Your '64 CASSETTE Book

A Data Becker Book from First Publishing Limited



1st
FIRST PUBLISHING LTD

Paulissen

Your Cassette Book on the Commodore 64 and Vic 20

A Data Becker book from
First Publishing Ltd.

First English Edition, October 1984

Printed in Germany

Copyright © 1984 Data Becker GmbH
Merowingerstrasse 30
4000 Dusseldorf
West Germany

Copyright © 1984 First Publishing Ltd
Unit 20B
Horseshoe Park
Pangbourne, Berkshire

This book is copyrighted. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic or otherwise, without the prior written permission of First Publishing Ltd or Data Becker GmbH.

Important Note

This book and the programs contained therein have been very carefully compiled by the author and checked during reproduction. Nevertheless, errors cannot totally be avoided. First Publishing Ltd are therefore obliged to point out that they cannot give any guarantee or accept legal responsibility for any consequences attributable to program errors or incorrect information in the book. We shall be pleased to learn of any errors at any time.

ISBN 0-948015-357

Foreword

This book is a must for every cassette owner. It will explain in detail the internal workings and how you can get much faster loading, better storage and much more.

We hope you enjoy it.

IMPORTANT NOTE FOR READERS

The circuits, procedures and programs reproduced in this manual are passed on to the reader without regard to the patent situation. They are exclusively intended for the use of amateurs and for tuition purposes, and may not be used for commercial purposes

All the circuits, technical data and programs used in this book have been produced or assembled by the author with the greatest care, and subjected to the closest scrutiny before publication. Nevertheless, the possibility of errors can never be entirely ruled out, and FIRST PUBLISHING feel obliged to point out that they offer no guarantee, and accept no liability or responsibility of any kind, in respect of any consequences arising out of the use of faulty data or information contained in this book.

The author will be grateful to readers for any errors found.

LIST OF CONTENTS

1.	Introduction	4
	PART 1	5
2.	Instructions for Datasette handling	5
2.1	SAVE	6
2.2	LOAD	8
2.3	VERIFY	9
2.4	OPEN	9
2.5	PRINT# and CLOSE	10
2.6	INPUT#	12
2.7	GET#	17
3.	Secondary address	18
4.	Status variable	20
4.1	Saving programs after 'LOAD ERROR'	21
5.	Loading and storage under programe control	24
5.1	Overlay technique	25
6.	Storage of machine language programs	27
7.	Cassette buffer	31
7.1	Creating a list of cassette contents	32
7.2	Display of files found	33
7.3	Self-catering programs and program protection	34
8.	Storage format in cassette storage	39
9.	Appending Basic programs	40
10.	Program control of Datassette	44
11.	Datassette hardware	50
11.1	Care of Datassette	50
11.2	Choice and handling of cassettes	51
11.3	Datassette operation	52
11.4	Loudspeaker for Datassette	54

11.5	Adjusting the head	55
11.6	Other cassette recorders for data storage	59
	PART 2	62
12	New 'FastTape' cassette operating system	62
12.1	Program description	87
13.	Data processing with 'FastTape'	94
13.1	Program description	113
14.	Cassette contents and catalogue of FastTape cassettes	118
15.	Disk back-up for cassette and vice versa	120
15.1	Disk back-up for cassette	120
15.2	Program description of disk back-up for cassette	127
15.3	Cassette back-up for diskette	128
15.4	Program description of cassette back-up for disk	156

APPENDIX

A.	Important storage locations	158

1. INTRODUCTION

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

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

However, if the computer is to be able to read all the information it had previously put into storage again, it must store it 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 it all.

However, if you read through the operating instructions, you will soon realize that they say very little about the way the C-64 or VIC 20 with Datassette functions, or how to operate them.

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

Using lots of small sample programs, I would like to make it clear to you that one can get a great deal more out of the cassette than the manual suggests. The crowning glory, in the second part of the book, is a complete new operating system for storing the program and any data on the Datassette. This system is considerably quicker than the built-in operating system, indeed, it is even quicker than a disk drive. With the aid of this program package, I would like to show you that the much cheaper cassette recorder is usually perfectly adequate for use 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 the VIC 20. As 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, a Basic loader for the C-64 and a Basic loader for the VIC 20, in order to give users maximum scope.

In the Basic programs, all the control characters have been placed in square brackets in order to make the programs easier to read. 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, to give just one example.

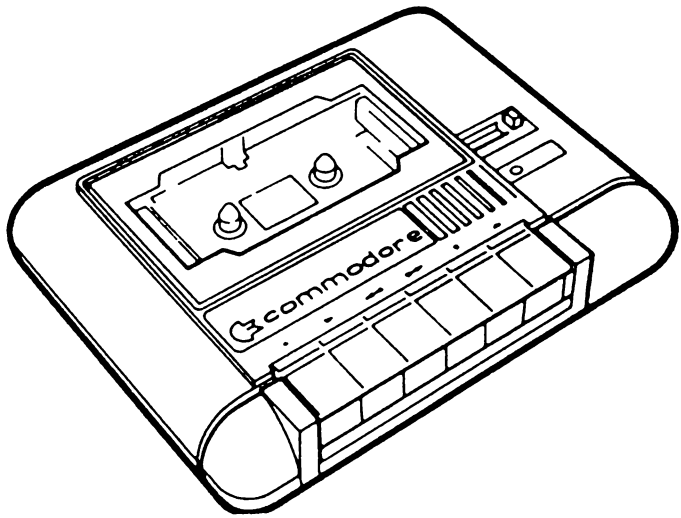
PART 1

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 an expression
7.	INPUT#	Read an expression
8.	GET#	Read in a single byte

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



2.1 SAVE

This instruction causes the computer to store the program it currently has in its memory. In order to be able to tell in which part of its memory the program is held, your computer reserves a few memory locations in which the beginning and end of each program is noted.

Immediately after being switched on, your computer sets up a sort of 'notebook'. This consists of four pages (0-3) and is located right at the beginning of the memory area. Under the control of the computer's own storage management facility, 256 storage locations are grouped together to form each page, as each storage location or each byte can have exactly 256 (0-255) different values. On page 0, i.e. from 0-255, the computer makes a note of the values it often needs.

STRUCTURE OF BASIC RAM AND POINTERS

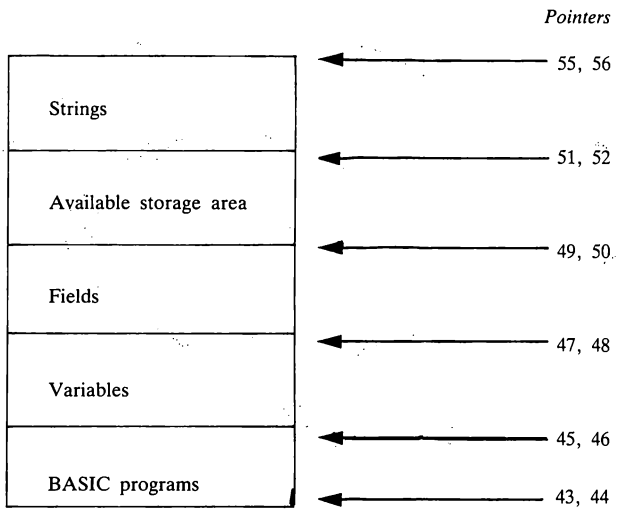


Fig. 1: General diagrammatic structure of BASIC RAM and pointers

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

Just check that out. 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 structure diagram above.

As the computer has 65, 536 storage locations, addresses are always expressed in two bytes. The first of these, the LSB, or least significant byte, 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 storage location, i.e. storage 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: In order that the computer can tell where the end of a Basic program is, it always adds a couple of noughts 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 noughts. You can easily check this for yourself. 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 if you now read out the contents of storage locations 43, 44, 45, and 46, you will get

1 8 17 8

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

After this little excursion into storage management, let's get back to the SAVE instruction.

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

However, in order that you find the program again when you have stored it on tape, you can give it a name. The syntax for this is as follows:

```
SAVE"NAME"
```

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

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

This facility can prove useful if you want to use this particular instruction in a program. But more of 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 it should select. It serves 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.

However, to save you a bit of work, the computer 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 stored 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, too. The rules are the same in both cases.

Following every LOAD instruction in direct mode, the computer automatically updates its notebook so that it corresponds to the program just loaded. This is important when you are loading machine language program so for example, if you are

loading a maching language program which is located at the end of the Basic storage area, then the Basic end vector 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 do want to make any further entries afterwards, you must either store the old value POKEn in pointers 45 and 46, or you must enter a NEW instruction so that the program in the Basic storage will be deleted, but not the machine language program.

One can get round the NEW instruction by loading the machine 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 written into storage, but compared, byte for byte, with the program in storage. In this way, one can test whether a program has been successfully stored.

If storage 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. However, one can also load and store data. In order to tell the computer that it is data which is to be loaded or stored, one must first open a file using the OPEN instruction.

With this instruction, one enters all the parameters the computer needs, namely, the device address, the file name and the secondary address, which indicates whether it is a loading or storing operation. In order 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 it should work with.

The various codes used are as follows:

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

LF stands for logical file number:

This number, which may be anywhere between 0 and 255, represents the index for this OPEN instruction. In order not to have to enter the parameters afresh with every input and output instruction, the computer sets up a table for itself of all the parameters, storing them under this index 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, first 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 A"DATA TO INTERMEDIATE STORAGE"
 80 PRINT:PRINT"FURTHER DATA"
 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 an empty 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 contain the name of the file. 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. That the string is not immediately written onto tape will be apparent from the fact that the tape has not moved.

Only when you have entered a whole series of data will the tape move, and then everything you have entered will be written onto the tape. I shall 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 overwritten with "space" (CHR$(32)), ready to receive more data.

When you conclude the input, 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 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 again, otherwise the 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 an STR instruction automatically with each PRINT# instruction. It is the same thing 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, if you wish to store several expressions with a single instruction. But I shall return to this particular problem after I have dealt with the INPUT# instruction, when it will be easier to understand.

2.6 INPUT#

The behaviour of this instruction is completely analogous to that of the 'normal' INPUT instruction, except that, in this case, it is not the keyboard which is set up to receive. If, for example, you define the keyboard as the transmitting 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 a little program. So wind 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 A"DATA READ IN"
 80 PRINT:PRINT"FURTHER DATA"
 90 GET F$: IF F$ = "" THEN 90
100 IF F$ = "Y" THEN 40
110 CLOSE1
120 PRINT:PRINT"FILE CLOSED"
130 END
```

If you now 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 looked for the file "DATA FILE" and then, on

receiving the INPUT# instruction in line 40, read in the first data block and transferred the first string of the variable A$, which was 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 in when storing 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 them onto the screen. The INPUT# instruction recognizes, from the comma, or 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	31
Content	S	T	R	1	CR	S	T	R	2	CR	S	T		R	5	CR	00	

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

You will get this message because you have read in the last string in storage 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 storage locations (you can read in a maxmimum 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 puts out an error message.

However, in order to prevent you getting this error message with every 'normal' file program, there is one way of recognizing the end of the file with the aid of the status variable. You will find out how to program it in Chapter 4.

However, you will get this error message, not only if you try to carry on reading 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 was automatically added onto the end of each one. However, it is possible to store several expressions with a single PRINT instruction. It is important to ensure in this case that the individual expressions are separated from one 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, namely,

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

However, 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	16	17
Content	S	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 just 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 it is saying the same thing, but in fact it is not, so I shall go into a bit more detail about it.

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

1. Department 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.

2. Department Storage 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 is not bothered whether it finds a comma or a CR. It 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 the CR as the starting point of a new expression.

The whole thing will become much clearer if we have a little program to work with. Enter the following statements, insert a 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, wind your cassette back to the start of the file and start the program with RUN100. When the computer has read in the file, then the strings will be arranged in the buffer in the following way:

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	33	34	35	
I	N	G	3	CR	S	T	R	I	N	G	4	CR	00				

On the screen, you will get:

```
STRING1
STRING4
```

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

Now change lines 110 and 120 by adding ', B$, C$'. If you now read the file in again, you will get 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 in the same format as it was stored in. However, if you wish to free yourself from this format when reading it in, then the separation string should be defined with CR.

```
20    T$=CHR$(13)
```

With data stored in this way, you will always get 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, however, it may make 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 read in a little while ago, 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 RUN100, you will get:

```
STRING1STRING2STRING3STRING4
```

However, if this leads to strings which are longer than 80 characters, then these cannot be read in 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. It is used to read in byte by byte. First enter the following program:

```
10   OPEN1, 1, 0, "TEST"
20   GET#1, A$
30   PRINTA$;
40   GETA$: IFA$"" THEN 40
50   GOTO 20
```

Now wind back 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 up with "SPACE". The characters can be made visible with the following program:

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

In this way, you can have every character printed out as a letter and as an ASCII value. If the GET$ instructions stumbles across a zero byte, it treats it as an empty string. In order not to trigger an error message on converting to ASCII code, CHR$(0) is always added on to the A$.

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

```
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 should like to make a few fundamental remarks on the two different types of program. On the one hand, there 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 storage area will depend on how the computer is expanded. In the basic version, this storage area starts at storage location 1025. If the storage area is expanded by 8k bytes, for example, then the Basic storage area will commence at storage location 1609. But in spite of that, programs written for the basic version will run on the expanded version.

