHR$(0))

```

This will print every character as a letter and as an ASCII value. When the GET# instruction encounters a zero byte, an empty string is created, i.e. "". To prevent an error message when converting to ASCII code, CHR\$(0) is always added to the variable, i.e. A\$+CHR\$(0).

Now rewind and start the program again. The following will now appear on the screen:

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

S	83	
T	84	
R	82	
I	76	
N	78	
G	71	
1	49	
		44
S	83	
.		
.		
N	78	
G	71	
4	52	
		13
		0
		32
.		
.		
.		

The zero after STRING4 indicates end of file. After that, there is nothing but "SPACE".

3. SECONDARY ADDRESS

As I mentioned in Chapter 2, there is, in addition to the device address, also a secondary address. This address is generally used to transfer, to the computer or peripheral device, when operating in a receiving mode, an additional instruction regarding the type of operation.

Before going any further, I would like to make a few fundamental remarks on the two different types of programs. The first type are BASIC programs. These are not dependent in any way on the area they are stored in. They are relocatable.

With the VIC 20, for example, the position of the Basic memory area will depend on how the computer is expanded. In the basic version, this area starts at location 1025. When the memory is expanded by 8k bytes, then the Basic area will start at location 1609. But in spite of this, most programs written for the basic version will run on the expanded version.

To ensure that these programs will always be loaded at the start of the Basic memory area, both the VIC 20 and C-64 have a relocating loader, which automatically loads all Basic programs at the start of the Basic memory area.

The other type of program is the machine language program. This type of program runs only in the memory area for which it was written. Relocatable loading is usually not permitted.

To distinguish between Basic and machine language programs, the secondary address is used in cassette operations. With the SAVE function, the secondary address performs the following functions:

I. SAVE

0 - The stored program is identified as a Basic program, ensuring that this program is automatically loaded at the beginning of the Basic area available at the time, i.e. relatively loaded, by the LOAD instruction.

1 - It identifies a program as a machine language program. When you load a program that is stored in this way, the LOAD instruction automatically loads it at the address from which it was saved.

2 - It stores a program as a Basic program, in the same way as secondary address 0. In addition, the computer writes an EOT (End Of Tape) block after the program. This block tells the computer when loading, that it should stop searching for further programs at that point. If the computer should come across such a block without having first found a program, it prints out a message "?FILE NOT FOUND ERROR".

3 - It identifies a program as a machine language program and writes an EOT block after it.

With the LOAD instruction, the secondary address has the following functions:

II. LOAD

0 - The program is loaded into the start of BASIC, which are loaded relocatably.

1 - Every program, using the header information, is loaded absolutely, i.e. at the address from which it was saved.

With the OPEN instruction there are the following secondary addresses:

III. OPEN

0 - Opens a file for reading.

1 - Opens a file for writing.

2 - Opens a file for writing and writes an EOT block after the file

4. STATUS VARIABLE

As I have explained in the previous chapters, there is a reserved variable that your Commodore uses to provide you with information about the progress of a file operation. This status variable (ST) is set each time a cassette or other peripheral device is accessed. It enables you to specify more precisely the nature of any error that may occur. It will also help you to determine when you have reached the end of a file.

This variable consists of eight individual "flags", which are set according to the errors which occur. Each individual bit of this variable can be tested using an AND operation. Further details of the number representations and logical functions will be found in the book Tips and Tricks for the Commodore 64 from ABACUS Software.

The precise meaning of each flag is explained in Table 1.

In order to be able to understand the fourth bit of the status variable, one must know that the computer writes all programs and data on tape twice, and compares the two versions with each other when loading. Further details of this will be found in Chapter 8.

How can you use the status variable in your programs? As you can see from the Table 1, bit 6 tells you whether you have come to the end of a file. You can check this bit during the course of a program and stop reading in data after the last string.

Table 1

ST bit	ST decimal equivalent	Meaning
0	1	No meaning for cassette operations
1	2	Ditto
2	4	Short block. Block shorter than it should be.
3	8	Long block. Block longer than it should be.
4	16	Second pass error. Data in first pass does not agree with data in second pass.
5	32	Check sum error. Check sum in memory does not agree with computed check sum.
6	64	End of file.
7	128	End of tape. EOT read.

Now enter the following program and read the file set up in Chapter 2. After STRING4 has been printed out, the message READY and the cursor will appear.

```

10 OPEN1,1,0,"TEST"
20 INPUT#1,A$
30 PRINT A$
40 IF (ST AND 64) THEN 60
50 GOTO20
60 CLOSE1
70 END
    
```

4.1 SAVING PROGRAMS AFTER "LOAD ERROR"

No doubt it has happened that your computer has printed a "?LOAD ERROR" after loading, while the program has been only partially loaded.

The reason for this is, usually, that the position of the sound head relative to the tape during writing was different to what it was during reading. The only remedy for this is to adjust the head as described in Chapter 11.5.

If your program still refuses to load after readjustment of the sound head, then the fault will lie with the tape or faulty storage of the program. In certain circumstances, it may still be possible to rescue at least part of the program.

When you store a program with your computer, it is written to tape twice in succession. When reading, the first version is loaded into memory and then compared with the second version, if the computer discovers any discrepancy or error, it tries to correct it. If correction is not possible, it sets bit 4 of the status variable and prints a "?LOAD ERROR" message.

Sometimes the computer is unable to read individual bits, or the byte identification, which has resulted in bad synchronization between the reading routine and the verify routine. If this occurs, then bits 2 and/or 3 are set.

When an error occurs the Basic vectors 45 and 46 (end of program) are not set in the usual way, if the error has

occurred late in the program or only on the second pass, then you will be able to list all of the program or at least part of it.

This portion that you have been able to list can be saved by means of an UNNEW instruction.

This instruction is not included in Commodore Basic 2.0. If you do not have a Basic extension which includes this instruction, then you can enter the program printed at the end of this chapter. If you do not have an ASSEMBLER program, then enter the Basic loader and save it before running the program. Start the loader, you can now save the machine language program from the indicated start address to the end of the Basic RAM (see Chapter 6). You can now load the saved machine language program, whenever needed, simply by means of the LOAD "NAME",1,1 instruction and start it with SYS (start address), thus enabling you to restore a deleted program or rescue at least part of a program that could not be loaded without errors.

Now that you have restored the program and vectors with the UNNEW instruction, save it on another cassette. We now come to the most difficult part: you have to find out which part of the program is damaged. This debugging is done in the same way as for any other Basic program.

As I remarked at the beginning of the chapter, the most frequent cause of a "?LOAD ERROR" is faulty adjustment of the sound head. In such cases you should first try to set the head correctly and if necessary repeat the procedure described above with different settings of the head until you can load the program with the minimum number of errors.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
100 REM UNNEW 64
110 PS=0:E=256*PEEK(56)+PEEK(55)-1:A=E-51
120 FOR I=A TO E:READ X:PS=PS+X:POKE I,X:NEXT
130 IF PS<>5274 THEN PRINT"DATA ERROR":END
140 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
150 PRINT"START ADDRESS"A:NEW
160 DATA 165,43,164,44,133,34,132,35,160,
      3,200,177,34,208,251,200,152,24,101
170 DATA 34,160,0,145,43,165,35,105,0,200
      ,145,43,32,51,165,165,34,105,2,133
180 DATA 45,165,35,105,0,133,46,32,99,166
      ,76,123,227
```

```
100 REM UNNEW 20
110 PS=0:E=256*PEEK(56)+PEEK(55)-1:A=E-51
120 FOR I=A TO E:READ X:PS=PS+X:POKE I,X:NEXT
130 IF PS<>5319 THEN PRINT"DATA ERROR":END
140 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
150 PRINT"START ADDRESS"A:NEW
160 DATA 165,43,164,44,133,34,132,35,160,
      3,200,177,34,208,251,200,152,24,101
170 DATA 34,160,0,145,43,165,35,105,0,200
      ,145,43,32,51,197,165,34,105,2,133
180 DATA 45,165,35,105,0,133,46,32,99,198
      ,76,103,228
```

ASSEM-MON 64 V2.0 Page 1

```

120:                ;
130:                ; UNNEW INSTRUCTION
140:                ;
150:                ; CALL UP WITH SYS START ADDRESS
160:                ;
170:                ;
172:    7000                .OPT P1,00
180:    7000                .PAG 61
182:    A663                CLR    =    $A663    ;($C663)
184:    E37B                BASIC =    $E37B    ;($E467)
186:    A533                TIE   =    $A533    ;($C533)
200:    7000                *=    $7000
210:    7000 A5 2B          LDA    $2B        ;BASIC START IN
                                   BUFFER
220:    7002 A4 2C          LDY    $2C
230:    7004 85 22          STA    $22        ;WRITE
240:    7006 84 23          STY    $23
250:    7008 A0 03          LDY    #3        ;NULL BYTE SEARCH
260:    700A C8             NULL INY
270:    700B B1 22          LDA    (*22),Y
280:    700D D0 FB          BNE    NULL
290:    700F C8             INY                ;LINE LENGTH
292:    7010 98             TYA
294:    7011 18             CLC
300:    7012 65 22          ADC    $22        ; + BASIC START
                                   GIVES
310:    7014 A0 00          LDY    #0        ;LINK ADDRESS
                                   OF FIRST LINE
320:    7016 91 2B          STA    ($2B),Y   ;WRITE LINK
                                   ADDRESS TO
330:    7018 A5 23          LDA    $22 + 1   ;BASIC START
340:    701A 69 00          ADC    #0
    
```


ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
350:    701C CB          INY
360:    701D 91 2B      STA    ($2B),Y
370:    701F 20 33 A5   JSR    TIE          ;RECALCULATE
                                LINK ADDRESSES
460:    7022 A5 22      LDA    $22          ;END ADDRESS IN
                                $22/$23
470:    7024 69 02      ADC    #2           ; + 2 EQUALS BASIC
                                END VECTOR
480:    7026 85 2D      STA    $2D
490:    7028 A5 23      LDA    $22 + 1
500:    702A 69 00      ADC    #0
510:    702C 85 2E      STA    $2E
520:    702E 20 63 A6   JSR    CLR          ;SETS REMAINING
                                POINTERS
530:    7031 4C 7B E3   JMP    BASIC        ;RETURN TO BASIC
U7000-7034
```


5. LOADING AND SAVING UNDER PROGRAM CONTROL

When discussing the use of the SAVE and LOAD instructions in Chapter 2, I described how programs were saved and loaded in direct mode. These instructions can also be used in program mode.

When used in a program, the SAVE instruction operates in the same way as in direct mode. It is therefore possible for a program to SAVE itself. You will probably use this only on rare occasions. You will use it much more frequently for storing only part of a Basic program, or a whole machine language program.

Imagine that you have written a program which generates data statements from line 10000 onwards, and that you want to save this part of the program alone, to be able to add it to other programs. You must first determine the memory location of line 10000 in memory, since it is from here that you wish to start saving. The problem can be solved with the following program:

```

1000 S = PEEK(43) + 256 * PEEK(44) : BS = S
1010 LN = PEEK(S + 2) + 256 * PEEK(S + 3)
1020 IF LN = 10000 THEN 1050
1030 S = PEEK(S) + 256 * PEEK(S + 1)
1040 GOTO 1010

```

To understand these lines of programming, a little explanation is required on how Basic lines are stored. The first two bytes of a Basic line form the line-link address. This gives the starting address of the next Basic line, i.e. points to the next line address. The next two bytes contain

the line number. This is followed by the contents of the Basic line, which concludes with a zero byte. The Basic program is thus stored in the following manner:

```
ADL ADH LNL LNH Basic text 00 ADL ADH LNL LNH ....
```

where

```
ADL  = line-link address LSB  
ADH  = line-link address MSB  
LNL  = line number LSB  
LNH  = line number MSB
```

In our program, the program start address, from the vectors in S and BS, is read in by line 1000. In line 1010, the line number is transferred to the variable LN. Line 1020 checks whether the required line number was reached, if so, control branches to a further part of the program. If not, then line 1030 causes the starting location of the next Basic line to be transferred from the line-link address to S, and control returns to 1010 via 1040.

Using this program, you will have all the information required to save the last part of your program. There is only one important thing still to attend to, on completion of the saving process, the old values must be written back to the Basic start vectors. The sub-program "DATA LINES" can be saved with the following subroutine:

```
1050 POKE 43,S AND 255: POKE 44,INT(S/256) :REM SET  
      POINTER TO LINE 10000  
1060 SAVE"DATA LINES" :REM STORE DATA LINES  
1070 POKE 43,BS AND 255: POKE 44,INT(BS/256) :REM RESET  
      POINTER
```

If you wish to store machine language programs, the whole process becomes a bit more complicated. You will find more about that in Chapter 6.

5.1. OVERLAY TECHNIQUE

The LOAD instruction operates in a different manner when used in program mode than it does in direct mode. Chapter 2 explained that the Basic pointers are set to the new program after a LOAD instruction. However, if the LOAD instruction is given as part of a program, then the vectors are retained, as well as all variables defined up to that point.

It is therefore possible to call up various subroutines from a main program, all of them operating with the same variables. This method is known as an overlay technique.

There are a number of things you must know when using this overlay technique:

1. As the Basic vectors are not changed, the first program must be at least as long as the program that will be overlaid.

The variables are stored directly after a Basic program and vectors 45 and 46 is also the pointer for the start of the variable table. However, if the program overlaid is longer, then the variables are overwritten and the pointer points to the new program. As soon as a variable operation takes place, the variable is sought in the program, which could lead to a system "crash".

However, there is a technique which will enable you to overlay a longer program from a shorter one.

First load the longest program to be overlaid in direct mode and determine its length by reading out memory locations 45 and 46. Make a note of these values, to be on the safe side. Then load the calling program, prefacing it with the following Basic line:

```
10 POKE 45,W1: POKE 46,W2: CLR
```

The line number is optional, but it must be the lowest number in the program. W1 and W2 stand for the values you had noted from locations 45 and 46. As a result of the POKE instructions, you have artificially lengthened your program. The CLR causes the other pointers to be changed accordingly.

2. You must be careful that, after a LOAD instruction, the program is executed from the beginning again.

This only makes sense, and does not cause any problems if you are calling up Basic programs. If you wish to call up a machine language program, the Basic program in memory does not change and will start again repeatedly, to prevent this you will have to use a branch table. In the following program, three machine programs are loaded at the beginning of the program, before reaching the main program.

```
20 IF A = 0 THEN A=1: LOAD"MPR1",1,1
30 IF A = 1 THEN A=2: LOAD"MPR2",1,1
40 IF A = 2 THEN A=3: LOAD"MPR3",1,1
50 REM MAIN PROGRAM
```

3. The last of thing to watch out for is a peculiarity of string storage.

Your Commodore operating system is designed so that as little memory as possible is taken up by strings. Every time you assign a string to a string variable, a pointer indicating the corresponding string is filed in the variable table under the variable name. If you assign a string to a variable by means of an INPUT instruction, then this string is filed in a special table at the end of the Basic RAM. If this assignment takes place during a program, e.g. by means of;

```
100 A$="STRING"
```

the string is not transferred to the string table, but the pointer is set to the position of the string in the program.

If you overlay a program, something quite different will be found at the corresponding location and on receiving a PRINT A\$ instruction, the computer will print out garbage. The situation can be remedied by the simulated definition of a new string, by linking the string defined in the program with an empty string. So use;

```
100 A$="STRING": A$=A$+""
```

This way, the operating system will transfer the string to the string table.

6. STORAGE OF MACHINE LANGUAGE PROGRAMS

When you have spent a little more time working with your computer, you will one day come up against the problem of transferring the contents of certain memory areas to a storage device. Such memory areas could be, for example, the contents of a screen display, or a machine language program. Unfortunately, Commodore Basic does not support this operation at all. Proper programming techniques can do this, as you have seen in Chapter 5.

In the last chapter, I mentioned that, with a SAVE instruction, the computer always saves the contents of memory between the pointers in 43, 44 and 45, 46. What could be more obvious than to change these pointers in such a way that they point to the beginning and end of the memory area you were interested in. However, if you still want to operate 'normally' with your computer after this SAVE operation, then you will have to be very careful.

If the computer is to operate normally, after this type of saving, the old values must be stored in the correct vectors. In short, you will have to store these values and transfer them back to pointers 43 - 46 when the SAVE operation is complete.

To save the contents of the pointers, you must not transfer them to variables, as the program end vector is also the variable start vector. When you change this pointer, the computer is unable to find any of the variables. For this reason, the contents of the pointers must be POKEd into memory locations that are not affected by the program operation or the operating system.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

In the following example, I will be using the memory locations 251 - 254. These locations are not used by the operating system. This program will enable you to write the contents of the screen onto a cassette:

```
10 POKE251, PEEK(43): POKE252, PEEK(44): REM SAVE START
20 POKE253, PEEK(45): POKE254, PEEK(46): REM SAVE END
30 POKE43,0: POKE 44,4: REM VIDEO RAM START AS START
40 POKE45,232: POKE 46,3: REM VIDEO RAM END AS END
50 SAVE "SCREEN",1,1 : REM STORE VIDEO RAM
60 POKE43, PEEK(251): POKE44, PEEK(252):REM RESTORE START
70 POKE45, PEEK(253): POKE46, PEEK(254):REM RESTORE END
```

As you can see, the whole thing is quite complicated. It is much easier if you use a machine language program:

```
SYS adr"NAME",DA,sadr,eadr
```

where

- adr = start address of machine language program
- NAME = name of area to be stored
- DA = device address (cassette = 1)
- sadr = start address of area to be stored
- eadr = end address

The program listed below will save the memory area, without having to manipulate the vectors in any way.

The Basic loaders shown store the machine code program at the end of the Basic RAM and print out the start address. But before you start the loader, you should also save it, since it automatically clears itself.

ABACUS Software . . . CASSETTE BOOK for the COMMODORE 64 and VIC

I should like at this point to explain a few things about the machine code program.

In line 310, an operating system routine, which reads the name of the screen and writes it into the appropriate memory locations, is called. The routine which is called in line 320 checks whether the next character is a comma and transfers the following expression to the X-register.

In line 330, the secondary address is loaded into the Y-register and in line 340 the logical file number is loaded into the accumulator. With these parameters, a routine is called which sets the file parameters. This routine must be called before each load and save operation.

In lines 350 - 380, a check is made for a comma, to read in and process the start address. This is filed away in memory locations \$14 and \$15. The contents of \$14/\$15 are saved in \$FB/\$FC, after which the end address is read in.

These parameters are transferred to the SAVE routine by transferring the end addresses to the X and Y registers in absolute form, while the accumulator contains the zero page position which holds the start address.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
100 REM SAVE ADDRESS 64
110 E=256*PEEK(56)+PEEK(55)-1:A=E-47
120 FOR I=A TO E:READ X:POKE I,X:NEXT
130 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
140 PRINT"START ADDRESS"A:NEW
150 DATA 32,87,226,32,0,226,160,1,169,0,3
    2,186,255,32,14,226,32,138,173,32,247
160 DATA 183,166,20,164,21,134,251,132,25
    2,32,14,226,32,138,173,32,247,183,169
170 DATA 251,166,20,164,21,76,216,255
```

```
100 REM SAVE ADDRESS VIC 20
110 E=256*PEEK(56)+PEEK(55)-1:A=E-47
120 FOR I=A TO E:READ X:POKE I,X:NEXT
130 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
140 PRINT"START ADDRESS"A:NEW
150 DATA 32,84,226,32,253,225,160,1,169,0,3
    2,186,255,32,11,226,32,138,205,32
160 DATA 247,215,166,20,164,21,134,251,132,
    252,32,11,226,32,138,205,32,247,215
170 DATA 169,251,166,20,164,21,76,216,255
```

```

101:    C000
102:    C000
103:      ;                SAVE ADDRESS
104:      ;
105:      ;                START WITH
107:      ;
110:      ;    SYS ADDRESS "NAME",DA,START ADD., END ADD.
111:      ;
112:    C000
112:    C000
114:      ;    VALUES IN BRACKETS APPLY TO
117:    C000
117:    C000
119:      ;                VIC 20
120:    C000
120:    C000
125:    C000                *=    $C000
130:    E257  SETFNAME      =    $E257  ; ($E254)
140:    E200  HBYTE         =    $E200  ; ($E1FD) COMMA TEST
                                   AND BYTE => X

150:    FFBA  SETLFS       =    $FFBA
160:    E20E  CHRIN        =    $E20E  ; ($E20B) COMMA TEST
170:    ADBA  FRMNUM       =    $ADBA  ; ($CDBA)
180:    B7F7  FACINT       =    $B7F7  ; ($D7F7) FAC TO
190:    FFD8  SAVE         =    $FFD8  ;          $14,$15
200    00FB  BUFFER      =    $FB
300:    C000
300:    C000
310:    C000 20 57 E2      JSR    SETFNAME ; GET FILE NAME
320:    C003 20 00 E2      JSR    HBYTE   ; GET GA
330:    C006 A0 01        LDY    #1      ; SA
340:    C008 A9 00        LDA    #0      ; LFN

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
350:    C00A 20 BA FF      JSR   SETLFS ; SET FILE PARAMETER
360:    COOD 20 0E E2      JSR   CHRIN  ; GET START ADDRESS
370:    C010 20 BA AD      JSR   FRMNUM ; AND EVALUATE
380:    C013 20 F7 B7      JSR   FACINT
390:    C016 A6 14         LDX   $14
400:    C018 A4 15         LDY   $15
410:    C01A 86 FB         STX   BUFFER ; SAVE IN BUFFER
420:    C01C 84 FC         STY   BUFFER+1
430:    C01E 20 0E E2      JSR   CHRIN  ; GET END ADDRESS
440:    C021 20 BA AD      JSR   FRMNUM ; AND EVALUATE
450:    C024 20 F7 B7      JSR   FACINT
460:    C027 A9 FB         LDA   #BUFFER; TRANSFER START ADD
470:    C029 A6 14         LDX   $14      ; TRANSFER END ADD.
480:    C02B A4 15         LDY   $15
490:    C02D 4C D8 FF      JMP   SAVE
```

UC000-C030

7. CASSETTE BUFFER

In this chapter, I would like to look at the cassette buffer in greater detail. This buffer takes up the memory area from 828 (hexadecimal 033C) to 1019 (hexadecimal 03FB). This buffer has two functions, in the first the file header is generated and then stored before each file, irrespective of whether it is a program or data file. The buffer then serves as file identification and contains the following parameters:

- | | | | | |
|----|----------------|-----------|---|---------|
| 1. | File type | byte 0 | = | 828 |
| 2. | Start address | byte 1, 2 | = | 829,830 |
| 3. | End address | byte 3, 4 | = | 831,832 |
| 4. | File name from | byte 5 | = | 833.... |

The remaining bytes are filled with SPACES.

The cassette buffers second function is as intermediate storage for data protection purposes. The data to be saved is first written into this memory area and, when the buffer is full, written onto tape. When you load data, the data is loaded a block at a time into the buffer and then read from the buffer by means of INPUT# or GET#, as described in Chapter 2.4.

The file type is coded in the following manner:

- | | | |
|---|---|------------------------------------|
| 1 | = | Basic program, loaded relatively |
| 2 | = | Data block |
| 3 | = | Machine program, loaded absolutely |
| 4 | = | Data header |
| 5 | = | EOT block |

7.1 CREATING A LIST OF CASSETTE CONTENTS

The header contains all the important data required for purposes of identification. This data can be used to catalog the contents of cassettes, on which programs and data are stored.

The program at the end of this section will output a list of the contents of a cassette. If you have a printer, you can print the contents.

PROGRAM DESCRIPTION

Lines 100 - 160 are intended for initialization. The individual tabulator stops are read into Tx\$ and the various file types into field TN(x)\$.

In case output is to be transmitted to a printer as well, a printer channel is opened in line 180 and the heading transmitted to this device in lines 190 - 220. A file is then opened in line 230, for reading in a cassette file without a name. This causes the next header on tape to be read in.

In lines 240 - 270, the type byte, start and end address and file name are read from the buffer by the PEEK statement and formatted. In lines 280 - 290, the length of the program in Kbytes is computed from the start and end addresses.

Lines 300 - 320 cause the data to be output on the printer and the screen.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

In line 330, the counter reading is computed on the basis of the program lengths. When the file is closed, the program branches back to the OPEN instruction.

The last program or file on the tape should have an EOT (End Of Tape) marker or the program will continue to run even after the tape stops.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

100 REM                LIST OF CONTENTS
110 PA=828:REM START ADDRESS IN CASSETTE BUFFER
120 PRINT"[DOWN2]PRINT ? [Y/N] ";D$
130 GET D$:IF D$=""THEN 130
140 T1$=CHR$(16)+"05":T2$=CHR$(16)+"10":T3$=CHR$(16)+"20":
    T4$=CHR$(16)+"30"
150 T5$=CHR$(16)+"38":T6$=CHR$(16)+"50":T7$=CHR$(16)+"63"
    :REM TABULATOR STOPS
160 TN$(0)="RELATIVE":TN$(2)="ABSOLUTE":TN$(3)="FILE"
170 IF DS<>"Y" THEN 230
180 OPEN 2,4
190 PRINT#2,T1$"LFN" T2$"COUNTER" T3$"TYPE" T4$"K-BYTE" T5$"STAR
    T" T6$" END";
200 PRINT#2,T7$" NAME"
220 PRINT#2
230 OPEN 1,1
240 TY=PEEK(PA):TY$=TN$(TY-1)
250 A$=" :"+RIGHT$(" "+STR$(PEEK(PA+1)+(256*PEEK(PA+2))
    -1),6)
260 B$=" -"+RIGHT$(" "+STR$(PEEK(PA+3)+(256*PEEK(PA+4))
    -1),6)
270 C$="":FOR I=5 TO 20:C$=C$+CHR$(PEEK(PA+I)):NEXT
280 T=VAL(RIGHT$(B$,6))-VAL(RIGHT$(A$,6))
290 K$=RIGHT$(" "+STR$(INT(T/1024*100)/100),6)
300 PRINT Z;TY$;K$;A$B$:PRINT C$:PRINT
310 N=N+1:N$=RIGHT$(" "+STR$(N),3)
315 IF D$<>"Y"THEN 330
320 PRINT#2,T1$;N$;T2$;Z;T3$;TY$;T4$;K$;T5$;A$;T6$;B$;T7$;C$
330 Z=INT(Z+(T/212)+4+2*(N/100))
    :REM COMPUTE TAPE COUNTER
340 CLOSE 1
350 GOTO 230

```

7.2 DISPLAY OF FILES FOUND

The ability to read the name of a file by means of the PEEK instruction during the course of a file management program can also be useful. Have you ever been annoyed to find that, when you tried to read in a certain file during the course of a program, the computer took an age to go through the files, only to throw up a message "?FILE NOT FOUND"? The reason may have been that you had rewound too far. However, if the computer lets you know what file it has found, then you can quickly tell whether you had rewound too far or not. Always assuming, of course, that you have a reasonable idea of the sequence of files on the tape.

The following lines of Basic will induce the computer to display the name of every file it finds.

```

100 INPUT"FILE NAME";F$
110 L=LEN(F$)
120 FE$=""
130 OPEN 4,1,0,FE$
140 FOR I=0 TO 15:FE$=FE$+CHR$(PEEK(833+I)):NEXT
150 PRINT:PRINT"FOUND ";FE$
160 IF LEFT$(FE$,L)<>LEFT$(F$,L) THEN CLOSE 4:GOSUB
    200:GOTO 120
170 PRINT:PRINT"FILE MAY BE LOADED":END
200 ?"PRESS ANY KEY TO CONTINUE"
210 POKE 198,0:WAIT 198,1:RETURN
    
```

The OPEN instruction without a name, in line 130 causes the computer to load every data header into the buffer. In line 140 the file name is transferred from the buffer to FE\$ by means of the PEEK command and displayed in line 150. If comparison of FE\$ and the required file name in line 160 produces a negative result, the search is continued, otherwise, the program ends.

7.3 SELF-STARTING PROGRAMS AND PROGRAM PROTECTION

So far, we have only read out data from the cassette buffer. However, this buffer is admirably suited to take data or programs. It is therefore possible to use this buffer to store other data, or a program, using another program.

The buffer covers the area in memory from 828 to 1019. It is therefore 191 characters long. A maximum of 21 bytes is required to transfer header information (file name, start address and end address, type byte = 16+2+2+1 bytes), leaving no less than 170 bytes free, quite enough to accommodate a small machine language program.

As an example, I would like to describe a small program which generates a program which will automatically load and start the next program on cassette, just as you wish.

Such programs can also be used to protect your programs. With such a program one could relocate the Basic RAM before loading the main program, in order to file a few pages of graphics before the main program for example. There is no limit to what the programmer can do here. But let's get back to our example.

I have chosen this example because, on the one hand, it illustrates the general principle, while, on the other hand, it can still be used for general purposes by a user with little interest in machine language programming.

This is a description of the program:

In line 10, the file name under which the loader program is to be stored is checked. In lines 20 and 30, the file name is extended to a length of 16 characters by filling it up with "SPACES". A machine code program, in the data lines from 180 onwards, is appended to the file name by lines 40 to 70.

As the programs for VIC 20 and C-64 are identical except for the data, only the data lines for the C-64 have been printed out by the loader program.

Lines 80 and 90 give you a chance to enter Basic instructions which will be automatically executed after loading the program with "NAME",1,1.

If you want to have a Basic program loaded and started in this way, then enter LOAD RETURN RUN RETURN RETURN using the following brief instructions:

L (SHIFTED-O)↑ R (SHIFTED-U)↑↑

For reasons which I shall explain at a later stage, it is not possible in this program to transfer more than 10 characters. These characters are then appended to the file name following the machine program (lines 100 - 150).

In line 160, the pointer to a system routine - the get character loop - is changed to the machine language program in the cassette buffer.

Line 170 then stores this vector with the aid of the machine language program described in the previous chapter (SAVE ADDRESS).

You must therefore write the start address of your SAVE ADDRESS program after the SYS instruction and be sure that the SAVE ADDRESS program is in memory.

The load instruction causes the file name together with the machine language program and Basic instructions to be written into the cassette buffer and onto tape. When the end of the program is reached, the computer normally jumps via the vector in 770 and 771 to the input holding loop. However, since this was set to our machine program in the cassette buffer in line 160, the computer jumps to the cassette buffer and processes the program. As we gave the computer a LOAD instruction in the output string, it now tries to load the next program. This can be interrupted by depressing the RUN/STOP key in order to store the main program behind the loading program which is to be loaded by the program just created.

When you have saved a program after the loader, rewind the tape to the start of the loading program and enter LOAD and RETURN.

Your computer will now find the loading program and load it, and immediately afterwards load and start the following program.

For those of you who are more advanced in programming or are particularly interested in such matters, I shall now explain the program and techniques used in detail.

The three fundamental aspects of this technique are as follows:

1. Transfer of a machine code program to the cassette buffer by appending to a file name
2. Changing of vectors, e.g. the get character loop vector to point to one's own machine program, and storage of these changed vectors-- Before the computer jumps to the loaded program, the vector pointing to one's own machine language program must be reset to its old value.
3. Memory areas stored as machine programs are always loaded at the point from which they were transferred to storage.

In memory locations 768 - 819, the computer has a series of vectors which enable it to jump to certain subroutines. By changing these pointers it is possible to insert your own program.

The get character loop is the subroutine which waits patiently for your input. Whenever input is presented, the get character loop evaluates and executes it before returning to the loop again.

Our program generates a program which overwrites the get character loop vector with the start address of your own program, written into the cassette buffer with the file name.

If you want to experiment on your own, you could also place your machine program in the area \$2C0 - \$2FF (704 - 766), for example, and store the contents of area \$2C0 - 330 as the program.

But let us now take a look at the machine language program. The vector for the get character loop is reinstated in lines 200 - 230. Your input is stored in storage locations commencing at \$036C.

This input is transferred to the keyboard buffer in lines 250 - 290. The jump to the get character loop at the end of the program causes the computer to execute the instructions in the keyboard buffer.

The example given here is pretty simple but has the disadvantage that, due to the limited space available in the keyboard buffer, ten characters, only a few instructions can be transferred. However, if you have understood the principle, it would not be too difficult to alter this program to suit your own requirements by writing your own routine, starting with line 240 of the machine program, and concluding it with a jump to the get character loop.

I am sure you will already have realized that this technique lends itself well to program protection, as the program starts itself. If you also suppress abnormal termination of a program by means of the RUN/STOP key and change the main program in such a way that it will only run in conjunction with the loader, then an unauthorized user would be unable to gain access to your program.

The whole purpose of this protection would be frustrated if I were to give precise details of how to go about it. I hope that what I have taught you here, together with the experience you will acquire in time in machine language programming, will enable you to write your own autostart routines.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

ASSEM-MON 64 V2.0

Page 1

```

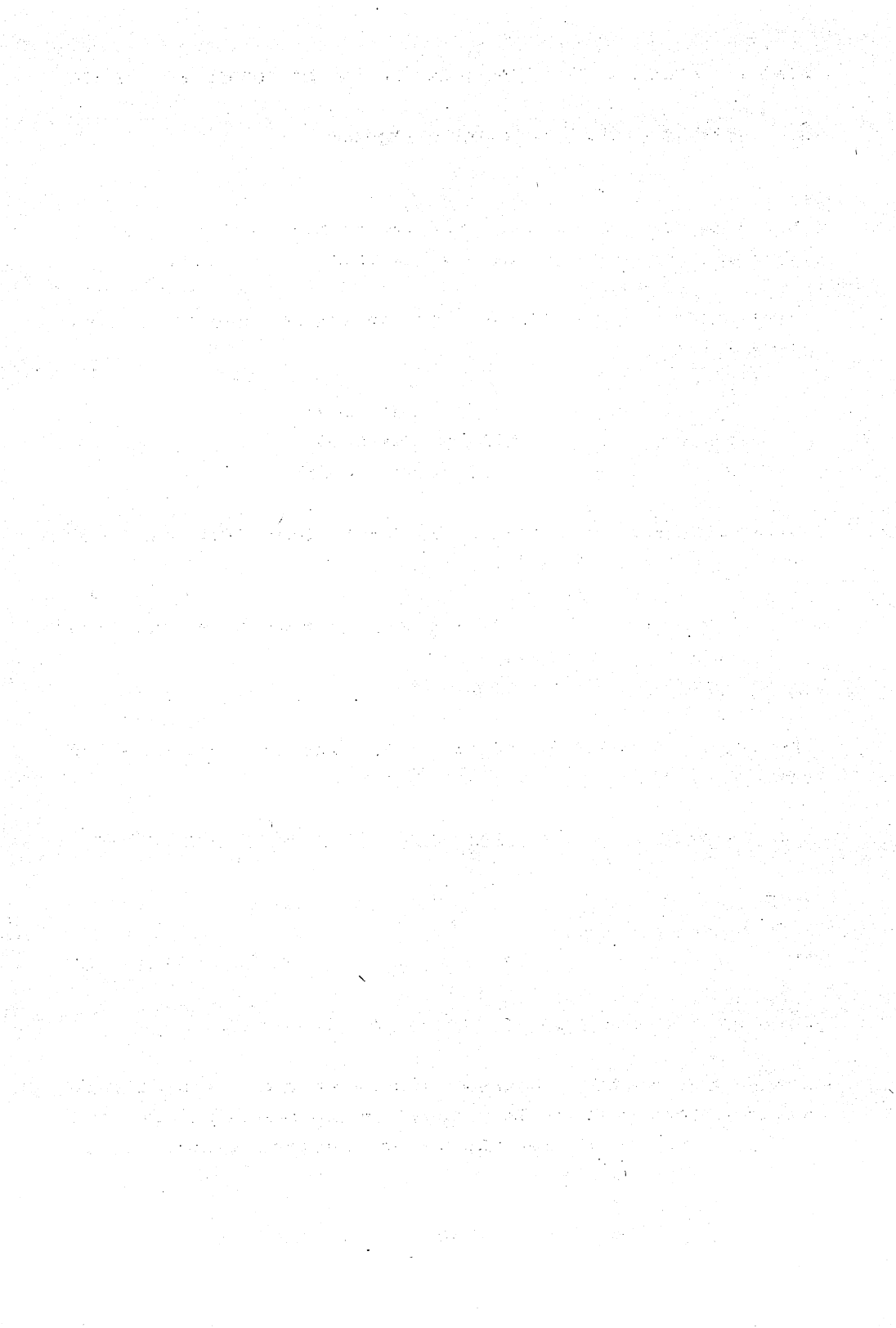
130: 0351
140: ;LOADER PROGRAM
150: 0351
150: 0351
160: A483 VECTOR = $A483 ; ($C483)VECTOR
165: ; TO GET CHARACTER LOOP
170: 00C6 NUMK = $C6
180: 0277 BUFFER = $277 ; KEYBOARD BUFFER
185: 036C TABLE = $36C ; TABLE OF BASIC
      INSTRUCTIONS
190: 0351
200: 0351 A9 83 LDA #<VECTOR ; SETTING OF
210: 0353 8D 02 03 STA $302 ; OLD VECTOR
220: 0356 A9 A4 LDA #>VECTOR
230: 0358 8D 03 03 STA $303
234: ;
235: ; TRANSFER VALUES FROM
237: ; TABLE TO KEYBOARD
238: ; BUFFER
239: ;
240: 035B AE 6C 03 LDX TABL ;NUMBER OF
      CHARACTERS
250: 035E 86 C6 STX NUMK
260: 0360 BD 6C 03 LOOP LDA TABLE,X
270: 0363 9D 77 02 STA BUFFER,X
280: 0366 CA DEX
290: 0367 D0 F7 BNE LOOP
294: ;
295: ; JUMP BACK TO GET
297: ; CHARACTER LOOP
299: ;
300: 0369 4C 83 A4 JMP VECTOR
U0351-036C