In order to ensure that these programs will always be loaded at the start of the Basic storage area, both the VIC 20 and C-64 have a so-called relative loader, which

automatically reads all Basic programs in at the start of the Basic storage area.

The other type of program is the machine program. This type of program runs only in the storage area for which it was written. Relative loading is not permitted.

In order to be able to distinguish between Basic and machine language programs, the secondary address is used in cassette operations.

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 storage area available at the time, i.e. relatively loaded, by the LOAD instruction.

1—It identifies a program as a machine code 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 transferred to storage.

2—It stores a program as a Basic program, in the same way as secondary address 0. In addition, the computer writes a so-called EOT (End Of Tape) block after the program. This block tells the computer when reading, 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, then it puts out a message "?FILE NOT FOUND ERROR".

3—It identifies a program as a machine code program and writes an EOT block after it.

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

II LOAD

0—The program is loaded in accordance with the information in the program header, i.e. programs stored as macine code programs are loaded as such. The same applies to Basic programs, which are loaded relatively.

1—Every program, irrespective of identification, is loaded absolutely, i.e. at the address from which it was transferred to storage.

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 certain hard and fast variable with your Commodore which provides you with information about the progress of a cassette operation. This so-called status variable ST is set each time a cassette or other peripheral device is accessed. It therefore enables you to specify more precisely the nature of any fault that may occur. It will also help you to determine when a file has come to an end.

This variable consists of eight individual "flags", which are set according to the faults 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", available from First Publishing.

The precise meaning of each flag can be seen from Table 1.

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

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 total error. Check total in storage does not agree with computed check total

6	64	End of file
7	128	End of tape. EOT read

Well now, how can you use this variable in your programs? As you can see from the table, bit 6 tells you whether you have come to the end of a file. You can therefore check during the course of a program and stop reading in after the last string. Now enter the following program and use it to 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) = 64 THEN60
50   GOTO20
60   CLOSE1
70   END
```

4.1 Saving Programs after 'LOAD ERROR'

No doubt it has some time happened to you that your computer has thrown up a "?LOAD ERROR" after loading, while the program has been only partially read in, or not at all.

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. However, there is no need to despair. 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 onto tape twice in succession. On reading, the first version is loaded into storage and then compared with the second version. If the computer discovers any discrepancy or fault, it first tries to correct it. If that is not possible, it sets bit 4 of the status variable and gives out a "?LOAD ERROR".

It could be that the computer is unable to read individual bits or byte identif ication which has resulted in the synchronisation between the reading routine and the information read getting out of step. If this occurs, then bits 2 and/or 3 are set.

Every error which occurs means that 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 the program wholly or at least in part. This portion that you have been able to list can be saved by means of a UNNEW instruction.

This is an instruction that 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 a MONITOR program, then enter it with the Basic loader and store it. Then start the loader. You can now store the machine program from the indicated start address to the end of the Basic RAM (see Chapter 6). You can now load the stored machine program, whenever needed, simply by means of the LOAD instruction with SYS (start address), thus enabling you to call a deleted program back to life or to rescue at least part of a program that could not be loaded without errors.

Now that you have set the program and vectors with the UNNEW instruction, store 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 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 always first try to set the head better and if necessary repeat the procedure described above with different settings of the head until you can load the program with the minimum possible number of faults.

```
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
```

PROFI-ASS 64 V20 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      ;SEEK ZERO BYTE
260:  700A C8     ZERO  INY
270:  700B B1 22        LDA   ($22), Y
280:  700D D0 FB        BNE   ZERO
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      ;COUPLING ADDRESS OF
                                        FIRST LINE
320:  7016 91 2B        STA   ($2B), Y;WRITE COUPLING
                                        ADDRESS TO
330:  7018 A5 23        LDA   $22 + 1 ;BASIC START
340:  701A 69 00        ADC   #0
```

```
350:   701C CB          INY
360:   701D 91 2B       STA   ($2B), Y
370:   701F 20 33 A5    JSR   TIE     ;RECALCULATE COUPLING
                                        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 STORAGE UNDER PROGRAM CONTROL

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

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 have itself put into storage. You will probably use it for this only on rare occasions. You will use it much more frequently for storing only a part of a Basic program, or a whole machine code program.

Imagine that you have written a program which generates data lines from line 10000 onwards, and that you want to store this part of the program alone, in order to be able to add it on to other programs. You must first of all determine the storage location of line 10000 in storage, since it is only from here that you wish to start storing. The problem can be solved with the following bit of programming:

```
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
```

In order to understand these lines of programming, a little explanation is required as to how Basic lines are stored.

The first two bytes of a Basic line form the coupling address. This gives the starting address of the next Basic line, i.e. points to the next coupling 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:

CAL CAH LNL LNH Basic text 00 CAL CAH LNL LNH

where

CAL = coupling address LSB
CAH = coupling address LSB
LNL = line number LSB
LNH = line number MSB

In our little 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 program. If not, then line 1030 causes the starting location of the next Basic line to be transferred from the coupling address to S, and control returns to 1010 via 1040.

Using this program, you will have all the information required to enable you to store the last part of your program. There is only one important thing still to attend to, namely, on completion of the storing process, the old value must be written back to the Basic start vector. The data lines can be stored with the following section of programming:

```
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 your wish to store machine language programs, the whole thing 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 a program to what it does in direct mode. In Chapter 2 I explained that the Basic pointer is 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.

However, there are a number of things to watch with this technique:

1. As the Basic vectors are not changed, the calling program must be at least as long as the program called up. The variables are stored directly after a Basic program and vector 45, 46 is thus also the pointer for the start of the variable table. However, if the program called up 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 trick which will enable you to call up a longer program from a short one.

   First load the longest program to be called up in direct mode and determine its length by reading out storage 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 previously noted.

   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 to see that, after a LOAD instruction, the program is executed right from the beginning again.

This only makes sense and does not cause any problems if you are calling up Basic programs. It you wish to call up a machine code program, the Basic program in storage does not change and will start again repeatedly. If that is to be prevented, then you will have to use a branch table. In the following program, three machine code programs are loaded at the beginning of the program, before reaching the main program. program.

```
10  REM START
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 these things to watch out for is a peculiarity of string storage.

   Your Commodore operating system is designed so that as little space 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 again to the string table, but the pointer is set to the position of the string in the program.

   If you call up a program, something quite different will be found at the corresponding location and on receiving a PRINT A$ instruction, the computer will print out a load of rubbish. 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 write:

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

   In this way, you will cause the operating system to 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 storage areas to a storage device. Such storage areas could be, for example, the contents of a screen display, or a machine code program. Unfortunately, Commodore Basic does not support this operation at all. However, as you have seen in Chapter 5, it can be done, with the aid of a few tricks.

In the last chapter, I mentioned that, with a SAVE instruction, the computer

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

If the computer is to operate after storing in the same way as it did before, the old values must be stored in the vectors. In short, you will have to save these values beforehand and transfer them back to pointers 43 46 when the storage operation is complete.

In order 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, as explained in previous chapters. As soon as 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 storage locations which are not affected by the storage operation or the operating system.

In the following example, I shall be using the storage locations 251-254. These locations are not used by the operating system. These lines 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 BACK START
70  POKE45, PEEK(253): POKE46, PEEK(254):REM BACK END
```

As you can see, the whole thing is quite complicated. It is much easier if you use a little machine code 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

This is sufficient to store the area without having to manipulate the vectors in any way.

The Basic loaders shown store the little machine code program at the end of the Basic RAM and print out the start address with which it can be called up. But before you start the loader, you should also transfer it to storage, since it automatically clears itself.

I should like at this point to explain a few things to do with the machine code programs.

In line 310, an operating system routine, which reads the name of the screen and writes it into the appropriate storage locations, is called up. The routine which is called up 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 up which sets the file parameters. This routine must be called up before each load and storage operation.

In lines 350 380, a check is made for a comma, in order to read in and process the start address. This is filed away in storage 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.

```
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, 32, 186, 255, 32, 14, 226, 32, 138,
173, 32, 247
160  DATA 183, 166, 20, 164, 21, 134, 251, 132, 252, 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-256H*
140  PRINT"START ADDRESS"A:NEW
150  DATA 32, 84, 226, 32, 253, 225, 160, 1, 169, 0, 32, 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
```

PROFI-ASS 64 V2.0 Page 1

```
 80:  C000                    .PAG 61
100:  C000                    .OPT P1, 0
102:  C000
102:  C000
103:          ;                    SAVE ADDRESS
104:          ;
105:          ;                    CALL 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          =    $AD8A     ;($CD8A)
180:  B7F7    FACINT          =    $B7F7     ;($D7F7) FAC TO $14, $15
190:  FFD8    SAVE            =    $FFD8
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
350:   C00A 20 BA FF      JSR  SETLFS     ; SET FILE PARAMETER
360:   C00D 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 storage area from 828 (hexadecimal 033C) to 1019 (hexadecimal 03FB). This buffer has two basic functions. On the one hand, the file header is generated in this area and then stored before each file, irrespective of whether it is a program or data file. It 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 SPACE.

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

The file type is coded in the following manner:

1. = Basic program, loaded relatively
2. = Data block
3. = Machine code program, loaded absolutely
4. = Data header
5. = EOT block..The header contains all the important data required for purposes of identification. What could be more obvious, than that these data should be used to catalogue the contents of cassettes on which programs and data are stored.

7.1 Creating a List of Cassette Contents

The program at the end of 7.1 will print out a list of the contents of a cassette. If you have a printer, you can print it out on this

PROGRAM DESCRIPTION

Lines 100-160 are intended for initialisation. The individual tabulator stops are read into TX$ and the various file types into field TN$.

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 without a name. This causes the next header to be read in.

In lines 240-270, the type byte, start and end address and file name are read from the buffer and formatted by the PEEK. 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.

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.

```
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$"K-BYTE "T5$"
      START"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 COMPUTER TAPE
    COUNTER
340 CLOSE 1
350 GOTO 230
```

7.2 Display of Files Found

The ability to read out the name of a file by means of a PEEK instruction during the course of a file management program can also be useful. Have you never 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 wound the tape on too far.

However, if the computer lets you know what file it has found, then you can quickly tell whether you had wound on 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:GOTO120
170  PRINT:PRINT"FILE BEING LOADED"
200  POKE 198, 0:WAIT 198, 1:RETURN
```

The OPEN instruction in line 130 without a name causes the computer to load every data header into the buffer.

In lines 140-160, the file name is transferred from the buffer to FE$ by means of the PEEK and displayed in line 170. If comparison of FE$ and the required file name in line 180 produces a negative result, the search is continued. Otherwise, the file is loaded.

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 storage 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 program.

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

Such interludes can also be used to protect your programs. With such an interlude, one could relocate the Basic RAM before loading the main program, in order to file a few pages of graphics before the main program. 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 the niceties of programming.

This is a description of the program:

In line 10, the file name under which the main 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 "SPACE". 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 interlude.

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

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 input holding loop is changed over to the machine program in the cassette buffer.

Line 170 then stores this vector with the aid of the machine code program described in the previous chapter. You must therefore write the start address of your SAVE ADDRESS program after the SYS instruction.

This storage instruction causes the file name together with the machine code program and Basic instructions to be written into the 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 code 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 interlude.

If you have written 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 it 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 input holding loop vector to point to one's own machine code program, and storage of these changed vectors. Before the computer jumps to the loaded program, the vector pointing to one's own machine code program must be reset to its old value.

3 Storage areas stored as machine code programs are always loaded at the point from which they were transferred to storage.

In storage 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 input holding loop is the subroutine which waits patiently for your input. Whenever input is presented, the input holding loop evaluates and executes it before returning to the loop again.

Our program generates an interlude which overwrites the input holding loop vector with the start address of your own program, written into storage with the file name.

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

But let us now take a look at the machine code program. The vector for the input holding loop is reinstated in lines 200-230. Your input is stored in storage locations commencing $036C. This input is transferred to the keyboard buffer in lines 250-290. The jump to the input holding 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, 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 code program, and concluding it with a jump to the input holding 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.

However, 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.

ASSEMBLER 64 V2.0 PAGE 1

```
110:  0351          .OPT  P1, O
120:  0351     *=   $0351
130:  0351
130:  0351
140:                ;LOADER PROGRAM
150:  0351
150:  0351
160:  A483     VECTOR        =    $A483      ; ($C483)VECTOR
165:                ;                         TO INPUT HOLDING LOOP
170:  00C6     PANZ          =    $C6
180:  0277          BUFFER   =    $277       ; KEYBOARD BUFFER
185:  036C          TABLE    =    $36C       ; TABLE OF BASIC
                                                 INSTRUCTIONS
190:  0351
190:  0351
200:  0351 A9 83              LDA #<VECTOR ; SETTING OF OLD
210:  0353 8D 02 03           STA  $302    ; 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 TABLE      ;NUMBER OF CHARAC-
                                               TERS
250:   035E 86 C6             STX PANZ
260:   0360 BD 6C 03    LOOPLDA 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 INPUT
297:                    ; HOLDING LOOP
299:                    ;
300:   0369 4C 83 A4                         JMP VECTOR
U0351-036C
```

```
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.
**********************************************************************
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 synchronisation is required between tape and counter. For this reason, each block is preceded by synchronisation and a countdown.

A block is constructed as follows:

1. Synchronisation bytes and countdown
   This part is responsible for the synchronisation of the read components on the tape. The synchronisation also contains information as to what sort of a block it is.

2. Data

3. Check total byte
   During storage and loading, the computer forms an EXOR check total 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 total 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 total byte (as 3)

6. End of block mark

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 link up a number of programs in storage.

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 storage and then 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 storage locations 43, 44.

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

The whole thing can be performed more elegantly with a machine 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 storage.

PROGRAM DESCRIPTION

The loading vector is changed to this program by the initialisation 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.

ASSEMBLER 64 V2.0 PAGE 1

```
 25:  C000                      *=   $C000
 30:  C000                      .0PT P1, 0
 32:  C000
 32:  C000
 33:        ;******** MERGE ********
 34:  C000
 34:  C000                           ;VALUE IN BRACKETS APPLIES TO
 35:              ;                  VIC 20
 36:  C000
 36:  C000
 40:  0330    VECTOR        =    $330        ;LOADING VECTOR
 50:  F4A5     LOAD         =    $F4A5       ;($F549) LOAD ROUTINE
 60:  00BB    FILEADDR      =    $BB         ;POINTER TO FILE NAME
 70:  00B7    FILELNG            $B7         ;FILE NAME LENGTH
 80:  00C3    STARTADDR     =    $C3         ;POINTER TO PROG.START
 90:  002D    PROGR END     =    $2D         ;POINTER TO PROG.END
 92:  C000
 92:  C000
 93:  C000
 93:  C000
 95:              ;INITIALISATION 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:  C00B
150:  C00B 48     START       PHA
160:  C00C A2 00          LDX #0
170:  C00E A1 BB          LDA (FILEADDR, 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 PROGEND
210:  C017 E9 02      SBC #2                ;REDUCE POINTER BY 2
220:  C019 85 C3      STA STARTADDR         ;IN LOAD START ADDR.
230:  C01B A5 2E      LDA PROG END+1        LDA PROG.END+1
240:  C01D E9 00      SBC #0
250:  C01F 85 C4      STA START ADDR+1
260:  C021 3 18       CLC
270:  C022 E6 BB      INC FILEADDR          ;DELETE '@'
280:  C024 C6 B7      DEC FILELNG
290:  C026 68    END  PLA
300:  C027 4C A5 F4   JMP LOAD
]C000-C02A
```

```
10 REM MERGE C64
*************************************************************************

  20  E=256*PEEK(56)+PEEK(55)-1:A=E-41
  30  FOR I=A TO E:READ X:POKE I, X:NEXT
  z40  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 MERGE 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, 161, 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 storage processes with the Datassette. We have seen that the tape can be started and stopped automatically. The computer has the ability to control the motor, so, assuming that we can put this to good use, I shall now describe how to exploit this facility in your programs.

Using the following storage 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 draws up a catalogue 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 on from the end of the cataloguing program to the program to be catalogued, 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 on 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 cataloguing 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 adjust the fast forward wind 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 cataloguing program at the beginning of the cassette again.

You can now select a program in comfort from those shown in the catalogue 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 initialised in lines 180 to 340 and the menu is written on the screen. During initialisation, 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 colour 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.

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 program smaller than the cataloguing 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.

```
 10  GOTO 190
 20  ************************************************
 30  *                                              *
 40                      CATALOGUE                   *
 50  *                                              *
 60  *                    FOR CASSETTE              *
 70  *                                              *
 80  ************************************************
 90  :REM
************************************************************
100  REM              HOLDING LOOP
************************************************
110  POKE 198, 0:WAIT 198, 1:GET A$:RETURN
120  POKE 192, 0:POKE 1, PEEK(1)AND 223:REM MOTOR ON C-64
130  :REM POKE 37148, 252:REM MOTOR ON VIC20
140  RETURN
150  POKE 192, 1:POKE 1, PEEK(1)OR 32:REM MOTOR OFF C-64
160  :REM POKE 37148, 0:REM MOTOR OFF VIC20
170  RETURN
************************************************
180  REM "              INITIALISATION
************************************************
190  HD$="*C A T A L O G U E*"
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]NEW INPUT":PRINT
320  PRINT TAB(SP/2-6)"[RVS]L[RVOFF]LOAD"
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";:IF SP(40 THENPRINT: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"
     :PRINT"[DOWN, RVS]STOP[RVOFF, SPACE]KEY"
440  PRINT"[DOWN]ON THE RECORDER":WAIT WZ, WW, 255-WW
450  PRINT"[CLEAR]PRESS THE ";:IF SP(40 THEN PRINT
     :PRINT
460  PRINT"[RVS]F.FWD[RVOFF, SPACE]KEY"
470  PRINT:PRINT"START THE RECORDER";
     :IF SP(40 THEN PRINT:PRINT:PRINT" ";
480  PRINT"[LEFT]BY PRESSING"
490  PRINT:PRINT"A KEY AND ";:IF SP(40 THEN PRINT:PRINT
500  PRINT"STOP AT THE "
510  PRINT:PRINT"START OF REQUIRED";
     :IF SP(40 THEN PRINT: PRINT:PRINT" ";
520  PRINT"[LEFT]FILE IN SAME":PRINT"WAY."
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$(A)
570  GOSUB 110:GOSUB 150:DI=TI-R+DI
     :REM CC OFF, COMPUTE TIME DIFFERENCE
580  ZS(A)=DI-10:DI=DI+10:A=A+1
590  IF A(ZE THEN 540:REM LAST FILE NAME?
*****************************************************************
600  REM * STORAGE IN DATA LINES **
*****************************************************************
610  PRINT"[CLEAR, DOWN3]2000DATA"A
     :REM NUMBER IN DATA LINE
.620  FOR I=ZA TO ZE-1:PRINT2100+I"DATA"N$(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 WIND BACK"
     :PRINT"[DOWN]AND STORE"
     :PRINT"[DOWN] THE PROGRAM AT THE"
660  PRINT:PRINT"BEGINNING":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 = UP."
     :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
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 "
      :PRINT"[DOWN, SPACE6]PLAY "
 980  PRINT:PRINT"KEY"
 990  PRINT"[DOWN4]":PRINT TAB(SP/2-6)"[RVS, SPACE2]S P A C E
        [RVOFF]":
      GOSUB 110
1000  PRINT"[CLEAR DOWN3] L[SHIFT 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 Datassette

Like every other cassette recorder, the datassette requires a certain amount of care and attention. The frequence occurrence of a "?LOAD ERROR" is usually the result of a dirty, badly adjusted or magnetised 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 (read/write head and delete head) with the swab sticks dipped in the cleaning agent by rubbing the front face of the heads until the swab no longer turns brown and following up with a dry swab.

   You are 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. Demagnetisation
   For demagnetisation, you should use a commercial demagnetiser or a demagnetising 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 anything like 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 magnetised, 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 in such a way that they are protected against dust. Likewise, they should not be stored for too 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.

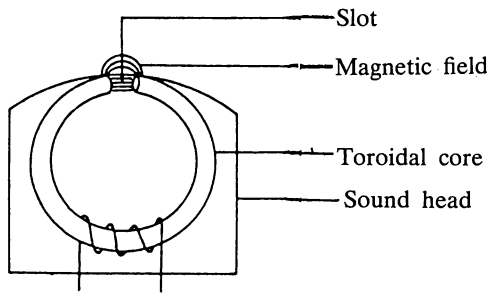
11.3 Datassette Operation

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 port bit 3 of the C-64 processor (storage location 1). In the case of the VIC 20, it is bit 3 of storage 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:

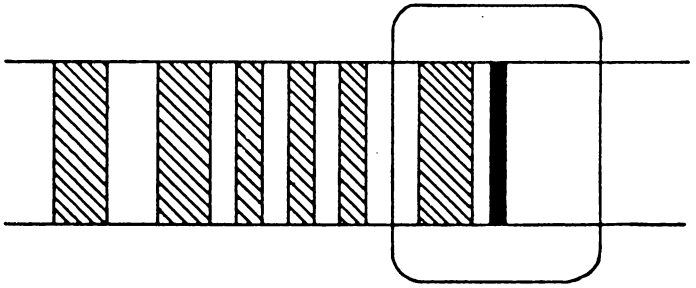


By way of a number of electronic stages which I will not explain in greater detail here, the signal arrives at the sound head. This head consists basically of a coil with a toroidal core and a small slot on the side facing the tape.



Depending on the voltage applied to the coil contacts, a magnetic field is created

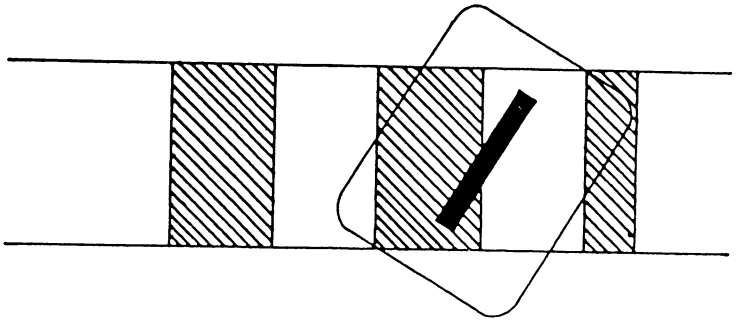
at the slot which aligns the magnetic particles on the tape in a certain direction. The idealized magnetization pattern on the tape would then look like this:



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 storage 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 storage, we get the following picture:



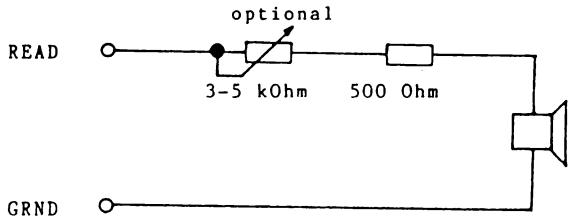
In the top part of the slot, the direction of magnetization is different to what it is in the bottom part. The two effects erase one another, with the result that the data is not read in correctly.

11.4 A Loudspeaker for the Datassette

No doubt you have occasionally been annoyed that, apart from the none-too-reliable 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 on 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 up 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 up to the READ line. In order not to overload the connection, a resistor of approx. 500 0hm would have to be connected in series. You could also couple up a potentiometer of 2-5 K0hm in series as well, in order to control the volume.



Connection of the loudspeaker

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 within 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 again and insert it into the cassette port on the computer.

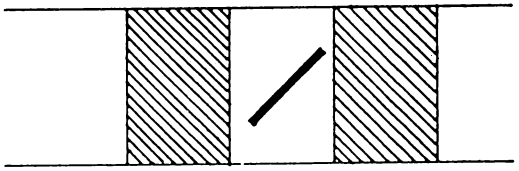
Switch on the computer, place a used cassette in the recorder and press the PLAY key. You will now hear either a whistling sound, all on one note, or a chirping sound.

The whistling is associated with the synchronisation and characterises 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.

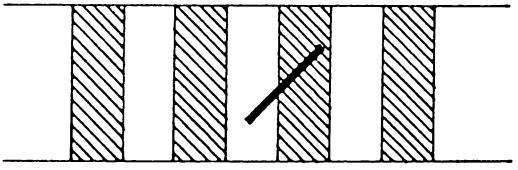
This misalignment can be heard quite clearly with the aid of a loudspeaker. Even if the head is only at a slight angle to the tape, this can be detected straight away from the high frequencies. I should like to illustrate that with a couple of drawings:



LOW FREQUENCY

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.



HIGH FREQUENCY

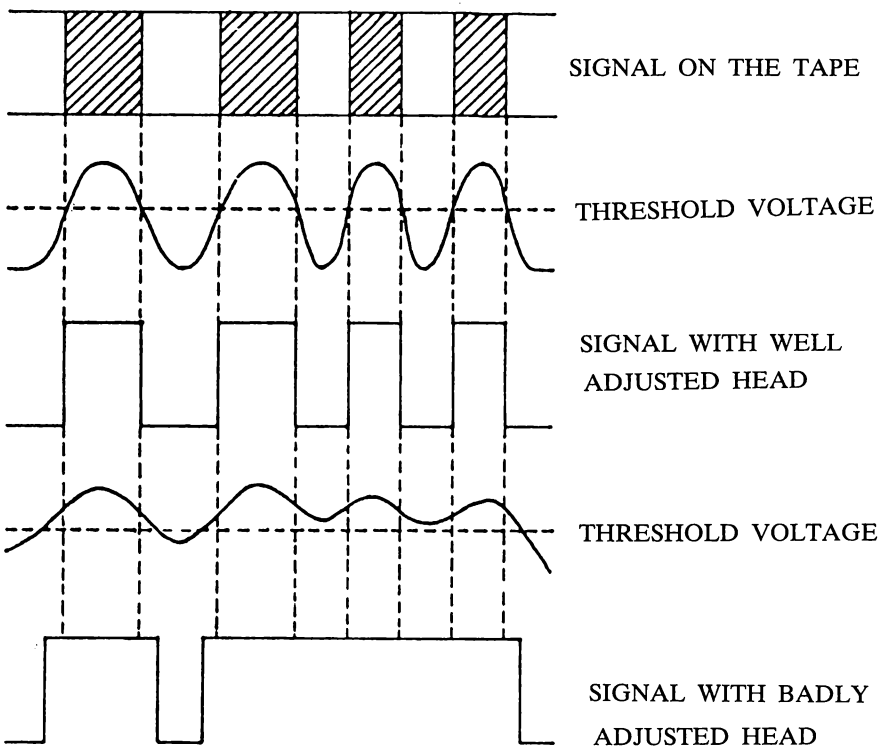


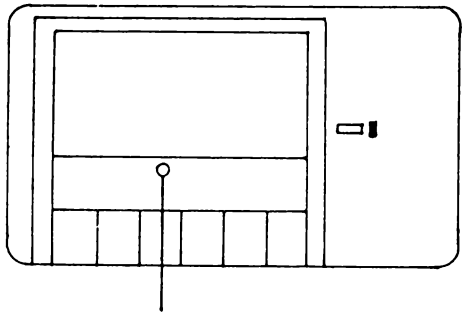
Fig. 2 Output signals in various positions of the head

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.

After this brief theoretical introduction, let us look at the practical side of things.

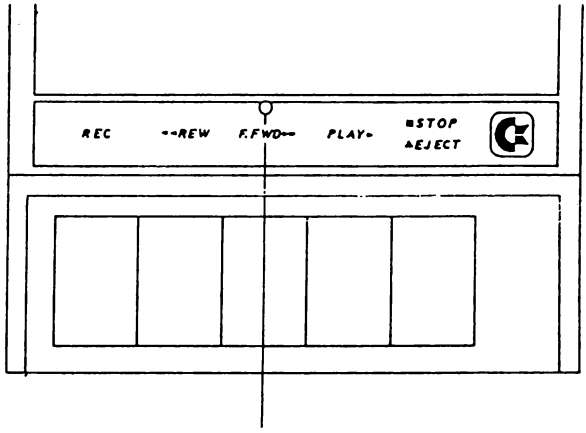
If you have the latest version of the datassette, you will find a small hole approximately in the centre of the cassette compartment.



:ACCESS TO SOUND HEAD ADJUSTER SCREW

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.



DRILL SMALL HOLE HERE

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 store it. Then record the chirping generated by this program on a clean tape for about two minutes.