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
5      REM LOADER FOR VIC 20
10     INPUT"FILE NAME";F$
20     SP$="                "
30     F$=LEFT$(F$+SP$,16):S=833
40     FOR I=0 TO 26
50         :   READ F
60         F$=F$+CHR$(F)
70     NEXT
80     PRINT"OUTPUT STRING MAX. 10 CHARACTERS"
90     PRINT"↑= RETURN, TWICE AT THE END"
95     INPUT A$
100    LA=LEN(A$):IF LA>10 THEN 90
110    F$=F$+CHR$(LA)
120    FOR I=1 TO LA
130        :   W$=MID$(A$,I,1):IF W$="↑"THEN W$=CHR$(13)
140        :   F$=F$+W$
150    NEXT
160    POKE 770,81:POKE 771,3
170    SYS 24528F$,1,768,772
180    DATA 169,131,141,2,3,169,196,141,3,3,174,108,3,134,
198,189,108,3,157,119
190    DATA 2,202,208,247,76,196,131
```

```
5      REM LOADER FOR C-64
170    SYS 49152F$,1,768,772:REM START ADDRESS OF SAVE
ADDRESS PROGRAM
180    DATA 169,131,141,2,3,169,164,141,3,3,174,108,3,134,
198,189,108,3,157,119
190    DATA 2,202,208,247,76,131,164
```



8. STORAGE FORMAT IN CASSETTE STORAGE

When data or programs are recorded on tape, the individual bytes and bits are recorded in the form of impulses.

Three different lengths of impulse are defined for coding purposes:

- S = short (176 microseconds)
- M = medium (256 microseconds)
- L = long (336 microseconds)

Three different combinations are formed from these impulses, with the following meanings:

- LLMM = byte - this combination precedes each byte
- MMSS = bit set = 1
- SSMM = bit not set = 0

The value 65 would be recorded on tape in the following form:

LLMM	MMSS	SSMM	SSMM	SSMM	SSMM	SSMM	MMSS	SSMM	MMSS
BYTE	1	0	0	0	0	0	1	0	1
BIT	0	1	2	3	4	5	6	7	PARITY ODD

This gives a time span of 8.96 ms per character.

During the reading process, a counter reduces a certain value as long as there is a signal on the reading line. The computer can tell from the values reached whether it is

dealing with a bit that is set or not set, or a byte. In order to ensure that the counter always starts at the right time, precise synchronization is required between tape and counter. For this reason, each block is preceded by synchronization bytes and a countdown.

A block is constructed as follows:

1. Synchronization bytes and countdown
This part is responsible for the synchronization of the read components on the tape. The synchronization also contains information as to what sort of a block it is.
2. Data
3. Check sum byte
During saving and loading, the computer forms an EXOR check sum of all the bytes by linking the last value with the next by means of a logical "Exclusive OR" function and storing it. If a read error occurs, this check sum will not agree with the value stored.
4. Repetition of Data
In the reading process, this repetition takes the form of comparison with the bytes already read in, in order to detect any reading errors.
5. Check sum byte (as 3)
6. End of block mark

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

The complete files are stored as follows:

Program files	Data files
1. Header	Header
2. Program	Data block, possibly further data blocks
3. Possible EOT block	Possible EOT block

9. APPENDING BASIC PROGRAMS

No doubt you have on occasions been annoyed that the Commodore Basic 2.0 has no instruction that would enable you to combine a number of programs on tape or disk.

Once again, the Basic pointers 43 and 44/45 and 46 come to the rescue. They "bend" the Basic RAM start pointer to the end of the program in memory and then you reload the program to be appended. Finally, pointer 43,44 must be reset to its original value.

The detailed procedure is as follows:

1. Load the first program. The line numbers must all be smaller than those of the program to be appended. If they are not, then the composite program will not function as well as it ought to.

2. Fix the Basic start address by

```
PRINT PEEK(43),PEEK(44)
```

and make a note of it for yourself.

3. Enter

```
POKE43, (PEEK(45)+256*PEEK(46)-2)AND255  
POKE44, (PEEK(45)+256*PEEK(46)-2)/256
```

As there are always two zero bytes at the end of a Basic program to mark the end, the end vector

must be reduced by two bytes (see also Chapter 2.1). Now there is apparently no program left in storage.

This can be checked by entering LIST.

4. Now load the second program. If you now enter LIST, only the second program will appear.
5. Now write back the old values into memory locations 43,44.

With the help of this procedure, you have joined two programs.

The whole thing can be performed more elegantly with a machine language program, of course. The following short program recognizes from the @, as the first character in the file name, that the program to be loaded should be appended to the program in memory.

PROGRAM DESCRIPTION

The loading vector is changed to this program by the initialization process. Accordingly, control branches to this program on every loading operation. In this program, the first character of the file name is checked for @. If this character is not found, control returns to the old routine. If this character is found, the program vector minus 2 is transferred to the load start address vector \$C3/\$C4 and the @ is deleted from the file name. The program then jumps to the old loading routine.

```

25:  C000          *=  $C000
30:  C000          .OPT P1,0
32:  C000
33:          ;*****  APPEND  *****
34:  C000          ;VALUE IN BRACKETS APPLIES TO
35:          ;          VC 20
36:  C000
40:  0330          VECTOR  =    $330    ;LOADING VECTOR
50:  F4A5          LOAD    =    $F4A5    ;($F549) LOAD
                                   ROUTINE
60:  00BB          FILE ADDR =    $BB    ;PTR TO FILE NAME
70:  00B7          FILELNG          $B7    ;FILE NAME LENGTH
80:  00C3          START ADDR =    $C3    ;PTR TO PROG.START
90:  002D          Progr.END =    $2D    ;PTR TO PROG.END
92:  C000
92:  C000
93:  C000
93:  C000
95:          ;INITIALIZATION OF
96:          ;PROGRAM
97:  C000
97:  C000
100: C000 A9 0B          INIT  LDA #<START
110: C002 A2 C0          LDX #>START
120: C004 8D 30 03          STA VECTOR
130: C007 8E 31 03          STX VECTOR+1
140: C00A 60          RTS
142: C00B
142: C00B
144:          ;MAIN PROGRAM
146: C00B
146: COOB

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

150:  C00B 48          START PHA
160:  C00C A2 00          LDX #0
170:  C00E A1 BB          LDA (FILE ADDR.X)
180:  C010 C9 40          CMP #"@" ;FIRST CHARACTER '@'
190:  C012 D0 12          BNE END      ;NO => NORMAL LOAD
195:  C014 38          SEC
200:  C015 A5 2D          LDA PROG.END
210:  C017 E9 02          SBC #2      ;REDUCE POINTER BY 2
220:  C019 85 C3          STA START ADDR      ;IN LOAD
                                START ADDR.

230:  C01B A5 2E          LDA PROG.END+1
240:  C01D E9 00          SBC #0
250:  C01F 85 C4          STA START ADDR+1
260:  C021 18          CLC
270:  C022 E6 BB          INC FILE ADDR.      ;DELETE '@'
280:  C024 C6 B7          DEC FILELNG
290:  C026 68          END      PLA
300:  C027 4C A5 F4          JMP LOAD

]C000-C02A

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
10 REM APPEND C64
20 E=56*PEEK(56)+PEEK(55)-1:A=E-41
30 FOR I=A TO E:READ X:POKE I,X:NEXT
40 READ X,Y,Z:X=A+X:Y=A+Y:Z=A+Z:H=INT(Z/256)
45 POKE X,Z-256*H:POKE Y,H
46 POKE E-1,PEEK(816):POKE E,PEEK(817)
50 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
65 SYS A:NEW
32000 DATA 169,224,162,127,141,48,3,142,49,3,96,72,162,0,
161,187,201,64,208,18
32001 DATA 56,165,45,233,2,133,195,165,46,233,0,133,196,
24,230,187,198,183,104
32002 DATA 76,165,244,1,3,11
```

```
10 REM APPEND VIC 20
20 E=256*PEEK(56)+PEEK(55)-1:A=E-41
30 FOR I=A TO E:READ X:POKE I,X:NEXT
40 READ X,Y,Z:X=A+X:Y=A+Y:Z=A+Z:H=INT(Z/256)
45 POKE X,Z-256*H:POKE Y,H
46 POKE E-1,PEEK(816):POKE E,PEEK(817)
50 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
65 SYS A:NEW
100 DATA 169,224,162,127,141,48,3,142,49,3,96,72,162,0,1,
61,187,201,64,208,18
110 DATA 56,165,45,233,2,133,195,165,46,233,0,133,196,24,
230,187,198,183,104
120 DATA 76,73,245,1,3,11
```


10. PROGRAM CONTROL OF DATASSETTE

Up to now, I have only written about loading and saving with the Datasette. We have seen that the tape can be started and stopped automatically. The computer has the ability to control the motor, assuming that we can put this to good use, I shall now describe how to exploit this facility in your programs.

Using the following memory locations, you can control the motor and check whether a key on the recorder has been pressed:

VIC 20 Addr.	Value	C-64 Addr.	Value	Function
37148	252	1	AND 223	Motor on
37148	0	1	OR 32	Motor off
192	0	192	0	Motor on
192	1	192	1	Motor off
37151	64	1	16	Key pressed

Waiting for a key on the recorder:

```

                VIC 20                C-64
            WAIT 37151, 64,64        WAIT 1, 16,16
    
```

As an example of how you can make use of this control facility, I will now describe a program that will save you the trouble of searching for a specific program stored along with other programs on a cassette. It creates a catalog of all the programs stored on your cassette and automatically finds any given program.

The program operates on the following principle:

After you have stored one or more programs on tape after this sample program, it will determine the time required by the cassette recorder to wind from the end of the cataloging program to the program to be cataloged, with a little help from you and using the variable TI.

This time is stored in data lines, together with the file name of the program.

If a certain program is required, then the cassette drive motor is run to wind the tape until the actual running time matches the time stored. After pressing the PLAY key, the required file is loaded.

The program is operated as follows:

This program is stored at the beginning of the cassette. The other programs can then follow in the usual way, at an interval of approx. 10 seconds. If the programs are to be catalogued, then you should note down the starting numbers on the counter together with the file names of the individual programs. Then rewind the cassette to the beginning and load the cataloging program. After starting the program, you should select the point for fresh input.

First of all, you will be requested to enter the appropriate file names in the correct sequence. You will then be requested by the program to press the fast forward key on the recorder. You can now start the recorder by pressing the SPACE key and stop it again on reaching the counter values you have noted down; you can then restart and stop

again, until you have assigned each of the programs on the cassette an appropriate time value. Finally, you store the cataloging program at the beginning of the cassette again.

You can now select a program in comfort from those shown in the catalog displayed in the screen menu after specifying "LOAD". When the FFWD key is pressed, the computer will wind the tape forward to the start of the required program. When the PLAY and SPACE key is pressed, the program will be loaded.

PROGRAM DESCRIPTION

Lines 120 to 170 contain the subroutines for starting and stopping the motor. If the program is to run on a VIC 20, then the REM instructions in lines 130 and 160 must be deleted. In addition, a REM must be inserted after POKE 192,0: in lines 120 and 150.

The program is initialized in lines 180 to 340 and the menu is written on the screen. During initialization, the name and time fields are dimensioned and the repeat function for all the keys is switched on. The computer type is determined with the aid of the operating system routine SCREEN (65517) and the start of the video and color ram is set accordingly. Finally, the variables for motor control are allocated, depending on the computer (lines 240 and 260). Depending on the input, a branch is taken to "new input" or to "Load".

The block "New input" will be found in lines 350 to 660. In line 390, the number of files already entered is read in from the data lines via READ. The new file names are read in and stored in field N(n) in lines 400 - 420.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

In lines 530 to 580 the winding time between the starts of the individual programs is established and, once assigned to the appropriate file names, stored in field ZS(n). Beforehand, however, a value of 10 is deducted in line 580 to compensate for the run-down time of the motor. For the same reason, the same time value is added when summing the individual times.

In order to prevent any errors in operation, a check is made in line 420 to determine whether a key has been pressed on the recorder, and the operator will be requested, if necessary, to press the STOP key.

In lines 600 - 660, the file names and times are written in data lines and the program is terminated.

In line 670, the program reaches the loading stage. The number of data and the file data are read in, and in lines 720 - 820 the files, eight to a page, are displayed on the screen with a (>) which is controlled by lines 770 - 810. If the result of the check for key F7 in line 790 is positive, then a branch is taken to the seek and load routine after line 830.

After the test to determine whether a key has been pressed on the recorder (840), the computer waits in line 900 (WAIT instruction) for the FFWD key on the recorder to be pressed, after which the motor runs for a period of time equal to the stored time for the appropriate program (lines 920 - 940).

In lines 950 - 1010 the required program is loaded. If the program were to be loaded in program mode, then only programs smaller than the cataloging program could be loaded. In order to avoid any such thing, the LOAD

instruction of the FastTape LOAD instruction <-L is written on the screen and RETURN is written into the keyboard buffer. In this way the program is loaded in direct mode and the Basic vectors set according to the program to be loaded.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

10 GOTO 190
20 *****
30 *
40 *          CATALOG          *
50 *
60 *          FOR CASSETTE     *
70 *
80 *****
90 :REM
100 REM          HOLDING LOOP
110 POKE 198,0:WAIT 198,1:GET A$:RETURN
120 POKE 192,0:POKE 1,PEEK(1)AND 223:REM MOTOR ON C-64
130 :REM POKE 37148,252:REM MOTOR ON VIC 20
140 RETURN
150 POKE 192,1:POKE 1,PEEK(1)OR 32:REM MOTOR OFF C-64
160 :REM POKE 37148,0:REM MOTOR OFF VIC 20
170 RETURN
180 REM "          INITIALIZATION
190 HD$=" * * C A T A L O G * * "
200 SYS 65517:SP=PEEK(781):ZE=PEEK(782):
          REM GET SCREEN DATA
210 BS=1024:BF=55296:FW=1:IF SP>25 THEN 260
220 BS=4*(PEEK(36866)AND 128)+64*(PEEK(36869)AND 120)
230 BF=37888+4*(PEEK(36866)AND 128)
240 WZ=37151:WW=64:REM VIC20 VALUES WAIT FOR RECORDER KEY
250 FW=6:GOTO 270
260 WZ=1:WW=16:REM C-64 VALUES 'WAIT FOR RECORDER KEY'
270 DIM N$(50),ZS(50):REM NAME AND START POSITION
280 P1=3*SP+1:P2=2*SP
290 POKE 650,128:REM REPEAT KEYING
300 PRINT"[CLEAR]";:PRINT TAB (SP/2-11)HD$:PRINT"[DOWN7]"
310 PRINT TAB(SP/2-6)"[RVS]N[RVOFF]EW INPUT":PRINT
320 PRINT TAB(SP/2-6)"[RVS]L[RVOFF]OAD"
330 GOSUB 110:IF A$="L"THEN 680

```

```

340 IF A$<>"N"THEN 330
350 REM *****NEW INPUT*****
360 PRINT"[CLEAR,DOWN2]THE INPUT OF FILES":PRINT
370 PRINT"MUST TAKE PLACE":PRINT
380 PRINT"SUCCESSIVELY. @@ = END"
390 RESTORE:READ A:ZA=A
400 INPUT"[DOWN]FILE NAMES";N$
410 IF N$ <>"@"THEN N$(A)=N$:A=A+1:GOTO 400
420 ZE=A:A=ZA:IF(PEEK(WZ)AND WW)=WW THEN 450
430 PRINT"[CLEAR]PRESS THE [RVS]STOP[RVOFF] KEY"
440 PRINT"[DOWN]ON THE RECORDER":WAIT WZ,WW,255-WW
450 PRINT"[CLEAR]PRESS THE [RVS]F.FWD[RVOFF] KEY!"
460 PRINT:PRINT"START AND STOP THE":PRINT
470 PRINT:PRINT"RECORDER BY PRESSING"
480 PRINT:PRINT "A KEY. PRESS A"
490 PRINT:PRINT"KEY AT THE START"
500 PRINT"OF A PROGRAM"
510 PRINT:PRINT"(WATCH COUNTER)"
520 PRINT:PRINT"HIT A KEY FOR NEXT"
525 PRINT:PRINT"PROGRAM"
530 WAIT WZ,WW,WW:GOSUB 150:DI=0
540 GOSUB 110:GOSUB 120:R=TI:REM CC ON, NOTE TIME
550 PRINT"[CLEAR,DOWN5]STOP AT START:"
560 PRINT"[DOWN]"N$(A)
570 GOSUB 110:GOSUB 150:DI=TI-R+DI
:REM CC OFF, COMPUTE TIME DIFFERENCE
580 ZS(A)=DI-10:DI=DI+10:A=A+1
590 IF A<ZE THEN 540:REM LAST FILE NAME?
600 REM * STORAGE IN DATA LINES **
610 PRINT"[CLEAR,DOWN3]2000DATA"A
:REM NUMBER IN DATA LINE
620 FOR I=ZA TO ZE-1:PRINT 2100+I"DATA"N$(I)", "ZS(I)
630 NEXT:PRINT"GOTO650"
640 POKE 198,11:FOR I=0 TO 10:POKE 631+I,13:NEXT

```

```

:PRINT"[HOME]":END
650 PRINT"[CLEAR]NOW REWIND AND SAVE"
:PRINT"[DOWN]THE PROGRAM AT THE "
:PRINT"[DOWN] BEGINNING"
660 PRINT:PRINT:GOSUB 120:END
670 REM ***** LOADING THE FILES *****
680 RESTORE:READ A:FOR I=0 TO A-1:READ N$(I),ZS(I):NEXT
690 PRINT"[CLEAR]";:PRINT TAB(SP/2-11)HD$
700 PRINT"[DOWN3,SPACE2]F1 = UPW."
:PRINT"[DOWN2,SPACE2]F2 = DOWN."
:PRINT"[DOWN2,SPACE2]F7 = LOAD"
710 PRINT"[DOWN4]":PRINT TAB(SP2-6)"[RVS,SPACE2]S P A
C E [RVOFF]":GOSUB 110:ZA=0
720 PRINT"[CLEAR]";:PRINT TAB(SP/2-11)HD$:Z=0:PRINT
730 FOR I=ZA TO ZA+8:IF N$(I)<>""THEN PRINT
:PRINT TAB(3)N$(I)
740 NEXT
750 POKE BS+P1+Z*P2,62:POKE BF+P1+Z*P2,FW
760 GOSUB 110:REM CHECK KEYBOARD
770 IF(ASC(A$)=133)AND(Z>0)THEN POKE BS+P1+Z*P2,32
:Z=Z-1:GOTO 750
780 IF(ASC(A$)=134)AND(Z<8)AND(N$(ZA+Z+1)<>"" )THEN POKE
BS+P1+Z*P2,32:Z=Z+1:GOTO 750
790 IF ASC(A$)=136 THEN 830
800 IF ASC(A$)=133 AND Z=0 AND ZA THEN ZA=ZA+9:GOTO 720
810 IF(ASC(A$)=134)AND(N$(I)<>""AND(Z=8)THEN ZA=ZA+9:
GOTO 720
820 GOTO 750
830 REM LOAD
840 IF(PEEK(WZ) AND WW)=WW THEN 870
850 PRINT"[CLEAR]PRESS THE"
:PRINT"[DOWN,RVS]STOP[RVOFF,SPACE]KEY"
860 PRINT"[DOWN]ON RECORDER":WAIT WZ,WW,255-WW
870 PRINT"[CLEAR]";:PRINT TAB(SP/2-11)HD$:PRINT:PRINT

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
880 PRINT"PRESS THE ":PRINT"[DOWN,SPACE4,RVS,SPACE]F.FWD
    [RVOFF]"
890 PRINT:PRINT"KEY"
900 WAIT WZ,WW,WW:REM WAIT FOR KEY ON RECORDER
910 PRINT" OK "
920 R=TI:R2=R+ZS(Z+ZA)
930 IF R2<TI THEN 950
940 GOTO 930
950 GOSUB 150
960 PRINT"[CLEAR]";:PRINT TAB(SP/2-11)HD$
970 PRINT:PRINT"PRESS THE [RVSON]PLAY[RVSOFF]"
980 PRINT:PRINT"KEY"
990 PRINT"[DOWN4]":PRINTTAB(SP/2-6)"[RVS,SPACE2]S P A C E
    [RVOFF]":GOSUB 110
1000 PRINT"[CLEAR,DOWN3]<-L";CHR$(34)(N$(Z+ZA))CHR$(34)
1010 POKE 198,1:POKE 631,13:POKE 37148,252:PRINT"[HOME]"
:END
1020 REM *****NUMBER OF PROGRAMS*****
2000 DATA 0
2005 REM *****PROGRAM DATA*****
```


11. DATASSETTE HARDWARE

So far, I have only described the software used to operate the datassette. In this chapter, I should like to go into the datassette itself.

11.1 CARE OF THE DATASSETTE

Like every other cassette recorder, the datassette requires a certain amount of care and attention. The occurrence of a "LOAD ERROR" is usually the result of a dirty, badly adjusted or magnetized head.

1. Cleaning of heads and capstan idler:

Normal commercially available swab sticks are ideally suited for this task. If they are not obtainable, one could wrap cotton wool round the end of a matchstick or similar sliver of wood. Alcohol, petroleum spirit or spirit are suitable cleaning agents.

To carry out this cleaning, open the cassette compartment and press the PLAY key. You can now clean the heads with the swab sticks dipped in the cleaning agent. You clean the heads by rubbing the front face of the heads until the swab no longer turns brown and then follow up with a dry swab.

It is recommended to clean the rubber capstan idler at the same time, as a slippery coating of material rubbed off the tape becomes deposited over a long

period of operation; as a result, the tape is no longer drawn smoothly past the sound head. To clean the idler, take a small piece of lintfree material (pocket handkerchief or similar), place it over the end of your index finger, moisten it with one of the cleaning fluids mentioned above and hold it against the capstan idler from the right-hand side with the datassette running on PLAY.

2. Demagnetization

For demagnetization, you should use a commercial demagnetizer or a demagnetizing cassette of the type used for audio equipment. The procedure is then the same as described in the operating instructions.

11.2 CHOICE AND HANDLING OF CASSETTES

The demands made on a cassette used for storing data and programs need not be as high as those made on an audio cassette. You do not need tapes with a particularly good frequency response or high signal-to-noise ratio. The cheapest cassette you can find will serve your purpose perfectly well.

The only important point to watch is that the tape is free from "drop outs". These are areas of tape where the coating of magnetic particles is imperfect. These areas cannot be properly magnetized, with the result that information stored in such parts is lost. This is, unfortunately, a common risk with very cheap tapes, so in my opinion, you would be better off buying one of the simple range of tapes produced by a reputable manufacturer.

In order to avoid long rewinding times, I would recommend C60 cassettes as a maximum.

As you know, there are leader tapes at the beginning or end of every magnetic tape to take the strain when the tape comes to an end and stops. In spite of this, the tape is always subjected to more strain at the beginning and end than in between, and this can lead to stretching. I would therefore recommend that nothing to be stored on the first and last 10 - 20 seconds of the tape.

In other respects, the same applies to data cassettes as to audio and video cassettes. They should not be exposed to strong magnetic fields and should be stored so that they are protected against dust. Likewise, they should not be stored

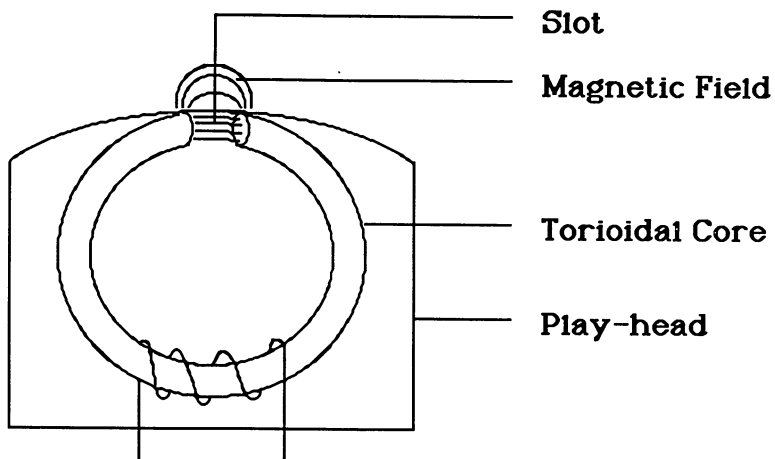
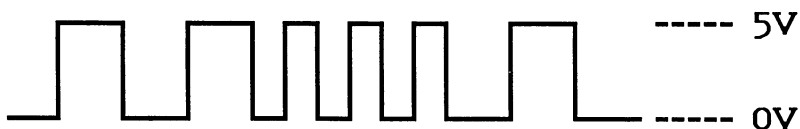
for long (periods of several months) without rewinding them. Otherwise the magnetic coatings on the tape could interact with one another and changes could take place.

In order to protect your programs against external influences and also against any mistakes on your own part, I would recommend that you make a copy of all important programs and data which you keep on a protected cassette. This copy is never used as a working tape, but serves as a back-up in the event that your working tape should be destroyed or damaged.

11.3 DATASSETTE OPERATION

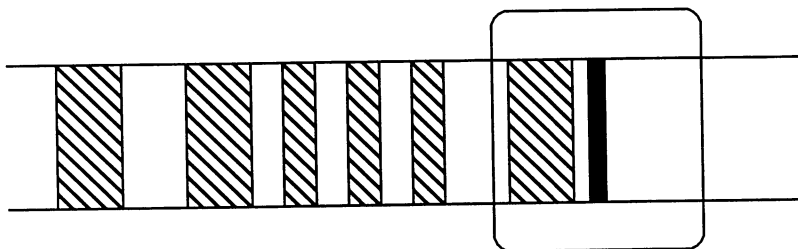
As described in Chapter 8, the information to be stored is transferred to the datassette in the form of impulses. This results from the fact that there is a voltage of either .5V = 1 or 0 V = 0 on bit 3 of the C-64 processor (memory location 1). In the case of the VIC 20, it is bit 3 of memory location 37152. With both processors, there is a direct connection between this contact and the cassette port contacts E and 5.

Ideally, the signal found there will show the following pattern:



By way of a number of electronic stages which I will not explain in greater detail here, the signal arrives at the sound head. This head consists basically of a coil with a toroidal core and a small slot on the side facing the tape.

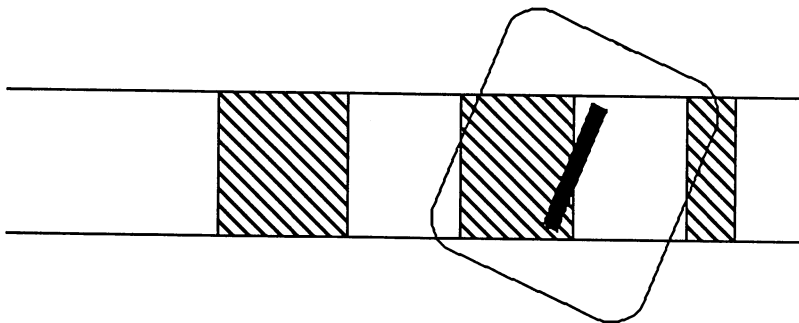
Depending on the voltage applied to the coil contacts, a magnetic field is created at the slot which aligns the magnetic particles on the tape in a certain direction. The idealized magnetization pattern on the tape would then look like this:



When the tape is read, the process is reversed.

A voltage is created in the coil by the magnetized particles, via the slot in the sound head. This voltage leads to the same sort of wave pattern, having passed through a number of other stages, as during the saving process. This voltage is connected to cassette port contacts D and 4.

If, for whatever reason, the alignment of the sound head and tape is different when reading to what it is for saving, we get the following picture:



In the top part of the slot, the direction of magnetization is different to what it is in the bottom part. The two effects erase one another, with the result that the data is not read in correctly.

11.4 A LOUDSPEAKER FOR THE DATASSETTE

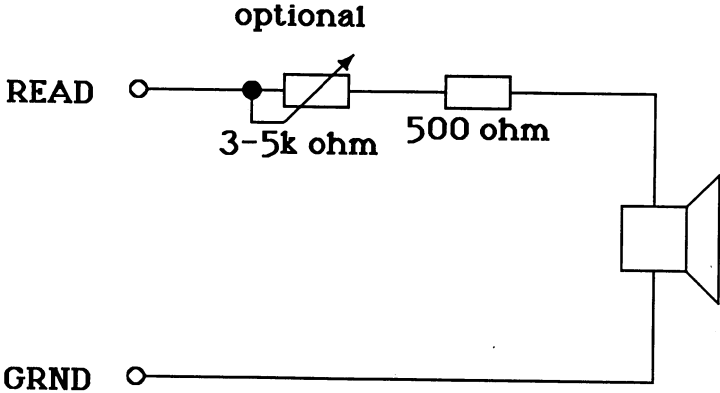
No doubt you have occasionally been annoyed that, apart from the counter, there is no other way of determining the precise position of the tape. It would be useful to be able to tell whether the tape is positioned at the beginning of a program, or in the middle, or wherever, since this would reduce the time taken to search for a specific program.

Well, connecting up a loudspeaker to the datassette provides you with just such a possibility. Using the loudspeaker, you can hear when you are at the beginning of a program. By winding a little and "listening in" again with PLAY, it is much easier to find the start of the next program.

Any loudspeaker is suitable for the purpose and can be connected to the datassette. As the sound quality is not really important, you could remove the loudspeaker from an old portable radio. This is then connected to the READ line. In order not to overload the connection, a resistor of approx. 500 Ohm will have to be connected in series. You could also couple up a potentiometer of 2 - 5 ohm in series as well, to control the volume.

The simplest way to connect the loudspeaker up, is to the datassette plug. All you need to do is to open up the plug, which is connected to the computer, and attach one of the loudspeaker leads to the GROUND line (A/1) and the other one to the READ line (D/4).

Now put the plug back together and insert it into the cassette port on the computer.



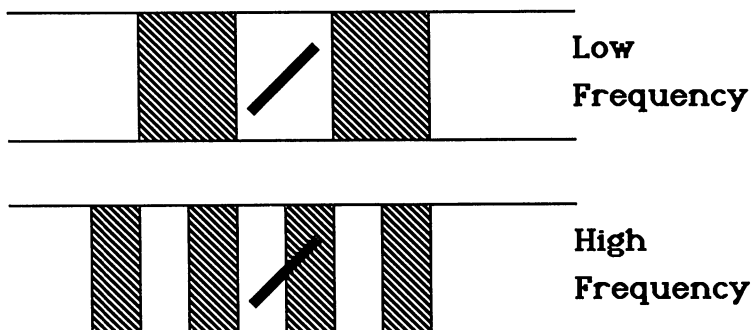
Connection of the loudspeaker

Switch on the computer, place a used cassette in the recorder and press the PLAY key. You will now hear either a whistling sound, all on one note, or a chirping sound. The whistling is associated with the synchronization and characterizes the start of a block. The chirping is associated with the data recorded on the tape.

11.5 ADJUSTING THE HEAD

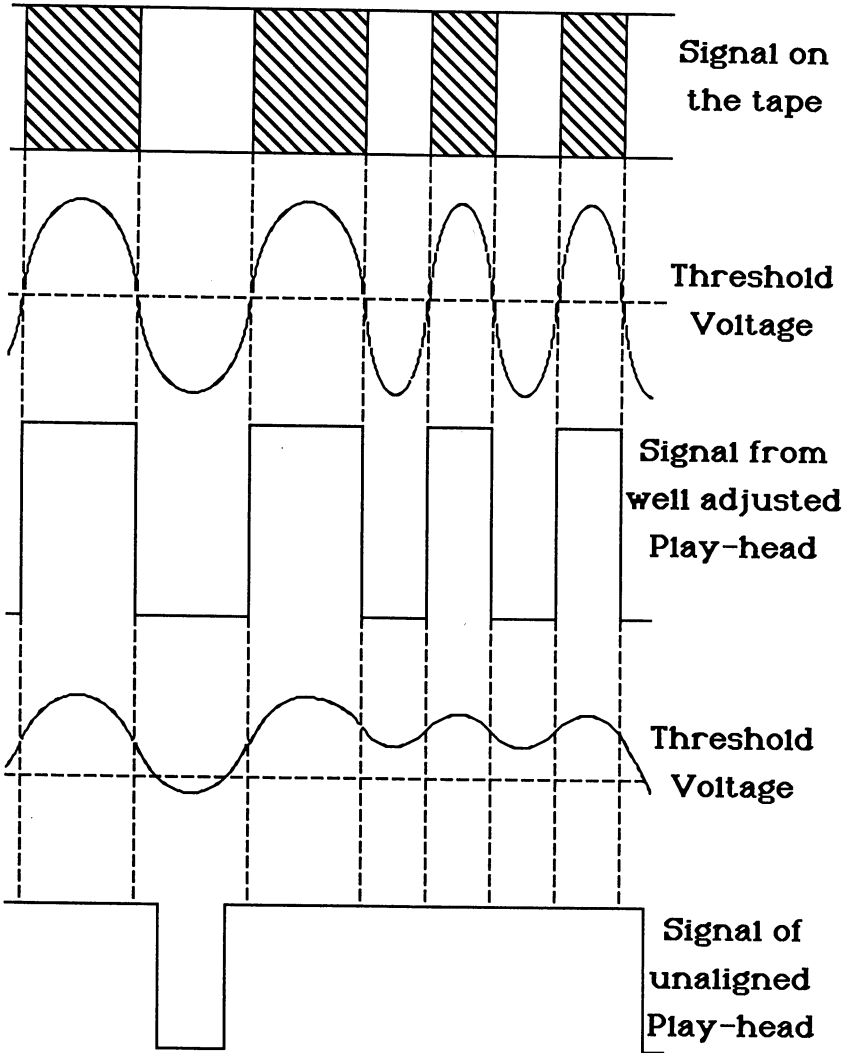
For various reasons it can happen that the sound head gets out of alignment with the tape, with the result that loading cannot be carried out successfully. This could happen, for example, when you try to read in programs which were recorded on a recorder other than your datassette. But it can also happen that over a long period of time, the sound head gradually slips out of alignment of its own accord.