```
 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 most high 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.

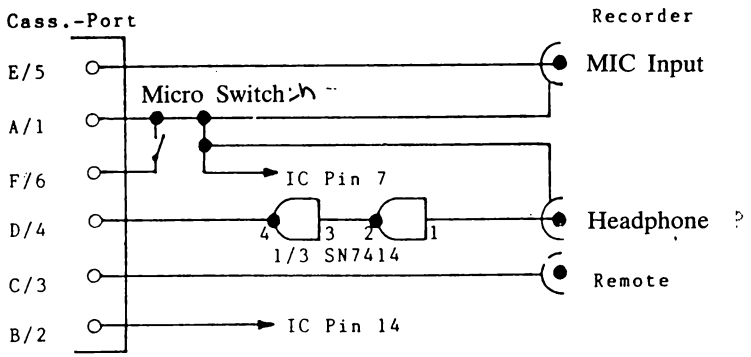
For the sake of completeness, I should like to show you a little circuit which you can make for yourself. The components you need are listed below.

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

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 uses 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.

PART 2

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 better 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 storing 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 possible with *one* instruction
4. Loading into or storage 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 when included in programs.

Operating the Program

When the program has been loaded, it is initialised with SYS7*4096 in the case of the VIC 20, and SYS12*4096 in the case of the C-64. The program is automatically initialised by the Basic loader. After a RESET, however, the program must be initialised afresh. Each Fast Tape 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 in 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 storage
S"name", 1, Sadr, Eadr	The storage 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 storage area, stored with S, is loaded either absolutely or relocatably, depending on the SA given with S. If this instruction is given indirect mode, then the Basic vectors are set according to the program loaded.
L"name", 1	Programs and storage 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 storage area is loaded at Sadr, irrespective of the address from which it was transferred to storage. 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 storage with the bytes on the tape.

M ("NAME") FASTTAPE MERGE

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

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 code 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 storage location $C000 onwards by means of the Basic loader FT64.

The Basic loader FT20 stores the program in the VIC 20, from storage location $7000 onwards, i.e. the program will only work on a VIC 20 if it is fully extended. It is, however, possible to re-locate the Fast Tape program into another storage area with the aid of a MONITOR program and the ASSEMBLER listing.

First Publishers Margaret

ASSEMBLER 64 V2.0 PAGE 1

```
 82:  C000                        .PAG 61
 90:  C000                        .OPT P2
 92:  C000
 92:  C000
 94:        ; FASTTAPE FOR VC20 AND C-64
 95:  C000
 95:  C000
 95:  C000
 96:        ; VALUES IN BRACKETS APPLY TO
 97:  C000
 97:  C000
 98:        ;                     VC 20
100:  C000                 *-    $C000
110:  0000  COMPU          =     $00       ; (2)
120:  0001  PORT           =     $1        ; (*9120)
130:  0002  ABSFLS         =     $2        ;
140:  00C0  MOFLAG         =     $C0
150:  00A3  BITC           =     $A3
153:  0004  BENFANZ        =     4
155:  005F  CODE           =     "←"
157:  0073  CHRGET         =     $73
160:  030B  VECTOR         =     $308      ; FETCH INSTR. ADDRESS
165:  02A0  IRQFLG         =     $2A0      ; IRQ FLAG
170:  0090  STATUS         =     $90       ; STATUSBYTE
175:  00C3  STARTV         =     $C3       ; LOAD START VECTOR
180:  00AE  ENDVEC         =     $AE       ; LOAD END VECTOR
185   00D7  PSUMME         =     $D7       ; CHECK TOTAL BYTE
186:  033C  CASPUF         =     $033C     ; START ADDRESS OF CASS.
                                             BUFFER
187:  C000
187:  DD00  CIA1           =     $DD00     ; C-64 = START ADDR. OF CIA1
188:  DC00  CIA2           =     $DC00     ; C-64 = START ADDR. 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) CHECKS STOP KEY
220:  FC93  ENDEN       =  $FC93   ; ($FCCF) TERMINATE LOADING
230:  E17A  BEFEND      =  $E17A   ; ($E177) BRANCH BACK FROM
                                     LOADING
240:  F750  FOUND       =  $F750   ; ($F7D3) OUTPUTS FOUND
                                     'NAME'
260:  F5A9  STOEND      =  $F5A9   ; ($F461) END ADDR. AFTER X/Y
300:  A7E7  OLD         =  $A7E7   ; ($C7E7) EVALUATE INSTRUC-
                                     TION
310:  A7AE  INTER       =  $A7AE   ; ($A7AE) INTERPRETING LOOP
320:  F838  RPTASTE     =  $F838   ; ($F8B7) WAITS FOR PLAY KEY
330:  F817  PTASTE      =  $F817   ; ($F894) WAITS FOR KEY
335:  B526  GABCOL      =  $B526   ; ($D526) GARBAGE COLLECTION
340:  E206  WZEICH      =  $E206   ; ($E203) PRINT FURTHER
                                     CHARACTERS
350:  E257  HFNAM       =  $E257   ; ($E254) COLLECT FILE NAMES
360:  E200  HOLPAR      =  $E200   ; ($E1FD) COLLECT FILE PARA-
                                     METERS
365:  E20E  CHRIN       =  $E20E   ; ($E20B) COLLECT CHARACTERS
370:  AF08  SYNTAX      =  $AF08   ; ($CFO8) PRINT OUT 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  FEHLAUS     =  $A437   ; ($C437) OUTPUT ERROR
410:  A677  BEENDEN     =  $A677   ; ($C677)
420:  C000
420:  C000
430:  FFB7  HOLSTA      =  $FFB7   ; COLLECT STATUS
440:  FFBD  SETNAM      =  $FFBD   ; SET FILE NAME PARAMETERS
450:  FFBA  SETLFS      =  $FFBA   ; SET FILE PARAMETERS
460:  FFE4  GET         =  $FFE4   ; READ IN A CHARACTER
500:  C000
500:  C000
510:  C000
510:  C000
520:  ;INITIALISE PROGRAM
```

```
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
602:  C00B
603:  C00B
603:  C00B
604:     ; PROGRAM START
606:  C00B
606:  C00B
610:  C00B 20 73 00 START    JSR  CHRGET    ; FETCH CHARACTER
620:  C00E F0 04             BEQ END
630:  C010 C9 5F             CMP #CODE      ; COMPARE WITH
640:  C012 F0 03             BEQ FTAPE      ; FASTTAPE CODE
650:  C014 4C E7 A7 END      JMP  OLD       ; UNEQUAL =⟩ OLD
                                              ROUTINE
654:  C017
654:  C017
655:              ; EVALUATE FASTTAPE INSTRUCTION

657:  C017
657:  C017
660:  C017 20 73 00 FTAPE    JSR  CHRGET
670:  C01A 20 20 C0          JSR  SUCH
680:  C01D 4C A3 A7          JMP  INTER
690:  C020 A2 00 SUCH        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    ; NO FASTTAPE
                                              INSTRUCTION
755:  C02F
755:  C02F
760:  C02F 8A BEFEHL         TXA
770:  C030 0A                ASL
```

```
 770:  C031 AA              TAX             ; BEFZHAL * 2
 780:  C032 BD 5A C0        LDA BEFADR+1, X; INSTRUCTION ADDRESS
                                              TO
 790:  C035 48              PHA             ; STACK
 800:  C036 BD 59 C0        LDA BEFADR, X
 800:  C039 48              PHA
 810:  C03A 4C 73 00        JMP  CHRGET     ; AND EXECUTE INSTRUC-
                                              TION
 815:  C03D
 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
 887:  C054
 890:  C054 53 4C 56 TAB    .ASC"SLVMD"
 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           .WORDMERGE1-1
 910:  C061 3C C0           .WORDDATA-1
 920:  C063
 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 PARAMETERS
1692:  C06E A5 02           LDA ABSFLG
1694:  C070 29 02           AND #2          ; TEST FOR BIT 2
1695:  C072 D0 0B           BNE ABSOLUT     ;SET=)LOAD ABSOLUTELY
1700:  C074 A2 04           LDX $04
1710:  C076 B5 2A LOOP1     LDA $2A, X      ; RE-STORES BASIC START
1720:  C078 95 AB           STA $AB, X
1725:  C07A 95 A6           STA $A6, X
1730:  C07C CA              DEX
1740:  C07D D0 F7           BNE LOOP1
1750:  C07F 20 38 F8 ABSOLUTJSR RPTASTE
1760:  C082 20 8F F6        JSR SAVING      ; OUTPUT 'SAVING NAME'
1770:  CO85 20 FC CO        JSR MOTOR       ; SWITCH ON MOTOR
1774:  CO88
1774:  CO88
1775:                ; WRITE LEADER
1776:  C088
1776:  C088
1780:  C088 20 13 C1        JSR SYNCH       ;WRITE SYNCHRONISATION
1790:  C08B A5 B9           LDA $B9         ;INCREMENT SECONDARY
                                              ADDRESS
1800:  C08D 18              CLC
1810:  C08E 69 01           ADC #$01
1820:  C090 CA              DEX
1830:  C091 20 33 C1        JSR WBYTE       ; WRITES SA
1840:  C094 A2 08           LDX #$08+COMPU
1850:  C096 B9 A7 00 LOOP2  LDA $A7, Y      ; WRITES START AND END
                                              ADDRESSES
1860:  C099 20 33 C1        JSR WBYTE
1870:  C09C A2 06           LDX #$06+COMPU
1880:  C09E CB              INY
1890:  C09F C0 05           CPY #$05
1900:  C0A1 EA              NOP
```

```
1910:  C0A2 D0 F2            BNE LOOP2
1920:  C0A4 A0 00            LDY #$00        ; WRITES FILE NAMES AND
                                               'SPACES'
1930:  C0A6 A2 04            LDX #$04+COMPU
1940:  C0AB B1 BB LOOP3      LDA ($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 #$05
2010:  C0B6 CB               INY
2020:  C0B7 C0 BB            CPY #$BB
2030:  C0B9 DO ED            BNE LOOP 3
2034:  C0BB
2034:  C0BB
2035:                ; WRITE SYNCHRONISATION 2
2036:  C0BB
2036:  COBB
2040:  COBB A9 02            LDA #$02
2050:  C0BD 85 AB            STA  $AB
2060:  C0BF 20 13 C1         JSR  SYNCH  ; WRITE SYNCHRONISATION
2070:  C0C2 98               TYA
2080:  C0C3 20 33 C1         JSR  WBYTE  ; 0BYTE =〉 END OF LEADER
2090:  C0C6 84 D7            STY  PSUMME  ; DELETE CHECK TOTAL BY
2100:  COC8 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 ONTO
                                               TAPE
2130:  C0CD 20 33 C1         JSR  WBYTE
2140:  C0D0 A2 03            LDX #$03+COMPU
2150:  C0D2 E6 AC            INC  $AC        ; INCREMENT PROGRAM
                                               POINTER
2160:  C0D4 D0 04            BNE NOHI
2170:  C0D6 E6 AD            INC  $AD
2180:  C0D8 CA               DEX
2190:  C0D9 CA               DEX
```

```
2200:  C0DA A5 AC NOHI    LDA $AC
2210:  C0DC C5 A7         CMP ENDVEC   ;PROGRAM END REACHED
2220:  C0DE A5 AD         LDA $AD
2230:  C0E0 E5 AF         SBC ENDVEC+1
2240:  C0E2 90 EU         BCC PRGLOOP  ; NO =⟩ CONTINUE
2250:  C0E4 EA            NOP
2260:  C0E5 A5 D7  LOOP4  LDA PSUMME   ; WRITE CHECK TOTAL
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 SWITCH OFF
                                         MOTOR
2330:  C0F2 58            CLI
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:  COFC
2375:  C0FC202CAB 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 REQUIRED WITH
2420:  C108 8D 11 D0      STA $D011      ; VC 20
2430:  C20B CA    ANLOOP  DEX ; WAITS TILL MOTOR HAS
2440:  C10C D0 FD         BNE ANLOOP     ; RUN UP
2450:  C10E 88            DEY
2460:  C10F D0 FA         BNE ANLOOP
2470:  C111 78            SEI
2480:  C112 60            RTS
2481:  C113
2481:  C113
2482: QWQWQWQW; GENERATE SYNCHRONISATION
2482:  C113
2484:  C113
2490:  C113 A0 00         SYNCH LDY#$00
```

```
2500:  C115 A9 02 LOOP5     LDA #$02        ; WRITE START BYTE 255-9
                                              TIMES
2510:  C117 20 33 C1        JSR  WBYTE
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
2500:  C125 DO 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
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 PSUMME
2690:  C137 85 D7          STA  PSUMME
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 TIME
2880:  C15C CA    T2LOOP     DEX ; LOOP
2890:  C15D DO FD            BNE T2LOOP
2900:  C15F 85 01    BIT0    STA PORT
2902:  C161 60               RTS
2903:  C162
2903:  C162
2904:  C162
2904:  C162
2905:                ; MERGE
2906:  C162
2906:  C162
2909:                ; MERGE
2910:  C162 A5 2D MERGE 1 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 MERGE2
2915:  C170
2915:  C170
2916:  C170
2916:  C170
2917:                ; LOAD
2918:  C170
2918:  C170
2919:  C170
2919:  C170
2920:  C170 A2 00 LOAD       LDX #00
2925:  C172 2C               .BYTE2C
2930:  C173 A2 01 VERIFY     LDX #$01
2940:  C175 A4 2B            LDY $2B        ;BASIC START VECTORS IN
2950:  C177 A5 2C            LDA $2C        ;START STORAGE LOCA-
                                             TIONS
```

```
2960:  C179 86 0A MERGE2  STX $0A          ;SET LOAD FLAG
2970:  C17B 86 93         STX $93          ; (0=LOAD/1=VERIFY)
2975:  C17D A2 00         LDX #0
2980:  C17F 86 02         STX ABSFLG       ; CLEAR ABSOLUTE FLAG
2985:  C181 84 C3         STY STARTV
2990:  C183 85 C4         STA STARTV+1
3000:  C185 20 B0 C2      JSR GETPARA
3010   C188 20 BE C1      JSR LOADR        ; LOAD AND VERIFY
3020:  C18B 4C 7A E1      JMP BEFEND
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
                                             ABSOLUTELY
3070:  C195 F0 08         BEQ ABSOL        ; =⟩ LOAD ABSOLUTELY
3080:  C197 C9 01         CMP #$01         ; BYTE NOT 1=⟩ CONTINUE
                                             SEARCH
3090:  C199 DO F3         BNE LOADR
3100:  C19B A5 B9         LDA $B9          ; SECONDARY ADDRESS=0
3110:  C19D F0 10         BEQ RELLOD; =⟩ LOAD RELOCATABLY
3112:  C19F A9 02 ABSOL   LDA #2           ; SET ABSOLUTE FLAG
3113:  C1A1 05 02         ORA ABSFLG
3114:  C1A3 85 02         STA ABSFLG
3120:  C1A5 AD 3C 03      LDA CASPUF       ;START ADDRESS FROM
                                             BUFFER
3130:  C1A8 85 C3         STA STARTV       ; IN START VECTOR
3140:  C1AA AD 3D 03      LDA CASPUF+1
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 #
3158: C1B6 DO 08          BNE NOWAIT
3160: C1B8 20 50 F7       JSR FOUND        ; OUTPUT FOUND 'NAME'
3170: C1BB 20 E4 FF WAIT  JSR GET          ; NOT REQUIRED WITH
3180: C1BE FO FB          BEQ WAIT         ; VC 20
3190: C1C0 20 2C AB NOWAITJSR STOP
3200: C1C3 A4 B7          LDY $B7          ; FILE NAME LENGTH
3120: C1C5 FO OB          BEQ NONAME
```

```
3220: C1C7 88 TESTNA      DEY             ;FILE NAME TESTS
3230: C1C8 B1 BB          LDA ($BB), Y
3240: C1CA D9 41 03       CMP CASPUF+5, Y
3250: C1CD DO BF          BNE LOADR
3260: C1CF 98             TYA
3270: C1D0 DO F5          BNE TESTNA
3280: C1D2 84 90 NONAME   STY STATUS    ; CLEAR STATUS
3290: C1D4: 20 D2 F5      JSR LOADING
3300: C1D7 AD 3E 03       LDA CASPUF+2  ; COMPUTE END
                                          ADDRESS FROM
3310: C1DA 38             SEC           ; DIFFERENT START+
                                          END ADDRESSES
3320: C1DB ED 3C 03       SBC CASPUF    ; 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 CASPUF+3
3380: C1E7 65 C4          ADC STARTV+1
3390: C1E9 28             PLP
3400: C1EA ED 3D 03       SBC CASPUF+1
3410: C1EC 85 AF          STA ENDVEC+1
3420: C1EF 20 1F C2       JSR PLOAD
3430: C1F2 A5 BD          LDA $BD
3440: C1F4 45 D7          E0R PSUMME    ; TEST CHECK TOTAL
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    OK    LDA ABSFLG    ; TEST FLAG
3484: C200 D0 03          BNE ALTADR    ; IF SET =〉 OLD END
                                          VECTORS
3486: C202 4C A9 F5       JMP STOEND    ; RECEIVED
3488: C205 A6 2D  ALTADR  LDX $2D
3490: C207 A4 2E          LDY $2E
3492: C209 60             RTS
3500: C20A 20 58 C2 SSYNCH JSR SCOUNT   ; SEEK SYNCHRONISA-
                                          TION
3510: C20D C9 00          CMP #$00      ; SYNCHR. BEFORE
                                          HEADER #〉
3520: C20F F0 F9          BEQ SSYNCH    ; SEEK SYNCHR. AFTER
                                          HEADER
```

```
3530: C211 85 AB              STA $AB        ;SECONDARY
                                               ADDRESS+1
3540: C213 20 86 C2 SPSTART   JSR HOLBYT     ; SEEK START OF
3550: C216 91 B2              STA ($B2), Y   ; PROGRAM
3560: C218 C8                 INY
3570: C219 C0 C0              CPY #$C0
3580: C21B D0 F6              BNE SPSTART
3590: C21D F0 2D              BEQ GEFUN      ; END LEADER, LOAD
                                               PROGRAM
3600: C21F 20 58 C2 PLOAD     JSR SCOUNT     ; LOAD PROGRAM
3610: C222 20 86 C2 LLOOP     JSR HOLBYT
3620: C225 C4 93              CPY $93
3630: C227 D0 02              BNE VERGL      ; WITH VERIFY ONLY
                                               COMPARE
3640: C229 91 C3              STA (STARTV), Y
3650: C22B D1 C3 VERGL        CMP (STARTV), Y
3660: C22D F0 02              BEQ GLEICH
3670: C22F 86 90              STX STATUS
3680: C231 45 D7 GLEICH       EOR PSUMME     ; COMPUTE CHECK
                                               TOTAL
3690: C233 85 D7              STA PSUMME
3700: C235 E6 C3              INC STARTV     ; INCREASE ADDRESS
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
                                               REACHED
3750: C23F A5 C4              LDA STARTV+1
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 #$000
3850: C252 8D A0 02           STA IRQFLG
3860: C255 4C 93 FC           JMP ENDEN
3870: C258 20 17 F8 SCOUNT    JSR PTASTE     ; SEEK CONDITION
3880: C25B 20 FC C0           JSR MOTOR
3890: C25E 84 D7              STY PSUMME
```

```
3900: C260 A9 07              LDA #$07       ; ($27) VALUE FOR
                                               TIMER LO
3910: C262 8D 06 DD           STA CIAI+6     ; 9PORT+8) IN TIMER LO
3920: C265 A2 01              LDX #$01       ; VALUE FOR TIMER HI
3930: C267 20 99 C2 START     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' - BYTES
4000: C277 C9 02              CMP $02        ;
4010: C279 F0 F9              BEQ ENDE2
4020: C27B C4 BD COUNTL       CPY $BD        ; TESTS THE COUNT-
                                               DOWN
4030: C27D D0 E8              BNE SSTART
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:              ; COLLECT BYTE FROM TAPE

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
4110: C28D 26 BD              ROL $BD        ; MOVES CARRY TO IN-
                                               PUT BUFFER
4120: C28F EA                 NOP
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 WAITDABIT                ; WAITS FOR SIGNAL TO
                             CIA2 +13          'FLAG'
4210: C29E FoFB              BEQ WAITDA
4220: C2A0 AD OD DD          LDA CIAI+13     ; LOAD ICR, HAS
                                               REACHED   TBO
4225:                                        ; BIT 1 IS SET
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 ONCE
4260: C2A9 8D OF DD          STA CIA1+15
4270: C2AC 68                PLA             ; FETCH ICR
4280: C2AD 4A                LSR A           ; MOVE BIT 1 TO CARRY
4290: C2AE 4A                LSR A
4300: C2AF 60                RTS
4310: C2B0
4320:                ; 'HOLBIT' FOR VC20

4330: C2B0
4330: C2B0
4340:                ;HOLBIT LDA $02
4350:                ;WAITDA BIT  PORT+13    ; TEST IFR BIT 1
4360:                ;       BEQ WAITDA
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  SETLFS
```

```
4510: C2BC 20 06 E2          JSR  WZEICH    ; FURTHER
                                              CHARACTERS
4520: C2BF 20 57 E2          JSR  HFNAM
4540: C2C2 20 06 E2          JSR  WZEICH
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  WZEICH
4610: C2D0 20 F4 C2          JSR  HOLWERT
4620: C2D3 86 AC             STX  $AC
4622: C2D5 86 47             STX  $A7
4624: C2D7 84 A8             STY  $A8
4630: C2D9 84 AD             STY  $AD
4632: C2DB A2 01             LDX #1         ; FLAG FOR LOAD TO
                                CERTAIN
4634: C2DD 86 02             DTX ABSFLG     ; ADDRESS AND SAVE
4650: C2DF 20 06 E2          JSR  WZEICH
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 OE E2          HOL WERT JSR CHRIN;FETCH NEXT
                                          VALUE
4720: C2F7 20 8A AD          JSR  FRMNUM    ;  EVALUATE  EXPRES-
                                              SION
4730: C2FA 20 F7 B7          JSR  FACINT    ; CONVERT TO 2BYTE
                                VALUE
4740: C2FD A6 14             LDX $14
4750: C2FF A4 15             LDY $15
4760: C301 60                RTS
4770: C302 A5 02 ADTEST      LDA ABSFLG     ; LOAD FLAG
4780: C304 29 01             AND #1
4790: C306 F0 08             BEQ NORM       ; UNEQUAL =⟩ LOAD
                                              NORMALLY
4800: C308 A5 AC             LDA $AC
4810: C30A 85 C3             STA  STARTV
```

```
4820: C30C A5 AD              LDA $AD
4830: C30E 85 C4              STA STARTV+1
4840: C310 60        NORM     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 BO C2           JSR GETPARA
5040: C31B A9 01              LDA #1          ; SA=1
5050: C31D 85 B9              STA $B9
5060: C31F A2 04              LDX #4
5070: C321 B5 2C ULOOP        LDA $2C, X      ; RESTORE START AND
                                                END
5080: C323 95 AB              STA $AB, X      ; ADDRESSES OF VARI-
                                                ABLES
5085: C325 95 A6              STA $A6, X
5090: C327 CA                 DEX
5100: C328 D0 F7              BNE ULOOP
5110: C32A 20 7F C0                JSR        ABSOLUTE; 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
                                                START
                                    AND
5140: C338 95 AB              STA $AB, X      ; END ADDRESSES
5145: C33A 95 A6              STA $A6, X      ; RE-STORE END
                                                ADDRESSES
5150: C33C CA                 DEX
5160: C33D D0 F7              BNE U2LOOP
5170: C33F 20 7F CO           JSR ASOLUTE     ; STORE
5175: C342 A2 05              LDX #5
5177: C344 86 AB              STX $AB
5190: C346 A5 33              LDA $33         ; RE-STORE STRING
                                                START
```

```
5200: C348 A6 34              LDX $34         ;
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 $3418       ;
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 CERTAIN
                                                ADDRESS
5430: C366 85 02              STA  ABSFLG
5440: C368 A5 2D              LDA $2D         ; RE-STORE VARIABLE
                                                START
5450: C36A A6 2E              LDX $2E         ; ADDRESS AS START
5460: C36C 85 AC              STA  $AC        ; 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; LOAD
5486: C375 85 02              STA  ABSFLG     ; TO CERTAIN ADDRESS
5490: C377 A9 00              LDA #0
5500: C379 85 B7              STA  $B7        ; NO FILE NAME
5510: C37B A6 AE              LDX $AE         ; END OF VARIABLES
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
                                                VECTOR
5560: C385 84 AD              STY  $AD
5570: C387 2083C1             JSR  LOADR      ; LOAD
5580: C38A A6 AE              LDX $AE         ; ARRAY END
```

```
5590: C38C A4 AF          LDX $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: C999 AD 3C 03       LDA CASPUF
5642: C39C 85 33          STA $33
5644: C39E AD 3D 03       LDA CASPUF+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 FEHLAUS      ; OUTPUT ERROR
5700: C3AF A2 19  NOFEHL  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 CONT
5740: C3B9 60             RTS
UC000-C3BA
```

```
**********************************************************************
100 REM BASICLOADER FT20
**********************************************************************

110  E=29611:A=28672:PS=0
120  FOR I=A TO E:READ X:POKE I, X:PS=PS+X:NEXT
130  IF PS<>114760 THEN PRINT" ERROR IN DATA ":END
140  POKE 55, 0:POKE 56, 112
150  SYS A:NEW
160  DATA 169, 11, 141, 8, 3, 169, 112, 141, 9, 3, 96, 32, 115, 0, 240, 4, 201, 95, 240,
          3, 76, 231
170  DATA 199, 32, 115, 0, 32, 32, 112, 76, 174, 199, 162, 0, 221, 84, 112, 240, 8
          232, 228, 4
180  DATA 208, 246, 76, 8, 207, 138, 10, 170, 189, 90, 112, 72, 189, 8, 9, 112, 72, 76,
          115, 0, 201
190  DATA 83, 240, 7, 201, 76, 240, 9, 76, 8, 207, 32, 115, 0, 76, 3, 11, 5, 32, 115, 0,
          76, 83, 115
```

```
200 DATA 83, 76, 86, 77, 68, 98, 112, 105, 113, 108, 113, 91, 113, 60, 112, 162, 5,
    134, 171, 162
210 DATA 0, 134, 2, 32, 162, 114, 165, 2, 41, 2, 208, 11, 162, 4, 181, 42, 149, 171,
    149, 166, 202
220 DATA 208, 247, 32, 183, 248, 32, 40, 247, 32, 252, 112, 32, 11, 113, 165, 185, 24,
    105, 1
230 DATA 202, 32, 43, 113, 162, 10, 185, 167, 0, 32, 43, 113, 162, 8, 200, 192, 5, 234,
    208, 242
240 DATA 160, 0, 162, 6, 177, 187, 196, 183, 144, 3, 169, 32, 202, 32, 43, 113, 162, 5,
    200, 192
250 DATA 187, 208, 237, 169, 2, 133, 171, 32, 11, 113, 152, 32, 43, 113, 132, 215,
    162, 9, 234
260 DATA 177, 172, 32, 43, 113, 162, 5, 230, 172, 208, 4, 230, 173, 202, 202, 165,
    172, 197, 174
270 DATA 165, 173, 229, 175, 144, 231, 234, 165, 215, 32, 43, 113, 162, 9, 136, 208,
    246, 200
280 DATA 132, 192, 88, 24, 169, 0, 141, 160, 2, 76, 207, 252, 32, 44, 200, 160, 0, 132,
    192, 202
290 DATA 208, 253, 136, 208, 250, 120, 96, 160, 0, 169, 2, 32, 43, 113, 162, 9, 136,
    192, 9, 208.
300 DATA 244, 162, 7, 198, 171, 208, 238, 152, 32, 43, 113, 162, 9, 136, 208, 247,
    202, 202, 96
310 DATA 133, 189, 69, 215, 133, 215, 169, 8, 133, 163, 6, 189, 173, 32, 145, 41, 247,
    32, 78
320 DATA 113, 162, 19, 234, 9, 8, 32, 78, 113, 162, 16, 198, 163, 208, 232, 96, 202,
    208, 253
330 DATA 144, 5, 162, 11, 202, 208, 253, 141, 32, 145, 96, 165, 45, 56, 233, 2, 168,
    165, 46, 233
340 DATA 0, 162, 0, 240, 9, 162, 0, 44, 162, 1, 164, 43, 165, 44, 134, 10, 134, 147,
    162, 0, 134
350 DATA 2, 132, 195, 133, 196, 32, 162, 114, 32, 136, 113, 76, 119, 225, 32, 255,
    113, 165, 171
360 DATA 201, 2, 240, 8, 201, 1, 208, 243, 165, 185, 240, 16, 169, 2, 5, 2, 133, 2, 173,
    60, 3, 133
370 DATA 195, 173, 61, 3, 133, 196, 32, 244, 114, 165, 2, 41, 4, 208, 3, 32, 211, 247,
    32, 44, 200
380 DATA 164, 183, 240, 11, 136, 177, 187, 217, 65, 3, 208, 196, 152, 208, 245, 132,
    144, 32
390 DATA 106, 246, 173, 62, 3, 56, 237, 60, 3, 8, 24, 101, 195, 133, 174, 173, 63, 3,
    101, 196
400 DATA 40, 237, 61, 3, 133, 175, 32, 20, 114, 165, 189, 69, 215, 5, 144, 240, 4, 169,
    255, 133
```

```
410 DATA 144, 165, 2, 208, 3, 76, 65, 246, 166, 45, 164, 46, 96, 32, 77, 114, 201, 0,
    240, 249
420 DATA 133, 171, 32, 123, 114, 145, 178, 200, 192, 192, 208, 246, 240, 45, 32, 77,
    114, 32
430 DATA 123, 114, 196, 147, 208, 2, 145, 195, 209, 195, 240, 2, 134, 144, 69, 215,
    133, 215
440 DATA 230, 195, 208, 2, 230, 196, 165, 195, 197, 174, 165, 196, 229, 175, 144,
    221, 32, 123
450 DATA 114, 32, 252, 112, 200, 132, 192, 88, 24, 169, 0, 141, 160, 2, 76, 207, 252,
    32, 148
460 DATA 248, 32, 252, 112, 132, 215, 169, 39, 141, 40, 145, 162, 1, 32, 142, 114, 38,
    189, 165
470 DATA 189, 201, 2, 208, 245, 160, 9, 32, 123, 114, 201, 2, 240, 249, 196, 189, 208,
    232, 32
480 DATA 123, 114, 136, 208, 246, 96, 169, 8, 133, 163, 32, 142, 114, 38, 189, 234,
    234, 234
490 DATA 198, 163, 208, 244, 165, 189, 96, 169, 2, 44, 45, 145, 240, 251, 173, 45,
    145, 142, 41
500 DATA 145, 44, 33, 145, 10, 10, 10, 96, 169, 0, 32, 189, 255, 162, 1, 160, 0, 32,
    186, 255, 32
510 DATA 3, 226, 32, 84, 226, 32, 3, 226, 32, 253, 225, 138, 168, 32, 186, 255, 32, 3,
    226, 32
520 DATA 230, 114, 134, 172, 134, 167, 132, 168, 132, 173, 162, 1, 134, 2, 32, 3, 226,
    32, 230
530 DATA 114, 134, 174, 134, 169, 132, 170, 132, 175, 165, 2, 9, 2, 133, 2, 96, 32, 11,
    226, 32
540 DATA 138, 205, 32, 247, 215, 166, 20, 164, 21, 96, 165, 2, 41, 1, 240, 8, 165, 172,
    133, 195
550 DATA 165, 173, 133, 196, 96, 32, 38, 213, 162, 5, 134, 171, 32, 162, 114, 169, 1,
    133, 185
560 DATA 162, 4, 181, 44, 149, 171, 149, 166, 202, 208, 247, 32, 127, 112, 162, 0,
    134, 183, 162
570 DATA 5, 134, 171, 202, 181, 46, 149, 171, 149, 166, 202, 208, 247, 32, 127, 112,
    162, 5, 134
580 DATA 171, 165, 51, 166, 52, 133, 172, 133, 167, 134, 173, 134, 168, 165, 55, 166,
    56, 133
590 DATA 174, 133, 169, 134, 175, 134, 170, 76, 127, 112, 32, 162, 114, 169, 1, 133,
    2, 165, 45
600 DATA 166, 46, 133, 172, 134, 173, 32, 136, 113, 169, 5, 133, 2, 169, 0, 133, 183,
    166, 174
610 DATA 164, 175, 134, 47, 132, 48, 134, 172, 132, 173, 32, 136, 113, 166, 174, 164,
    175, 134
```

```
620 DATA 49, 132, 50, 169, 4, 133, 2, 32, 136, 113, 173, 60, 3, 133, 51, 173, 61, 3,
    133, 52, 32
630 DATA 183, 255, 37, 191, 240, 5, 162, 29, 76, 55, 196, 162, 25, 134, 22, 169, 0,
    133, 58, 133
640 DATA 16, 96


1000 REM LOADER FT64
1010 E=50105:A=49152:PS=0
1020 FOR I=A TO E:READ X:POKE I, X:PS=PS+X:NEXT
1030 IF PS < >120504 THEN PRINT"ERROR IN DATA":END
1040 SYS A
10000 DATA 169, 11, 141, 8, 3, 169, 192, 141, 9, 3, 96, 32, 115, 0, 240, 4, 201, 95,
    240, 3, 76, 231
10010 DATA 167, 32, 115, 0, 32, 32, 192, 76, 174, 167, 162, 0, 221, 84, 192, 240, 8,
    232, 228, 4
10020 DATA 208, 246, 76, 8, 175, 138, 10, 170, 189, 90, 192, 72, 189, 89, 192, 72, 76,
    115, 0, 201
10030 DATA 83, 240, 7, 201, 76, 240, 9, 76, 8, 175, 32, 115, 0, 76, 17, 195, 32, 115, 0,
    76, 97, 195
10040 DATA 83, 76, 86, 77, 68, 98, 192, 111, 193, 114, 193, 97, 193, 60, 192, 162, 5,
    134, 171, 162
10050 DATA 0, 134, 2, 32, 176, 194, 165, 2, 41, 2, 208, 11, 162, 4, 181, 42, 149, 171,
    149, 166, 202
10060 DATA 208, 247, 32, 56, 248, 32, 143, 246, 32, 252, 192, 32, 19, 193, 165, 185,
    24, 105, 1
10070 DATA 202, 32, 51, 193, 162, 8, 185, 167, 0, 32, 51, 193, 162, 6, 200, 192, 5,
    234, 208, 242
10080 DATA 160, 0, 162, 4, 177, 187, 196, 183, 144, 3, 169, 32, 202, 32, 51, 193, 162,
    5, 200, 192
10090 DATA 187, 208, 237, 169, 2, 133, 171, 32, 19, 193, 152, 32, 51, 193, 132, 215,
    162, 7, 234
10100 DATA 177, 172, 32, 51, 193, 162, 3, 230, 172, 208, 4, 230, 173, 202, 202, 165,
    172, 197, 174
10110 DATA 165, 173, 229, 175, 144, 231, 234, 165, 215, 32, 51, 193, 162, 7, 136,
    208, 246, 200
10120 DATA 132, 192, 88, 24, 169, 0, 141, 160, 2, 76, 147, 252, 32, 44, 168, 160, 0,
    132, 192, 173
10130 DATA 17, 208, 41, 239, 141, 17, 208, 202, 208, 253, 136, 208, 250, 120, 96,
    160, 0, 169, 2
10140 DATA 32, 51, 193, 162, 7, 136, 192, 9, 208, 244, 162, 5, 198, 171, 208, 238,
    152, 32, 51, 193
10150 DATA 162, 7, 136, 208, 247, 202, 202, 96, 133, 189, 69, 215, 133, 215, 169, 8,
    133, 163, 6
```

```
10160 DATA 189, 165, 1, 41, 247, 32, 85, 193, 162, 17, 234, 9, 8, 32, 85, 193, 162, 14,
      198, 163
10170 DATA 208, 233, 96, 202, 208, 253, 144, 5, 162, 11, 202, 208, 253, 133, 1, 96,
      165, 45, 56
10180 DATA 233, 2, 168, 165, 46, 233, 0, 162, 0, 240, 9, 162, 0, 44, 162, 1, 164, 43,
      165, 44, 134
10190 DATA 10, 134, 147, 162, 0, 134, 2, 132, 195, 133, 196, 32, 176, 194, 32, 142,
      193, 76, 122
10200 DATA 225, 32, 10, 194, 165, 171, 201, 2, 240, 8, 201, 1, 208, 243, 165, 185,
      240, 16, 169
10210 DATA 2, 5, 2, 133, 2, 173, 60, 3, 133, 195, 173, 61, 3, 133, 196, 32, 2, 195, 165,
      2, 41, 4, 208
10220 DATA 8, 32, 80, 247, 32, 228, 255, 240, 251, 32, 44, 168, 164, 183, 240, 11,
      136, 177, 187
10230 DATA 217, 65, 3, 208, 191, 152, 208, 245, 132, 144, 32, 210, 245, 173, 62, 3,
      56, 237, 60
10240 DATA 3, 8, 24, 101, 195, 133, 174, 173, 63, 3, 101, 196, 40, 237, 61, 3, 133,
      175, 32, 31, 194
10250 DATA 165, 189, 69, 215, 5, 144, 240, 4, 169, 255, 133, 144, 165, 2, 208, 3, 76,
      169, 245, 166
10260 DATA 45, 164, 46, 96, 32, 88, 194, 201, 0, 240, 249, 133, 171, 32, 134, 194,
      145, 178, 200
10270 DATA 192, 192, 208, 246, 240, 45, 32, 88, 194, 32, 134, 194, 196, 147, 208, 2,
      145, 195, 209
10280 DATA 195, 240, 2, 134, 144, 69, 215, 133, 215, 230, 195, 208, 2, 230, 196, 165,
      195, 197
10290 DATA 174, 165, 196, 229, 175, 144, 221, 32, 134, 194, 32, 252, 192, 200, 132,
      192, 88, 24
10300 DATA 169, 0, 141, 160, 2, 76, 147, 252, 32, 23, 248, 32, 252, 192, 132, 215,
      169, 7, 141, 6
10310 DATA 221, 162, 1, 32, 153, 194, 38, 189, 165, 189, 201, 2, 208, 245, 160, 9, 32,
      134, 194

10320 DATA 201, 2, 240, 249, 196, 189, 208, 232, 32, 134, 194, 136, 208, 246, 96,
      169, 8, 133, 163
10330 DATA 32, 153, 194, 38, 189, 234, 234, 234, 198, 163, 208, 244, 165, 189, 96,
      169, 16, 44
10340 DATA 13, 220, 240, 251, 173, 13, 221, 142, 7, 221, 72, 169, 25, 141, 15, 221,
      104, 74, 74
10350 DATA 96, 169, 0, 32, 189, 255, 162, 1, 160, 0, 32, 186, 255, 32, 6, 226, 32, 87,
      226, 32, 6
10360 DATA 226, 32, 0, 226, 138, 168, 32, 186, 255, 32, 6, 226, 32, 244, 194, 134,
      172, 134, 167
```