This misalignment can be heard quite clearly with the aid of a loudspeaker. Even if the head is only at a slight angle to the tape, this can be detected from the high frequencies. I should like to illustrate this with a couple of drawings:



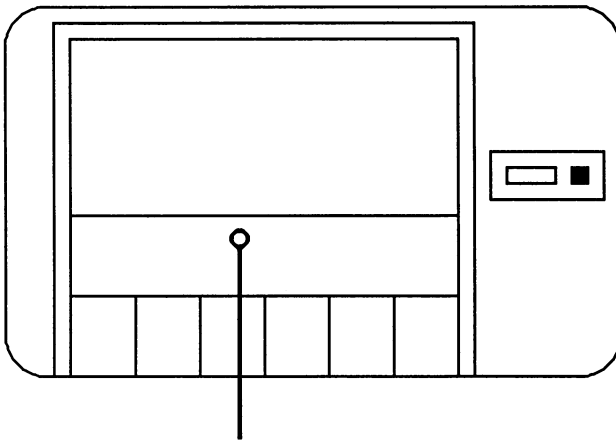
As can be seen from the illustrations, there are tape positions at low frequencies in which the same degree of magnetization is present over the entire length of the slot.

At high frequencies, however, the slot straddles a mixture of different magnetic areas, which results in these frequencies being dampened.



When reading data, the pattern is as illustrated in Fig. 2. The signal picked up by the sound head is approximately sinusoidal. This signal is converted in the datassette by means of a Schmitt trigger into rectangular pulses which can be read by the computer. A Schmitt trigger is nothing more than a switch which switches itself on and off at a certain voltage - the threshold voltage - and thus generates a square-wave voltage at the output.

It is clear from Fig. 2 that, if the head is badly adjusted, the time intervals between the individual pulses, which carry the information, will no longer provide the correct spacing. There will therefore be frequent read errors when reading the tape.

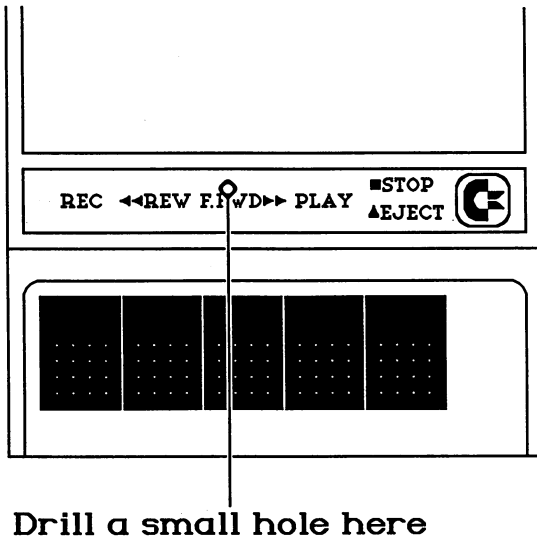


**Access opening to the Play-head
Adjustment Screw**

After this brief theoretical introduction, let us look at the practical side of things. If you have the latest version of the datassette, you will find a small hole approximately in the center of the cassette compartment.

When the PLAY key is pressed, you can adjust the sound head setting screw through this hole with the aid of a Phillips screwdriver.

On older versions of the datassette, there is no such hole. In order to be able to adjust the sound head on these models, you will have to drill a small hole in the plastic casing yourself. The precise position for this hole can be seen in the following illustration.



However, before you start adjusting the head, a word of caution, or rather, two:

1. It is possible to turn the screw so far in the wrong direction that it is no longer possible to adjust properly without expensive tools (measuring equipment), or at least, only with considerable difficulty. You must also take care to see that you do not screw the adjuster screw right out. Normally, two turns in either direction at the maximum is sufficient to adjust the head. Usually, half a turn is quite enough.
2. When you have readjusted the head, you may find that the programs stored with the head in its previous setting can now no longer be loaded.

In order to be able to find the original setting again with the minimum of trouble and to set a badly adjusted head more easily, you should use the following short program. It will record a chirping on the cassette, with a large proportion of high frequencies.

As it is the high frequencies which are dampened first, this chirping makes it much easier to adjust the setting of the sound head, by matching the chirping of the original recording. It also permits more precise adjustment of the head.

So enter the program and save it. Then record the chirping generated by this program on a clean tape for about two minutes.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```
10 REM AUXILIARY PROGRAM FOR ADJUSTING HEAD C64
20 E=256*PEEK(56)+PEEK(55)-1:A=E-17
30 FOR I=A TO E:READ X:POKE I,X:NEXT
40 PRINT"START ADDRESS"A
50 PRINT:PRINT"INSERT CLEAN CASSETTE AND"
60 PRINT:PRINT"PRESS RECORD AND PLAY"
70 WAIT 1,16,16
80 PRINT:PRINT"END VIA RUN/STOP KEY"
90 SYS A:END
1000 DATA 160,255,169,255,162,255,32,177,251,136,208
,246,32,225,255,208,239,96
```

```
10 REM AUXILIARY PROGRAM FOR ADJUSTING HEAD VIC20
20 E=256*PEEK(56)+PEEK(55)-1A=E-17
30 FOR I=A TO E:READ X:POKE I,X:NEXT
40 PRINT"START ADDRESS"A
50 PRINT:PRINT"INSERT CLEAN CASSETTE AND"
60 PRINT:PRINT"PRESS RECORD AND PLAY"
70 WAIT 37151,64,64
80 PRINT:PRINT"END VIA RUN/STOP KEY"
90 SYS A:END
1000 DATA 160,255,169,255,162,255,32,177,251,136,208
,246,32,225,255,208,239,96
```

Now rewind the tape, pick up the Phillips screwdriver and press the PLAY key. Insert the screwdriver vertically into the hole until it engages in the screw. Turn the screw half a turn in either direction from the original position and listen to the characteristic change in the sound. At only one point, namely, the original position, will you hear the highest frequencies.

Now insert a used cassette and repeat the process. You will not detect any change in the sound over a wide range. The best position is mid-way between the points where changes become audible.

Having got your ear accustomed to the sound in this way, you can now adjust the head in the same way to programs that will not load. Try to find the best setting of the head purely by the sound. Then try loading the program. If it still won't load, change the position of the head slightly and try again.

If, in spite of all your efforts, it still will not load properly, then proceed as described in Chapter 4.1.

However, before you load the UNNEW program, or store the fragment of program, you must return the sound head to the original position, using the adjustment cassette you prepared for the purpose.

11.6 OTHER CASSETTE RECORDERS FOR DATA STORAGE

So far, I have always referred to the Commodore Datassette in my discussions of the available hardware. However, it is perfectly possible to use other cassette recorders, suitably adapted, for recording purposes. Adaptation is by means of an interface which converts the incoming signals from the tape into the square-wave pulses that the computer understands. Such interfaces are manufactured by a number of different firms and can be obtained at quite reasonable cost.

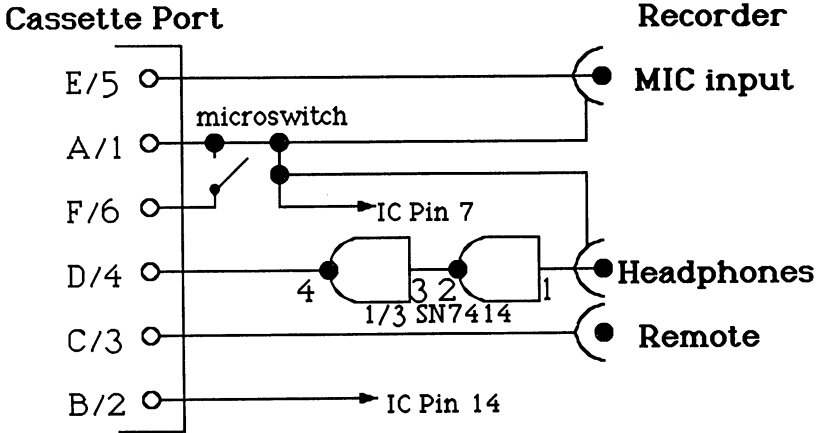
For the sake of completeness, I should like to show you a little circuit which you can make for yourself. The components you need are listed below.

Components for Cassette Recorder Interface

- 1 flat plug for the cassette port on the VIC 20/C64
- 1 plug for the headphone output on your cassette recorder
- 1 plug for the microphone input on your cassette recorder
- 1 IC SN 7414 (6 * inverting Schmitt triggers)
- 1 microswitch (on/off) if necessary
- 1 plug for the remote input of your recorder

Only those recorders which operate on 4 - 6 V and have their negative pole connected to ground are suitable for use with this circuit.

The wiring up of the components can be seen from the following diagram.



Description of connections at cassette port:

- A/1 = earth
- B/2 = +5V
- C/3 = cassette motor (+5V = on)
- D/4 = read line
- E/5 = switch line for recorder key

The IC SN 7414 contains 6 inverting Schmitt triggers. Schmitt triggers are circuits which form suitable square-wave pulses from any voltage form, as described in Chapter 11.5

Of these six Schmitt triggers, only two are required for this circuit, for inversion purposes.

The operating voltage for this module is taken from the cassette port (pin 7 - A/1, pin 14 - B/2).

If the cassette recorder is used exclusively for your computer, you could install this circuit in the battery compartment and solder the microphone and headphone lines permanently in position. In this way, you could save yourself the jack plugs.

It would be very practical if you could install the microswitch in the recorder in such a way that it could be operated by pressing a key. If this is not possible, then the switch would have to be mounted on the outside and operated manually before any of the recorder keys.

12. NEW 'FASTTAPE' CASSETTE OPERATING SYSTEM

In the first part of this book, I explained how to operate the datassette with the computer's operating system. I also showed you a number of programs which simplify working with the data recorder. I should now like to introduce you to a new operating system which not only includes the simplifications I have already explained but also operates 10 - 20 times faster, and is thus even faster than a disk drive.

I admit that your fingers will be a bit sore from entering the program, such is its length, but I am sure you will agree that it is worth the effort. The program has the following characteristics:

1. 10 times faster in loading and saving programs.
2. Support for data storage
Data can be loaded and stored up to 20 times faster than with the normal operating system
3. Appending of Basic programs is possible with one instruction
4. Loading into or saving from certain data areas is also possible with a single instruction.
5. Programs and data stored with FastTape can be read by both VIC 20 and C-64
6. All FastTape instructions work both in direct mode and program mode.

Operating the Program

When the program has been loaded, it is initialized with SYS7*4096 in the case of the VIC 20, and SYS12*4096 in the case of the C-64. The program is automatically initialized by the Basic loader. After a RESET, however, the program must be initialized again. Each FastTape instruction is preceded by an arrow pointing to the left (<-).

The following instructions are now at your disposal (expressions in brackets are optional, i.e. do not have to be given):

SA	=	secondary address
Sadr, Eadr	=	start or end address
"name"	=	can also be transferred as a string variable

<-S ("NAME", SA, SADR, EADR) FASTTAPE SAVE

<-S("name") : Basic program is stored as relocatable file as program and loaded with <-L at the Basic RAM start

<-S"name", 1 : Basic program stored as absolute file and loaded with <-L at the address from which it was transferred to memory

<-S"name",1,Sadr,Eadr : The memory area between Sadr and Eadr is stored as an absolute file

<-L ("NAME", SA, SADR)

FASTTAPE LOAD

If this instruction is given in a program, the Basic program starts again at the first line, analogous to the LOAD instruction.

<-L("name")

: A program or memory area, stored with <-S, is loaded either absolutely or relocatably, depending on the SA given with S. If this instruction is given in direct mode, then the Basic vectors are set according to the program loaded.

<-L"name",1

: Programs and memory areas are loaded absolutely. Even in direct mode, the Basic vectors are not changed by this instruction. As a result, no NEW instruction is required after a machine code program has been loaded.

<-L"name",1,Sadr

: A memory area is loaded at Sadr, irrespective of the address from which it was transferred to memory. In

this way it is possible for you, for example, to re-read the contents of a screen, which have already been stored into the video RAM, even though you have relocated the video RAM.

<-V ("NAME", SA, SADR) FASTTAPE VERIFY

Analogous to L in operation, but compares only the contents of memory with the bytes on the tape.

<-A ("NAME") FASTTAPE APPEND

Using this instruction, a FastTape program can be attached to the end of the program in memory.

<-DS "NAME" DATASAVE

This instruction stores all the variables, fields and strings so far defined but, so far as the strings are concerned, only those in the string storage. If your program contains string assignments such as

```
100 AS="TEST"
```

then you can only properly load the corresponding strings again with the same program. The situation can be eased by linking with a blank string. The same applies here as in Chapter 5.1.

<-DL "NAME"

DATALOAD

This instruction is used to load the variables, fields and strings stored with DS. All the variables, fields and strings defined prior to this instruction are deleted.

ATTENTION!

<-DS and <-DL must always be used in conjunction with a file name. These instructions result in all the variables, fields and strings being stored or loaded in three blocks. The file name serves to help the computer identify the first of these blocks.

ATTENTION!

The Basic RAM end must be the same with <-DS and the corresponding <-DL. The string table is stored and loaded as an absolute block. If you reduce the end of the Basic RAM, after you have given <-DS, and then load with <-DL, you might possibly destroy a machine program written after the new Basic RAM end.

Just so that you can see just how easy it is to process data using these instructions, the next chapter contains a very detailed description of a data processing program. The program is stored in the C-64 from memory location \$C000 onwards by means of the Basic loader FT64.

The Basic loader FT20 stores the program in the VIC 20, from location \$7000 onwards, i.e. the program will only work on a VIC 20 if it is fully expanded. It is, however, entirely possible to re-write the FastTape program into another memory area with the aid of a MONITOR program.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

ASSEMBLER 64 V2.0 PAGE 1

```

82:    C000                .PAG 44,0
90:    C000                .OPT P130
92:    C000
92:    C000
94:
95:    C000                ; FASTTAPE FOR VIC20 AND C-64
95:    C000
95:    C000
96:
97:    C000                ;VALUES IN BRACKETS ARE FOR
97:    C000
98:    C000                ;
98:    C000                ;
98:    C000                ;
100:   C000                ;
110:   0000                COMPU    =    $C000
120:   0001                PORT     =    $00    ; (2)
130:   0002                ABSFLG   =    $1     ; ($9120)
140:   00C0                MOFLAG   =    $C0
150:   00A3                BITC     =    $A3
153:   0004                BEFANZ   =    4
155:   005F                CODE     =    "<->"
157:   0073                CHRGET   =    $73
160:   0308                VECTOR   =    $308    ; FETCH INST.
165:   02A0                IRQFLG   =    $2A0    ; IRQ FLAG
170:   0090                STATUS   =    $90     ; STATUS BYTE
175:   00C3                STARTV   =    $C3     ; LOAD START
                                VECTOR
180:   00AE                ENDVEC   =    $AE     ; LOAD END VECTOR
185:   00D7                CHKSUM   =    $D7     ; CHECK SUM BYTE
186:   033C                CASBUF   =    $033C    ; START ADR. OF
                                CASSETTE BUFFER
187:   C000
187:   DD00                CIA1     =    $DD00    ; C-64 = START
                                ADDRESS OF CIA1
188:   DC00                CIA2     =    $DC00    ; C-64 = START
                                ADDRESS OF CIA2
189:   C000
189:   C000
190:   F68F                SAVING  =    $F68F    ; ($F728) OUTPUTS
                                "SAVING NAME"
195:   F5D2                LOADING =    $F5D2    ; ($F66A) OUTPUTS
                                "LOADING"
200:   A82C                STOP    =    $A82C    ; ($C82C) TEST
                                STOP KEY
220:   FC93                ENDEN   =    $FC93    ; ($FCCF) STOP
                                LOADING
230:   E17A                BRAEND  =    $E17A    ; ($E177) BRANCH
                                BACK FROM LOADING
240:   F750                FOUND   =    $F750    ; ($F7D3) OUTPUTS

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

                FOUND 'NAME'
260:  F5A9      STOEND  =   $F5A9  ; ($F641) END ADR.
                AFTER X/Y
300:  A7E7      OLD      =   $A7E7  ; ($C7E7)
                EVALUATE INSTRUCTION
310:  A7AE      INTER   =   $A7AE  ; ($A7AE)
                INTERPRETERS LOOP
320:  F838      PLKEY   =   $F838  ; ($F8B7) WAIT
                FOR PLAY-KEY
330:  F817      AKEY    =   $F817  ; ($F894) WAITS FOR
                KEY
335:  B526      GABCOL  =   $B526  ; ($D526) GARBAGE
                COLLECTION
340:  E206      WRCHAR  =   $E206  ; ($E203) PRINT
                FURTHER CHARACTERS
350:  E257      HFNAM   =   $E257  ; ($E254) COLLECT
                FILE NAMES
360:  E200      HOLPAR  =   $E200  ; ($E1FD) FILE
                PARAMETER
365:  E20E      CHRIN   =   $E20E  ; ($E20B) COLLECT
                CHARACTERS
370:  AF08      SYNTAX  =   $AF08  ; ($CF08) PRINT
                SYNTAX ERROR
380:  AD8A      FRMNUM  =   $AD8A  ; ($CD8A)
                EVALUATE TERM
390:  B7F7      FACINT  =   $B7F7  ; ($D7F7) CONVERT
                TO INTEGER
395:  A660      CLEAR   =   $A660  ; ($C660) CLR
400:  A437      OUTERR  =   $A437  ; ($C437) OUTPUT
                ERROR
410:  A677      BEENDEN =   $A677  ; ($C677)
420:  C000
420:  C000
430:  FFB7      HOLSTA  =   $FFB7  ; COLLECT STATUS
440:  FFBD      SETNAM  =   $FFBD  ; FILE NAME
                PARAMETER SET
450:  FFBA      SETLFS  =   $FFBA  ; FILE PARAMETER
                SET
460:  FFE4      GET     =   $FFE4  ; READ CHARACTER
500:  C000
510:  C000
520:                ;PROGAMM INITIALIZATION
530:  C000
530:  C000
560:  C000 A9 0B      INIT    LDA  #<START
570:  C002 8D 08 03      STA  VECTOR
580:  C005 A9 C0      LDA  #>START
590:  C007 8D 09 03      STA  VECTOR+1
600:  C00A 60      RTS
602:  C00B
603:  C00B
604:                ; PROGRAM START

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

606:  C00B
606:  C00B
610:  C00B 20 73 00 START      JSR  CHRGET  ; FETCH CHARACTER
620:  C00E F0 04                BEQ  ENDE
630:  C010 C9 5F                CMP  #CODE   ; COMPARE WITH
                                     FAST TAPE CODE
640:  C012 F0 03                BEQ  FTAPE   ; UNEQUAL
650:  C014 4C E7 A7 ENDE        JMP  OLD     ; => OLD ROUTINE
654:  C017
654:  C017
655:                                     ;EVALUATE FAST TAPE INSTRUCTION
657:  C017
657:  C017
660:  C017 20 73 00 FTAPE      JSR  CHRGET
670:  C01A 20 20 C0            JSR  SUCH
680:  C01D 4C AE A7            JMP  INTER
690:  C020 A2 00              LDX  #0
700:  C022 DD 54 C0 SUCHL      CMP  TAB,X   ; COMPARE WITH
                                     INSTRUCTION
710:  C025 F0 08                BEQ  BEFEHL  ; LETTERS FROM
                                     TABLE
720:  C027 E8                  INX
730:  C028 E4 04              CPX  BEFANZ
740:  C02A D0 F6              BNE  SUCHL
750:  C02C 4C 08 AF            JMP  SYNTAX  ; NOT FASTTAPE
755:  C02F                      INSTRUCTION
755:  C02F
760:  C02F 8A                BEFEHL  TXA
770:  C030 0A                ASL
770:  C031 AA                TAX          ; BEFZAHL * 2
780:  C032 BD 5A C0          LDA  BEFADR+1,X ; INSTRUCTION
790:  C035 48                PHA          ;ADDRESS TO STACK
800:  C036 BD 59 C0          LDA  BEFADR,X
800:  C039 48                PHA
810:  C03A 4C 73 00          JMP  CHRGET  ; AND EXECUTE
815:  C03D                      INSTRUCTION
815:  C03D
820:  C03D C9 53              DATA  CMP  #"S"   ; STORE DATA
830:  C03F F0 07              BEQ  DS
840:  C041 C9 4C              CMP  #"L"   ; LOAD DATA
850:  C043 F0 09              BEQ  DL
860:  C045 4C 08 AF            JMP  SYNTAX
865:  C048
865:  C048
870:  C048 20 73 00 DS        JSR  CHRGET
870:  C04B 4C 11 C3          JMP  DASAV1
880:  C04E 20 73 00 DL        JSR  CHRGET
880:  C051 4C 61 C3          JMP  DATLOD
885:  C054
885:  C054
886:                                     ;TABLE OF INSTRUCTION LETTERS
887:  C054

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

887:  C054
890:  C054 53 4C 56 TAB      .ASC "SLVAD"
895:  C059
895:  C059
896:                                ;TABLE OF INSTRUCTION ADDRESSES
897:  C059
897:  C059
900:  C059 62 C0      BEFADR  .WORDSAVE-1
900:  C05B 6F C1      .WORDLOAD-1
900:  C05D 72 C1      .WORDVERIFY-1
910:  C05F 61 C1      .WORDAPPND1-1
910:  C061 3C C0      .WORDDATA-1
920:  C063
920:  C063
1650: C063
1650: C063
1660: C063
1660: C063
1665:                                ;SAVE ROUTINE
1666:  C063
1666:  C063
1670: C063 A2 05      SAVE    LDX  #$05  ; NUMBER OF SYNCHR.
1680: C065 86 AB      STX   $AB    ; REPETITIONS
1682: C067 A2 00      LDX  #$00    ; CLEAR FLAG
1684: C069 86 02      STX   ABSFLG
1690: C06B 20 B0 C2    JSR   GETPARA ; FETCH
1692: C06E A5 02      LDA   ABSFLG ; PARAMETERS
1694: C070 29 02      AND  #2     ; TEST FOR BIT 2
1695: C072 D0 0B      BNE  ABSOLUT ; GESETZT =>
1700: C074 A2 04      LDX  #$04    ; ABSOLUTE LOAD
1710: C076 B5 2A      LOOP1  LDA  $2A,X  ; RESTORE BASIC
1720: C078 95 AB      STA  $AB,X  ; START
1725: C07A 95 A6      STA  $A6,X
1730: C07C CA      DEX
1740: C07D D0 F7      BNE  LOOP1
1750: C07F 20 38 F8 ABSOLUT JSR  PLKEY
1760: C082 20 8F F6    JSR  SAVING ; OUTPUT
                                'SAVING NAME'
1770: C085 20 FC C0    JSR  MOTOR  ; SWITCH ON MOTOR
1774: C088
1774: C088
1775:                                ;WRITE LEADER
1776:  C088
1776:  C088
1780: C088 20 13 C1    JSR  SYNCH  ; WRITE SYNC. 1
1790: C08B A5 B9      LDA  $B9    ; INCREMENT
1800: C08D 18      CLC      ; SECONDARY ADDRESS
1810: C08E 69 01      ADC  #$01
1820: C090 CA      DEX
1830: C091 20 33 C1    JSR  WBYTE  ; WRITE SA
1840: C094 A2 08      LDX  #$08+COMPU
1850: C096 B9 A7 00 LOOP2  LDA  $A7,Y  ; WRITE START-

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

1860: C099 20 33 C1      JSR  WBYTE   ; AND END ADDRESS
1870: C09C A2 06      LDX  #$06+COMPU
1880: C09E C8        INY
1890: C09F C0 05      CPY  #$05
1900: C0A1 EA        NOP
1910: C0A2 D0 F2      BNE  LOOP2
1920: C0A4 A0 00      LDY  #$00      ;WRITE FILE NAME
1930: C0A6 A2 04      LDX  #$04+COMPU ; AND 'SPACES'
1940: C0A8 B1 BB      LDA  ($BB),Y
1950: C0AA C4 B7      CPY  $B7
1960: C0AC 90 03      BCC  TEXT
1970: C0AE A9 20      LDA  #$20
1980: C0B0 CA        DEX
1990: C0B1 20 33 C1 TEXT JSR  WBYTE
2000: C0B4 A2 05      LDX  #$05
2010: C0B6 C8        INY
2020: C0B7 C0 BB      CPY  #$BB
2030: C0B9 D0 ED      BNE  LOOP3
2034: C0BB
2034: C0BB
2035:                ; WRITE SYNCHRONIZATION 2
2036: C0BB
2036: C0BB
2040: C0BB A9 02      LDA  #$02
2050: C0BD 85 AB      STA  $AB
2060: C0BF 20 13 C1   JSR  SYNCH   ; WRITE
2070: C0C2 98        TYA      ; SYNCHRONIZATION
2080: C0C3 20 33 C1   JSR  WBYTE   ; 0 BYTE =>END OF
                                LEADER
2090: C0C6 84 D7      STY  CHKSUM  ; DELETE CHECK
                                TOTAL BYTE
2100: C0C8 A2 07      LDX  #$07+COMPU
2104: C0CA
2104: C0CA
2105:                ;WRITE PROGRAM
2106: C0CA
2106: C0CA
2110: C0CA EA        NOP
2120: C0CB B1 AC      PRGLOOP LDA  ($AC),Y ;WRITE PROGRAM
2130: C0CD 20 33 C1   JSR  WBYTE   ; ON TO TAPE
2140: C0D0 A2 03      LDX  #$03+COMPU
2150: C0D2 E6 AC      INC  $AC      ; INCREMENT
2160: C0D4 D0 04      BNE  NOHI    ; PROGRAM POINTER
2170: C0D6 E6 AD      INC  $AD
2180: C0D8 CA        DEX
2190: C0D9 CA        DEX
2200: C0DA A5 AC      NOHI  LDA  $AC
2210: C0DC C5 AE      CMP  ENDVEC  ; PROGRAM END
2220: C0DE A5 AD      LDA  $AD      ; REACHERED
2230: C0E0 E5 AF      SBC  ENDVEC+1
2240: C0E2 90 E7      BCC  PRGLOOP ; NOT=> CONTINUE
2250: C0E4 EA        NOP

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

2260: C0E5 A5 D7      LOOP4      LDA  CHKSUM  ; WRITE CHECK SUM
2270: C0E7 20 33 C1      JSR  WBYTE
2280: C0EA A2 07      LDX  #$07+COMPU
2290: C0EC 88          DEY
2300: C0ED D0 F6      BNE  LOOP4
2310: C0EF C8          INY
2320: C0F0 84 C0      STY  MOFLAG  ; PREPARE TO
2330: C0F2 58          CLI           ; SWITCH OFF MOTOR
2340: C0F3 18          CLC
2350: C0F4 A9 00      LDA  #$00
2360: C0F6 8D A0 02      STA  IRQFLG
2370: C0F9 4C 93 FC      JMP  ENDEN
2372: C0FC
2372: C0FC
2373:                ; START MOTOR
2374: C0FC
2374: C0FC
2375: C0FC 20 2C A8 MOTOR JSR  STOP
2380: C0FF A0 00      LDY  #$00
2390: C101 84 C0      STY  MOFLAG
2400: C103 AD 11 D0      LDA  $D011  ; SCREEN OFF
2410: C106 29 EF      AND  #$EF   ; NOT WITH
2420: C108 8D 11 D0      STA  $D011  ; VIC 20
2430: C10B CA          ANLOOP     DEX           ; WAITS TILL
2440: C10C D0 FD      BNE  ANLOOP ; MOTOR HAS
2450: C10E 88          DEY           ; RUN UP
2460: C10F D0 FA      BNE  ANLOOP
2470: C111 78          SEI
2480: C112 60          RTS
2481: C113
2481: C113
2482:                ; GENERATE SYNCHRONIZATION
2484: C113
2484: C113
2490: C113 A0 00      SYNCH     LDY  #$00
2500: C115 A9 02      LOOP5     LDA  #$02   ; WRITE START
2510: C117 20 33 C1      JSR  WBYTE ; BYTE 255-9
2520: C11A A2 07      LDX  #$07+COMPU ; BYTE VALUE
2530: C11C 88          DEY
2540: C11D C0 09      CPY  #$09
2550: C11F D0 F4      BNE  LOOP5
2560: C121 A2 05      LDX  #$05+COMPU
2570: C123 C6 AB      DEC  $AB
2580: C125 D0 EE      BNE  LOOP5
2581: C127
2581: C127
2582:                ; WRITE COUNTDOWN
2584: C127
2584: C127
2590: C127 98          COUNTS    TYA           ; COUNTDOWN
2600: C128 20 33 C1      JSR  WBYTE
2610: C12B A2 07      LDX  #$07+COMPU