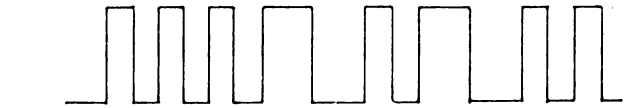
```
10370 DATA 132, 168, 132, 173, 162, 1, 134, 2, 32, 6, 226, 32, 244, 194, 134, 174,
      134, 169, 132
10380 DATA 170, 132, 175, 165, 2, 9, 2, 133, 2, 96, 32, 14, 226, 32, 138, 173, 32, 247,
      183, 166
10390 DATA 20, 164, 21, 96, 165, 2, 41, 1, 240, 8, 165, 172, 133, 195, 165, 173, 133,
      196, 96, 32
10400 DATA 38, 181, 162, 5, 134, 171, 32, 176, 194, 169, 1, 133, 185, 162, 4, 181, 44,
      149, 171
10410 DATA 149, 166, 202, 208, 247, 32, 127, 192, 162, 0, 134, 183, 162, 5, 134, 171,
      202, 181
10420 DATA 46, 149, 171, 149, 166, 202, 208, 247, 32, 127, 192, 162, 5, 134, 171,
      165, 51, 166
10430 DATA 52, 133, 172, 133, 167, 134, 173, 134, 168, 165, 55, 166, 56, 133, 174,
      133, 169, 134
10440 DATA 175, 134, 170, 76, 127, 192, 32, 176, 194, 169, 1, 133, 2, 165, 45, 166,
      46, 133, 172
10450 DATA 134, 173, 32, 142, 193, 169, 5, 133, 2, 169, 0, 133, 183, 166, 174, 164,
      175, 134, 47
10460 DATA 132, 48, 134, 172, 132, 173, 32, 142, 193, 166, 174, 164, 175, 134, 49,
      132, 50, 169
10470 DATA 4, 133, 2, 32, 142, 193, 173, 60, 3, 133, 51, 173, 61, 3, 133, 52, 32, 183,
      255, 37, 191
10480 DATA 240, 5, 162, 29, 76, 55, 164, 162, 25, 134, 22, 169, 0, 133, 58, 133, 16, 96
```