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

2620: C12D 88          DEY
2630: C12E D0 F7      BNE  COUNTS
2640: C130 CA          DEX
2650: C131 CA          DEX
2660: C132 60          RTS
2662: C133
2662: C133
2664:                  ;WRITE BYTE ON TAPE
2666: C133
2666: C133
2670: C133 85 BD      WBYTE  STA  $BD
2680: C135 45 D7      EOR   CHKSUM
2690: C137 85 D7      STA  CHKSUM
2700: C139 A9 08      LDA  #$08
2710: C13B 85 A3      STA  BITC
2720: C13D 06 BD      SHIFT  ASL  $BD
2730: C13F A5 01      LDA  PORT
2740: C141 29 F7      AND  #$F7
2750: C143 20 55 C1   JSR  T1LOOP
2760: C146 A2 11      LDX  #$11+COMPU
2770: C148 EA          NOP
2780: C149 09 08      ORA  #$08
2790: C14B 20 55 C1   JSR  T1LOOP
2800: C14E A2 0E      LDX  #$0E+COMPU
2810: C150 C6 A3      DEC  BITC
2820: C152 D0 E9      BNE  SHIFT
2830: C154 60          RTS
2840: C155 CA          T1LOOP  DEX
2850: C156 D0 FD      BNE  T1LOOP
2860: C158 90 05      BCC  BIT0      ; IF CARRY SET
2870: C15A A2 0B      LDX  #$0B      ; INCREASE THE
2880: C15C CA          T2LOOP  DEX          ; LOOP
2890: C15D D0 FD      BNE  T2LOOP
2900: C15F 85 01      BIT0  STA  PORT
2902: C161 60          RTS
2903: C162
2903: C162
2904: C162
2904: C162
2905:                  ;APPEND
2906: C162
2906: C162
2909:                  ;APPEND
2910: C162 A5 2D      APPND1  LDA  $2D
2911: C164 38          SEC
2912: C165 E9 02      SBC  #2
2912: C167 A8          TAY
2912: C168 A5 2E      LDA  $2E
2912: C16A E9 00      SBC  #0
2913: C16C A2 00      LDX  #0
2914: C16E F0 09      BEQ  APPND2
2915: C170

```


ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

2915: C170
2916: C170
2916: C170
2917: ;LOAD
2918: C170
2919: C170
2919: C170
2920: C170 A2 00 LOAD LDX #$00
2925: C172 2C .BYTE$2C
2930: C173 A2 01 VERIFY LDX #$01
2940: C175 A4 2B LDY $2B ; BASIC STRT VCTR
2950: C177 A5 2C LDA $2C ;IN STRT LOCATIONS
2960: C179 86 0A APPND2 STX $0A ; LOAD FLAG
2970: C17B 86 93 STX $93 ; (0=LOAD/1=VERIFY)
2975: C17D A2 00 LDX #0 ; SET
2980: C17F 86 02 STX ABSFLG ; CLEAR ABSOLUTE
2985: C181 84 C3 STY STARTV ; FLAG
2990: C183 85 C4 STA STARTV+1
3000: C185 20 B0 C2 JSR GETPARA
3010: C188 20 8E C1 JSR LOADR ; LOAD AND VERIFY
3020: C18B 4C 7A E1 JMP BRAEND
3022: C18E
3022: C18E
3024: ;LOAD ROUTINE
3026: C18E
3026: C18E
3040: C18E 20 0A C2 LOADR JSR SSYNCH
3050: C191 A5 AB LDA $AB
3060: C193 C9 02 CMP #$02 ; IS PRG STORED
3070: C195 F0 08 BEQ ABSOL ; => LOAD ABSOLUT
3080: C197 C9 01 CMP #$01 ; BYTE NOT 1 =>
3090: C199 D0 F3 BNE LOADR ; CONTINUE
3100: C19B A5 B9 LDA $B9 ; SEC. ADRS = 0
3110: C19D F0 10 BEQ RELLOD ; => LOAD RELOCAT
3112: C19F A9 02 ABSOL LDA #2 ; SET ABSLT FLAG
3113: C1A1 05 02 ORA ABSFLG
3114: C1A3 85 02 STA ABSFLG
3120: C1A5 AD 3C 03 LDA CASBUF ; START ADDRESS
3130: C1A8 85 C3 STA STARTV ; OF BUFFERIN
3140: C1AA AD 3D 03 LDA CASBUF+1 ;START VECTOR
3150: C1AD 85 C4 STA STARTV+1
3155: C1AF 20 02 C3 RELLOD JSR ADTEST
3156: C1B2 A5 02 LDA ABSFLG ; TEST FOR 'WAIT'
3157: C1B4 29 04 AND #4
3158: C1B6 D0 08 BNE NOWAIT
3160: C1B8 20 50 F7 JSR FOUND ;OUTPUT FOUND NAME
3170: C1BB 20 E4 FF WAIT JSR GET ; NOT REQUIRED
3180: C1BE F0 FB BEQ WAIT ; WITH VIC 20
3190: C1C0 20 2C A8 NOWAIT JSR STOP
3200: C1C3 A4 B7 LDY $B7 ; FILE NAME LENGTH
3210: C1C5 F0 0B BEQ NONAME
3220: C1C7 88 TESTNA DEY ; FILE NAME TESTS

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

3230: C1C8 B1 BB          LDA  ($BB),Y
3240: C1CA D9 41 03      CMP  CASBUF+5,Y
3250: C1CD D0 BF          BNE  LOADR
3260: C1CF 98             TYA
3270: C1D0 D0 F5          BNE  TESTNA
3280: C1D2 84 90          STY  STATUS ; CLEAR STATUS
3290: C1D4 20 D2 F5      JSR  LOADING
3300: C1D7 AD 3E 03      LDA  CASBUF+2 ; COMPUTE END
                                ADR. FROM
3310: C1DA 38             SEC
                                ; DIF. START- +END
3320: C1DB ED 3C 03      SBC  CASBUF ; ADR. FROM HEADER
3330: C1DE 08             PHP
                                ; PLUS CERTAIN
3340: C1DF 18             CLC
                                ; START ADDRESS
3350: C1E0 65 C3          ADC  STARTV
3360: C1E2 85 AE          STA  ENDVEC
3370: C1E4 AD 3F 03      LDA  CASBUF+3
3380: C1E7 65 C4          ADC  STARTV+1
3390: C1E9 28             PLP
3400: C1EA ED 3D 03      SBC  CASBUF+1
3410: C1ED 85 AF          STA  ENDVEC+1
3420: C1EF 20 1F C2      JSR  PLOAD
3430: C1F2 A5 BD          LDA  $BD
3440: C1F4 45 D7          EOR  CHKSUM ; TEST CHECK SUM
3450: C1F6 05 90          ORA  STATUS
3460: C1F8 F0 04          BEQ  OK
3470: C1FA A9 FF          LDA  #$FF
3475: C1FC 85 90          STA  STATUS
3480: C1FE A5 02          LDA  ABSFLG ; TEST FLAG
3484: C200 D0 03          BNE  ALTADR ; IF SET => OLD
3486: C202 4C A9 F5      JMP  STOEND ; END VECTORS
3488: C205 A6 2D          LDX  $2D ; RECIEVED
3490: C207 A4 2E          LDY  $2E
3492: C209 60             RTS
3500: C20A 20 58 C2      JSR  SSYNCH
3510: C20D C9 00          CMP  #$00 ; SEEK SYNCHR.
                                BEFORE
3520: C20F F0 F9          BEQ  SSYNCH ; SEEK SYNCHR
                                AFTER HEADER
3530: C211 85 AB          STA  $AB ; SEC. ADRS. + 1
3540: C213 20 86 C2      JSR  SPSTART ; SEEK START OF
                                PRG
3550: C216 91 B2          STA  ($B2),Y
3560: C218 C8             INY
3570: C219 C0 C0          CPY  #$C0
3580: C21B D0 F6          BNE  SPSTART
3590: C21D F0 2D          BEQ  GEFUN ; END LEADER
3600: C21F 20 58 C2      JSR  PLOAD
3610: C222 20 86 C2      JSR  LLOOP
3620: C225 C4 93          CPY  $93
3630: C227 D0 02          BNE  VERGL ; WITH VERIFY
3640: C229 91 C3          STA  (STARTV),Y ; ONLY COMPARE
3650: C22B D1 C3          CMP  (STARTV),Y
                                VERGL

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

3660: C22D F0 02          BEQ  GLEICH
3670: C22F 86 90          STX  STATUS
3680: C231 45 D7          GLEICH EOR  CHKSUM ; COMPUTE CHECK
3690: C233 85 D7          STA  CHKSUM ; SUM
3700: C235 E6 C3          INC  STARTV ; INCREASE ADRS.
3710: C237 D0 02          BNE  NOTHI
3720: C239 E6 C4          INC  STARTV+1
3730: C23B A5 C3          NOTHI LDA  STARTV
3740: C23D C5 AE          CMP  ENDVEC ; END ADDRESS
3750: C23F A5 C4          LDA  STARTV+1 ; REACHED
3760: C241 E5 AF          SBC  ENDVEC+1
3770: C243 90 DD          BCC  LLOOP ; NO => CONTINUE
3780: C245 20 86 C2      JSR  HOLBYT
3790: C248 20 FC C0      JSR  MOTOR
3800: C24B C8            INY
3810: C24C 84 C0          GEFUN STY  MOFLAG
3820: C24E 58            CLI
3830: C24F 18            CLC
3840: C250 A9 00          LDA  #$00
3850: C252 8D A0 02      STA  IRQFLG
3860: C255 4C 93 FC          JMP  ENDEN
3870: C258 20 17 F8      SCOUNT JSR  AKEY ; SEEK COUNTDOWN
3880: C25B 20 FC C0      JSR  MOTOR
3890: C25E 84 D7          STY  CHKSUM
3900: C260 A9 07          LDA  #$07 ; ($27) VALUE FOR
                                     TIMER LO
3910: C262 8D 06 DD      STA  CIA1+6 ; (PORT+8) IN
                                     TIMER LO
3920: C265 A2 01          LDX  #$01 ;VALUE FOR TIMER HI
3930: C267 20 99 C2      SSTART JSR  HOLBIT ; SEEK START
3940: C26A 26 BD          ROL  $BD
3950: C26C A5 BD          LDA  $BD
3960: C26E C9 02          CMP  #$02 ; START FOUND
3970: C270 D0 F5          BNE  SSTART
3980: C272 A0 09          LDY  #$09 ; => SEEK COUNTDOWN
3990: C274 20 86 C2      ENDE2 JSR  HOLBYT ;SEEK END '2' -
4000: C277 C9 02          CMP  #$02 ; BYTES
4010: C279 F0 F9          BEQ  ENDE2
4020: C27B C4 BD          COUNTL CPY  $BD ; TEST THE
4030: C27D D0 E8          BNE  SSTART ;COUNTDOWN
4040: C27F 20 86 C2      JSR  HOLBYT
4050: C282 88            DEY
4060: C283 D0 F6          BNE  COUNTL
4070: C285 60            RTS
4072: C286
4072: C286
4075: ; BYTE FROM HOLEN
4076: C286
4076: C286
4080: C286 A9 08          HOLBYT LDA  #$08 ; 8 BIT
4090: C288 85 A3          STA  BITC
4100: C28A 20 99 C2      SHIFT7 JSR  HOLBIT

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

4110: C28D 26 BD          ROL  $BD      ; MOVES CARRY TO
4120: C28F EA          NOP                ; INPUT BUFFER
4130: C290 EA          NOP
4140: C291 EA          NOP
4150: C292 C6 A3       DEC  BITC
4160: C294 D0 F4       BNE  SHIFT7
4170: C296 A5 BD       LDA  $BD
4180: C298 60         RTS
4182: C299
4182: C299
4184:                    ; FETCH BIT FROM TAPE FOR C-64
4185: C299
4185: C299
4190: C299 A9 10       HOLBIT LDA  #$10
4200: C29B 2C 0D DC WAITDA BIT  CIA2+13 ;WAITS FOR SIGNAL
4210: C29E F0 FB       BEQ  WAITDA   ; TO FLAG
4220: C2A0 AD 0D DD       LDA  CIA1+13 ; LOAD ICR, HAS
4225:                    ; => BIT 1 IS SET ; REACHED TB 0
4230: C2A3 8E 07 DD       STX  CIA1+7  ; VALUE IN TB HI
4240: C2A6 48          PHA                ; SAVE ICR
4250: C2A7 A9 19       LDA  #$19      ; START TB, COUNT
4260: C2A9 8D 0F DD       STA  CIA1+15 ; ONCE
4270: C2AC 68          PLA                ; FETCH ICR
4280: C2AD 4A          LSR  A          ; MOVE BIT 1 TO
4290: C2AE 4A          LSR  A          ; CARRY
4300: C2AF 60         RTS
4310: C2B0
4310: C2B0
4320:                    ; 'HOLBIT' FOR VC20
4330: C2B0
4330: C2B0
4340:                    ;HOLBIT LDA #$02
4350:                    ;WAITDA BIT PORT+13 ; TEST FOR IFR BIT
4360:                    ; BEQ WAITDA ; 1 TEST
4370:                    ; LDA PORT+13
4380:                    ; STX PORT+9 ; SET TIMER HI
4390:                    ; BIT PORT+1
4400:                    ; ASL
4410:                    ; ASL
4420:                    ; ASL
4430:                    ; RTS
4432: C2B0
4432: C2B0
4435:                    ; FETCH PARAMETERS
4436: C2B0
4436: C2B0
4450: C2B0 A9 00       GETPARA LDA  #0
4460: C2B2 20 BD FF       JSR  SETNAM
4470: C2B5 A2 01       LDX  #1      ; DEFAULT FOR GA
4480: C2B7 A0 00       LDY  #00     ; DEFAULT FOR SA
4490: C2B9 20 BA FF       JSR  SETLE'S
4510: C2BC 20 06 E2       JSR  WRCHAR  ; FURTHER CHARS

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

4520: C2BF 20 57 E2      JSR  HFNAM
4540: C2C2 20 06 E2      JSR  WRCHAR
4550: C2C5 20 00 E2      JSR  HOLPAR
4560: C2C8 8A           TXA
4570: C2C9 A8           TAY
4580: C2CA 20 BA FF      JSR  SETLFS
4600: C2CD 20 06 E2      JSR  WRCHAR
4610: C2D0 20 F4 C2      JSR  HOLWERT
4620: C2D3 86 AC        STX  $AC
4622: C2D5 86 A7        STX  $A7
4624: C2D7 84 A8        STY  $A8
4630: C2D9 84 AD        STY  $AD
4632: C2DB A2 01        LDX  #1          ; FLAG FOR LOAD TO
4634: C2DD 86 02        STX  ABSFLG     ; CERTAIN ADDRESS
4650: C2DF 20 06 E2      JSR  WRCHAR     ; AND SAVE
4660: C2E2 20 F4 C2      JSR  HOLWERT
4670: C2E5 86 AE        STX  $AE
4672: C2E7 86 A9        STX  $A9
4674: C2E9 84 AA        STY  $AA
4680: C2EB 84 AF        STY  $AF
4685: C2ED A5 02        LDA  ABSFLG
4690: C2EF 09 02        ORA  #2
4695: C2F1 85 02        STA  ABSFLG
4700: C2F3 60           RTS
4710: C2F4 20 0E E2      JSR  CHRIN     ; FETCH NEXT VALUE
4720: C2F7 20 8A AD      JSR  FRMNUM    ; EVALUATE EXPREN
4730: C2FA 20 F7 B7      JSR  FACINT    ; CONVERT TO
4740: C2FD A6 14        LDX  $14      ; 2BYTE VALUE
4750: C2FF A4 15        LDY  $15
4760: C301 60           RTS
4770: C302 A5 02        LDA  ABSFLG    ; LOADFLAG
4780: C304 29 01        AND  #1
4790: C306 F0 08        BEQ  NORM     ; UNEQUAL => LOAD
4800: C308 A5 AC        LDA  $AC      ; NORMALLY
4810: C30A 85 C3        STA  STARTV
4820: C30C A5 AD        LDA  $AD
4830: C30E 85 C4        STA  STARTV+1
4840: C310 60           RTS
4940: C311
4940: C311
4945: C311
4945: C311
4950:                      ; DATASAVE
4970: C311
4970: C311
5000: C311 20 26 B5      DASAV1 JSR  GABCOL
5010: C314 A2 05        LDX  #5
5020: C316 86 AB        STX  $AB
5030: C318 20 B0 C2      JSR  GETPARA
5040: C31B A9 01        LDA  #1      ; SA=1
5050: C31D 85 B9        STA  $B9
5060: C31F A2 04        LDX  #4

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

5070: C321 B5 2C      ULOOP   LDA  $2C,X   ; RE-STORE START
5080: C323 95 AB      STA  $AB,X   ; AND END ADRS
5085: C325 95 A6      STA  $A6,X   ; OF VARIABLES
5090: C327 CA          DEX
5100: C328 D0 F7      BNE  ULOOP
5110: C32A 20 7F C0    JSR  ABSOLUT ; STORE
5112: C32D A2 00      LDX  #0
5114: C32F 86 B7      STX  $B7     ; NO FILE NAME
5115: C331 A2 05      LDX  #5
5117: C333 86 AB      STX  $AB
5119: C335 CA          DEX
5130: C336 B5 2E      U2LOOP  LDA  $2E,X   ; RE-STORE ARRAY
5140: C338 95 AB      STA  $AB,X   ; STRT & END ARDS
5145: C33A 95 A6      STA  $A6,X   ; RE-STORE END
5150: C33C CA          DEX         ; ADDRESSES
5160: C33D D0 F7      BNE  U2LOOP
5170: C33F 20 7F C0    JSR  ABSOLUT ; STORE
5175: C342 A2 05      LDX  #5
5177: C344 86 AB      STX  $AB
5190: C346 A5 33      LDA  $33     ; RESTORE STRING
5200: C348 A6 34      LDX  $34     ; START
5210: C34A 85 AC      STA  $AC
5215: C34C 85 A7      STA  $A7
5220: C34E 86 AD      STX  $AD
5225: C350 86 A8      STX  $A8
5230: C352 A5 37      LDA  $37     ; AND END ADDRESSES
5240: C354 A6 38      LDX  $38     ;
5250: C356 85 AE      STA  $AE
5255: C358 85 A9      STA  $A9
5260: C35A 86 AF      STX  $AF
5265: C35C 86 AA      STX  $AA
5270: C35E 4C 7F C0    JMP  ABSOLUT ; STORE
5300: C361
5300: C361
5305: C361
5305: C361
5310:                ; DATALOAD
5320: C361
5320: C361
5410: C361 20 B0 C2 DATLOD JSR  GETPARA
5420: C364 A9 01      LDA  #1     ; LOAD TO CERTIAN
5430: C366 85 02      STA  ABSFLG ; ADDRESS
5440: C368 A5 2D      LDA  $2D    ; RE-STORE VARIABLE
5450: C36A A6 2E      LDX  $2E    ; START ADDRESS AS
5460: C36C 85 AC      STA  $AC    ; START ADDRESS
5470: C36E 86 AD      STX  $AD
5480: C370 20 8E C1    JSR  LOADR  ; LOAD
5482: C373 A9 05      LDA  #4+1   ; DO NOT WAIT ;
5486: C375 85 02      STA  ABSFLG ; LOAD TO ADDRESS
5490: C377 A9 00      LDA  #0
5500: C379 85 B7      STA  $B7    ; NO FILENAME
5510: C37B A6 AE      LDX  $AE    ; END OF VARIABLES

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

5520: C37D A4 AF          LDY $AF
5530: C37F 86 2F          STX $2F          ; IN VECTOR
5540: C381 84 30          STY $30
5550: C383 86 AC          STX $AC          ; AND LOAD START
5560: C385 84 AD          STY $AD          ; VECTOR
5570: C387 20 8E C1      JSR LOADR        ; LOAD
5580: C38A A6 AE          LDX $AE          ; ARRAY END
5590: C38C A4 AF          LDY $AF
5600: C38E 86 31          STX $31          ; IN VECTOR
5610: C390 84 32          STY $32
5620: C392 A9 04          LDA #4           ; LOAD FROM ADDRESS
5630: C394 85 02          STA ABSFLG      ; FROM HEADER
5640: C396 20 8E C1      JSR LOADR        ; LOAD
5642: C399 AD 3C 03      LDA CASBUF
5642: C39C 85 33          STA $33
5644: C39E AD 3D 03      LDA CASBUF+1
5644: C3A1 85 34          STA $34
5650: C3A3 20 B7 FF      JSR HOLSTA      ; FETCH STATUS
5660: C3A6 25 BF          AND $BF         ; CLEAR EOF-BIT
5670: C3A8 F0 05          BEQ NOFEHL      ; NO ERROR
5680: C3AA A2 1D          LDX #$1D        ; OFFSET FOR
                    'LOAD ERROR'
5690: C3AC 4C 37 A4      JMP OUTERR      ; OUTPUT ERROR
5700: C3AF A2 19          LDX #$19        ; STRING-
                    DESCRIPTOR
5710: C3B1 86 16          STX $16         ; RESET INDEX
5720: C3B3 A9 00          LDA #00
5730: C3B5 85 3A          STA $3A
5730: C3B7 85 10          STA $10         ; BLOCK COUNT
5740: C3B9 60          RTS
1C000-C3BA
NO ERRORS

```

```

1000 REM LOADER FOR C-64
1010 E=50105:A=49152:PS=0
1020 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
1030 IFPS<>120492THENPRINT"ERROR IN DATA
":END
1040 SYSA
10000 DATA169,11,141,8,3,169,192,141,9,3
,96,32,115,0,240,4,201,95,240,3,76,231
10010 DATA167,32,115,0,32,32,192,76,174,
167,162,0,221,84,192,240,8,232,228,4
10020 DATA208,246,76,8,175,138,10,170,18
9,90,192,72,189,89,192,72,76,115,0,201
10030 DATA83,240,7,201,76,240,9,76,8,175
,32,115,0,76,17,195,32,115,0,76,97,195
10040 DATA83,76,86,65,68,98,192,111,193,
114,193,97,193,60,192,162,5,134,171,162
10050 DATA0,134,2,32,176,194,165,2,41,2,
208,11,162,4,181,42,149,171,149,166,202
10060 DATA208,247,32,56,248,32,143,246,3
2,252,1,92,32,19,193,165,185,24,105,1
10070 DATA202,32,51,193,162,8,185,167,0,
32,51,193,162,6,200,192,5,234,208,242
10080 DATA160,0,162,4,177,187,196,183,14
4,3,169,32,202,32,51,193,162,5,200,192
10090 DATA187,208,237,169,2,133,171,32,1
9,193,152,32,51,193,132,215,162,7,234
10100 DATA177,172,32,51,193,162,3,230,17
2,208,4,230,173,202,202,165,172,197,174
10110 DATA165,173,229,175,144,231,234,16
5,215,32,51,193,162,7,136,208,246,200
10120 DATA132,192,88,24,169,0,141,160,2,
76,147,252,32,44,168,160,0,132,192,173
10130 DATA17,208,41,239,141,17,208,202,2
08,253,136,208,250,120,96,160,0,169,2
10140 DATA32,51,193,162,7,136,192,9,208,
244,162,5,198,171,208,238,152,32,51,193
10150 DATA162,7,136,208,247,202,202,96,1
33,189,69,215,133,215,169,8,133,163,6
10160 DATA189,165,1,41,247,32,85,193,162
,17,234,9,8,32,85,193,162,14,198,163
10170 DATA208,233,96,202,208,253,144,5,1
62,11,202,208,253,133,1,96,165,45,56
10180 DATA233,2,168,165,46,233,0,162,0,2
40,9,162,0,44,162,1,164,43,165,44,134
10190 DATA10,134,147,162,0,134,2,132,195
,133,196,32,176,194,32,142,193,76,122
10200 DATA225,32,10,194,165,171,201,2,24
0,8,201,1,208,243,165,185,240,16,169
10210 DATA2,5,2,133,2,173,60,3,133,195,1
73,61,3,133,196,32,2,195,165,2,41,4,208
10220 DATA8,32,80,247,32,228,255,240,251
,32,44,168,164,183,240,11,136,177,187

```


10230 DATA17,65,3,208,191,152,208,245,1
 32,144,32,210,245,173,62,3,56,237,60
 10240 DATA3,8,24,101,195,133,174,173,63,
 3,101,196,40,237,61,3,133,175,32,31,194
 10250 DATA165,189,69,215,5,144,240,4,169
 ,255,133,144,165,2,208,3,76,169,245,166
 10260 DATA45,164,46,96,32,88,194,201,0,2
 40,249,133,171,32,134,194,145,178,200
 10270 DATA192,192,208,246,240,45,32,88,1
 94,32,134,194,196,147,208,2,145,195,209
 10280 DATA195,240,2,134,144,69,215,133,2
 15,230,195,208,2,230,196,165,195,197
 10290 DATA174,165,196,229,175,144,221,32
 ,134,194,32,252,192,200,132,192,88,24
 10300 DATA169,0,141,160,2,76,147,252,32,
 23,248,32,252,192,132,215,169,7,141,6
 10310 DATA221,162,1,32,153,194,38,189,16
 5,189,201,2,208,245,160,9,32,134,194
 10320 DATA201,2,240,249,196,189,208,232,
 32,134,194,136,208,246,96,169,8,133,163
 10330 DATA32,153,194,38,189,234,234,234,
 198,163,208,244,165,189,96,169,16,44
 10340 DATA13,220,240,251,173,13,221,142,
 7,221,72,169,25,141,15,221,104,74,74
 10350 DATA96,169,0,32,189,255,162,1,160,
 0,32,186,255,32,6,226,32,87,226,32,6
 10360 DATA226,32,0,226,138,168,32,186,25
 5,32,6,226,32,244,194,134,172,134,167
 10370 DATA132,168,132,173,162,1,134,2,32
 ,6,226,32,244,194,134,174,134,169,132
 10380 DATA170,132,175,165,2,9,2,133,2,96
 ,32,14,226,32,138,173,32,247,183,166
 10390 DATA20,164,21,96,165,2,41,1,240,8,
 165,172,133,195,165,173,133,196,96,32
 10400 DATA38,181,162,5,134,171,32,176,19
 4,169,1,133,185,162,4,181,44,149,171
 10410 DATA149,166,202,208,247,32,127,192
 ,162,0,134,183,162,5,134,171,202,181
 10420 DATA46,149,171,149,166,202,208,247
 ,32,127,192,162,5,134,171,165,51,166
 10430 DATA52,133,172,133,167,134,173,134
 ,168,165,55,166,56,133,174,133,169,134
 10440 DATA175,134,170,76,127,192,32,176,
 194,169,1,133,2,165,45,166,46,133,172
 10450 DATA134,173,32,142,193,169,5,133,2
 ,169,0,133,183,166,174,164,175,134,47
 10460 DATA132,48,134,172,132,173,32,142,
 193,166,174,164,175,134,49,132,50,169
 10470 DATA4,133,2,32,142,193,173,60,3,13
 3,51,173,61,3,133,52,32,183,255,37,191
 10480 DATA240,5,162,29,76,55,164,162,25,
 134,22,169,0,133,58,133,16,96

```

100 REM BASIC LOADER FOR VIC-20
110 E=29611:A=28672:PS=0
120 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
130 IFPS<>114748THENPRINT"ERROR IN DATA"
:END
140 POKE55,0:POKE56,112
150 SYSA:NEW
160 DATA169,11,141,8,3,169,112,141,9,3,9
6,32,115,0,240,4,201,95,240,3,76,231
170 DATA199,32,115,0,32,32,112,76,174,19
9,162,0,221,84,112,240,8,232,228,4
180 DATA208,246,76,8,207,138,10,170,189,
90,112,72,189,89,112,72,76,115,0,201
190 DATA83,240,7,201,76,240,9,76,8,207,3
2,115,0,76,3,115,32,115,0,76,83,115
200 DATA83,76,86,65,68,98,112,105,113,10
8,113,91,113,60,112,162,5,134,171,162
210 DATA0,134,2,32,162,114,165,2,41,2,20
8,11,162,4,181,42,149,171,149,166,202
220 DATA208,247,32,183,248,32,40,247,32,
252,112,32,11,113,165,185,24,105,1
230 DATA202,32,43,113,162,10,185,167,0,3
2,43,113,162,8,200,192,5,234,208,242
240 DATA160,0,162,6,177,187,196,183,144,
3,169,32,202,32,43,113,162,5,200,192
250 DATA187,208,237,169,2,133,171,32,11,
113,152,32,43,113,132,215,162,9,234
260 DATA177,172,32,43,113,162,5,230,172,
208,4,230,173,202,202,165,172,197,174
270 DATA165,173,229,175,144,231,234,165,
215,32,43,113,162,9,136,208,246,200
280 DATA132,192,88,24,169,0,141,160,2,76
,207,252,32,44,200,160,0,132,192,202
290 DATA208,253,136,208,250,120,96,160,0
,169,2,32,43,113,162,9,136,192,9,208
300 DATA244,162,7,198,171,208,238,152,32
,43,113,162,9,136,208,247,202,202,96
310 DATA133,189,69,215,133,215,169,8,133
,163,6,189,173,32,145,41,247,32,78
320 DATA113,162,19,234,9,8,32,78,113,162
,16,198,163,208,232,96,202,208,253
330 DATA144,5,162,11,202,208,253,141,32,
145,96,165,45,56,233,2,168,165,46,233
340 DATA0,162,0,240,9,162,0,44,162,1,164
,43,165,44,134,10,134,147,162,0,134
350 DATA2,132,195,133,196,32,162,114,32,
136,113,76,119,225,32,255,113,165,171
360 DATA201,2,240,8,201,1,208,243,165,18
5,240,16,169,2,5,2,133,2,173,60,3,133
370 DATA195,173,61,3,133,196,32,244,114,
165,2,41,4,208,3,32,211,247,32,44,200
380 DATA164,183,240,11,136,177,187,217,6

```

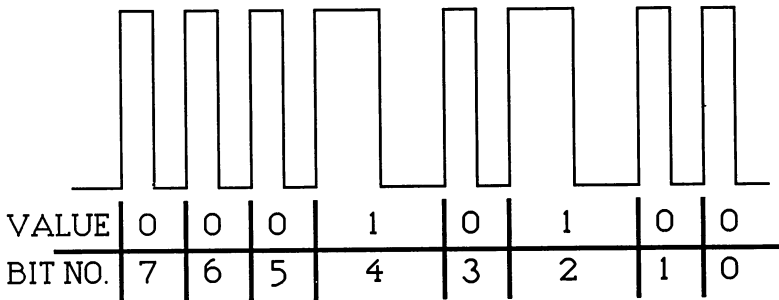
5, 3, 208, 196, 152, 208, 245, 132, 144, 32
 390 DATA106, 246, 173, 62, 3, 56, 237, 60, 3, 8, 2
 4, 101, 195, 133, 174, 173, 63, 3, 101, 196
 400 DATA40, 237, 61, 3, 133, 175, 32, 20, 114, 16
 5, 189, 69, 215, 5, 144, 240, 4, 169, 255, 133
 410 DATA144, 165, 2, 208, 3, 76, 65, 246, 166, 45
 , 164, 46, 96, 32, 77, 114, 201, 0, 240, 249
 420 DATA133, 171, 32, 123, 114, 145, 178, 200, 1
 92, 192, 208, 246, 240, 45, 32, 77, 114, 32
 430 DATA123, 114, 196, 147, 208, 2, 145, 195, 20
 9, 195, 240, 2, 134, 144, 69, 215, 133, 215
 440 DATA230, 195, 208, 2, 230, 196, 165, 195, 19
 7, 174, 165, 196, 229, 175, 144, 221, 32, 123
 450 DATA114, 32, 252, 112, 200, 132, 192, 88, 24
 , 169, 0, 141, 160, 2, 76, 207, 252, 32, 148
 460 DATA248, 32, 252, 112, 132, 215, 169, 39, 14
 1, 40, 145, 162, 1, 32, 142, 114, 38, 189, 165
 470 DATA189, 201, 2, 208, 245, 160, 9, 32, 123, 1
 14, 201, 2, 240, 249, 196, 189, 208, 232, 32
 480 DATA123, 114, 136, 208, 246, 96, 169, 8, 133
 , 163, 32, 142, 114, 38, 189, 234, 234, 234
 490 DATA198, 163, 208, 244, 165, 189, 96, 169, 2
 , 44, 45, 145, 240, 251, 173, 45, 145, 142, 41
 500 DATA145, 44, 33, 145, 10, 10, 10, 96, 169, 0,
 32, 189, 255, 162, 1, 160, 0, 32, 186, 255, 32
 510 DATA3, 226, 32, 84, 226, 32, 3, 226, 32, 253,
 225, 138, 168, 32, 186, 255, 32, 3, 226, 32
 520 DATA230, 114, 134, 172, 134, 167, 132, 168,
 132, 173, 162, 1, 134, 2, 32, 3, 226, 32, 230
 530 DATA114, 134, 174, 134, 169, 132, 170, 132,
 175, 165, 2, 9, 2, 133, 2, 96, 32, 11, 226, 32
 540 DATA138, 205, 32, 247, 215, 166, 20, 164, 21
 , 96, 165, 2, 41, 1, 240, 8, 165, 172, 133, 195
 550 DATA165, 173, 133, 196, 96, 32, 38, 213, 162
 , 5, 134, 171, 32, 162, 114, 169, 1, 133, 185
 560 DATA162, 4, 181, 44, 149, 171, 149, 166, 202
 , 208, 247, 32, 127, 112, 162, 0, 134, 183, 162
 570 DATA5, 134, 171, 202, 181, 46, 149, 171, 149
 , 166, 202, 208, 247, 32, 127, 112, 162, 5, 134
 580 DATA171, 165, 51, 166, 52, 133, 172, 133, 16
 7, 134, 173, 134, 168, 165, 55, 166, 56, 133
 590 DATA174, 133, 169, 134, 175, 134, 170, 76, 1
 27, 112, 32, 162, 114, 169, 1, 133, 2, 165, 45
 600 DATA166, 46, 133, 172, 134, 173, 32, 136, 11
 3, 169, 5, 133, 2, 169, 0, 133, 183, 166, 174
 610 DATA164, 175, 134, 47, 132, 48, 134, 172, 13
 2, 173, 32, 136, 113, 166, 174, 164, 175, 134
 620 DATA49, 132, 50, 169, 4, 133, 2, 32, 136, 113
 , 173, 60, 3, 133, 51, 173, 61, 3, 133, 52, 32
 630 DATA183, 255, 37, 191, 240, 5, 162, 29, 76, 5
 5, 196, 162, 25, 134, 22, 169, 0, 133, 58, 133
 640 DATA16, 96

12.1 PROGRAM DESCRIPTION

For the benefit of those of you who are interested and Assembler programming, I should like to go into greater detail on the FastTape program. You may come up with a few ideas of your own in the way of improvements which you would like to incorporate into the program for your own use. By way of additional reading matter on the subject, I would recommend the corresponding chapters on the input and output modules in the ABACUS Software book ANATOMY of the COMMODORE 64.

Principle of Operation

The individual bits are recorded on tape by means of square-wave pulses. Byte 20 - in binary form %00010100 - looks like this:



A program is stored in the following format:

1. Synchronization 1

5 times (246 times byte 2). In this way, synchronization with the start of a byte is achieved during loading.

Countdown bytes 9, 8, 7, 6, 5, 4, 3, 2, 1. This identifies the end of synchronization.

2. Leader, consisting of

Secondary address 1
Start and end address
File Name

The leader is always filled up with "SPACE", to take up 192 characters, and is stored in the cassette buffer during loading. So here again, it is possible to transfer machine code programs with the file name, as described in Chapter 7.3.

3. Synchronization 2

As Synchronization 1, but only twice.

4. Zero byte as indication that data following.

5. Data

6. Check sum, 255 times

In the INIT subroutine, the vector for the interpreter loop is re-directed to the FastTape program. In this way, we can ensure that, every time an instruction is evaluated, control jumps to the FastTape program. In the START subroutine, the program checks whether the first character encountered is the FastTape code (← arrow pointing to the left). If this is not the case, control returns to the interpreter loop.

However, if the FastTape code is found, then the corresponding start address is determined by means of the SEEK subroutine, by comparing with the TAB table of letters. If the corresponding letter is not found, then a "?SYNTAX ERROR" is output.

Otherwise, the corresponding start address of the instruction is pushed onto the STACK. This causes the program to branch after RTS in the CHRGET routine to the corresponding address plus one, and to return after execution to the address \$C10D(\$701D).

Before I proceed to explain the individual main programs, I should like to describe some of the more frequently used subroutines.

Write Byte on Tape: WBYTE \$C133

Transfer of the byte to be written to tape takes place via the accumulator. This byte is written to the storage location \$BD and then linked with the EXOR check sum by means of an EXCLUSIVE OR.

In \$C139, 8 (one byte = 8 bits) is written into the bit counter. The loop, which writes 8 bits onto tape, starts with SHIFT. The highest-value bit is first of all moved to

the Carry flag. The PORT byte is now loaded into the accumulator and the write byte (bit 3) is deleted.

The write line is now "high", corresponding to the time loop T1LOOP (if the Carry flag is set, then T2LOOP as well). At the end of the time loop, the contents of the accumulator are transferred to the port, with the result that the write line goes "low".

In \$C149, bit three in the accumulator is set again, and the program branches once again to the time loop T1LOOP. At the end of the loop, the write line becomes "high" again when the contents of the accumulator are transferred to the port.

In the \$C150 the bit counter is decremented and tested. If all eight bits have been transferred, the program branches back.

Read Byte from Tape: HOLBYTE \$C286

Starting at \$C286, the bit counter is set to 8. Then eight bits in succession are loaded by means of HOLBIT and moved into \$BD via the Carry flag. At the end (\$C269), the byte read in is transferred via the accumulator.

Read Bit from Tape: HOLBIT \$C299

Loop \$C299 - \$C29E waits until bit 4 in the ICR2 of CIA2 goes "high". This bit is set when there is a signal on the Read line (D/4) of the cassette port. The ICR1 is now loaded, the X-register is written into counter B of CIA2 as a high byte, and ICR1 is stored into the STACK.

In \$C2A9, bits 1, 3 and 4 of the control register B are set, with the result that Counter B counts down once and is

started. The contents of ICR1, which were saved, are brought back again and bit 1 is moved to the Carry Flag position by a double right shift. This bit goes "high" when the counter has reached zero, before being started again.

In this way one can ascertain whether it took a long time for the read line to go "high" again, which corresponds to the setting of a bit.

Fetch Parameters: GETPARA \$C2B0

This routine fetches the parameters which have been transferred with the instructions and writes them into the corresponding storage locations. In addition, flag ABSFLG is set, depending on the parameters.

This flag is a binary flag and controls the following functions:

Bit	Value	Effect
0	1	1 = Load file at address given with L instruction
1	2	1 = Load or store file absolutely
2	4	1 = With C 64, do not wait but load immediately

Write Synchronization: SYNCH \$C113

This subroutine writes byte 2 onto tape 246 times as often as specified by \$AB. Finally, a countdown of the bytes 9, 8, 7, 6, 5, 4, 3, 2, 1 is written onto the tape.

Seek Synchronization: SCOUNT \$C258

First of all, the recorder key is checked. The motor is then started, the screen is switched off (only C-64) and the check sum byte is set to zero. From \$C260 onwards, 7 is written into the low byte and 1 is loaded into the X register for the timer's high byte.

Starting with SSTART, bits are read in until the byte value of the bits read in is exactly 2. All subsequent two's bytes for the synchronization are then read in, before determining whether a countdown from 9 follows.

If an error-free countdown is found, then control branches back to the main program.

Read in leader: SSYNCH \$C20A

This subroutine uses SCOUNT to look for the next synchronization. With the aid of the last byte read in by SCOUNT, it determines whether Synchronization 1 or 2 is involved. If the synchronization found is not Synchronization 1, the search continues. Otherwise, the last byte is stored as a secondary address in \$AB. The leader is then read in and stored in the cassette buffer. The motor is switched off via the operating system routine (from \$FC93) and, in the case of the C-64, the screen is switched on again.

Load Program: PLOAD \$C21F

The program is read in using this subroutine. First of all, Synchronization 2 is sought using SCOUNT, and in this way the read routine is synchronized to the tape.

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

The actual load routine used is LLOOP. A byte is read in with HOLBYT and, indirectly indexed, written into storage via the start vector and compared. With V (VERIFY), only comparison takes place. If any error occurs, the status byte is set. The EXOR check sum is then formed and the start vector incremented.

Loop LLOOP is then repeated until the difference between the start and end vectors is zero. The stored check sum is read in, the motor is switched off and, in the case of the C-64, the screen switched on. Control returns to the main program via the operating system routine \$FC93.

Finally, I should like to describe the main programs in greater detail.

SAVE ROUTINE \$C063

First of all, the repetition count for Synchronization 1 is stored in \$AB and flag ABSFLG is cleared. The individual file parameters are then read in with GETPARA. Depending on the setting of ABSFLG, the basic vectors are then transferred to the load, start and end vectors.

Starting with \$C07F, the recorder key is checked, "SAVING name" is output, the motor started and, with the C-64, the screen switched off. The synchronization is now written onto tape via the subroutine SYNCH. The secondary address, incremented by 1, is then transferred to tape. Following this, the leader, consisting of start and end addresses, file name and filler bytes, is written onto tape, starting with \$C094. Finally comes Synchronization 2, which for identification purposes concludes with a zero byte.

The actual program storage routine starts at \$C0CB (PRGLOOP). The program is read out of storage, indirectly indexed, via the program start vector \$AC/\$AD and transferred to the datassette. The vector \$AC/\$AD is then incremented and compared with the program end vector.

This loop is repeated until the start vector is identical to the end vector. To conclude, the EXOR check sum is written onto tape 255 times. Value 1 is now written into the motor flag, in order to prepare for switching off the motor, and the storage operation is now terminated via the operating system routine in \$FC93.

LOAD ROUTINE \$C18E

First of all, subroutine SSYNCH is used to search for a synchronization. If the last byte of the synchronization found - which was written in \$AB - is zero, then this is Synchronization 2, and control branches back to the start of this routine. If \$AB is not equal to zero, then Synchronization 1 has been found and the value in \$AB is the secondary address.

If the file has been stored with secondary address 1, or if the secondary address transferred with the load instruction is equal to 1, then bit 1 is set in ABSFLG and the start address from the cassette buffer is written to the loading start vector. "Found name" is then output, starting with RELLOD.

Subroutine ADTEST now checks bit zero of ABSFLG to determine whether another address transferred by the L instruction should be used as loading start vector instead of the start address in the cassette buffer. If bit zero of ABSFLG is

set, the address transferred with L, which has been stored in \$AC/\$AD, is written to the start vector.

Depending on bit 2 of ABSFLG, the computer will wait for you to press the key or not, as the case may be. After testing to see whether the STOP key was pressed, the computer compares the name of the file found with the name of the file required, if a name was transferred. If the names agree, the loading process can begin and the status byte is deleted.

The end address is calculated by adding the program length, obtained from the difference between the start and end addresses in the leader, to the start vector and then written to the end vector. Subroutine PLOAD then loads the program.

The check totals are compared and the status byte set if required. If there is no bit set in ABSFLG, then the Basic vectors are set to the program end by the operating system routine STOEND.

The APPEND, LOAD and VERIFY routines all run via the LOAD routine.

In the case of APPEND, the start vector is set according to the Basic program vector. Subsequent loading is relative. With VERIFY, the VERIFY flag is set to 1 via the X register, and to zero in the case of LOAD.

Before the LOAD routine, further parameters are read in by GETPARA and the corresponding value is set in ABSFLG.

DATA STORAGE AND READ DATA

Data is stored in three blocks.

Block 1: Variables
area between pointers 45, 46 and 47, 48

Block 2: Fields
area between pointers 47, 48 and 49, 50

Block 3: Strings
area between pointers 51, 52 and 55, 56

The length of the individual blocks is given by the start and end addresses in the leader.

DATASAVE \$C311

Before the data can be stored, they have to be compressed so as to take up the minimum space. For this reason, Garbage Collection is called at the beginning.

The file name is now set using GETPARA. Then 1 is given as the secondary address, since a certain memory area is to be transferred to storage. The pointers to the start and end addresses of the variable area are transferred to the start and end vectors by means of ULOOP.

Finally, this area is stored absolutely by means of subroutine ABSOLUTE. The fields and strings are then stored without file names by means of ABSOLUTE, analogously to the variables.

DATALOAD \$C361

First of all, the file name is set by means of GETPARA. Setting bit 0 in ABSFLG causes the area to be loaded at a certain address and the addresses in the leader to be ignored.

The variable start address is transferred to the start vector as a load address. The variable area is now read in by LOADR. After that, bit 2 in ABSFLG is also set, so that the computer will not wait for the operator to press a key after finding the next Synchronization 1. No file name is entered. The contents of \$AE and \$AF set the pointer to the end of the area written. This pointer is also transferred to the load start vector as a start vector for the fields, so that when LOADR is called up again, the fields can be read in, \$AE and \$AF are then written into the pointer to the end of the fields.

As the string table gradually extends downwards from the end of the Basic RAM, its length is not influenced by the length of the Basic program. For this reason, this area is always loaded back to the point from which it was transferred to storage. Bit zero is therefore deleted in ABSFLG. The table is then loaded by the routine LOADR.

The pointer to the lower end of the string table is then read from the cassette buffer and stored in 51, 52.

And finally, the status is checked and an error message output if necessary. If no error occurs, the string descriptor is reset and control returns to the main program.

13. DATA PROCESSING WITH FASTTAPE

In this chapter, I should like to show you that efficient data processing is also possible with a cassette recorder, using suitable software.

The data processing program reproduced in this chapter has the following specification:

- data storage and loading with FastTape. Depending on the size of the file, the storage and loading process will last between 20 seconds and 1.5 minutes.
- the number of data records is left to your own discretion. The maximum number will depend on the free memory capacity of the computer.
- A data record can be divided up into any required number of fields.
- A field may contain a maximum of 80 characters.
- The names of the individual fields can be selected at will and can be subsequently changed if required.
- A search can be made in individual fields or in a number of fields. You also have a choice whether one or all of the search criteria should be met.
- The speed of searching is the same in all fields.
- A search can be made on the basis of equality, inequality, or greater or lesser value.

- If a search is made in several fields, then a different search function can be determined for each field.
- A HELP PAGE explaining the individual instructions can be displayed at any time.
- You can draw up lists of the data found and format them for output.
- You can freely determine which fields should be output.
- The individual fields can be output with or without field identification.
- Lists can be sorted by any field.

I have written this program so that it will run on both the VIC 20 and the C-64 without any alterations. However, these general adjustments require a certain amount of memory space and can be either altered or deleted as required, so that more memory is available. You will find a detailed description at the end of this chapter, so that you can adapt the program to meet your own requirements.

But let's now take a look at the operation of the program. It is possible to use this program to set up any sort of file and to process it. You could set up a literary index or a catalog of recipes or phonograph records, or record all the addresses of your friends and acquaintances.

To explain and illustrate the function and operation of this program, it would be best for you to set up a trial file.

As a catalog of recipes or something of that sort would be too lengthy for our purposes, we will set up an address file. I freely admit that this type of file is severely overworked, but it does provide a simple way of writing a file, having a number of data records, with relatively little writing.

After starting the program with RUN, the menu will appear on your screen.

MENU

1. File care and maintenance
2. Page up or down
3. Output list
4. Sort list
5. Enter data
6. Load file
7. Store file
8. Set up file
9. Change field identification
- e. END

As you have not yet set up the file, let us start with point 9 on the menu.

SETTING UP A FILE

With this item on the menu, you must enter the basic characteristics of your file. You are requested to enter the number of fields and the maximum number of data records. These values cannot be changed.

You will next be asked to enter field identification for the individual fields. For our address file, you should enter the maximum number of data records as 10 or more and the number of fields as 7. The individual fields could be best be identified as follow:

1. First Name
2. Last Name
3. Address
4. ZIP code
5. City/State
6. Telephone
7. Hobby

When you have entered these details, go back to the main menu. It is now no longer possible to select this particular item on the menu.

As you have not properly tested the program yet - and it is always possible to make mistakes when entering the details - it would be safest to store the file without data. To do this, you merely select item 7 on the menu, "Store file".

First of all, you will be asked for the file name. Enter "addresses" and press RETURN. You will now be asked to insert a data cassette, so insert a cassette in your

recorder. Make sure when doing so that the tape is not completely rewound to the beginning, to ensure that the file is completely stored. Press RETURN and follow the instruction "PRESS RECORD & PLAY ON TAPE".

After all the variables have been entered, the computer returns to the menu. We now want to enter some data records.

ENTER DATA

When you select this item on the menu, the following display will appear on your screen:

ENTER DATA

There are 0 data records

First Name:

Last Name:

Address:

ZIP code:

City/State:

Telephone:

Hobby:

The cursor will be positioned under the field identification "First Name". You can now write up to 80 characters in each field. So long as you remain within a single field, the complete screen editor facility of the Commodore is at your disposal. Fill in the fields in the following way:

ENTER DATA

There are 0 data records:

First Name:

ABACUS

Last Name:

Software

Address:

P.O. Box 7211

ZIP code:

49510

City/State:

Grand Rapids Mi.

Telephone:

616-241-5510

Hobby:

COMPUTERS

When you have concluded the input in the last field with RETURN, "RETURN" appears on the last line on the screen.

You now have the following input possibilities:

- RETURN - The data record is accepted by the file and you can now enter the next data record

- Q - The data record is not accepted and the program returns to the menu

- A - The cursor appears in the first field again and you can alter the input

Any - The data record is accepted and the program
other key returns to the menu.

If the input is free from errors, then enter RETURN. The heading will now show that there is a data record and the cursor will appear in the first input field again. The fields still contain the last input, which you could overwrite if necessary. If some of the fields do not change, you can input them over by pressing RETURN. So now enter 4 addresses in the manner described above.

PAGE UP OR DOWN

Select item 2 on the menu. In this mode, you can review the data in the sequence in which they were entered and change them if necessary.

By pressing the RETURN key you can page up a data record one at a time. You can page down by using the "-" key. If you press the "A" key, the cursor will appear in the first field and you can change the data record displayed. You can return to the menu by pressing "Q".

PAGE UP OR DOWN

Data record No. 2

First Name:

Peter

Last Name:

Nameless

Address:

123 Easy Street

ZIP Code:

1234

City/State:

Nowhere GA

Telephone:

123456

Hobby:

Computers

FILE CARE AND MAINTENANCE

This part of the program comes under item 1 on the menu. This is where you can select, delete and change data and create lists.

First of all, we shall search for all data records in which the name begins with an "S". To do this, respond to the question as to which fields are to be sought by entering a 2. The corresponding field identification will then appear inversely on the screen. In the second to last line on the screen, the following will appear:

```
1 : = / 2 : <> / 3 : > / 4 : <
```

These are the options that you have.

1. Equal

You are seeking a significant sequence of characters which is exactly the same as your search string. The computer will restrict the search to precisely the number of characters you have entered, i.e. if you enter "Ma", it could come up with "man", "Mama", "marmalade" etc. The same applies by analogy to the following possibilities.

2. Unequal

The computer will find the strings which are not identical to the search string.

3. Greater than or equal to

This time the computer will find all those strings which are greater than or equal to the one you have given. If you have given the letter "M" for the name, then the computer will pick out all names starting with M or any of the following letters (N, O, P, Q...).

4. Smaller than or equal to

If you enter M as the search string, then all names beginning with A - M will be picked out.

If you press "1", you will get the following display:

SEEK / CHANGE

1. First Name
2. Last Name
3. Address
4. ZIP code
5. City/State
6. Telephone
7. Hobby

Which fields are to be sought?

= Last Name

If you wish to search other fields, then all you need to do is to enter the appropriate number.

To start with, however, we will only search for the name. The input is concluded by pressing the RETURN key. You will

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

now be asked whether a data record is to be found in one case or in all cases of matching. As you have only entered one search criterion anyway, you merely press RETURN.

The specified field identification and the cursor now appear. Since we wish to find all names beginning with "S", enter "S" and RETURN. "RETURN" will now appear again on the last line of the screen. If you now press "A", you can change the input. If you press RETURN, the computer will start the search and display the number of matches, the number of the data record displayed and the data record itself.

SEEK / CHANGE

1 matches

Data record No. 0

First Name:

ABACUS

LAST Name:

Software

Address

P.O. Box 7211

ZIP code:

49510

City/State:

Grand Rapids MI

Telephone:

616-241-5510

Hobby:

Computers

<<RETURN>>

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

You can now search through the records found, as described under Page Up and Down.

You can also change them by entering "A" or delete them by entering "D". You can get back to the menu by entering "Q".

You should now familiarize yourself with items 1 and 2 on the menu, so that you can see for yourself just how quick and easy it is to find and correct the required data records. Then try searching with a criterion which applies to several data records.

When you have assembled a list of several data records, you then sort them by any field you wish, e.g. "Last Names", using menu item 4.

Menu item 3 will enable you to output the list on the screen or a printer.

OUTPUT LIST

With this item on the menu, you will be asked which fields are to be printed. By entering the field indices, you can select the required fields and conclude input with return.

However, if you enter "R", the fields displayed will be removed and you can repeat the input. When you have selected the required fields, press RETURN.

As the next step, you will be asked for a heading. This will appear at the top of your list. With printer output, the heading will appear in wide-spaced lettering.

After that, you can choose whether the field identification WILL be printed out as well.

Finally, you will be asked whether output should be displayed on the screen or printed out on a printer.

The program is designed so that each field appears on a separate line. The output format can be changed to meet your individual wishes by altering lines 2870 - 2930 of the program.

If the output is displayed on the screen, the output can be halted and restarted by pressing the "SPACE" key.

I hope that after a little practice, you will have no difficulty in operating this program.


```

1230 gosub1010:a=val(a$):ifa$="e"then1270
1250 onagosub2130,2000,2700,2990,1860,1570,1460,1700,1780
1260 goto1110
1265 rem "terminate program"
1270 if(fland2)=2or(fland4)=0thenend
1280 print"{CLR}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{
DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{
}":printtab(sp/2-8)"{WHT}still terminate":gosub1310
1290 iff1<>255then1110
1300 end
1305 rem error
1310 pokec1,5*c2
1320 print"{HOM}":print"{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN
}"
1330 print:printtab(sp/2-3)"{RED}{RVS}file"
1340 print:printtab(sp/2-5)"{RVS}was not"
1350 print:printtab(sp/2-4)"{RVS}stored"
1360 fori=0to15:geta$:ifa$<>"theni=15:next:goto1440
1370 next
1380 print"{HOM}":print"{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN
}"
1390 print:printtab(sp/2-3)"{VEL}file"
1400 print:printtab(sp/2-5)"was not"
1410 print:printtab(sp/2-4)"stored"
1420 fori=0to15:geta$:ifa$=""theni=15:next:goto1310
1430 next
1440 ifa$="y"thenf1=255
1450 return
1460 rem "store
1470 if(fland1)=0thenreturn
1480 pokec1,7*c2:print"{BLU}":gosub3590
1490 print:print:print" f$"{ UP}":input"filename";f$
1500 print:print"insert data cassette"
1510 print:print:print" << return{SH }>>"
1520 gosub1010
1530 ifa$="r"thenprint"{HOM}":goto1490
1540 ifa$="q"thenreturn
1550 ifa<>13then1520
1560 <-dsf$:f1=f1or2:return
1570 rem "load"
1580 if(fland6)=4thengosub1680:iff1<>255thenreturn
1590 pokec1,7*c2:print"{BLU}":gosub3600
1600 print:print:print" f$"{ UP}":input"Filename";f$
1610 print:print"insert data cassette"
1620 print:print:print" << return{SH }>>"
1630 gosub1010
1640 ifa$="r"thenprint"{HOM}":goto1600
1650 ifa$="q"thenreturn
1660 ifa<>13then1630

```



```

gsub1010
2060 ifa$="-"thenr1=r1-1:goto2025
2070 ifa$="q"thenreturn
2080 ifa$="a"thengosub2560:goto2060
2090 ifa<>13then2050
2100 ifr1<rt-1thenr1=r1+1:goto2030
2110 gosub3610:print"{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{
DN}{RHT}{RVS} no further data":gosub 1010
2120 goto2020
2130 rem "maintain file
2135 if(fland8)=0thenreturn
2140 poke1,4*c2
2150 fori=0tofa-1:sf(i)=0:next:as=0
2160 gosub3620:fori=0tofa-1:printi+1"{LFT} "m$(i):next
2170 print:print"which fields are ";:ifsp<40thenprint:prin
t
2180 print"to be sought":print:print
2190 gosub1010:ifa=13then2262
2200 ifa$="q"thenreturn
2210 ifa$="r"then2150
2220 i=val(a$):ifi<1ori>fathen2190
2230 ifsf(i-1)then2190
2240 d6=peek(214):rem "save cursor line
2250 sf(i-1)=1:printtab(3)"{RVS}"m$(i-1):gosub3710
2260 printleft$(dw$,d6+1)z$(sf(i-1)-1):as=as+1:goto2190
2262 gosub3620:print:print"how many matched fields"
2264 print:printtab(4)"1 = one"
2265 print:printtab(4)"0 = all"
2266 print:input"is found ";ue
2267 ifuethenas=ue
2270 gosub3620:close1:open1,0
2280 print:print"then enter ";:ifsp<40thenprint:print
2290 print"search strings":print:print
2300 forf=0tofa-1:ifsf(f)=0then2330
2310 print"{RVS}"m$(f)" : "
2320 prints$(f)"{ UP}":input#1,s$(f):print
2330 next
2340 printleft$(dw$,ze)"<< RETURN >>":gosub1010:ifa$="r"t
henprint"{HOM}":goto2280
2350 ifa$="q"thenreturn
2360 ifa<>13goto2340
2370 rem "seek
2372 forf=0tofa-1:ifsf(f)thenfb=f:f=fa
2374 next:rem "determine lowest and highest
2376 forf=fa-1to0step-1:ifsf(f)thenfe=f:f=0
2378 next:rem "search field
2380 z=0:for r =0 to rt-1
2390 forf=fbtofe-1:ifsf(f)thenu$(f)=e$(f,r)
2400 next:rem re-store examination strings

```



```

2410 n=0:f=fb:gosub100:ifn>=asthengf%(z)=r:z=z+1
2420 next:ifz=0then2130
2430 rem "display data records
2440 i=0
2450 gosub3620:print:printz" hit":print"data record no."gf
%(i)
2460 r1=gf%(i):gosub3650
2470 printleft$(dw$,ze)"{GRN} << RETURN{SH }>> {WHT}";:g
osub1010
2480 ifa$="-"andi>0theni=i-1:goto2450
2490 ifa$="q"thenreturn
2500 ifa$="a"thengosub2560:goto2480
2510 ifa$="d"then2640
2520 ifa<>13then2470
2530 ifi<z-1theni=i+1:goto2450
2540 gosub3620:print"{ DN}{ DN}{ DN}{ DN}{ DN}{ DN}{
DN}{RHT}{RHT}{RHT}{RVS} no further data"
2550 printleft$(dw$,ze)"{GRN} << RETURN >> {WHT}";:g
osub1010:goto2450
2560 rem "change"
2570 close1:open1,0:r2=r1:gosub3670
2580 printleft$(dw$,ze)"{GRN} << RETURN >> {WHT}";:g
osub1010
2600 ifa$="r"then2570
2610 fl=flor4:fl=fland253:return
2620 return
2630 rem delete
2640 fori1=gf%(i)+1tort-1:rem re-store
2650 forj=0tofa-1:rem data
2660 e$(j,i1-1)=e$(j,i1)
2670 nextj,i1
2680 z=z-1:rt=rt-1:fl=flor4:goto2450
2690 rem output list
2695 ifz=0thenreturn
2700 pokec1,7*c2:print"{BLU}"
2710 fori=0tofa-1:sf(i)=0:next
2720 gosub3630
2725 fori=0tofa-1:printi+1"{LFT}. "m$(i):next
2730 print:print"which fields are to be ";:ifsp<40thenprin
t:print
2740 print"printed out":print:print
2750 gosub1010:ifa=13then2810
2760 ifa$="q"thenreturn
2770 ifa$="r"then2710
2780 i=val(a$):ifi<1ori>fathen2750
2790 ifsf(i-1)then2750
2800 sf(i-1)=1:print"{RVS}"m$(i-1):print:goto2750
2810 print:input"type in heading";ub$
2820 print:input"field description [yes=1/no=0]";fm$

```

```

2830 fm=val(fm$)
2840 print:input"printer=4/screen=0";dv
2850 ifdvthenopen4,dv,7:cmd4
2860 printub$chr$(15):print:print
2870 forr=0toz-1
2880 forf=0tofa-1:ifsf(f)=0then2910
2890 iffmthenprint:print"{RVS}"m$(f)" : {OFF}"
2900 print$(f,gf%(r))
2910 next:ifdvthen2930
2920 geta$:ifa$<>"thenpoke198,0:wait198,1:geta$
2930 print:print:next
2940 ifdvthenprint#4:close4:return
2950 print:print" << RETURN >>"
2960 gosub1010:ifa$<>13then2960
2970 return
2980 rem sort
2990 ifz=0thenreturn
3000 gosub3640
3010 fori=0tofa-1:printi+1"{LFT}. "m$(i):next
3020 print"{ DN}on what field ";:ifsp<40thenprint:print
3030 input"should sort take place";sf:sf=sf-1
3040 f1=0:fh=0:fh=peek(49)+peek(50)*256:rem save upper lim
it
3050 f1=peek(47)+peek(48)*256:rem save lower limit
3060 dimn$(z),z(z+1)
3070 q=1:fori=0toz-1:rem sort field
3080 n$(q)=e$(sf,gf%(i)):rem re-store
3090 z(q)=q:q=q+1:next
3100 gosub3240:rem sort routine
3110 forx=0tofa-1
3130 fory=0toz-1
3140 n$(y+1)=e$(x,gf%(y))
3160 next
3170 fory=0toz-1
3180 e$(x,gf%(y))=n$(z(y+1))
3200 nexty,x
3210 fori=0toz-1:n$(i)="" :next:rem clear out the array
3220 gosub3520:rem restore old pointers
3230 return
3235 rem sort routine
3240 ti$="000000":print"{CLR}"
3250 l=int(z/2)+1
3260 r=z:print"{CLR}"r"{LFT} "
3270 ifl>1then3340
3280 ifr<=1then3330
3290 h2$=n$(1):h1=z(1)
3300 n$(1)=n$(r):z(1)=z(r)
3310 n$(r)=h2$:z(r)=h1
3320 r=r-1

```

```

3330 goto3350
3340 l=1-1
3350 j=1
3360 i=2*j
3370 h2$=n$(j):h1=z(j)
3380 ifi>rthen3480
3390 ifi>=rthen3420
3400 ifn$(i)>=n$(i+1)then3420
3410 i=i+1:print"{CLR}"i{LFT}    "
3420 ifi>rthen3480
3430 ifh2$>=n$(i)then3480
3440 n$(j)=n$(i):z(j)=z(i)
3450 j=i
3460 i=2*j
3470 goto3390
3480 n$(j)=h2$:z(j)=h1
3490 ifr<>1then3270
3500 print"{CLR}{ DN}{ DN}sort time";ti$
3510 return
3515 rem restore pointer
3520 j=peek(47)+peek(48)*256+(fh-f1)
3530 poke49,jand255:poke50,int(j/256)
3540 return
3545 rem headings
3550 print"{CLR}";:printtab(sp/2-7)"* Set Up File *":retur
n
3560 print"{CLR}";:printtab(sp/2-10){WHT}* * * M E N U *(
SH }* *":return
3570 print"{CLR}";:printtab(sp/2-9)"* Maintain File *":ret
urn
3580 print"{CLR}{BLU}";:printtab(sp/2-10)"* Enter Data *":
return
3590 print"{CLR}";:printtab(sp/2-8)"* Store Data *":return
3600 print"{CLR}";:printtab(sp/2-6)"* Load Data *":return
3610 print"{CLR}";:printtab(sp/2-8)"* * Page{SH }* *":retu
rn
3620 print"{CLR}";:printtab(sp/2-8)"Seek/Change":return
3630 print"{CLR}";:printtab(sp/2-7)"Output List":return
3640 print"{CLR}";:printtab(sp/2-7)"sort list":return
3645 rem headings
3650 print"{HOM}{ DN}{ DN}{ DN}{ DN}{ DN}":forf=0tofa-1:pr
int"{RVS}"m$(f)": "
3660 printe$(f,r1):next:return
3670 print"{HOM}{ DN}{ DN}{ DN}{ DN}{ DN}":forf=0tofa-1:pr
int"{RVS}"m$(f)": "
3680 printe$(f,r1){ UP}":a=len(e$(f,r1))
3690 ifa>spthenforx=0toa/sp:print"{ UP}";:next
3700 input#1,e$(f,r2):print:next:return
3705 rem select search from

```

```

3710 printleft$(dw$,ze-1)"(RVS)1:=/2:< >/3:>/4:<(OFF)";
3720 gosub1010:j=val(a$):ifj<1orj>4then3720
3730 sf(i-1)=j
3740 printleft$(dw$,ze-1)"                                     ";;return
3750 rem help page
3760 print"<CLR>** ** ** ** ** H      E      L      P
** ** ** ** ** ** ** ** "
3770 print:print"If << RETURN >> appears, you have the"
3780 print:print"following options:":print:print
3790 print"RETURN = Continue":print
3800 print"R      = Repeat last entry":print
3810 print"Q      = Break out of sub-menu":print
3820 print"-      = Go back a record":print
3830 print"A      = Alter a record":print
3840 print"D      = Delete a record":print
3850 geta$:ifa$=""then3850
3860 return

5000 "*****
5010 "*          FASTTAPE DATA BASE          *
5020 "*          (C) DIRK PAULISSEN          *
5030 "*          *
5040 "* VARIABLES USED:                      *
5050 "* FA      = NUMBER OF FIELDS           *
5060 "* RA      = NUMBER OF RECORDS (MAX)*
5070 "* RT      = ' ' ' ' (ACTUAL)*
5080 "* F       = ACTUAL FIELD               *
5090 "* R       = ACTUAL RECORD              *
5100 "* SP      = NUMBER OF COLUMNS         *
5110 "* ZE      = NUMBER OF LINES            *
5120 "* A$      = INPUT STRING               *
5130 "* FL      = FLAG                       *
5140 "* E$(F,R) = DATQ STRING                *
5150 "* SF(F)   = SEARCH FIELD FLAG         *
5160 "*          *
5170 "*          1 : ENTIRE '='              *
5180 "*          2 : ENTIRE '<>'             *
5190 "*          3 : ENTIRE '>='           *
5200 "*          4 : ENTIRE '<='           *
5220 "* S$(F)   = SEARCH STRING              *
5230 "* U$(F)   = EXAMINATION STRING         *
5240 "* GF%( )  = NUMBER OF DATA           *
5250 "*          RECORDS FOUND              *
5300 "*****

```

READY.

13.1 PROGRAM DESCRIPTION

The program structure is as follows:

90	-	1030	Fast subroutines
1050	-	1090	Initialization
1100	-	1260	Main programs (menu)
1270	-	3540	Subroutines called by the menu
3550	-	3640	Headings
3650	-	3850	Subroutines which are called by other subroutines
5000	-	End	Copyright and list of variables used

Mode of Operation

Adaptation to the various computers takes place in line 1070 by calling up the operating system routine SCREEN (65517). This routine stores the number of columns and lines in storage locations 781 and 782. The storage locations and the values will be used for Color POKE instructions (C1, C2), corresponding to these values.

The main program consists solely of the menu and the branches to the selected subroutine. For data security purposes, a flag f1 was introduced; it is always tested at the beginning of a subroutine and set during the subroutine. The individual bits of f1 have the following meanings when they are set:

Bit	Value	Meaning
0	1	File set up
1	2	File stored
2	4	File changed
3	8	File contains data records

If an attempt is made to load a new file or to terminate the program before a changed file has been stored, the program branches to the error routine 1305 - 1450.

The "Load" and "Store" sections of the program require no further explanation.

Set Up File (1700 - 1850)

When the title has been written by line 1720, the number of fields (fa) and the number of data records (ra) are read in via INPUT. The query in 1750 enables any input error to be corrected.

The fields are dimensioned according to fa and ra in line 1760, and bit zero in f1 is set in line 1770.

Bit zero in f1 is tested in line 1780, since a branch to the routine for entering and changing the field descriptions can also be taken from the menu.

In lines 1800 and 1810 the string to be output is printed out in one or two lines, depending on the number of columns available on the computer.

The individual field descriptions are read into m\$ (fa) by means of a loop (lines 1820 - 1830). Any existing field

descriptions are displayed on the screen and can be input by RETURN.

The query in line 1840 again provides a possibility to make corrections.

Enter Data (1860 - 1990)

In line 1865, f1 is tested to determine whether a file has already been set up.

In line 1880, the keyboard is opened by OPEN 1,0. The result of this is that a following INPUT#1 is not accompanied by a question mark on the screen. For security reasons, the relevant file is always closed before every OPEN instructions.

The heading and, depending on the number of columns, a string are output by lines 1890 - 1990. In the subroutines which display the individual fields of a record (3650 - 3660) or which read in new fields (3670 - 3700), r1 or r2 is used as a record pointer. For this reason, the pointer is transferred in line 1910 to the record to be entered in rt, r1 or r2. If a record has already been entered (rt greater than zero), then r1 is decremented so that the fields of the last record entered are displayed. From line 1930, the computer queries whether it should terminate (a\$ = "q"), correct (a\$ = "a"), finish (a not equal 13) or continue. If a = 13, then line 1970 tests whether the maximum number of records has been reached.

Paging (2000)

With the help of the controlled variable r1, the individual data records are displayed by the subroutine "Output Fields" and the keyboard is checked in line (1010). Depending on the input, the computer will go on paging (2100), return to the menu 2070 or branch to the subroutine "Change" (2080).

Maintain File (2130)

After f1 and screen POKEs have been checked, the search flags (sf (i)) are set to zero (2150). Starting at line 2190, the computer checks via the subroutine at 1010 which field it should commence its search. If a valid and so far unselected field number is entered (the test for this is in lines 2220 - 2230), the actual cursor line is saved in d6 by line 2240 and the field description of the required field is printed out by line 2250.

With the aid of the cursor line saved in d6, the corresponding search form character is set in front of the field description in line 2260 and control branches back to check for further search fields.

In lines 2262 - 2267 the required number of matches is written into ue. The search strings are read into field s\$(f) via lines 2280 - 2360.

As the actual search routine examines the individual fields of a data record from a starting field to an end field, the lowest and highest field indices are established in lines 2372 - 2378 and saved in fb and fe, to optimize search time. The main search loop for the individual data records begins with line 2380. In line 2390 the strings to be examined are

transferred to field U\$(I). In line 2410 the index N for matches found is set to zero and f is set to the start field before the program branches to the actual search routine, starting at line 100.

The search routine is situated at the beginning of the program for reasons of time, as the target address will be sought after branching (GOTO,GOSUB) either after the calling line or from the beginning of the program.

If the search routine were at the required line number, it would have a very noticeable effect on search times with long files.

At the beginning of the search routine, a test is made to determine whether all required fields have been examined, or whether the required number of matches has been found. If this is not the case, then the program branches to the required comparison via field SF(F).

If the corresponding comparison is logically true, then the match counter is incremented. When the field index f has been incremented, control branches back to the start. After the return to line 2410, the number of data records is written into field GF%(Z) and the number of hits(z) incremented if the required number of matches has been reached.

On completion of the search, the hits are displayed, analogously to the subroutine "Paging" (2430 - 2550).

OUTPUT LIST (2690 - 2970)

This routine outputs the data records determined by field GF%(I) either on the screen or on the printer. The checking for the fields to be printed out is analogous to the determination of fields in the subroutine "Search".

Sorting (2980 - 3510)

After the inquiry as to whether a list was made up, and also after the output of the heading on the screen (2990 - 3000), the field on which the sort is to be is checked and the corresponding index written into sf.

A number of auxiliary fields are required for the following sort routine, their size depending on the size of the list. As fields may normally be dimensioned only once, I would have had to dimension the auxiliary fields N\$(I) and Z(I) to the maximum number of data records at the time of initialization. That would use up a great deal of memory and would also increase the time required for loading and storing on or from tape. However, there is a little trick which makes it possible to delete fields selectively and thus to make them redimensionable.

This trick consists in saving the upper and lower limits of the area in which the computer has stored the fields before a new dimensioning takes place. The fields dimensioned after that can be deleted by resetting the pointer to the old field table length.

For this reason, the pointers to the start and the end of the field table are saved in F1 and FH in lines 3040 and 3050.

In line 3060 the auxiliary fields are dimensioned according to the size of the list.

The corresponding data record field for each data record in the list is transferred to N\$(I) by means of the loop in lines 3070 to 3090.

This string field is sorted later. In order to make it possible for the corresponding other fields of a data record to be assigned to the sorted field later, a pointer is set up in Z(I) containing the position of the corresponding data record in the list.

The sort routine processes only fields with indices from one onwards. However, in the data field E\$(F,R) the data field are stored with indices from zero onwards. For this reason, the data and the pointer must be re-stored in a field variable with an index one higher than that in the data field.

The increased index is needed three time during the re-storing routine (index for n\$, index for z, pointer in z). In order to avoid having to compute "I+1" three times in the re-storing loop, I have defined an auxiliary variable q which only needs to be incremented once during the loop, thus saving time.

In line 3100, a branch is taken to the sort routine (3240-3510). This routine sorts the unidimensional field n\$(i) by a corresponding exchange of the contents of n\$(i). With each exchange, the contents of the pointer are also exchanged. This will become clearer if we take an example. Let us assume that we have four names stored in the following sequence, which have to be sorted:

```
n$ (1) = John      z (1) = 1
n$ (2) = Alfred   z (2) = 2
n$ (3) = Peter    z (3) = 3
n$ (4) = Richard  z (4) = 4
```

After sorting, we have:

```
n$ (1) = Alfred   z (1) = 2
n$ (2) = Richard  z (2) = 4
n$ (3) = John     z (3) = 1
n$ (4) = Peter    z (4) = 3
```

The index of z indicates the ascending sequence of the sorted string, while the content of each z is the number of the data record in the corresponding position.

In the loop formed by lines 3110 - 3200, the data fields are exchanged according to the pointer.

The outer loop counts the field. In the first of the inner loops, the corresponding field (denoted by x) in all data record is transferred in the old sequence into field n\$(I).

In the second of the inner loops, the contents of n\$(I) are written back into the data field in accordance with the pointer Z(I), having been duly sorted.

In line 3210 onwards, the auxiliary fields are clear again. First of all, the strings are deleted in order to reset the string vector in the computer's 'notebook'. By means of the subroutine in lines 3520 - 3540, the saved length of the field table is added to the actual start of the table and written into the vector 49, 50 as the end of the table.

14. CASSETTE CONTENTS AND CATALOG OF FASTTAPE CASSETTES

In Chapter 7, I described a program which will set up a catalog of the contents of all your cassettes. I have no doubt that you would also like to set up a similar catalogue for your FastTape cassettes, so here is the listing of a program which operates in exactly the same way as that in Chapter 7.

```

90 rem"(C) DIRK(SH )PAULSEN
100 rem fasttape catalog
108 rem v=.7:k1=.54 <><><>constants for counter calculatio
ns c60
109 v=.5:k1=.482:rem constants for counter calculations c9
0
110 pa=828: rem start address in cassette buffer
115 sa=171 :rem secondary address + 1
116 print"{CLR}          catalog of contents"
117 print:print"          for fasttape cassettes"
120 print"{ DN}{ DN}print ? (y/n) ";d$
130 get d$: if d$="" then 130
136 ta$=chr$(16)
140 t1$=ta$+"05":t2$=ta$+"10":t3$=ta$+"19":t4$=ta$+"27"
142 t5$=ta$+"35":t6$=ta$+"43":t7$=ta$+"61":rem tab stops
145 ifd$<>"y"then 170
150 open2,4:rem open printer
160 print#2,t1$"1fn"t2$"counter"t3$"k-byte"t4$"start"t5$"
end";
162 print#2,t6$"      name"t7$"2ndaddr"
165 print#2
170 if d$<>"y" then 230
180 open2,4

```

```

190 sys49674:rem vic20 ==> 29183
200 a$=" :"+right$("      "+str$(peek(pa)+(256*peek(pa+1))-
1),6)
210 b$=" -"+right$("      "+str$(peek(pa+2)+(256*peek(pa+3))
-1),6)
220 c$=" :":for i=5 to 20:c$=c$+chr$(peek(pa+i)):next:c$=c
$+": "
230 t=val(right$(b$,5))-val(right$(a$,5)):k$=str$(int(t/10
24*1000+.5)/1000)
235 k$=right$("      "+k$,6)
240 sa$=str$(peek(sa)-1)
250 print z;ty$;k$;a$;b$;print c$;print
252 fa=1+2*(6666/4400):rem factor for counter calculations
255 z1=z:z=int(z+v*fa+(t/1024)*(k1*fa)):rem calculate tape
counter
256 print"next counter reading"z
260 n=n+1:n$=right$("      "+str$(n),3)
270 if d$="y"thenprint#2,t1$;n$;t2$;z1;t3$;k$;t4$;a$;t5$;b$
;t6$;c$;t7$;sa$
280 t=val(right$(b$,6))-val(right$(a$,6))
290 k$=right$("      "+str$(int(t/1024*100)/100),6)
300 goto190
310 n=n+1:n$=right$("      "+str$(n),3)
315 if d$<>"y"then 330

320 print#2,t1$;n$;t2$;z;t3$;ty$;t4$;k$;t5$;a$;t6$;b$;t7$;
c$
330 z=int(z+(t/212)+4+2*(n/100)):rem compute tape counter
340 close1
350 goto 230

```

ready.

The cataloging program described in Chapter 10.1 will also operate with FastTape if suitably modified. It is only necessary to exchange the normal LOAD instructions for FastTape instructions. As the synchronization is considerably shorter in FastTape programs than in programs stored in the normal way, it is advisable not to store the programs immediately following one another, but to leave a 5 second space between each program.

15. DISK BACK-UP FOR TAPE AND VICE VERSA

This chapter is intended for those who have a disk drive as well as a data recorder. A data cassette is considerably more robust and less likely to develop faults than a diskette. It is therefore worth your while, even if you have a diskette drive, to store important programs on cassette. There are also plenty of people who have only a cassette recorder themselves, but who have friends who have diskette stations. Using the following programs, it will now be much easier to exchange programs between yourselves.

15.1 Tape Back-up for Disk

With this back-up program, you will be able to re-store a number of FastTape programs, currently stored one after another on cassette, on a diskette.

It is of no consequence whether the programs are in Basic or machine code. All programs will be copied in such a way that they can be run without any problems, irrespective of the storage medium from which they are loaded. It is merely important that all the programs you wish to copy should have a name, as the disk drive, in contrast to the data recorder, will not accept nameless files.