12.1 PROGRAM DESCRIPTION.

For the benefit of those of you who are interested and already have some basic foundation in 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 FIRST Publishing Book, The Anatomy of the C-64.

Principle of Operation.

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



Value	0	0	0	1	0	1	0	0
Bit No.:	7	6	5	4	3	2	1	0

A program is stored in the following format:.

1. Synchronisation 1.
   5 times (246 times byte 2). In this way, synchronisation 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 synchronisation.

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. Synchronisation 2
   As synchronisation 1, but only twice.

4. Zero byte as indication that data following.

5. Data

6 Check total, 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 $C01D($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 total 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 (D4) 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 got to zero, before being started afresh.

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 Synchronisation: 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 Synchronisation: 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 total 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 synchronisation 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 synchronisation. With the aid of the last byte read in by SCOUNT, it determines whether synchronisation 1 or 2 is involved. If the synchronisation found is not synchronisation 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, synchronisation 2 is sought using SCOUNT, and in this way the read routine is synchronised to the tape.

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 total 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 total 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 synchronisation 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 synchronisation 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 synchron isation 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 vendor is identical to the end vector. To conclude, the EXOR check total 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 synchronisation. If the last byte of the synchronisation found—which was written in $AB—is zero, then this is synchronisation 2, and control branches back to the start of this routine. If $AB is not equal to zero, then synchronisation 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 MERGE, LOAD and VERIFY routines all run via the LOAD routine.

In the case of MERGE, 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 readfl in by GETPARA and the corresponding value is set in ABSFLG.

DATA STORAGE AND READ DATA

Data are 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 up at the beginning.

The file name is now set using GETPARA. Then 1 is given as the secondary address, since a certain storage 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 fiinding the next synchronisation 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 put out 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 storage 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 searcing is the same in all fields.

— A search can be made on the basis of equality, inequality, or greater or lesser value.

— If you have an extended instruction facility with an INSTRING instruction, then fields can be searched for partial areas.

— 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 storage space and can be either altered or deleted as required, so that more storage space 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 catalogue of recipes or gramophone records, or record all the addresses of your friends and acquaintances.

In order better to explain and illustrate the function and operation of this program, it would be best for you to avail yourself of these operating instructions to set up a trial file. As a catalogue of recipes or something of that sort would be too lengthy for our purposes, it would be best for you to 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 with RUN, the menu will appear on your screen.

```
        * * * M E N U * * *

1 maintain file
2 paging
3 output list
4 sort list
5 input data
6 load file
7 store file
8 set up file
9 change field ident.
```

As you have not yet set up the file, let us start with option 8 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. Christian name
2. Surname
3. Street and house number
4. Postal code
5. Town
6. Telephone
7. Hobby

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

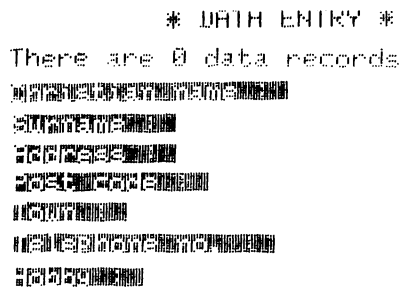
As you have not properly tested the program yet—and it is always possible to make mistakes when entering the details—it would be safest to store the file without data. To do this, you merely select item 7 on the menu, "Store file".

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:.



The cursor will be positioned under the field identification "Christian 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:



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.
K	—	The cursor appears in the first field again and you can correct the input.

Any other key — The data record is accepted and the program 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 take them over by pressing RETURN. So now enter 4 addresses in the manner described below.

If you have an "INSTRING" instruction at your disposal, then enter the next data record as well:.



Now return to the menu by pressing any key.

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 operating 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".

```
          * * PAGING    * *

Date record no. 2
[illegible]
Frank
[illegible]
Jones
[illegible]
2 Reading Road
[illegible]
RG3 7ZD
[illegible]
Reading
[illegible]
58434
[illegible]
Cycling
```

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 draw up lists.

First of all, we shall look out all data records in which the name begins with an "M". 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 : T/2 : = / 3 : ⟨⟩ / 4 : ⟩ / 5 : ⟨

These are the options that you have.

1. **Partial String (only with INSTRING instruction)**
   You are seeking a sequence of characters which may be anywhere within the field.

2. **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", "Mamma", "Marmalade" etc. etc. The same applies by analogy to the following possibilities.

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

4. **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...).

5. **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 "2", you will get the following display:.

```
          SEEK / CHANGE

1. Christian name
2. Surname :
3. Address :
4. Post code
5. Town :
6. TelePhone no.
7. Hobby :

Which fields are to be searched for ?


= Surname
```

If you wish to search through 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 now be asked whether a data record is to be found in one 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 "M", enter "M" and RETURN. "RETURN" will now appear again on the last line of the screen. If you now press "K", you can change the input. If



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 "L". 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 can then sort them by any field you wish, e.g. "Surnames", 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 "K", the fields displayed will be deleted 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 should be printed out as well.

Finally, you will be asked whether output should be displayed on the screen or printed out on a printer. With printer output you will get the following result:.

```
H E A D I N G


First
Publishing
Unit 20b Horseshoe Park
RG8 7JW
Pangbourne
07357 5244
Publishers


Frank
Jonesshing
2 Reading Road
RG3 72D
Reading
58434
Cycling


Willie
Makeit
100 Rush street
RG3 4BP
Reading
37552
Programming
```