As the back-up program operates with FastTape, you must load the FastTape program before you start the back-up program.

Operation couldn't be easier!

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

Enter the program in your computer and save it before starting, as it will delete itself automatically after start-up. When you start the program, the machine language program is first written into memory behind the FastTape program, starting at \$C3C0 (\$73C0). You will then be asked for the number of programs to be copied. This value is held in intermediate storage at location 767 for the machine language program.

The next thing to do is to insert a formatted diskette in the disk drive and to insert the cassette with the programs to be copied in the data recorder. When the RETURN key is pressed, the Basic program branches to the machine language program. This now reads the first program into the Basic RAM (the complete Basic RAM is used as a buffer). After that, the program is written from memory to diskette. This process is repeated as many times as there are programs. On completion, the computer reverts to direct mode.

If you now wish to copy any more programs, you will have to store the number of programs in question in memory location 767 with a POKE and start the machine program again by means of SYS 50112 (SYS 28672 in the VIC 20).

BACKUP T=>D ASSEMBLER 64 V2.0 PAGE 1

```

85: C3C0 .PAG 53,0
86: C3C0 .TIT "BACKUP T=>D"
90: C3C0 .OPT P130
100: C3C0 *= $C3C0 ;VC20 = $73C0
110: C3C0
110: C3C0
115: ; *****
120: ; * *
130: ; * BACKUP FOR CASSETTE TO DISK *
135: ; * *
140: ; * WITH FASTTAPE *
145: ; * *
146: ; *****
150: C3C0
150: C3C0
150: C3C0
150: C3C0
155: ; EXPRESSIONS IN BRACKETS APPLY
157: ; TO THE VIC 20
160: C3C0
160: C3C0
200: 000D CR = 13 ;
205: 0002 ABSFLG = 2
210: 00AC STARTV = $AC
220: 00AE ENDVEC = $AE
230: 033C CASPUF = $33C
240: 02FF ANZAHL = $2FF
245: 0803 CSTART = $803 ; ($1203) START
; OF BUFFER
250: C18E LOADR = $C18E ; ($7188) LOAD
; FASTTAPE
260: A642 NEW = $A642 ; ($C642) BASIC-
; INST. 'NEW'
270: A437 ERROR = $A437 ; ($C437) BASIC-
; WARM START
280: F693 MSSG = $F693 ; ($F72C) OUTPUT
; MESSAGE
290: C3C0
290: C3C0
300: FFB7 STATUS = $FFB7
310: FFBA SETLFS = $FFBA
320: FFBD SETNAM = $FFBD
325: FFC0 OPEN = $FFC0
330: FFC3 CLOSE = $FFC3
335: FFC9 CHKOUT = $FFC9
340: FFCC CLRCH = $FFCC
350: FFD2 BSOUT = $FFD2
955: C3C0
960: ; MAIN PROGRAM
970: C3C0

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

970:   C3C0
1000: C3C0 A9 00      START   LDA   #$00
1010: C3C2 20 42 A6      JSR   NEW
1020: C3C5 20 70 C4      JSR   CLOAD
1030: C3C8 20 D4 C3      JSR   DSAVE
1040: C3CB CE FF 02      DEC   ANZAHL
1050: C3CE F0 03      BEQ   ENDE
1060: C3D0 4C C0 C3      JMP   START
1070: C3D3 60          ENDE   RTS
1082: C3D4
1082: C3D4
1084: C3D4
1084: C3D4
1086:                ; STORE ON DISK
1088: C3D4
1088: C3D4
1089: C3D4 A9 10      DSAVE   LDA   #$10
1090: C3D6 A2 41      LDX   #$41      ; SET NAME
1092: C3D8 A0 03      LDY   #$03      ; PARAMETERS
1093: C3DA 20 BD FF      JSR   SETNAM
1095: C3DD 20 93 F6      JSR   MSSG      ; ' SAVING NAME '
1100: C3E0 A9 0D      LDA   #CR
1110: C3E2 20 D2 FF      JSR   BSOUT     ; NEW LINE
1120: C3E5 A2 10      LDX   #$10
1130: C3E7 CA          ELOOP   DEX
1140: C3E8 BD 41 03      LDA   CASPUF+5,X
1150: C3EB C9 20      CMP   #$20
1160: C3ED D0 04      BNE   ENDNAM
1170: C3EF E0 00      CPX   #$00
1180: C3F1 D0 F4      BNE   ELOOP
1190: C3F3 E8          ENDNAM  INX      ; SEEK END OF FILE
1200: C3F4 A9 2C      LDA   #", "    ; NAME AND APPEND
1210: C3F6 9D 41 03      STA   CASPUF+5,X
1220: C3F9 E8          INX      ; ',P,W'
1230: C3FA A9 50      LDA   #"P"
1240: C3FC 9D 41 03      STA   CASPUF+5,X
1250: C3FF E8          INX
1260: C400 A9 2C      LDA   #", "
1270: C402 9D 41 03      STA   CASPUF+5,X
1280: C405 E8          INX
1290: C406 A9 57      LDA   #"W"
1300: C408 9D 41 03      STA   CASPUF+5,X
1310: C40B E8          INX
1320: C40C 8A          TXA      ; LENGTH OF FILE
1330: C40D A2 41      LDX   #$41    ; NAME START ADDR
1340: C40F A0 03      LDY   #$03    ; OF FILE NAMES
1350: C411 20 BD FF      JSR   SETNAM
1360: C414 A9 08      LDA   #$08    ; FILE #
1370: C416 A2 08      LDX   #$08    ; DEVICE #
1380: C418 A0 01      LDY   #$01    ; SECONDARY ADDR.
1390: C41A 20 BA FF      JSR   SETLFS
1400: C41D 20 C0 FF      JSR   OPEN

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

1410: C420 A2 08          LDX  #$08
1420: C422 20 C9 FF          JSR  CHKOUT
1430: C425 A9 03          LDA  #<CSTART
1440: C427 A2 08          LDX  #>CSTART
1450: C429 85 AC          STA  STARTV
1460: C42B 86 AD          STX  STARTV+1
1470: C42D AD 3E 03        LDA  CASPUF+2 ; COMPUTE END
1480: C430 38          SEC          ; ADDRESS FROM
1490: C431 ED 3C 03        SBC  CASPUF ; DIFFERE. BETWEEN
1500: C434 08          PHP          ; START ANDEEND ADDR
1510: C435 18          CLC          ; FROM HEADER PLUS
1520: C436 65 AC          ADC  STARTV ; CERTAIN START
1530: C438 85 AE          STA  ENDVEC ; ADDRSTART ADDR
1540: C43A AD 3F 03        LDA  CASPUF+3
1550: C43D 65 AD          ADC  STARTV+1
1560: C43F 28          PLP
1570: C440 ED 3D 03        SBC  CASPUF+1
1580: C443 85 AF          STA  ENDVEC+1
1590: C445 AD 3C 03        LDA  CASPUF ; TRANSMIT PROGRAM
                                START ADDR. TO DISK
1600: C448 20 D2 FF          JSR  BSOUT ; SEND TO DISK
1610: C44B AD 3D 03        LDA  CASPUF+1
1620: C44E 20 D2 FF          JSR  BSOUT
1622: C451
1622: C451
1623:
                                ; STORAGE LOOP
1624: C451
1624: C451
1630: C451 A0 00          PSAVE  LDY  #$00
1640: C453 B1 AC          LDA  (STARTV),Y
1650: C455 20 D2 FF          JSR  BSOUT
1660: C458 E6 AC          INC  STARTV ; INCREMENT ADDR.
1670: C45A D0 02          BNE  NOTHI
1680: C45C E6 AD          INC  STARTV+1
1690: C45E A5 AC          NOTHI LDA  STARTV
1700: C460 C5 AE          CMP  ENDVEC ; END ADDR. REACHED
1710: C462 A5 AD          LDA  STARTV+1
1720: C464 E5 AF          SBC  ENDVEC+1
1730: C466 90 E9          BCC  PSAVE
1740: C468 20 CC FF          JSR  CLRCH
1750: C46B A9 08          LDA  #$08
1760: C46D 4C C3 FF          JMP  CLOSE
1762: C470
1762: C470
1764: C470
1764: C470
1766:
                                ; LOAD FROM CC
1768: C470
1768: C470
1770: C470 A2 00          CLOAD  LDX  #$00
1780: C472 A0 03          LDY  #<CSTART
1790: C474 A9 08          LDA  #>CSTART

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

1800: C476 86 0A          STX  $0A      ; LOAD/VERIFY FLAG
1810: C478 86 93          STX  $93
1820: C47A 84 AC          STY  $AC
1830: C47C 85 AD          STA  $AD
1840: C47E A9 05          LDA  #1+4     ; TO ADDR.
1850: C480 85 02          STA  ABSFLG  ; DO NOT WAIT
1860: C482 A9 00          LDA  #$00    ; NO FILE NAME
1870: C484 20 BD FF      JSR  SETNAM
1880: C487 A2 01          LDX  #$01    ; GA
1890: C489 A0 00          LDY  #$00    ; SA
1900: C48B 20 BA FF      JSR  SETLFS
1910: C48E 20 8E C1      JSR  LOADR
1922: C491
1924:                      ; TEST STATUS
1926: C491
1926: C491
1930: C491 20 B7 FF HOLSTA JST STATUS
1940: C494 BF            AND  #$BF
1950: C496 F0 05          BEQ  OK
1960: C498 A2 1D          LDX  #$1D
1970: C49A 4C 37 A4      JMP  ERROR
1980: C49D 60            RTS
UC3C0-C49E

```

```

100 GOSUB240
110
PRINT"[CLR]*****
*****[2 DOWN]
120 PRINT"* B A C K U P   W I T H   F A S T
T A P E ***"
130 PRINT"*           FROM CASSETTE TO
DISKETTE   ***"
140 PRINT"*           (C) DIRK PAULISSEN
**"
150
PRINT"*****"
160 PRINT:PRINT:PRINT"HOW MANY PROGRAMS
DO YOU WANT"
170 PRINT:INPUT"TO COPY";AZ
180 POKE767,AZ
190 PRINT:PRINT:PRINT"INSERT SOURCE
CASSETTE"
200 PRINT"AND PRESS << RETURN >>"
210 GETA$:IFA$<>CHR$(13)THEN210
220 SYS50112
230 END
240 REM LOAD PROGRAM FASTCOPY T-D 64
250 E=50333:A=50112:PS=0
260 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
270 IFPS<>27517THENPRINT"ERROR IN
DATA":END
280 RETURN
290 DATA169,0,32,66,166,32,112,196,32,21
2,195,206,255,2,240,3,76,192,195,96
300 DATA169,16,162,65,160,3,32,189,255,3
2,147,246,169,13,32,210,255,162,16
310 DATA202,189,65,3,201,32,208,4,224,0,
208,244,232,169,44,157,65,3,232,169
320 DATA80,157,65,3,232,169,44,157,65,3,
232,169,87,157,65,3,232,138,162,65
330 DATA160,3,32,189,255,169,8,162,8,160
,1,32,186,255,32,192,255,162,8,32,201
340 DATA255,169,3,162,8,133,172,134,173,
173,62,3,56,237,60,3,8,24,101,172,133
350 DATA174,173,63,3,101,173,40,237,61,3
,133,175,173,60,3,32,210,255,173,61
360 DATA3,32,210,255,160,0,177,172,32,21
0,255,230,172,208,2,230,173,165,172
370 DATA197,174,165,173,229,175,144,233,
32,204,255,169,8,76,195,255,162,0,160
380 DATA3,169,8,134,10,134,147,132,172,1
33,173,169,5,133,2,169,0,32,189,255
390 DATA162,1,160,0,32,186,255,32,142,19
3,32,183,255,41,191,240,5,162,29,76
400 DATA55,164,96
READY.

```

```

100 GOSUB240
110 PRINT"[CLR]*****"
120 PRINT"*BACKUP WITH FASTTAPE*"
130 PRINT"* FROM CASS. TO DISK *"
140 PRINT"* (C) DIRK PAULISSEN  *"
150 PRINT"*****"
160 PRINT:PRINT:PRINT"HOW MANY PROGRAMS
DO YOU WANT"
170 PRINT:INPUT"TO COPY";AZ
180 POKE767,AZ
190 PRINT:PRINT:PRINT"INSERT SOURCE
CASSETTE AND"
200 PRINT"PRESS << RETURN >>"
210 GETA$:IFA$(<>CHR$(13))THEN210
220 SYS29632
230 END
240 REM LOAD PROGRAM FASTCOPY T-D 20
250 E=29853:A=29632:PS=0
260 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
280 IFPS<>27173THENPRINT"ERROR IN
DATA":END
290 RETURN
300 DATA169,0,32,66,198,32,112,116,32,21
2,115,206,255,2,240,3,76,192,115,96
310 DATA169,16,162,65,160,3,32,189,255,3
2,44,247,169,13,32,210,255,162,16,202
320 DATA189,65,3,201,32,208,4,224,0,208,
244,232,169,44,157,65,3,232,169,80
330 DATA157,65,3,232,169,44,157,65,3,232
,169,87,157,65,3,232,138,162,65,160
340 DATA3,32,189,255,169,8,162,8,160,1,3
2,186,255,32,192,255,162,8,32,201,255
350 DATA169,3,162,18,133,172,134,173,173
,62,3,56,237,60,3,8,24,101,172,133
360 DATA174,173,63,3,101,173,40,237,61,3
,133,175,173,60,3,32,210,255,173,61
370 DATA3,32,210,255,160,0,177,172,32,21
0,255,230,172,208,2,230,173,165,172
380 DATA197,174,165,173,229,175,144,233,
32,204,255,169,8,76,195,255,162,0,160
390 DATA3,169,18,134,10,134,147,132,172,
133,173,169,5,133,2,169,0,32,189,255
400 DATA162,1,160,0,32,186,255,32,136,11
3,32,183,255,41,191,240,5,162,29,76
410 DATA55,196,96
READY.

```


15.2 PROGRAM DESCRIPTION OF DISK BACKUP FOR CASSETTE

For those of you who have at least a basic knowledge of ASSEMBLER programming, I should like to give a more detailed description of the ASSEMBLER listing.

The main program loop is formed from \$C30C0 to \$C3D3. First of all, the Basic vectors are all reset by the NEW instruction. Control then branches to the cassette load routine CLOAD; after execution of this subroutine, the program is stored on disk through DSAVE. After NUMBER has been decremented and tested, control returns to the start.

CLOAD (\$C469)

The start of the buffer CSTART is transferred to storage locations \$AC, \$AD. FastTape fetches the required loading address from these memory locations. In addition, the LOAD/VERIFY flags are set to 0 = LOAD. The setting of bits zero and two of ABSFLG causes the FastTape program to load into the address transferred to \$AC and \$AD and to continue after the program has been found.

After that, control branches to the FastTape subroutine LOADR. After loading, STATUS is tested and control reverts to the main loop.

DSAVE

Storage on diskette is more complicated. First of all, "SAVING name" is output via operating system subroutines. In \$C3DE to \$C3EA, the end of the file name, which is held in the cassette buffer, is determined by testing each

character for "SPACE", starting from the end, until another character is found.

In the following lines up to \$C401, ",P,W" (Program, Write) is appended to the file name. The length of the complete file name is determined via the X register.

The preparatory routines SETNAM, SETLFS, OPEN, CHKOUT, required for disk storage, start at \$C465. After that, the start vector is reset to the buffer start. The end address of the loaded program is computed from the difference between the start and the end addresses of the cassette buffer plus the start address of the buffer (\$C426 - \$C43C). With storage on diskette the first two bytes give the address where the program was transferred to disk. This address is held in the cassette buffer as the start address. It is transferred to the diskette in \$C43E - \$C447.

Actual program storage commences with PSAVE (\$C44A). 0 is loaded into the Y register and the individual bytes of the program are loaded, indirectly indexed, into the accumulator and transferred to the disk drive via BASOUT. The start vector is then incremented and compared with the end vector. If they are not equal, control branches to PSAVE again.

If the vectors are equal, the disk output channel is closed and control branches back to the main loop.

15.3 CASSETTE BACKUP FOR DISK

This program will enable you to transfer one or more programs from a diskette to one or more cassettes with the greatest of ease. Input is by means of the Basic loader which immediately follows the program description. With this Basic loader, don't forget to save it before starting.

This program will only run if you have previously loaded FastTape.

Operating the Program

The program is started by SYS 50176 (SYS 29696 in the case of the VIC 20). The title will immediately appear on the screen and you will be requested to insert the source diskette and to depress a key.

After a short time, the diskette title will appear and the first file entry in the list of the diskette's contents, with the query YES/NO. If you wish to copy the program, press "Y", whereupon YES appears after the file entry. The next entry is then displayed. If you do not wish to copy a program, then press "N", whereupon "NO" will appear after the entry. In this way, you can say "YES" to 48 programs.

If you should press the wrong key during the program, you can terminate the program by pressing RUN/STOP which returns to the title page.

When you have processed the entire list of contents, you will be asked whether you wish to copy the programs individually or continuously. If you wish to store all the

programs on one diskette, then you must enter "1". If you enter a "2", then the computer will always tell you which program it will read in next and will wait for a key to be pressed. In this way it is possible to insert a different cassette for each program.

When copying is completed, the computer returns to the title page. You then have the possibility of copying a new disk or of returning to Basic by pressing "E".

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

ASSEMBLER 64 V2.0 PAGE 1

```

55:    C400                                .PAGE53,0
60:    C400                                .OPT P130
95:                                     ;*****
96:                                     ;*
100:                                     ;*
110:                                     ;*
115:                                     ;*
117:                                     ;*
120:                                     ;*
122:                                     ;*
125:                                     ;*****
130:                                     ;
132:                                     ; START WITH SYS 50176 (SYS 29696)
134:    C400
134:                                     ; VIC-20 COMMENTS IN ( )'S
136:    C400
136:                                     ;
140:                                     ;
142:    C400                                *=    $C400    ; ($7400)
145:    C400
145:    C400                                START    =    *
150:    E544                                CLSCRN   =    $E544    ;($E55F) CLEAR
                                           SCREEN
152:    BDCD                                NUMOUT  =    $BDCD    ;($DDCD) OUTPUT
                                           POSITIVE INTEGER
154:    FCCA                                MOTAUS  =    $FCCA    ;($FD08) SWITCH
                                           OFF MOTOR
156:    F838                                PTASTE  =    $F838    ;($F8B7) CHECK
                                           RECORDER SWITCH
158:    F30F                                SFINUM  =    $F30F    ;($F3CF) SEEK
                                           FILE NUMBER
160:    AB1E                                STROUT  =    $AB1E    ;($CB1E) OUTPUT
                                           STRING
162:    F31F                                SETPAR  =    $F31F    ;($F3DF) SET FILE
                                           PARAMETERS
164:    C085                                ABSOLUT =    $C085    ;($707F) JUMP TO
169:    C400                                     ; FASTTAPE
169:    C400
170:    FFE4                                GETBYT  =    $FFE4    ;INPUT BYTE
180:    FFBA                                SETLFS  =    $FFBA    ;SET FILE UP
190:    FFBD                                SETNAM  =    $FFBD    ;SET NAME
200:    FFC0                                OPEN    =    $FFC0
210:    FFC3                                CLOSE   =    $FFC3
220:    FFB4                                TALK    =    $FFB4
230:    FF96                                SETALK  =    $FF96    ;SET SECONDARY
                                           ADDR. TO TALK
240:    FFA5                                IECIN   =    $FFA5    ;FETCH BYTE
250:    FFE7                                CLALL   =    $FFE7
260:    FFAB                                UNTALK  =    $FFAB    ;UNTALK

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

280: FFE1          STOP      =   $FFE1    ;CHECK STOP KEY
290: FFD2          BASOUT    =   $FFD2    ;OUTPUT BYTE
300: C400
300: C400
300: C400
320: 00BA          GA        =   $BA      ;DEVICE ADDR.
330: 00B9          SA        =   $B9      ;SECONDARY ADDR.
340: 0098          MAXBLK    =   $98      ;($5D) MAX BLOCKS
350: 000D          CR        =   $0D      ;RETURN
360: 00FB          COPANZ    =   $FB      ;LIST LENGTH
370: 00FC          TEMP      =   $FC      ;BUFFER
380: 00FE          FLAG      =   $FE      ;FLAG
390: 0041          LISTPT    =   $41      ;POINTER TO NAME LIST
400: 0340          FILPUF    =   $340     ;BUFFER
410: CA40          LNGTAB    =   START+$640 ;FILE NAMES
420: CB00          NAMTAB    =   START+$700 ;FILE LENGTHS
430: 0030          LSTMAX    =   $30      ;LIST LENGTH
440: 0009          TLSADR    =   $09      ;($13) HIBYTE LOAD
450: 0022          TEMP2     =   $22      ;AUX POINTER
460: 00AC          TPGSTA    =   $AC      ;START
470: 00AE          TPGEND    =   $AE      ;END
480: 00A7          PRGSTA    =   $A7      ;START
490: 00A9          PRGENG    =   $A9      ;END
495: 00C0          MOFLAG    =   $C0      ;MOTOR FLAG
520:
530:
560: C400 A9 01          LDA #1      ;(LDA #25)
570: C402 8D 20 D0       STA $D020 ;(STA $900F)
580: C405 8D 21 D0       STA $D021 ;(NOT USED)
590: C408 A9 06          LDA #6
600: C40A 8D 86 02       STA $286
610: C40D A9 5C          LDA #<TITEL
620: C40F A0 C8          LDY #>TITEL
630: C411 20 1E AB       JSR STROUT
640: C414 A9 34          LDA #<QUEINS
650: C416 A0 C8          LDY #>QUEINS
660: C418 20 1E AB       JSR STROUT
670: C41B 20 99 C7       JSR WAIT
680: C41E C9 45          CMP #"E"  ;STOP PROG
690: C420 D0 01          BNE OKCOPY
700: C422 60             RTS
701: C423
701: C423
702:
       ; READ IN CONTENTS
703: C423
703: C423
710: C423 20 B6 C7 OKCOPY JSR CARET
720: C426 20 B6 C7       JSR CARET
730: C429 20 9F C7       JSR INIT  ;INITILIZE DISK
740: C42C A9 08          LDA #$08
750: C42E AA             TAX
760: C42F A0 00          LDY #$00

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

770:  C431 20 BA FF      JSR  SETLFS
780:  C434 A9 01      LDA  #$01
790:  C436 A2 B6      LDX  #<CATALO
800:  C438 A0 C9      LDY  #>CATALO
810:  C43A 20 BD FF      JSR  SETNAM
820:  C43D 20 C0 FF      JSR  OPEN
830:  C440 A9 08      LDA  #$08
840:  C442 20 B4 FF      JSR  TALK
850:  C445 A9 00      LDA  #$00
860:  C447 20 96 FF      JSR  SETTALK
870:  C44A A0 04      LDY  #$04
880:  C44C 20 A5 FF LOOP1 JSR  IECIN      ;READ FIRST FOUR
890:  C44F 88          DEY          ; BYTES
900:  C450 D0 FA          BNE  LOOP1
910:  C452 20 A5 FF      JSR  IECIN      ;BLOCK NO.
920:  C455 85 22      STA  TEMP2
930:  C457 20 A5 FF      JSR  IECIN
940:  C45A A6 22      LDX  TEMP2
950:  C45C 20 CD BD      JSR  NUMOUT     ;OUTPUT
960:  C45F 20 B3 C7      JSR  SPACE
970:  C462 20 A5 FF LOOP2 JSR  IECIN      ;DISK NAME
980:  C465 F0 06      BEQ  NULL
990:  C467 20 D2 FF      JSR  BASOUT
1000: C46A 18          CLC
1010: C46B 90 F5      BCC  LOOP2
1020: C46D 20 B6 C7 NULL JSR  CARET
1030: C470 20 B6 C7      JSR  CARET
1040: C473 20 A5 FF      JSR  IECIN
1050: C476 20 A5 FF      JSR  IECIN
1060: C479 A0 00      LDY  #$00
1070: C47B 84 FB      STY  COPANZ
1071: C47D
1071: C47D
1073: C47D
1073: C47D
1080: C47D 20 A5 FF LSTART JSR  IECIN
1090: C480 85 FC      STA  TEMP
1100: C482 20 A5 FF      JSR  IECIN
1110: C485 85 FD      STA  TEMP+1
1120: C487 A6 FC      LDX  TEMP
1130: C489 20 CD BD      JSR  NUMOUT     ;OUTPUT BLOCK NO.
1140: C48C 20 B3 C7      JSR  SPACE
1150: C48F A0 00      LDY  #$00     ;READ IN
1160: C491 20 A5 FF WRFILE JSR  IECIN     ;FILE NAMES
1170: C494 48          PHA
1180: C495 20 BB C7      JSR  AUSGAB     ;OUTPUT THEM
1190: C498 68          PLA
1200: C499 99 40 03      STA  FILPUF,Y
1210: C49C F0 03      BEQ  NAMEND
1220: C49E C8          INY
1230: C49F D0 F0      BNE  WRFILE
1240: C4A1 20 A5 FF NAMEND JSR  IECIN

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

1250: C4A4 20 A5 FF      JSR  IECIN
1260: C4A7 A5 90      LDA  $90      ;CHECK STATUS
1270: C4A9 F0 03      BEQ  FRAGEN
1280: C4AB 4C 47 C5    JMP  INHEND
1290: C4AE A5 FD      LDA  TEMP+1   FRAGEN
1300: C4B0 D0 06      BNE  TOLONG
1310: C4B2 A5 FC      LDA  TEMP
1320: C4B4 C9 98      CMP  #MAXBLK
1330: C4B6 90 0A      BCC  LNGEOK   ;TEXT MAX LENGTH
1340: C4B8 A9 E8      LDA  #<SLONG  TOLONG
1350: C4BA A0 C8      LDY  #>SLONG
1360: C4BC 20 1E AB    JSR  STROUT   ;OUTPUT MESSAGE
1370: C4BF 4C 41 C5    JMP  NXFILE   FINISH
1380: C4C2 A5 FB      LDA  COPANZ   LNGEOK
1390: C4C4 C9 30      CMP  #LSTMAX
1400: C4C6 90 0A      BCC  FANZOK
1410: C4C8 A9 FB      LDA  #<LILONG
1420: C4CA A0 C8      LDY  #>LILONG
1430: C4CC 20 1E AB    JSR  STROUT   ;OUTPUT MESSAGE
1440: C4CF 18          CLC
1450: C4D0 90 ED      BCC  FINISH
1460: C4D2 A9 00      LDA  #$00     FANZOK
1470: C4D4 85 08      STA  $08
1480: C4D6 A9 1F      LDA  #$1F
1490: C4D8 85 D3      STA  $D3
1500: C4DA A9 0B      LDA  #<JANEIN
1510: C4DC A0 C9      LDY  #>JANEIN
1520: C4DE 20 1E AB    JSR  STROUT
1530: C4E1 20 E1 C7    JSR  INPUT    LOOP3
1540: C4E4 C9 4E      CMP  #"N"
1550: C4E6 F0 52      BEQ  NEIN
1560: C4E8 C9 4A      CMP  #"J"
1570: C4EA D0 F5      BNE  LOOP3
1580: C4EC A9 1D      LDA  #<STRJA
1590: C4EE A0 C9      LDY  #>STRJA
1600: C4F0 20 1E AB    JSR  STROUT
1610: C4F3 A5 FB      LDA  COPANZ
1620: C4F5 20 F1 C6    JSR  SETPTR
1630: C4F8 A2 00      LDX  #$00
1640: C4FA E8          INX          LOOP4
1650: C4FB BD 40 03    LDA  FILPUF,X
1660: C4FE C9 22      CMP  #$22     ;1ST ", " = START
1670: C500 D0 F8      BNE  LOOP4
1680: C502 86 FD      STX  TEMP+1   ;START SAVE
1690: C504 E8          INX
1700: C505 BD 40 03    LDA  FILPUF,X  UMSETZ
1710: C508 C9 22      CMP  #$22     ;2ND ", " = END
1720: C50A F0 06      BEQ  UMSSEND
1730: C50C 91 41      STA  (LISTPT),Y
1740: C50E E8          INX
1750: C50F C8          INY
1760: C510 D0 F3      BNE  UMSETZ

```


ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

1770: C512 8A          UMSEND TXA          ;END IN ACC.
1780: C513 A4 FB          LDY COPANZ ;INDEX IN Y-REG.
1790: C515 18          CLC
1800: C516 E5 FD          SBC TEMP+1 ;LENGTH
1810: C518 99 40 CA      STA LNGTAB,Y
1820: C51B BD 40 03 WETEST LDA FILPUF,X
1830: C51E D0 0A          BNE TYPTST
1840: C520 A9 29          ILL LDA #<STRILL
1850: C522 A0 C9          LDY #>STRILL
1860: C524 20 1E AB      JSR STROUT ;ILLEGAL FILE TYPE
1870: C527 4C 41 C5      JMP NXFILE
1880: C52A C9 53          TYPTST CMP # "S" ;TEST FOR PRG FILE
1890: C52C F0 F2          BEQ ILL
1900: C52E C9 50          CMP # "P"
1910: C530 F0 03          BEQ TYPOK
1920: C532 E8          INX
1930: C533 D0 E6          BNE WETEST
1940: C535 E6 FB          TYPOK INC COPANZ ;INC #OF FILES
1950: C537 18          CLC ;TO BE COPIED
1960: C538 90 07          BCC NXFILE
1970: C53A A9 3F          NEIN LDA #<STRNO
1980: C53C A0 C9          LDY #>STRNO
1990: C53E 20 1E AB      JSR STROUT
2000: C541 20 B6 C7 NXFILE JSR CARET
2010: C544 4C 7D C4      JMP LSTART
2020: C547 A9 08          INHEND LDA #$08 ;CLOSE
2030: C549 20 C3 FF      JSR CLOSE
2040: C54C A5 FB          LDA COPANZ
2050: C54E D0 03          BNE COPYST
2060: C550 4C 00 C4      JMP START
2062: C553
2062: C553
2062: C553
2064: ;PREPARE TO COPY
2066: C553
2066: C553
2066: C553
2070: C553 A9 4B          COPYST LDA #<FRAGE
2080: C555 A0 C9          LDY #>FRAGE ;
2090: C557 20 1E AB      JSR STROUT ;OUTPUT MESSAGE
2100: C55A 20 E1 C7 ABFRA2 JSR INPUT
2110: C55D C9 31          CMP # "1"
2120: C55F F0 07          BEQ EINZEL
2130: C561 C9 32          CMP # "2"
2140: C563 D0 F5          BNE ABFRA2
2150: C565 A9 00          LDA #$00 ;FLAG
2160: C567 2C          .BYT $2C
2170: C568 A9 FF          EINZEL LDA #$FF ;N
2180: C56A 85 FE          STA FLAG
2190: C56C A9 0F          LDA #$0F ;CLOSE ERROR CHANNEL
2200: C56E 20 C3 FF      JSR CLOSE
2202: C571

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

2202: C571
2202: C571
2204:                ;COPY - ROUTINE
2206: C571
2206: C571
2206: C571
2208:                ;PREPARE THE DISK
2209:                ;   FOR READING
2210: C571
2210: C571 A2 00                LDX #$00      ;TEMP DELETED
2220: C573 86 FC                STX TEMP     ;TEMP => PRG-CTER
2230: C575 A9 9C                LOLOOP LDA #<SREAD ;"READING"
2240: C577 A0 C9                LDY #>SREAD
2250: C579 20 1E AB             JSR STROUT   ; OUTPUT
2260: C57C A4 FC                LDY TEMP     ;POS. IN TABLE
2270: C57E BE 40 CA             LDX LNGTAB,Y;LENGTH IN X-REG.
2280: C581 A5 FC                LDA TEMP
2290: C583 20 F1 C6             JSR SETPTR
2300: C586 B1 41                WRTNAM LDA (LISTPT),Y ;FILENAME
2310: C588 20 D2 FF             JSR BASOUT
2320: C58B C8                    INY
2330: C58C CA                    DEX
2340: C58D D0 F7                BNE WRTNAM
2350: C58F A9 02                LDA #$02     ;FILE
2360: C591 A2 08                LDX #$08     ;OA
2370: C593 A8                    TAY         ;SA
2380: C594 20 BA FF             JSR SETLFS
2390: C597 A6 FC                LDX TEMP
2400: C599 BD 40 CA             LDA LNGTAB,X
2410: C59C 85 22                STA TEMP2
2420: C59E A5 FC                LDA TEMP
2430: C5A0 20 F1 C6             JSR SETPTR
2440: C5A3 A2 00                LDX #$00
2450: C5A5 B1 41                LOOP6 LDA (LISTPT),Y ;FILENAME IN
2460: C5A7 9D 40 03             STA FILPUF,X ; BUFFER
2470: C5AA C8                    INY
2480: C5AB E8                    INX
2490: C5AC C6 22                DEC TEMP2
2500: C5AE D0 F5                BNE LOOP6
2510: C5B0 A0 00                LDY #$00
2520: C5B2 B9 A7 C9             LOOP7 LDA PRGRE,Y ; ",P,R " APPEND
2530: C5B5 9D 40 03             STA FILPUF,X
2540: C5B8 C8                    INY
2550: C5B9 E8                    INX
2560: C5BA C0 04                CPY #$04
2570: C5BC 90 F4                BCC LOOP7
2580: C5BE 8A                    TXA
2590: C5BF A2 40                LDX #<FILPUF
2600: C5C1 A0 03                LDY #>FILPUF
2610: C5C3 20 BD FF             JSR SETNAM
2620: C5C6 20 C0 FF             JSR OPEN
2630: C5C9 A9 09                LDA #TLSADR

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

2640: C5CB A0 00          LDY  #$00
2650: C5CD 84 22          STY  TEMP2
2660: C5CF 85 23          STA  TEMP2+1
2670: C5D1 AD 11 D0      LDA  $D011 ; (LINES 2670-2690,
                                REMOVE FOR VIC)
2680: C5D4 29 EF          AND  #$EF ; OPEN SCREEN FROM
2690: C5D6 8D 11 D0      STA  $D011 ; CHANNEL 15
2700: C5D9 A9 0F          LDA  #$0F ;
2710: C5DB A2 08          LDX  #$08
2720: C5DD A8            TAY
2730: C5DE 20 BA FF      JSR  SETLFS
2740: C5E1 A9 03          LDA  #$03 ; "UI-" DISKETTE
2750: C5E3 A2 AC          LDX  #<UIMIN ; TO FAST
2760: C5E5 A0 C9          LDY  #>UIMIN
2770: C5E7 20 BD FF      JSR  SETNAM
2780: C5EA 20 C0 FF      JSR  OPEN
2790: C5ED A2 02          LDX  #$02 ; OPEN LOAD CHANNEL
2800: C5EF 20 0F F3      JSR  SFINUM
2810: C5F2 20 1F F3      JSR  SETPAR
2820: C5F5 A5 BA          LDA  GA
2830: C5F7 20 B4 FF      JSR  TALK
2840: C5FA A5 B9          LDA  SA
2850: C5FC 20 96 FF      JSR  SETALK
2860: C5FF A0 00          LDY  #$00
2862: C601
2862: C601
2864:                                ;LOAD PROGRAM IN BUFFER
2866: C601
2866: C601
2870: C601 20 A5 FF LDLOOP JSR  IECIN ; PRG LOADER
2880: C604 20 09 C7      JSR  STOBYT
2890: C607 A6 90          LDX  $90
2900: C609 F0 F6          BEQ  LDLOOP
2910: C60B 20 EB C7      JSR  FEHLER
2920: C60E 08            PHP
2930: C60F A9 02          LDA  #$02
2940: C611 20 C3 FF      JSR  CLOSE
2950: C614 AD 11 D0      LDA  $D011 ; (REMOVE)
2960: C617 09 10          ORA  #$10 ; (REMOVE)
2970: C619 8D 11 D0      STA  $D011 ; (REMOVE)
2980: C61C 28            PLP
2990: C61D 90 03          BCC  NOFEHL
3000: C61F 20 37 C7      JSR  FEMELD
3002: C622
3002: C622
3004:                                ;STORE ON CASSETTE
3006: C622
3006: C622
3010: C622 A6 FC          NOFEHL LDX  TEMP
3020: C624 A5 22          LDA  TEMP2
3030: C626 85 AE          STA  TPGEND
3040: C628 A5 23          LDA  TEMP2+1

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

3050: C62A 85 AF          STA  TPGEND+1
3060: C62C E8            INX
3070: C62D A5 FC          LDA  TEMP
3075: C62F D0 0D          BNE  NOMELD
3080: C631 A9 46          LDA  #<ZICASS;"INSERT CASSETTE
3090: C633 A0 C8          LDY  #>ZICASS
3100: C635 20 1E AB        JSR  STROUT
3105: C638 20 E1 C7        JSR  INPUT
3110: C63B 20 25 C8        JSR  TASTE ;CHECK RECORDER KEY
3115: C63E A9 B8          LDA  #<WRITI
3120: C640 A0 C9          LDY  #>WRITI
3125: C642 20 1E AB        JSR  STROUT
3130: C645 A4 FC          LDY  TEMP
3135: C647 BE 40 CA        LDX  LNGTAB,Y
3140: C64A A5 FC          LDA  TEMP
3145: C64C 20 F1 C6        JSR  SETPTR
3150: C64F B1 41          LDA  (LISTPT),Y
3155: C651 20 D2 FF        JSR  BASOUT
3160: C654 C8            INY
3160: C655 CA            DEX
3165: C656 D0 F7          BNE  WRNAM2
3210: C658 A2 01          LDX  #$01 ;FILEPARAMETER
3220: C65A A9 00          LDA  #0 ;FOR CASSETTE
3230: C65C A8            TAY
3240: C65D 20 BA FF        JSR  SETLFS
3250: C660 A6 FC          LDX  TEMP
3260: C662 BD 40 CA        LDA  LNGTAB,X
3270: C665 85 22          STA  TEMP2
3280: C667 A5 FC          LDA  TEMP
3290: C669 20 F1 C6        JSR  SETPTR
3300: C66C A2 00          LDX  #$00
3310: C66E B1 41          LDA  (LISTPT),Y ;FILE NAME
3320: C670 9D 40 03        STA  FILPUF,X ;CASSETTE
3330: C673 E8            INX ;BUFFER
3340: C674 C8            INY
3350: C675 C6 22          DEC  TEMP2
3360: C677 D0 F5          BNE  ULOOP
3370: C679 8A            TXA ;FILE NAME PARAMETER
3380: C67A A2 40          LDX  #<FILPUF ;IN
3390: C67C A0 03          LDY  #>FILPUF
3400: C67E 20 BD FF        JSR  SETNAM
3410: C681 A0 00          LDY  #$00
3420: C683 A9 09          LDA  #TLSADR
3425: C685 84 AC          STY  TPGSTA
3430: C687 85 AD          STA  TPGSTA+1
3440: C689 B1 AC          LDA  (TPGSTA),Y ;PRG STRT ADDR
3450: C68B 85 A7          STA  PRGSTA ;IN PRG STRT VCTR
3460: C68D C8            INY
3470: C68E B1 AC          LDA  (TPGSTA),Y
3480: C690 85 A8          STA  PRGSTA+1
3490: C692 C8            INY
3500: C693 84 AC          STY  TPGSTA

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

3520: C695 A5 AE          LDA  TPGEND  ;END ADDR
3530: C697 38            SEC
3540: C698 E5 AC          SBC  TPGSTA
3550: C69A 08            PHP
3555: C69E 18            CLC
3560: C69C 65 A7          ADC  PRGSTA
3570: C69E 85 A9          STA  PRGEND
3580: C6A0 A5 AF          LDA  TPGEND+1
3590: C6A2 65 A8          ADC  PRGSTA+1
3600: C6A4 28            PLP
3610: C6A5 E5 AD          SBC  TPGSTA+1
3620: C6A7 85 AA          STA  PRGEND+1
3622: C6A9
3622: C6A9
3624:                                ;STORE PROGRAM ON CASS.
3626: C6A9
3626: C6A9
3630: C6A9 A2 05          LDX  #5      ;SYNCHRONIZATIONS
3640: C6AB 86 AB          STX  $AB     ;REPEATED
3650: C6AD 20 85 C0        JSR  ABSOLUT ;FASTTAPE
3690: C6B0 20 E5 C6        JSR  SCRON   ;(REMOVE)
3700: C6B3 E6 FC          INC  TEMP
3710: C6B5 A6 FC          LDX  TEMP
3720: C6B7 E4 FB          CPX  COPANZ
3730: C6B9 B0 07          BCS  CENDEN
3732: C6BB 24 FE          BIT  FLAG
3733: C6BD 10 10          BPL  EINZE2
3740: C6BF 4C DF C6        JMP  NXTFIL
3755: C6C2 A9 C2          LDA  #<EOCOP ;"END OF COPING"
3760: C6C4 A0 C9          LDY  #>EOCOP
3770: C6C6 20 1E AB        JSR  STROUT
3780: C6C9 20 E1 C7        JSR  INPUT
3790: C6CC 4C 00 C4        JMP  START
3810: C6CF 20 55 C7        JSR  NNAMAU
3820: C6D2 A9 46          LDA  #<ZICASS;"INSERT CASSETTE
3830: C6D4 A0 C8          LDY  #>ZICASS
3840: C6D6 20 1E AB        JSR  STROUT
3845: C6D9 20 E1 C7        JSR  INPUT  ;WAIT FOR KEY
3850: C6DC 20 25 C8        JSR  TASTE
3870: C6DF 20 B6 C7        JSR  CARET
3880: C6E2 4C 75 C5        JMP  LOLOOP ;NEXT FILE
3882: C6E5
3882: C6E5
3882: C6E5
3884:                                ; SUBROUTINES
3886: C6E5
3886: C6E5
3886: C6E5
3887:                                ;SCREEN ON
3888: C6E5
3888: C6E5
3890: C6E5 20 73 C7        JSR  SEUIP  ;(REMOVE LINES

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

3900: C6E8 AD 11 D0          LDA  $D011    ;3890-3930)
3910: C6EB 09 10          ORA  #$10
3920: C6ED 8D 11 D0          STA  $D011
3930: C6F0 60              RTS
3932: C6F1
3932: C6F1
3936: C6F1
3936: C6F1
3940: C6F1 A0 00          SETPTR  LDY  #$00
3950: C6F3 0A              ASL  A
3960: C6F4 0A              ASL  A
3970: C6F5 84 42          STY  LISTPT+1
3980: C6F7 0A              ASL  A
3990: C6F8 26 42          ROL  LISTPT+1
4000: C6FA 0A              ASL  A
4010: C6FB 26 42          ROL  LISTPT+1
4020: C6FD 85 41          STA  LISTPT
4030: C6FF A5 42          LDA  LISTPT+1
4040: C701 18              CLC
4050: C702 69 CB          ADC  #>NAMTAB
4060: C704 85 42          STA  LISTPT+1
4070: C706 A0 00          LDY  #$00
4080: C708 60              RTS
4082: C709
4082: C709
4084:                      ;BYTE IN STORAGE
4086: C709
4086: C709
4090: C709 91 22          STOBYT STA  (TEMP2),Y
4100: C70B E6 22          INC  TEMP2    ;INC COUNT
4110: C70D D0 02          BNE  NOINC
4120: C70F E6 23          INC  TEMP2+1
4130: C711 58          NOINC  CLI
4140: C712 60              RTS
4142: C713
4142: C713
4144:                      ;ASCII-BYTE CONVERSION
4146: C713
4146: C713
4150: C713 20 A5 FF ASCHX  JSR  IECIN
4160: C716 29 0F          AND  #$0F
4170: C718 0A              ASL  A
4180: C719 0A              ASL  A
4190: C71A 0A              ASL  A
4200: C71B 0A              ASL  A
4210: C71C 85 57          STA  $57
4220: C71E 20 A5 FF          JSR  IECIN
4230: C721 29 0F          AND  #$0F
4240: C723 05 57          ORA  $57
4250: C725 60              RTS
4252: C726
4252: C726

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

4254:                                ;HEX IN ASCII-BYTE
4255:                                ;      AND OUTPUT
4256:  C726
4256:  C726
4260:  C726 48      HEXASC  PHA
4270:  C727 4A      LSR  A
4280:  C728 4A      LSR  A
4290:  C729 4A      LSR  A
4300:  C72A 4A      LSR  A
4310:  C72B 20 2F C7 JSR  HALBBT
4320:  C72E 68      PLA
4330:  C72F 29 0F      HALBBT AND  #$0F
4340:  C731 18      CLC
4350:  C732 69 30      ADC  #$30
4360:  C734 4C D2 FF      JMP  BASOUT ;OUTPUT
4362:  C737
4362:  C737
4364:                                ;ERROR OUTPUT
4366:  C737
4366:  C737
4370:  C737 AD 11 D0 FEMELD LDA  $D011 ; (REMOVE LINES
4380:  C73A 09 10      ORA  #$10 ;4370-4390)
4390:  C73C 8D 11 D0      STA  $D011
4400:  C73F A9 D7      LDA  #<WMACH ;"CONTINUE ? "
4410:  C741 A0 C9      LDY  #>WMACH
4420:  C743 20 1E AB      JSR  STROUT
4430:  C746 20 E1 C7 FLOOP JSR  INPUT ;WAIT FOR KEY
4440:  C749 C9 59      CMP  #$59
4450:  C74B D0 01      BNE  NEIN2
4460:  C74D 60      RTS
4470:  C74E C9 4E      NEIN2 CMP  #$4E
4480:  C750 D0 F4      BNE  FLOOP
4490:  C752 4C D0 C7      JMP  ENDE
4492:  C755
4492:  C755
4496:  C755
4496:  C755
4500:  C755 A9 E4      NNAMAU LDA  #<SNFILE ;"NEXT FILE"
4510:  C757 A0 C9      LDY  #>SNFILE
4520:  C759 20 1E AB      JSR  STROUT
4530:  C75C A5 FC      LDA  TEMP ;POS IN TABLE
4540:  C75E 0A      ASL  A
4550:  C75F 0A      ASL  A
4560:  C760 0A      ASL  A
4570:  C761 0A      ASL  A
4580:  C762 A6 FC      LDX  TEMP
4590:  C764 BC 40 CA      LDY  LNGTAB,X
4600:  C767 AA      TAX
4610:  C768 BD 00 CB WRNAM3 LDA  NAMTAB,X;FILE NAME OUTPUT
4620:  C76B 20 D2 FF      JSR  BASOUT
4630:  C76E E8      INX
4640:  C76F 88      DEY

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

4650: C770 D0 F6          BNE  WRNAM3
4660: C772 60            RTS
4662: C773
4662: C773
4666: C773
4666: C773
4670: C773 AD 11 D0 SEUIP LDA  $D011    ; (REMOVE LINES
                                4670-4820)
4680: C776 29 EF          AND  #$EF      ; SCREEN ON
4690: C778 8D 11 D0      STA  $D011
4700: C77B A9 0F          LDA  #$0F
4710: C77D 20 C3 FF      JSR  CLOSE
4720: C780 A9 0F          LDA  #$0F
4730: C782 A2 08          LDX  #$08
4740: C784 A8            TAY
4750: C785 20 BA FF      JSR  SETLFS
4760: C788 A9 03          LDA  #$03
4770: C78A A2 B0          LDX  #<UIPLU
4780: C78C A0 C9          LDY  #>UIPLU
4790: C78E 20 BD FF      JSR  SETNAM
4800: C791 20 C0 FF      JSR  OPEN
4810: C794 A9 0F          LDA  #$0F
4820: C796 4C C3 FF      JMP  CLOSE
4822: C799
4822: C799
4824:                    ; DELAY LOOP
4826: C799
4826: C799
4830: C799 20 E4 FF WAIT  JSR  GETBYT
4840: C79C F0 FB          BEQ  WAIT
4850: C79E 60            RTS
4852: C79F
4852: C79F
4854:                    ;DISK INITIALIZATION
4856: C79F
4856: C79F
4860: C79F A9 0F          INIT  LDA  #$0F    ; OPEN
4870: C7A1 A2 08          LDX  #$08    ; CHANNEL
4880: C7A3 A8            TAY
4890: C7A4 20 BA FF      JSR  SETLFS
4900: C7A7 A9 01          LDA  #$01    ; "I"
4910: C7A9 A2 B4          LDX  #<STRI
4920: C7AB A0 C9          LDY  #>STRI
4930: C7AD 20 BD FF      JSR  SETNAM
4940: C7B0 4C C0 FF      JMP  OPEN
4942: C7B3
4942: C7B3
4944:                    ;SPACE OR CARRIAGE RETURN
4946: C7B3
4946: C7B3
4950: C7B3 A9 20          SPACE LDA  #$20    ;SPACE
4960: C7B5 2C            .BYT $2C

```


ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

4970: C7B6 A9 0D   CARET   LDA #CR      ;CARRIAGE RETURN
4980: C7B8 4C D2 FF   JMP  BASOUT  ;SEND
4982: C7BB
4982: C7BB
4984:                                ;OUTPUT ROUTINE
4986: C7BB
4986: C7BB
4990: C7BB 20 D2 FF AUSGAB JSR  BASOUT
5000: C7BE 8A      STASTE   TXA      ;SAVE A,X,Y REGISTERS
5010: C7BF 48      PHA
5020: C7C0 98      TYA
5030: C7C1 48      PHA
5040: C7C2 20 E1 FF JSR  STOP   ;CHRCK FOR STOP KEY
5050: C7C5 18      CLC
5060: C7C6 D0 01    BNE  WEITER
5070: C7C8 38      SEC
5080: C7C9 68      WEITER   PLA      ;REPLACE REGISTERS
5090: C7CA A8      TAY
5100: C7CB 68      PLA
5110: C7CC AA      TAX
5120: C7CD B0 01    STOP2   BCS  ENDE
5130: C7CF 60      RETURN  RTS
5132: C7D0
5132: C7D0
5134:                                ;      RESTART
5136: C7D0
5136: C7D0
5140: C7D0 A2 F6    ENDE    LDX  #$F6
5150: C7D2 9A      TXS
5160: C7D3 20 E5 C6 JSR  SCRON  ; (REMOVE)
5170: C7D6 A9 0F    LDA  #$0F
5180: C7D8 20 C3 FF JSR  CLOSE
5190: C7DB 20 E7 FF JSR  CLALL
5200: C7DE 4C 00 C4 JMP  START
5202: C7E1
5202: C7E1
5204:                                ;  INPUT ROUTINE
5206: C7E1
5206: C7E1
5210: C7E1 20 E4 FF INPUT JSR  GETBYT
5220: C7E4 D0 E9    BNE  RETURN
5230: C7E6 20 BE C7 JSR  STASTE
5240: C7E9 90 F6    BCC  INPUT
5250: C7EB A2 0F    FEHLER  LDX  #$0F ;STATUS
5260: C7ED 20 0F F3 JSR  SFINUM
5270: C7F0 20 1F F3 JSR  SETPAR
5280: C7F3 A5 BA    LDA  GA
5290: C7F5 20 B4 FF JSR  TALK
5300: C7F8 A5 B9    LDA  SA
5310: C7FA 20 96 FF JSR  SETALK
5320: C7FD 20 13 C7 JSR  ASCHEX
5330: C800 C9 20    CMP  #$20

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

5340: C802 08 PHP
5350: C803 90 0B BCC MELAUS
5360: C805 48 PHA
5370: C806 20 B6 C7 JSR CARET
5380: C809 20 B6 C7 JSR CARET
5390: C80C 68 PLA
5400: C80D 20 26 C7 JSR HEXASC
5410: C810 20 A5 FF MELAUS JSR IECIN
5420: C813 C9 0D CMP #CR
5430: C815 F0 09 BEQ MELEND
5440: C817 28 PLP
5450: C818 08 PHP
5460: C819 90 F5 BCC MELAUS
5470: C81B 20 D2 FF JSR BASOUT
5480: C81E 90 F0 BCC MELAUS
5490: C820 20 AB FF MELEND JSR UNTALK
5500: C823 28 PLP
5500: C824 60 RTS
5502: C825 20 38 F8 TASTE JSR PTASTE
5502: C828 20 CD C7 JSR STOP2
5502: C82B D0 F8 BNE TASTE
5504: C82D A9 07 LDA #7
5504: C82F 85 C0 STA MOFLAG
5506: C831 4C CA FC JMP MOT AUS
5511: C834
5511: C834
5511: C834
5512: ; OUTPUT STRINGS
5513: C834
5513: C834
5513: C834
5520: C834 0D 0D 0D QUEINS .BYT 13,13,13
5530: C837 49 4E 53 .ASC "INSERT [RVS]DISK![OFF]"
5540: C845 00 .BYT 0
5550: C846 0D 0D ZICASS .BYT 13,13
5560: C848 49 4E 53 .ASC "INSERT [RVS]CASSETTE[OFF]!"
5570: C85A 0D 00 .BYT 13,0
5580: C85C 93 11 1C TITEL .ASC "[HOME] [DWN] [RED]
BACKUP FOR DISK"
5590: C879 0D 0D .BYT 13,13
5600: C87B 1F 20 20 .ASC "[BLUE] CASSETTE"
5602: C892 0D 0D .BYT 13,13
5604: C894 20 20 20 .ASC "[RVS] [ORG]
FASTTAP E[OFF]"
5606: C8B3 0D 0D 0D .BYT 13,13,13
5608: C8B6 20 20 20 .ASC "(C) DIRK PAULISSEN "
5610: C8D0 0D 0D 0D .BYT 13,13,13
5620: C8D3 1E 20 20 .ASC "[GRN] E = END[BLUE]"
5630: C8E7 00 .BYT 0
5640: C8E8 0D SLONG .BYT 13
5650: C8E9 1C 12 46 .ASC "[RED] [RVS]
FILE TO LONG!![BLUE]"

```

ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

```

5660: C8FA 00 .BYT 0
5670: C8FB 0D LILONG .BYT 13
5680: C8FC 1C 12 4C .ASC "[RED][RVS]LIST
FULL!![BLUE]"
5690: C90A 00 .BYT 0
5700: C90B 12 59 45 JANEIN .ASC "[RVS]YES/NO [7
LEFT][OFF][BLUE]"
5710: C91C 00 .BYT 0
5720: C91D 12 1E 20 STRJA .ASC "[RVS][GRN] YES
[OFF][BLUE]"
5730: C928 00 .BYT 0
5740: C929 0D STRILL .BYT 13
5750: C92A 1C 12 49 .ASC "[RED][RVS]ILLEGAL FILE
TYPE[BLUE]"
5760: C93E 00 .BYT 0
5770: C93F 1C 12 20 STRNO .ASC "[RVS] NO [OFF][BLUE]"
5780: C94A 00 .BYT 0
5790: C94B 0D 0D 0D FRAGE .BYT 13,13,13
5800: C94E 57 41 4E .ASC "WANT TO COPY PROGRAMS"
5810: C963 0D 0D .BYT 13,13
5820: C965 20 20 20 .ASC " 1 CONTINUE"
5830: C976 0D 0D .BYT 13,13
5840: C978 20 20 20 .ASC " 2 INDIVIDUALLY"
5850: C98D 0D 0D .BYT 13,13
5860: C98F 43 48 4F .ASC "CHOOSE ONE ?"
5870: C99B 00 .BYT 0
5880: C99C 0D 0D SREAD .BYT 13,13
5890: C99E 52 45 41 .ASC "READING "
5900: C9A6 00 .BYT 0
5910: C9A7 2C 50 2C PRGRE .ASC ",P,R"
5920: C9AB 00 .BYT 0
5930: C9AC 55 49 2D UIMIN .ASC "UI-"
5940: C9AF 00 .BYT 0
5950: C9B0 55 49 2B UIPLU .ASC "UI+"
5960: C9B3 00 .BYT 0
5970: C9B4 49 STRI .ASC "I"
5980: C9B5 00 .BYT 0
5985: C9B6 24 CATALO .ASC "$"
5986: C9B7 00 .BYT 0
5990: C9B8 0D 0D WRITI .BYT 13,13
6000: C9BA 53 41 56 .ASC "SAVING "
6010: C9C1 00 .BYT 0
6020: C9C2 0D 0D EOCOP .BYT 13,13
6030: C9C4 1C 12 45 .ASC "END OF COPYING "
6040: C9D6 00 .BYT 0
6045: C9D7 0D WMACH .BYT 13
6050: C9DF 43 4F 4E .ASC "CONTINUE ? "
6060: C9EA 00 .BYT 00
6070: C9EB 0D 0D SNFILE .BYT 13,13
6080: C9ED 4E 45 58 .ASC "NEXT FILE: "
6090: C9F8 00 .BYT 0
UC400-C9F9

```

```

10 REM          BACKUP FOR DISK TO TAPE
20 REM          [USING FASTTAPE (MUST BE
30 REM          IN MEMORY)]
40 FORI=50176TO51706:READX:POKEI,X:NEXT
50 SYS50176
1000 DATA169,1,141,32,208,141,33,208,169
,6,141,134,2,169,92
1010 DATA160,200,32,30,171,169,52,160,20
0,32,30,171,32,153,199
1020 DATA201,69,208,1,96,32,182,199,32,1
82,199,32,159,199,169
1030 DATA8,170,160,0,32,186,255,169,1,16
2,189,160,201,32,189
1040 DATA255,32,192,255,169,8,32,180,255
1,169,0,32,150,255,160
1050 DATA4,32,165,255,136,208,250,32,165
,255,133,34,32,165,255
1060 DATA166,34,32,205,189,32,179,199,32
,165,255,240,6,32,210
1070 DATA255,24,144,245,32,182,199,32,18
2,199,32,165,255,32,165
1080 DATA255,160,0,132,251,32,165,255,13
3,252,32,165,255,133,253
1090 DATA166,252,32,205,189,32,179,199,1
60,0,32,165,255,72,32
1100 DATA187,199,104,153,64,3,240,3,200,
208,240,32,165,255,32
1110 DATA165,255,165,144,240,3,76,71,197
,165,253,208,6,165,252
1120 DATA201,152,144,10,169,239,160,200,
32,30,171,76,65,197,165
1130 DATA251,201,48,144,10,169,2,160,201
,32,30,171,24,144,237
1140 DATA169,0,133,8,169,31,133,211,169,
18,160,201,32,30,171
1150 DATA32,225,199,201,78,240,82,201,89
,208,245,169,36,160,201
1160 DATA32,30,171,165,251,32,241,198,16
2,0,232,189,64,3,201
1170 DATA34,208,248,134,253,232,189,64,3
,201,34,240,6,145,65
1180 DATA232,200,208,243,138,164,251,24,
229,253,153,64,202,189,64
1190 DATA3,208,10,169,48,160,201,32,30,1
71,76,65,197,201,83
1200 DATA240,242,201,80,240,3,232,208,23
0,230,251,24,144,7,169
1210 DATA70,160,201,32,30,171,32,182,199
,76,125,196,169,8,32
1220 DATA195,255,165,251,208,3,76,0,196,
169,82,160,201,32,30
1230 DATA171,32,225,199,201,49,240,7,201

```

,50,208,245,169,0,44
 1240 DATA169,255,133,254,169,15,32,195,2
 55,162,0,134,252,169,163
 1250 DATA160,201,32,30,171,164,252,190,6
 4,202,165,252,32,241,198
 1260 DATA177,65,32,210,255,200,202,208,2
 47,169,2,162,8,168,32
 1270 DATA186,255,166,252,189,64,202,133,
 34,165,252,32,241,198,162
 1280 DATA0,177,65,157,64,3,200,232,198,3
 4,208,245,160,0,185
 1290 DATA174,201,157,64,3,200,232,192,4,
 144,244,138,162,64,160
 1300 DATA3,32,189,255,32,192,255,169,9,1
 60,0,132,34,133,35
 1310 DATA173,17,208,41,239,141,17,208,16
 9,15,162,8,168,32,186
 1320 DATA255,169,3,162,179,160,201,32,18
 9,255,32,192,255,162,2
 1330 DATA32,15,243,32,31,243,165,186,32,
 180,255,165,185,32,150
 1340 DATA255,160,0,32,165,255,32,9,199,1
 66,144,240,246,32,235
 1350 DATA199,8,169,2,32,195,255,173,17,2
 08,9,16,141,17,208
 1360 DATA40,144,3,32,55,199,166,252,165,
 34,133,174,165,35,133
 1370 DATA175,232,165,252,208,13,169,70,1
 60,200,32,30,171,32,225
 1380 DATA199,32,37,200,169,191,160,201,3
 2,30,171,164,252,190,64
 1390 DATA202,165,252,32,241,198,177,65,3
 2,210,255,200,202,208,247
 1400 DATA162,1,169,0,168,32,186,255,166,
 252,189,64,202,133,34
 1410 DATA165,252,32,241,198,162,0,177,65
 ,157,64,3,232,200,198
 1420 DATA34,208,245,138,162,64,160,3,32,
 189,255,160,0,169,9
 1430 DATA132,172,133,173,177,172,133,167
 ,200,177,172,133,168,200,132
 1440 DATA172,165,174,56,229,172,8,24,101
 ,167,133,169,165,175,101
 1450 DATA168,40,229,173,133,170,162,5,13
 4,171,32,133,192,32,229
 1460 DATA198,230,252,166,252,228,251,176
 ,7,36,254,16,16,76,223
 1470 DATA198,169,201,160,201,32,30,171,3
 2,225,199,76,0,196,32
 1480 DATA85,199,169,70,160,200,32,30,171
 ,32,225,199,32,37,200
 1490 DATA32,182,199,76,117,197,32,115,19

9,173,17,208,9,16,141
 1500 DATA17,208,96,160,0,10,10,132,66,10
 ,38,66,10,38,66
 1510 DATA133,65,165,66,24,105,203,133,66
 ,160,0,96,145,34,230
 1520 DATA34,208,2,230,35,88,96,32,165,25
 5,41,15,10,10,10
 1530 DATA10,133,87,32,165,255,41,15,5,87
 ,96,72,74,74,74
 1540 DATA74,32,47,199,104,41,15,24,105,4
 8,76,210,255,173,17
 1550 DATA208,9,16,141,17,208,169,222,160
 ,201,32,30,171,32,225
 1560 DATA199,201,89,208,1,96,201,78,208,
 244,76,208,199,169,235
 1570 DATA160,201,32,30,171,165,252,10,10
 ,10,10,166,252,188,64
 1580 DATA202,170,189,0,203,32,210,255,23
 2,136,208,246,96,173,17
 1590 DATA208,41,239,141,17,208,169,15,32
 ,195,255,169,15,162,8
 1600 DATA168,32,186,255,169,3,162,183,16
 0,201,32,189,255,32,192
 1610 DATA255,169,15,76,195,255,32,228,25
 5,240,251,96,169,15,162
 1620 DATA8,168,32,186,255,169,1,162,187,
 160,201,32,189,255,76
 1630 DATA192,255,169,32,44,169,13,76,210
 ,255,32,210,255,138,72
 1640 DATA152,72,32,225,255,24,208,1,56,1
 04,168,104,170,176,1
 1650 DATA96,162,246,154,32,229,198,169,1
 5,32,195,255,32,231,255
 1660 DATA76,0,196,32,228,255,208,233,32,
 190,199,144,246,162,15
 1670 DATA32,15,243,32,31,243,165,186,32,
 180,255,165,185,32,150
 1680 DATA255,32,19,199,201,32,8,144,11,7
 2,32,182,199,32,182
 1690 DATA199,104,32,38,199,32,165,255,20
 1,13,240,9,40,8,144
 1700 DATA245,32,210,255,144,240,32,171,2
 55,40,96,32,56,248,32
 1710 DATA205,199,208,248,169,7,133,192,7
 6,202,252,13,13,13,73
 1720 DATA78,83,69,82,84,32,18,68,73,83,7
 5,146,33,0,13
 1730 DATA13,73,78,83,69,82,84,32,18,67,6
 5,83,83,69,84
 1740 DATA84,69,146,33,13,0,147,17,28,32,
 32,32,32,32,32
 1750 DATA32,32,32,32,66,65,67,75,85,80,3

2, 70, 82, 79, 77
 1760 DATA32, 68, 73, 83, 75, 69, 84, 84, 69, 13, 1
 3, 31, 32, 32, 32
 1770 DATA32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 3
 2, 84, 79, 32, 67
 1780 DATA65, 83, 83, 69, 84, 84, 69, 13, 13, 32, 3
 2, 32, 32, 32, 32
 1790 DATA32, 32, 32, 32, 32, 32, 18, 129, 70, 32,
 65, 32, 83, 32, 84
 1800 DATA32, 84, 32, 65, 32, 80, 32, 69, 32, 146,
 13, 13, 13, 32, 32
 1810 DATA32, 40, 67, 41, 32, 32, 32, 32, 68, 73, 8
 2, 75, 32, 80, 65
 1820 DATA85, 76, 73, 83, 83, 69, 78, 32, 32, 13, 1
 3, 13, 30, 32, 32
 1830 DATA32, 32, 32, 32, 32, 32, 32, 32, 32, 69, 3
 2, 61, 32, 69, 78
 1840 DATA68, 31, 0, 13, 28, 18, 70, 73, 76, 69, 32
 , 84, 79, 32, 76
 1850 DATA79, 78, 71, 33, 33, 31, 0, 13, 28, 18, 76
 , 73, 83, 84, 160
 1860 DATA70, 85, 76, 76, 33, 33, 31, 0, 18, 89, 69
 , 83, 47, 78, 79
 1870 DATA32, 157, 157, 157, 157, 157, 157, 157,
 146, 31, 0, 18, 30, 32, 89
 1880 DATA69, 83, 32, 32, 146, 32, 31, 0, 13, 28, 1
 8, 73, 76, 76, 69
 1890 DATA71, 65, 76, 32, 70, 73, 76, 69, 32, 84, 8
 9, 80, 69, 31, 0
 1900 DATA28, 18, 32, 32, 78, 79, 32, 32, 146, 32,
 31, 0, 13, 13, 13
 1910 DATA87, 65, 78, 84, 32, 84, 79, 32, 67, 79, 8
 0, 89, 32, 80, 82
 1920 DATA79, 71, 82, 65, 77, 83, 13, 13, 32, 32, 3
 2, 32, 49, 32, 32
 1930 DATA32, 32, 67, 79, 78, 84, 73, 78, 85, 69, 1
 3, 13, 32, 32, 32
 1940 DATA32, 50, 32, 32, 32, 32, 73, 78, 68, 73, 8
 6, 73, 68, 85, 65
 1950 DATA76, 76, 89, 13, 13, 67, 72, 79, 79, 83, 6
 9, 32, 79, 78, 69
 1960 DATA32, 63, 0, 13, 13, 82, 69, 65, 68, 73, 78
 , 71, 32, 0, 44
 1970 DATA80, 44, 82, 0, 85, 73, 45, 0, 85, 73, 43,
 0, 73, 0, 36
 1980 DATA0, 13, 13, 83, 65, 86, 73, 78, 71, 32, 0,
 13, 13, 28, 18
 1990 DATA69, 78, 68, 32, 79, 70, 32, 67, 79, 80, 7
 3, 78, 71, 32, 146
 2000 DATA31, 0, 13, 67, 79, 78, 84, 73, 78, 85, 69
 , 32, 63, 32, 0
 2010 DATA13, 13, 78, 69, 88, 84, 32, 70, 73, 76, 6

9,58,32,0,115
2020 DATA65
READY.


```

10 REM DISK TO TAPE BACKUP FOR THE VIC
20 REM USING FASTTAPE
100 FORI=29696TO31231:READX:POKEI,X:NEXT
110 SYS29696
1000 DATA169,25,141,15,144,169,6,141,134
,2,169,6,160,120,32
1010 DATA30,203,169,222,160,119,32,30,20
3,32,70,119,201,69,208
1020 DATA1,96,32,99,119,32,99,119,32,76,
119,169,8,170,160
1030 DATA0,32,186,255,169,1,162,99,160,1
21,32,189,255,32,192
1040 DATA255,169,8,32,180,255,169,0,32,1
50,255,160,4,32,165
1050 DATA255,136,208,250,32,165,255,133,
34,32,165,255,166,34,32
1060 DATA205,221,32,96,119,32,165,255,24
0,6,32,210,255,24,144
1070 DATA245,32,99,119,32,99,119,32,165,
255,32,165,255,160,0
1080 DATA132,251,32,165,255,133,252,32,1
65,255,133,253,166,252,32
1090 DATA205,221,32,96,119,160,0,32,165,
255,72,32,104,119,104
1100 DATA153,64,3,240,3,200,208,240,32,1
65,255,32,165,255,165
1110 DATA144,240,3,76,68,117,165,253,208
,6,165,252,201,93,144
1120 DATA10,169,146,160,120,32,30,203,76
,62,117,165,251,201,48
1130 DATA144,10,169,165,160,120,32,30,20
3,24,144,237,169,0,133
1140 DATA8,169,31,133,211,169,181,160,12
0,32,30,203,32,139,119
1150 DATA201,78,240,82,201,74,208,245,16
9,199,160,120,32,30,203
1160 DATA165,251,32,204,118,162,0,232,18
9,64,3,201,34,208,248
1170 DATA134,253,232,189,64,3,201,34,240
,6,145,65,232,200,208
1180 DATA243,138,164,251,24,229,253,153,
64,122,189,64,3,208,10
1190 DATA169,211,160,120,32,30,203,76,62
,117,201,83,240,242,201
1200 DATA80,240,3,232,208,230,230,251,24
,144,7,169,233,160,120
1210 DATA32,30,203,32,99,119,76,122,116,
169,8,32,195,255,165
1220 DATA251,208,3,76,0,116,169,245,160,
120,32,30,203,32,139
1230 DATA119,201,49,240,7,201,50,208,245
,169,0,44,169,255,133

```

1240 DATA254,169,15,32,195,255,162,0,134
,252,169,73,160,121,32
1250 DATA30,203,164,252,190,64,122,165,2
52,32,204,118,177,65,32
1260 DATA210,255,200,202,208,247,169,2,1
62,8,168,32,186,255,166
1270 DATA252,189,64,122,133,34,165,252,3
2,204,118,162,0,177,65
1280 DATA157,64,3,200,232,198,34,208,245
,160,0,185,84,121,157
1290 DATA64,3,200,232,192,4,144,244,138,
162,64,160,3,32,189
1300 DATA255,32,192,255,169,19,160,0,132
,34,133,35,169,15,162
1310 DATA8,168,32,186,255,169,3,162,89,1
60,121,32,189,255,32
1320 DATA192,255,162,2,32,207,243,32,223
,243,165,186,32,180,255
1330 DATA165,185,32,150,255,160,0,32,165
,255,32,228,118,166,144
1340 DATA240,246,32,149,119,8,169,2,32,1
95,255,40,144,0,166
1350 DATA252,165,34,133,174,165,35,133,1
75,232,165,252,208,13,169
1360 DATA240,160,119,32,30,203,32,139,11
9,32,207,119,169,101,160
1370 DATA121,32,30,203,164,252,190,64,12
2,165,252,32,204,118,177
1380 DATA65,32,210,255,200,202,208,247,1
62,1,169,0,168,32,186
1390 DATA255,166,252,189,64,122,133,34,1
65,252,32,204,118,162,0
1400 DATA177,65,157,64,3,232,200,198,34,
208,245,138,162,64,160
1410 DATA3,32,189,255,160,0,169,19,132,1
72,133,173,177,172,133
1420 DATA167,200,177,172,133,168,200,132
,172,165,174,56,229,172,8
1430 DATA24,101,167,133,169,165,175,101,
168,40,229,173,133,170,162
1440 DATA5,134,171,32,127,112,230,252,16
6,252,228,251,176,7,36
1450 DATA254,16,16,76,198,118,169,111,16
0,121,32,30,203,32,139
1460 DATA119,76,0,116,32,40,119,169,240,
160,119,32,30,203,32
1470 DATA139,119,32,207,119,32,99,119,76
,114,117,160,0,10,10
1480 DATA132,66,10,38,66,10,38,66,133,6
5,165,66,24,105,123
1490 DATA133,66,160,0,96,145,34,230,34,2
08,2,230,35,88,96

1500 DATA32,165,255,41,15,10,10,10,10,13
3,87,32,165,255,41
1510 DATA15,5,87,96,72,74,74,74,32,10
,119,104,41,15
1520 DATA24,105,48,76,210,255,169,132,16
0,121,32,30,203,32,139
1530 DATA119,201,89,208,1,96,201,78,208,
244,76,125,119,169,145
1540 DATA160,121,32,30,203,165,252,10,10
,10,10,166,252,188,64
1550 DATA122,170,189,0,123,32,210,255,23
2,136,208,246,96,32,228
1560 DATA255,240,251,96,169,15,162,8,168
,32,186,255,169,1,162
1570 DATA97,160,121,32,189,255,76,192,25
5,169,32,44,169,13,76
1580 DATA210,255,32,210,255,138,72,152,7
2,32,225,255,24,208,1
1590 DATA56,104,168,104,170,176,1,96,162
,246,154,169,15,32,195
1600 DATA255,32,231,255,76,0,116,32,228,
255,208,236,32,107,119
1610 DATA144,246,162,15,32,207,243,32,22
3,243,165,186,32,180,255
1620 DATA165,185,32,150,255,32,238,118,2
01,32,8,144,11,72,32
1630 DATA99,119,32,99,119,104,32,1,119,3
2,165,255,201,13,240
1640 DATA9,40,8,144,245,32,210,255,144,2
40,32,171,255,40,96
1650 DATA32,183,248,32,122,119,208,248,1
69,7,133,192,76,8,253
1660 DATA13,13,13,73,78,83,69,82,84,32,1
8,68,73,83,75
1670 DATA146,33,0,13,13,73,78,83,69,82,8
4,32,18,67,65
1680 DATA83,83,69,84,84,69,146,33,13,0,1
47,17,28,32,32
1690 DATA32,32,32,32,32,66,65,67,75,85,8
0,32,70,79,82
1700 DATA32,68,73,83,75,69,84,84,69,13,1
3,31,32,32,32
1710 DATA32,32,32,32,32,32,32,32,32,32,3
2,67,65,83,83
1720 DATA69,84,84,69,13,13,32,32,32,32,3
2,32,32,32,32
1730 DATA32,32,32,18,129,70,32,65,32,83,
32,84,32,84,32
1740 DATA65,32,80,32,69,32,146,13,13,13,
32,32,32,40,67
1750 DATA41,32,32,32,32,68,73,82,75,32,8
0,65,85,76,73

1760 DATA83,83,69,78,32,32,13,13,13,30,3
 2,32,32,32,32
 1770 DATA32,32,32,32,32,32,69,32,61,32,6
 9,78,68,31,0
 1780 DATA13,28,18,70,73,76,69,32,84,79,3
 2,76,79,78,71
 1790 DATA33,33,31,0,13,28,18,76,73,83,84
 ,160,70,85,76
 1800 DATA76,33,33,31,0,18,89,69,83,47,78
 ,79,32,157,157
 1810 DATA157,157,157,157,157,146,31,0,18
 ,30,32,89,69,83,32
 1820 DATA32,146,32,31,0,13,28,18,73,76,7
 6,69,71,65,76
 1830 DATA32,70,73,76,69,32,84,89,80,69,3
 1,0,28,18,32
 1840 DATA32,78,79,32,32,146,32,31,0,13,1
 3,13,87,65,78
 1850 DATA84,32,84,79,32,67,79,80,89,32,8
 0,82,79,71,82
 1860 DATA65,77,83,13,13,32,32,32,32,49,3
 2,32,32,32,67
 1870 DATA79,78,84,73,78,85,65,76,76,89,1
 3,13,32,32,32
 1880 DATA32,50,32,32,32,32,73,78,68,73,8
 6,73,68,85,65
 1890 DATA76,76,89,13,13,67,72,79,79,83,6
 9,32,79,78,69
 1900 DATA32,63,0,13,13,82,69,65,68,73,78
 ,71,32,0,44
 1910 DATA80,44,82,0,85,73,45,0,85,73,43,
 0,73,0,36
 1920 DATA0,13,13,83,65,86,73,78,71,32,0,
 13,13,28,18
 1930 DATA69,78,68,32,79,70,32,67,79,80,7
 3,78,71,32,146
 1940 DATA31,0,13,67,79,78,84,73,78,85,69
 ,32,63,32,0
 1950 DATA13,13,78,69,88,84,32,70,73,76,6
 9,58,32,0,58
 1960 DATA32,0,32,119,32,190,160,190,0,11
 9,32,119,160,190,32
 1970 DATA254,32,127,0,87,0,190,0,190,0,1
 19,32,119,32,158
 1980 DATA160,0,0,119,0,119,160,190,0,190
 ,32,247,32,247,160
 1990 DATA254,160,254,32,119,0,87,0,254,0
 ,190,0,119,32,127
 2000 DATA32,254,32,190,32,119,0,119,0,19
 0,0,254,0,255,32
 2010 DATA127,0,254,0,190,0,119,0,127,0,2
 54,0,254,32,247

2020 DATA0,87,0,190,0,190
READY.

15.4 PROGRAM DESCRIPTION OF CASSETTE BACK-UP TO DISK

For those of your who are particularly interested in such matters and those of you who know Assembly language, here is a description of the program.

Preliminary Remark:

This program operates with two different loading and starting addresses. The actual starting address is the address of the buffer at which the program is placed in intermediate storage for copying. The proper starting address is the one at which the program is normally stored and where it operates.

The principle of the program is as follows:

After initialization of the diskette, the file names are first read into intermediate storage in the cassette buffer, starting at \$0340. The user is then asked whether the program is to be copied or not. If the question is answered with "Y", a test is made to see whether the maximum block length or the maximum length of the list of file names has been exceeded, the file name and its length are transferred to two tables (NAMTAB,LNGTAB) after the back-up program. The program also tests for the file type of the file to be copied. If it is not a program file, then an error message is output.

When the entire contents has been processed in this way, control passes to the actual copying routine.

When "READING names" is output, the corresponding file name is transferred to the cassette buffer again and ",P,R" attached to it. This extended file name is used to open a read file and the corresponding program is loaded after \$0900 (VIC 20 = \$1300), the actual program load address is (TLSADR). If no error has occurred during reading, the program start and end addresses, which are to be written in the program header, and the actual program start and end addresses, where the program is actually stored at the moment, are transferred to the FastTape storage routine. Thereafter, control branches "ABSOLUTELY" to the FastTape routine. Control branches back to the start again afterwards, if there are more files to be copied.

I have generously seasoned the ASSEMBLER listing with comments, so as to make the program as easy as possible to follow.

The disk drive can operate at two different transfer speeds. The faster of the two is specially for the VIC 20, while the slower speed is for the C-64, since this operates more slowly because of the screen control. If the screen is switched off, the C-64 will operate just as fast as the VIC 20 and is thus able to drive the diskette at the higher transfer rate. For this reason, the screen is switched off during the loading process and the diskette changed to the higher transfer rate.

This explains why the corresponding changeovers, which are clearly identified in the listing, are not required for VIC 20.

APPENDIX

Important Storage Locations

Hex	Decimal	Meaning
=0001	1	CPU output register Bit3 writing line for datassette Bit4 CASS-SENS, recorder key check Bit5 Motor Control
000A 10		LOAD/VERIFY flag, 0=LOAD/1=VERIFY
002B-002C	33-34	Pointer for Basic RAM start
002D-002E	35-36	Pointer for Basic program end/Start of variables
002F-0030	37-38	Pointer for beginning of arrays
0031-0032	49-50	Pointer for end of arrays
0033-0034	51-52	Pointer to end of strings
0037-0038	55-56	Pointer to end of Basic RAM
0090 144		Status byte
0093 147		LOAD/VERIFY flag; 0=LOAD/1=VERIFY
0098 152		Number of open files
0099 153		Input devices
009A 154		Output devices
00A6 166		Pointer in cassette buffer
00AC-00AD	172-173	Pointer in actual byte being written or read
00AE-00AF	174-175	Pointer to end of program when reading/writing
00B2-00B3	178-179	Pointer to start of cassette buffer
00B7 183		File name length
00B8 184		Actual logical file number
00B9 185		Actual secondary address
00BA 186		Actual device address

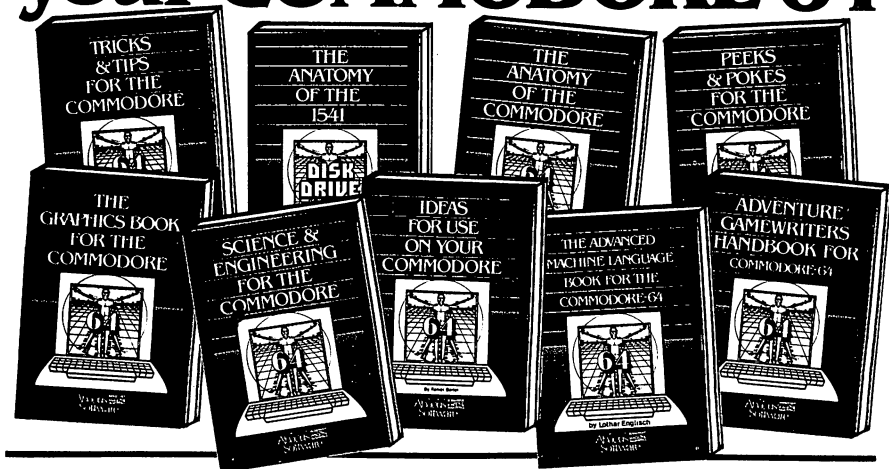
ABACUS Software CASSETTE BOOK for the COMMODORE 64 and VIC

00BE	190	Pass counter when reading and writing
00BD	191	Input/output byte
00C0	192	Cassette motor flag
00C1-00C2	193-194	Input/output start address
00C3-00C4	195-196	Input/output end address
033C-03FB	828-1019	Cassette buffer
033C	828	File type
033D-033E	829-830	Program start address
033F-0340	831-832	Program end address
0341-	833-	File names
*911c	37148	Bit5 cassette motor switch
*911F	37151	Bit6 CASS-SENS; recorder key check
*9120	37152	Bit3 Writing Line

+ applies only to C-64

* applies only to VIC 20

Required Reading for your COMMODORE 64



TRICKS & TIPS FOR YOUR C-64 - treasure chest of easy-to-use programming techniques. Advanced graphics, easy data input, enhanced BASIC, CPM, character sets, transferring data between computers, more.
ISBN# 0-916439-03-8 276 pages \$19.95

GRAPHICS BOOK FOR C-64 - from fundamentals to advanced topics this is most complete reference available. Sprite animation, Hires, Multicolor, lightpen, IRQ, 3D graphics, projections. Dozens of samples.
ISBN# 0-916439-05-4 350 pages \$19.95

SCIENCE & ENGINEERING ON THE C-64 - starts by discussing variable types, computational accuracy, sort algorithms, more. Topics from chemistry, physics, biology, astronomy, electronics. Many programs.
ISBN# 0-916439-05-7 280 pages \$19.95

ANATOMY OF 1541 DISK DRIVE - bestselling handbook available on using the floppy disk. Clearly explains disk files with many examples and utilities. Includes complete commented 1541 ROM listings.
ISBN# 0-916439-01-1 320 pages \$19.95

ANATOMY OF COMMODORE 64 - insider's guide to the '64 internals. Describes graphics, sound synthesis, I/O, kernel routines, more. Includes complete commented ROM listings. Fourth printing.
ISBN# 0-916439-003 300 pages \$19.95

IDEAS FOR USE ON YOUR C-64 - Wonder what to do with your '64? Dozens of useful ideas including complete listings for auto expenses, electronic calculator, store window advertising, recipe file, more.
ISBN# 0-916439-07-0 200 pages \$12.95

PEEKs & POKEs FOR THE C-64 - programming quickies that will simply amaze you. This guide is packed full of techniques for the BASIC programmer.
ISBN# 0-916439-13-5 180 pages \$14.95

ADVANCED MACHINE LANGUAGE FOR C-64 - covers topics such as video controller, timer and real time clock, serial and parallel I/O, extending BASIC commands, interrupts. Dozens of sample listings.
ISBN# 0-916439-08-2 210 pages \$14.95

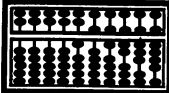
ADVENTURE GAMEWRITER'S HANDBOOK - is a step-by-step guide to designing and writing your own adventure games. Includes listing for an automated adventure game generator.
ISBN# 0-916439-14-3 200 pages \$14.95

Call today for the name of your nearest dealer Phone: (616) 241-5510

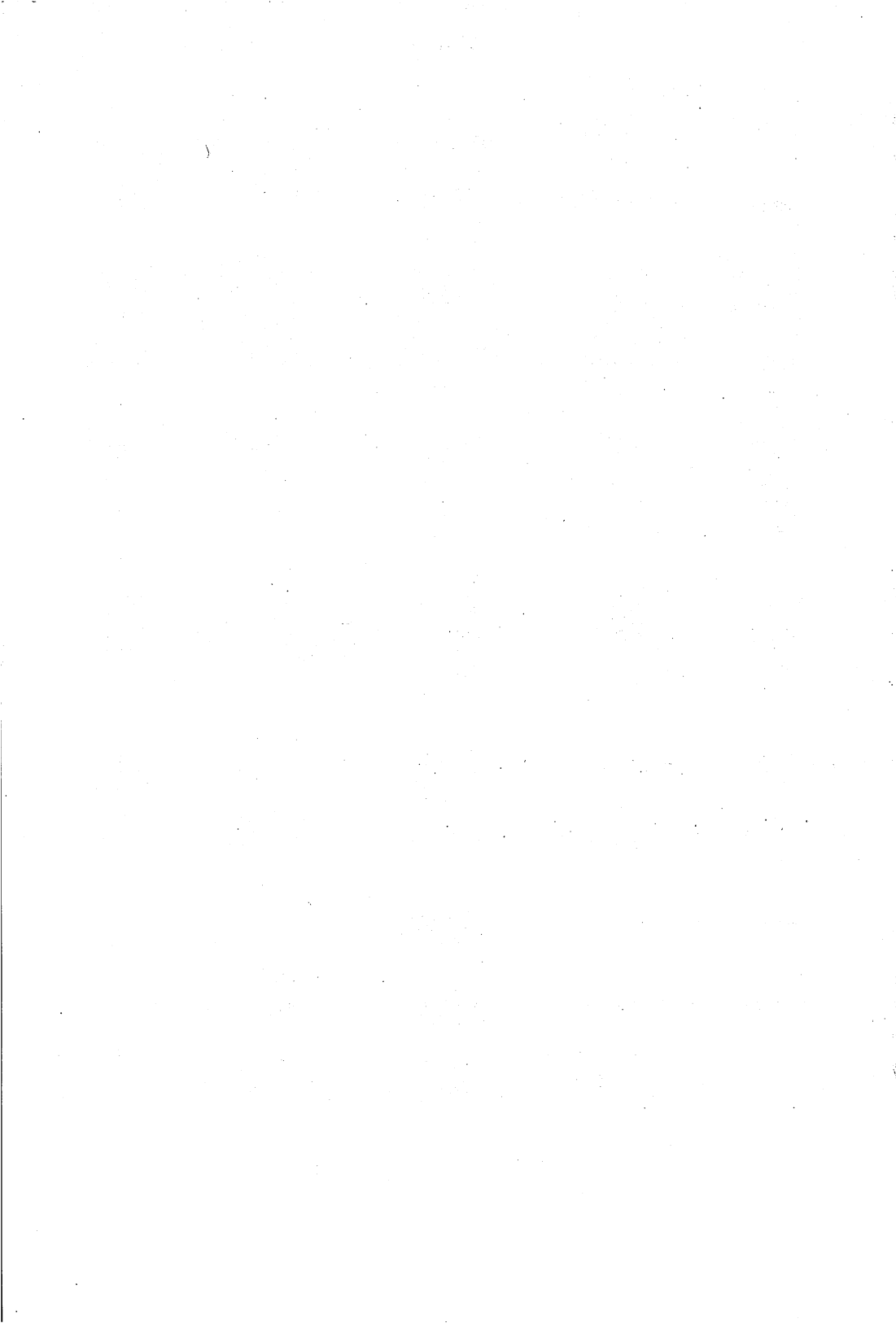
Other titles are available, call or write for a complete free catalog.

For postage and handling include \$4.00 (\$6.00 foreign) per order. Money order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted.

Michigan residents include 4% sales tax. CANADA: Book Center, Montreal Phone: (514) 332-4154

You Can Count On  **Abacus Software**

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

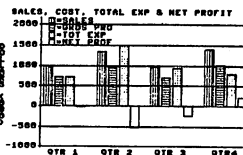


Make your '64 work fulltime

MAKE YOUR OWN CHARTS...

CHARTPAK-64

produces professional quality charts and graphs instantly from your data. 8 chart formats. Hardcopy in two sizes to popular dot matrix printers. \$39.95
ISBN# 0-916439-19-4



Also Available CHARTPLOT-64 for unsurpassed quality charts on plotters.
ISBN# 0-916439-20-8 \$84.95

DETAIL YOUR DESIGNS...

CADPAK-64

superb lightpen design tool. exact placement of object using our Accu-Point positioning. Has two complete screens. Draw LINES, BOXES, CIRCLES, ELLIPSES; pattern FILLING; freehand DRAW; COPY sections of screen; ZOOM in and do detail work. Hard copy in two sizes to popular dot matrix printers. ISBN# 0-916439-18-6 \$49.95

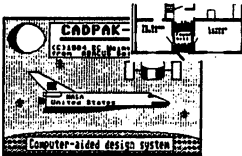
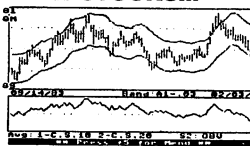


CHART YOUR OWN STOCKS...

TAS-64

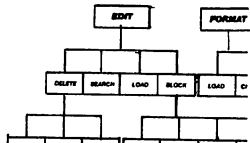
sophisticated technical analysis charting package for the serious stock market investor. Capture data from DJN/RS or Warner services or enter and edit data at keyboard. 7 moving averages, 3 oscillators, trading bands, least squares, 5 volume indicators, relative charts, much more. Hardcopy in two sizes, most printers. ISBN# 0-916439-24-0 \$84.95



DO YOUR OWN WORD PROCESSING

TEXTOMAT-64

flexible wordprocessing package supporting 40 or 80 columns with horizontal scrolling. Commands are clearly displayed on the screen awaiting your choice. Quickly move from editing to formatting to merging to utilities. Will work with virtually any printer.



ISBN# 0-916439-12-7 \$39.95

CREATE SPREADSHEETS & GRAPHS...

POWER PLAN-64

not only a powerful spreadsheet packages available, but with built in graphics too. The 275 page manual has tutorial section and HELP screens are always available. Features field protection; text formatting; windowing; row and column copy, sort; duplicate and delete. ISBN# 0-916439-22-4 \$49.95

Columns: C10	POWER PLAN-64		
111	A	B	C
1	Sales	Jan	Feb
2	Distributors	47.2	54.2
3	Retailers	27.9	35.4
4	Mail Order	18.5	23.7
5			
6		92.6	113.3
7			
8	Expenses		
9	Materials	8.2	9.2
10	Office	2.0	2.8
11	Shipping	4.4	5.0
12	Advertising	12.9	15.0
13	Payroll	10.5	10.7
14			
15		38.0	41.5
16			
17	Profit	55.4	71.8

FREE PEEKS & POKES POSTER WITH SOFTWARE
For name & address of your nearest dealer call (616) 241-5510

ORGANIZE YOUR DATA...

DATAMAT-64

powerful, yet easy-to-use data management package. Free form design of screen using up to 50 fields per record. Maximum of 2000 records per diskette. Complete and flexible reporting. Sorting on multiple fields in any combination. Select records for printing in desired format.

INVENTORY FILE

Item Number ___ Description ___

Onhand ___ Price ___

Location ___

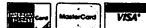
Reord. Pt. ___ Reord. Qty. ___

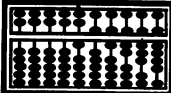
Cost ___

ISBN# 0-916439-16-X \$39.95

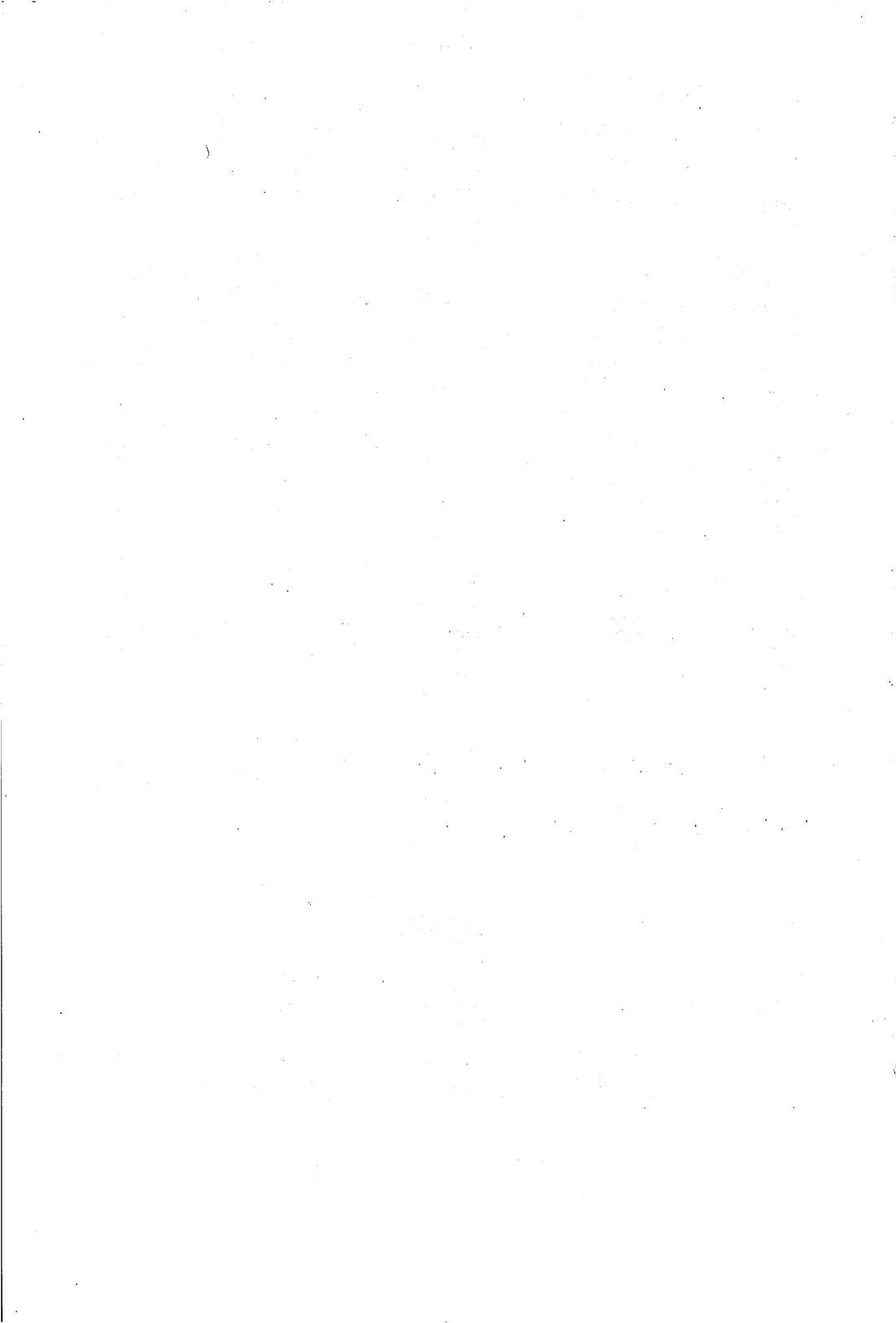
Other titles available. For FREE CATALOG and name of nearest dealer, write or call (616) 241-5510. For postage and handling, include \$4.00 (\$6.00 foreign) per order. Money Order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents include 4% sales tax.

CANADA: Book Center, Montreal (514) 332-4154

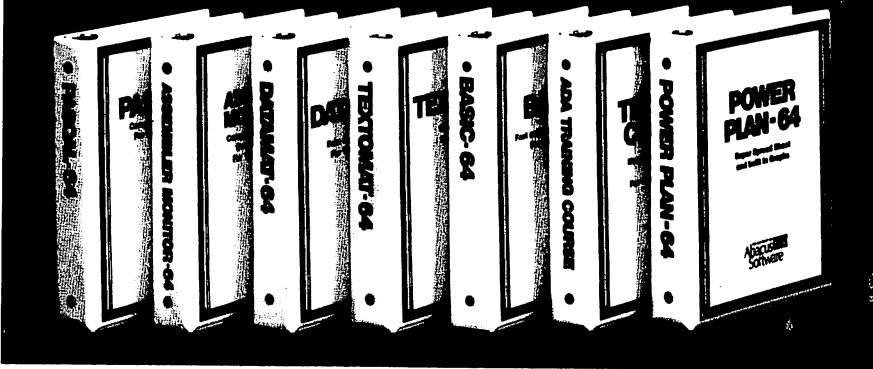


You Can Count On  Software

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510



Break the BASIC language barrier



VIDEO BASIC-64 - ADD 50+ graphic and sound commands to your programs with this super development package. You can distribute free RUN-TIME version without paying royalties!
 ISBN# 0-916439-28-7 \$59.95

BASIC COMPILER 64 - compiles the complete BASIC language into either fast 6510 machine language and/or compact speedcode. Get your programs into high gear and protect them by compiling.
 ISBN# 0-916439-17-8 \$39.95

MASTER-64 - professional development package for serious applications. Indexed file system, full screen management, programmer's aid, BASIC extensions, 100 commands.
 ISBN# 0-916439-21-6 \$39.95

PASCAL-64 - full Pascal with extensions for graphics, sprites, file management, more. Compiles to 6510 machine code and can link to Assembler/Monitor routines.
 ISBN# 0-916439-10-9 \$39.95

ADA TRAINING COURSE - teaches you the language of the future. Comprehensive subset of the language, editor, syntax checker/compiler, assembler, disassembler, 120+ page guide.
 ISBN# 0-916439-15-1 \$59.95

FORTH-64 - loaded with hires graphics, complete synthesizer control, full screen editor, programming tools, assembler.
 ISBN 0-916439-32-1 \$39.95

C LANGUAGE COMPILER - a full C language compiler. Conforms to the Kernighan & Ritchie standard, but without bit fields. Package includes editor, compiler and linker.
 ISBN# 0-916439-28-3 \$79.95

ASSEMBLER MONITOR-64 - a macro assembler and extended monitor package. Assembler supports floating point constants. Monitor supports bank switching, quick trace, single step, more.
 ISBN# 0-916439-11-8 \$39.95

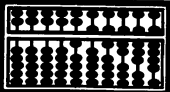
XREF-64 - indispensable tool for BASIC programmer cross-references all references to variable and line numbers.
 ISBN# 0-916439-27-5 \$17.95

OTHER TITLES ALSO AVAILABLE - WRITE OR CALL FOR A FREE COMPLETE CATALOG
 Call today for the name and address of your nearest local dealer.
PHONE: (616) 241-5510

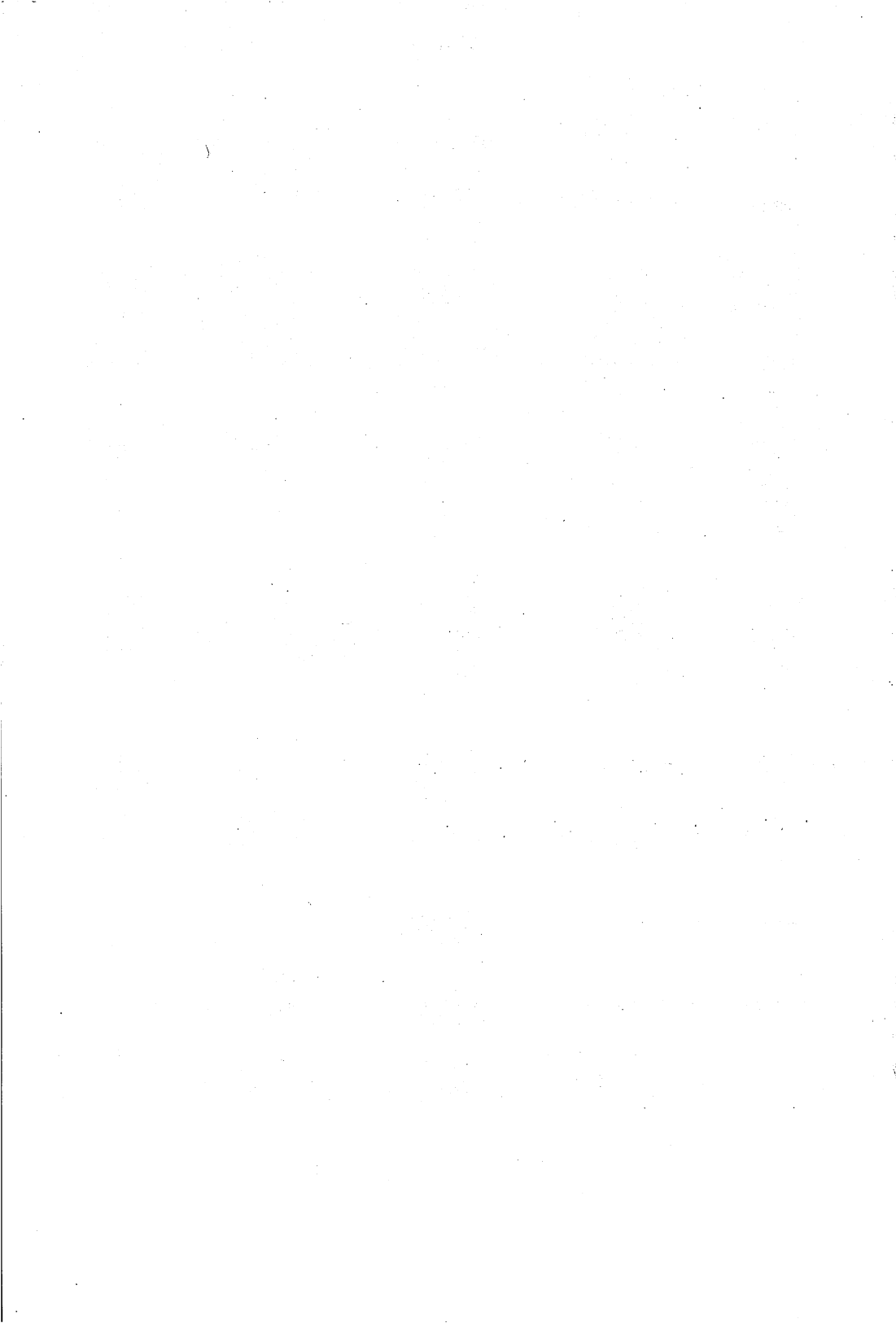
For postage and handling include \$4.00 (\$8.00 foreign) per order. Money order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents incl 4% sales tax.

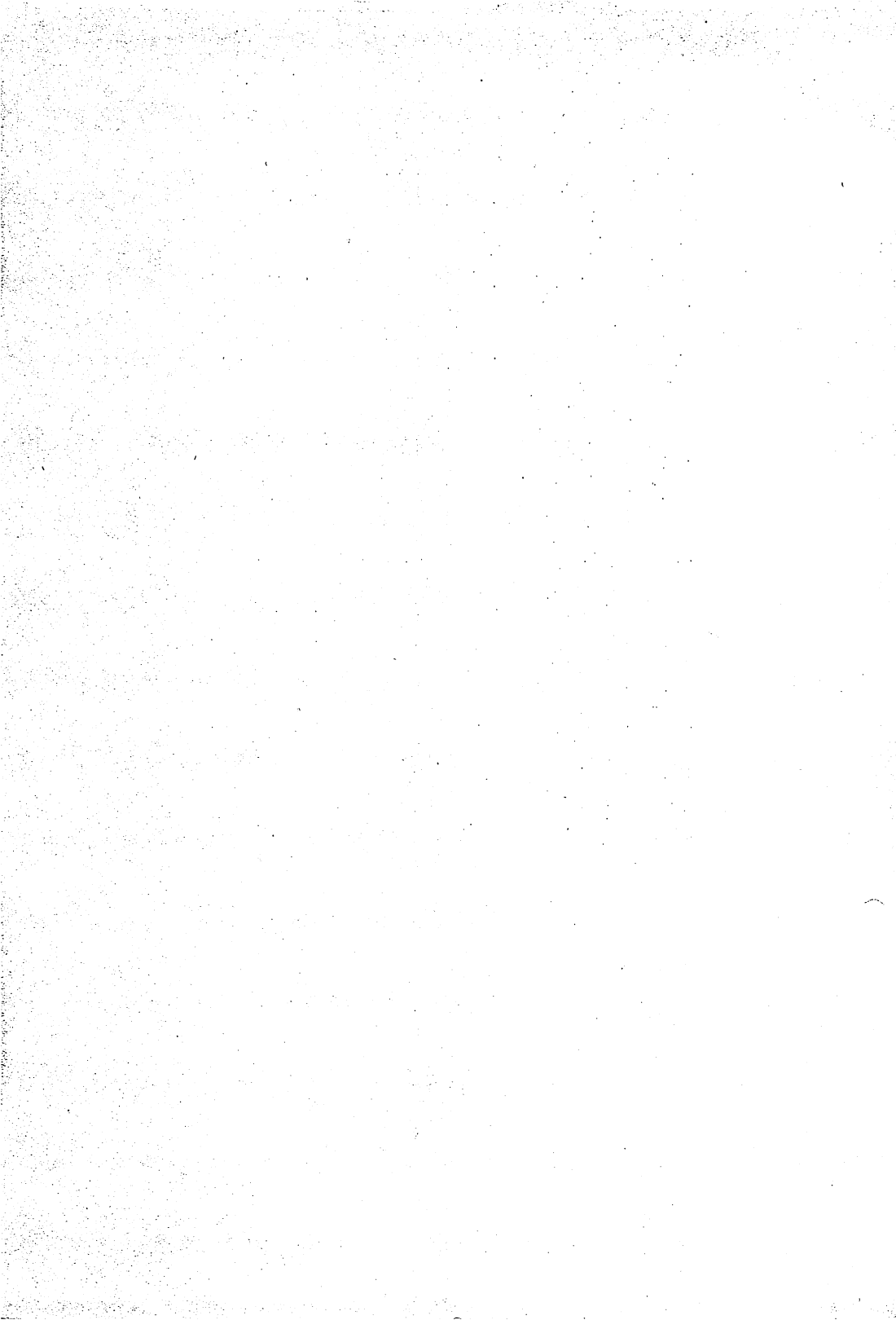


FREE PEEKS & POKES WALL POSTER INCLUDED WITH EVERY SOFTWARE PURCHASE

You Can Count On  **Abacus Software**

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510





CASSETTE BOOK

FOR THE

COMMODORE-64

AND VIC-20

This book is almost required if you own a Commodore Datasete. It completely explains the operations of the cassette: from the actual mechanical operation to a faster operating system.

Included is the "FAST TAPE" operating system which is quicker than the 1541 Disk Drive. Processing data with "FAST TAPE" is also explained.

Some of the topics covered include:

- Self starting programs
- Protected programs
- Other cassettes
- FAST TAPE
- Disk back-up to cassette
- Cassette back-up to disk

ISBN 0-916439-04-6

YOU CAN COUNT ON
Abacus



Software

P.O. BOX 7211 GRAND RAPIDS, MICH. 49510 PHONE 616-241-5510