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 A have no difficulty in operating this program.

```
  1  GOTO 1060
*********************************************************************
 90  REM " SEEK ROUTINE"
*********************************************************************
100  IF(F)FE)OR(N)=AS)THEN RETURN
110  A=SF(F)
120  ON A GOSUB 150, 190, 250, 280, 220
130  F=F+1:GOTO 100
140  :REM "PARTIAL AREA"
150  T=INSTR(U$(F), S$(F))
160  If T<>0 THEN N=N+1
170  RETURN
180  : REM "INTEGER'="
190  IF S$(F)=LEFT$(U$(F), LEN(S$(F)))THEN N=N+1
200  RETURN
210  : REM "INTEGER'(="
220  IF S$(F)>=LEFT$(U$(F), LEN(S$(F)))THEN N=N+1
230  RETURN
240  : REM "INTEGER' ""
250  IF S$(F) LEFT$(U$(F), LEN(S$(F)))THEN N=N+1
260  RETURN
270  : REM "INTEGER'>="
280  IF S$(F)(=LEFT$(U$(F), LEN(S$(F)))THEN N=N+1
290  RETURN
*********************************************************************
1000  REM "HOLDING LOOP"
*********************************************************************
1010  POKE 198, 0:WAIT 198, 1:GET A$:A=ASC(A$) :REM "HOLDING
        LOOP"
1020  IF A=133 THEN GOSUB 3750
1030  RETURN
```

```
*********************************************************************
1050  REM "INITIALISATION"
*********************************************************************
1060  DW$="[HOME, DOWN25]"
1070  SYS 65517:SP=PEEK(781):ZE=PEEK(782) :REM "FETCH SCREEN
      DATA"
1080  C1=53281:C2=1:IF SP=22 THEN C1=36879:C2=8
1090  DIM Z$(4):Z$(0)=" T":Z$(1)=" ="Z$(2)="⟨ ⟩":Z$(3)="⟩ =" :Z$(4)="#="
*********************************************************************
1100  REM "MAIN PROGRAM"
*********************************************************************
1110  PRINT CHR$(14):POKE C1, 6*C2
1120  GOSUB 3560:PRINT:PRINT:TA=INT(SP/5)-2
1130  PRINT TAB(TA)"[WHITE, RVS]1[RVOFF, SPACE]MAINTAIN FILE"
1140  PRINT TAB(TA)"[RVS]2[RVOFF, SPACE] PAGING"
1150  PRINT TAB(TA)"[RVS]3[RVOFF, SPACE] OUTPUT LIST"
1160  PRINT TAB(TA)"[RVS]4[RVOFF, SPACE] SORT LIST"
1170  PRINT TAB(TA)"[RVS]5[RVOFF, SPACE] INPUT DATA"
1180  PRINT TAB(TA)"[RVS]6[RVOFF, SPACE] LOAD FILE"
1190  PRINT TAB(TA)"[RVS]7[RVOFF, SPACE] STORE FILE"
1200  PRINT TAB(TA)"[RVS]8[RVOFF, SPACE] SET UP FILE"
1210  PRINT TAB(TA)"[RVS]9[RVOFF, SPACE] CHANGE FIELD IDENT."
1220  PRINT TAB(TA)"[RVS]E[RVOFF, SPACE4, RVS]END [RVOFF]"
1230  GOSUB 1010:A=VAL(A$):IF A$="E"THEN 1270
1240  IF (A<5) AND (FL AND 4)=0 THEN 1110
1250  ON A GOSUB 2130, 2000, 2700, 2990, 1860, 1570, 1460, 1700, 1780
1260  GOTO 1110
*********************************************************************
1265  REM "TERMINATE PROGRAM"
*********************************************************************
1270  IF(FL AND 2)=2 OR (F1 AND 4)=0 THEN END
1280  PRINT"[CLEAR, DOWN20]":PRINT TAB(SP/2-8)"[WHITE] STILL
      TERMINATE":GOSUB 1310
1290  IF FL 255 THEN 1110
1300  END
*********************************************************************
1305  REM "ERROR"
*********************************************************************
1310  POKE C1, 5*C2
1320  PRINT"[HOME]":PRINT"[DOWN7]"
1330  PRINT:PRINT TAB(SP/2-3)"[RED, RVS]FILE"
1340  PRINT:PRINT TAB(SP/2-6)"[RVS]WAS NOT"
```

```
1350  PRINT:PRINT TAB(SP/2-7)"[RVS]STORED"
1360  FOR I=0 TO 15:GET A$:IF A$<> ""THEN I=15:NEXT :GOTO 1440
1370  NEXT
1300  PRINT"[HOME]":PRINT"[DOWN7]"
1390  PRINT:PRINT TAB(SP/2-3)"[RED]FILE"
1400  PRINT:PRINT TAB(SP/2-6)"WAS NOT"
1410  PRINT:PRINT TAB(SP/2-7)"STORED"
1420  FOR I=0 TO 15:GET A$:IF A$=""THEN I=15:NEXT:GOTO 1310
1430  NEXT
1440  IF A$="Y"THEN FL=255
1450  RETURN
 ********************************************************************
1460  REM "STORE"
 ********************************************************************
1470  IF(FL AND 1)=0 THEN RETURN
1480  POKE C1, 7*C2:PRINT"[BLUE]":GOSUB 3590
1490  PRINT:PRINT:PRINT" "F$"[UP]":INPUT"FILE NAME";F$
1500  PRINT:PRINT"INSERT DATA CASSETTE"
1510  PRINT:PRINT:PRINT:" << RETURN >>"
1520  GOSUB 1010
1530  IF A$="K"THEN PRINT"[HOME]":GOTO 1490
1540  IF A$="Q"THEN RETURN
1550  IF A 13 THEN 1520
1560←DSF$:FL=FL OR 2:RETURN
 ********************************************************************
1570  REM "LOAD"
 ********************************************************************
1580  IF(FL AND 6)=4 THEN GOSUB 1680:IF FL 255 THEN RETURN
1590  POKE C1, 7*C2:PRINT"[BLUE]":GOSUB 3600
1600  PRINT:PRINT:PRINT" "F$"[UP]":INPUT"FILE NAME";F$
1610  PRINT:PRINT:"INSERT DATA CASSETTE"
1620  PRINT:PRINT:PRINT" << RETURN >>
1630  GOSUB 1010
1640  IF A$="K"THEN PRINT"[HOME]":GOTO 1600
1650  IF A$="Q"THEN RETURN
1660  IF A<> 13 THEN 1630
1670←DLF$:FL=FL AND 251:RETURN
1680  PRINT"[CLEAR, DOWN20]":PRINT TAB(SP/2-8)"[WHITE]STILL LOAD"
1690  GOSUB 1310:RETURN
 ********************************************************************
1700  REM "SET UP FILE"
 ********************************************************************
```

```
1710  POKE C1, 8*C2:IF F1 THEN RETURN
1720  GOSUB 3550:REM "WRITE TITLE"
1730  PRINT:PRINT:INPUT"NO. OF FIELDS";FA
1740  PRINT:INPUT"[DOWN]MAX DATA RECORDS";RA
1750  PRINT:PRINT"〈〈 RETURN 〉〉 ":GOS 1010 :IF A〈〉 13 THEN
        PRINT"[HOME]":GOTO 1730
1760  DIM M$(FA), E$(FA, RA), SF(FA), S$(FA), U$(FA), GF%(RA)
1770  FL=1:REM "SET FLAG"
 ********************************************************************
1775  REM "ENTER FIELD IDENTIFICATIONS
 ********************************************************************
1780  IF(FL AND 1)=0 THEN RETURN
1790  GOSUB 3550:REM "WRITE HEADING"
1800  PRINT:PRINT"ENTER THE FIELD";:IF SP〈40 THEN PRINT:PRINT
1810  PRINT"IDENTIFICATIONS"
1820  PRINT:FOR I=0 TO FA-1:PRINT:PRINT"[RVS] "I+1"[RVOFF]"
1830  PRINT" "M$(I)"[UP]":INPUT M$(I):NEXT
1840  PRINT LEFT$(DW$, ZE)"〈〈 RETURN 〉〉";:GOSUB 1010 :IF
        A$="K"THEN PRINT"[HOME]":GOTO 1800
1850  FL=FL OR 4:RETURN
 ********************************************************************
1860  REM "ENTER DATA"
 ********************************************************************
1865  IF FL AND 1=0 THEN RETURN
1870  POKE C1, 3*C2:PRINT"[BLUE]"
1880  CLOSE 1:OPEN 1, 0
1890  GOSUB 3580:PRINT:PRINT"THERE ARE"RT; :IF SP〈40 THEN PRINT
1900  PRINT"DATA RECORDS"
1910  R1=RT:R2=RT:IF(RT〉0)AND(A$〈〉"K")THEN R1=RT-1
1920  GOSUB 3650:GOSUB 3670:TEM "DISPLAY FIELDS"
1930  PRINT LEFT$(DW$, ZE)"[GREEN, SPACE3]〈〈 RETURN 〉〉 [WHITE]"
        ;:GOSUB 1010
1940  IF A$="Q"THEN RETURN
1950  IF A$="K" THEN 1890
1960  IF A〈〉13 THEN FL=FL OR 12:RT=RT+1:RETURN
1970  RT=RT+1:IF RT〈RA THEN 1890
1980  GOSUB 3580:PRINT:PRINT:PRINT"MAX NO. OF RECORDS"; :IF SP〈40
        THEN PRINT
1990  PRINT:PRINT"REACHED":GOSUB 1010:RETURN
 ********************************************************************
2000  REM "PAGING"
 ********************************************************************
```

```
2005  IF(FL AND 8)=0 THEN RETURN
2010  POKE C1, 6*C2:PRINT"[WHITE]"
2020  R1=0
2025  IF R1<0 THEN R1=R1+RT
2030  GOSUB 3610:PRINT:PRINT"DATA RECORD NO."R1
2040  GOSUB 3650
2050  PRINT LEFT$(DW$, ZE)"[GREEN, SPACE3]<< RETURN >>
          [WHITE]";:GOSUB 1010
2060  IF A$="-"THEN R1=R1-1:GOTO 2025
2070  IF A$="Q"THEN RETURN
2080  IF A$="A"THEN GOSUB 2560:GOTO 2060
2090  IF A<>13 THEN 2050
2100  IF R1<RT-1 THEN R1=R1+1:GOTO 2030
2110  GOSUB 3610:PRINT"[DOWN8, RIGHT, RVS, SPACE]NO FURTHER
          DATA"
2120  R1=0: GOTO 2050
 ********************************************************************
2130  REM "MAINTAIN FILE"
 ********************************************************************
2135  IF (FL AND 8)=0 THEN RETURN
2140  POKE C1, 4*C2
2150  FOR I=0 TO FA-1:SF(I)=0:NEXT:AS=0
2160  GOSUB 3620:FOR I=0 TO FA-1:PRINT I+1"[LEFT] "M$(I):NEXT
2170  PRINT:PRINT"WHICH FIELDS ARE"; :IF SP<40 THEN PRINT:PRINT
2180  PRINT"TO BE SOUGHT":PRINT:PRINT
2190  GOSUB 1010:IF A=13 THEN 2262
2200  IF A$="Q"THEN RETURN
2210  IF A$="K"THEN 2150
2220  I=VAL(A$):IF I<1 OR I>FA THEN 2190
2230  IF SF(I-1)THEN 2190
2240  D6=PEEK(214):REM "SAVE CURSOR LINE"
2250  SF(I-1)=1:PRINT TAB(3)"[RVS]"M$(I-1):GOSUB 3710
2260  PRINT LEFT$(DW$, D6+
1)Z$(SF(I-1)-1):AS=AS+1:GOTO 2190
2262  GOSUB 3620:PRINT:PRINT"IF AGREEMENT IN"
2264  PRINT:PRINT TAB(4)"1 = ONE"
2265  PRINT:PRINT TAB(4)"0 = ALL"
2266  PRINT:INPUT"IS FOUND"; UE
2267  IF UE THEN AS=UE
2270  GOSUB 3620:CLOSE 1:OPEN 1, 0
2280  PRINT:PRINT"THEN ENTER"; :IF SP<40 THEN PRINT:PRINT
2290  PRINT"SEARCH STRINGS:PRINT:PRINT"
```

```
2300  FOR F=0 TO FA-1:IF SF(F)=0 THEN 2330
2310  PRINT"[RVS]"M$(F)" : "
2320  PRINT S$(F)"[UP]":INPUT#1, S$(F):PRINT
2330  NEXT
2340  PRINT LEFT$(DW$, ZE)"〈〈 RETURN 〉〉";:GOSUB 1010
        :IFA$="K"THEN PRINT"[HOME]":GOTO 2280
2350  IF A$="Q"THEN RETURN
2360  IF A〈〉13 GOTO 2340
 ********************************************************************
2370  REM "SEEK"
 ********************************************************************
2372  FOR F=0 TO FA-1:IF SF(F)THEN FB=F:F=FA
2374  NEXT:REM "DETERMINE LOWEST AND HIGHEST"
2376  FOR F=FA-1 TO 0 STEP-1:IF SF(F)THEN FE=F:F=0
2378  NEXT:REM "SEARCH FIELD"
2380  Z=0:FOR R =0 TO RT-1
2390  FOR F=FB TO FE-1:IF SF(F)THEN U$(F)=E$(F, R)
2400  NEXT:REM "RE-STORE EXAMINATION STRINGS"
2410  N=0:F=FB:GOSUB 100:IF N〉=AS THEN GF%(Z)=R:Z=Z+1
2420  NEXT:IF Z=0 THEN 2130
 ********************************************************************
2430  REM "DISPLAY DATA RECORDS"
 ********************************************************************
2440  I=0
2450  GOSUB 3620:PRINT:PRINT Z"HIT" :PRINT"DATA RECORD
        NO."GF%(I)
2460  R1=GF%(I):GOSUB 3650
2470  PRINT LEFT$(DW$, ZE)"[GREEN, SPACE3]〈〈 RETURN 〉〉
        [WHITE]";:GOSUB 1010
2480  IF A$="-"AND I〉0 THEN I=I-1:GOTO 2450
2490  IF A$="Q"THEN RETURN
2500  IF A$="A"THEN GOSUB 2560:GOTO 2480
2510  IF A$="L"THEN 2640
2520  IF A〈〉13 THEN 2470
2530  IF I〈Z-1 THEN I=I+1:GOTO 2450
2540  GOSUB 3620:PRINT"[DOWN8, RIGHT3, RVS, SPACE]NO FURTHER
        DATA"
2550  PRINT LEFT$(DW$, ZE)"[GREEN, SPACE3]〈〈 RETURN 〉〉
        [WHITE]";:GOSUB 1010:GOTO 2450
 ********************************************************************
2560 REM "CHANGE"
 ********************************************************************
```

```
2570  CLOSE 1:OPEN 1, 0:R2=R1:GOSUB 3670
2580  PRINT LEFT$(DW$, ZE)"[GREEN, SPACE3]〈〈 RETURN 〉〉 [WHITE]";fl
        :GOSUB 1010
2600  IF A$="K"THEN 2570
2610  FL=FL OR 4:FL=FL AND 253:RETURN
2620  RETURN
 ********************************************************************
2630  REM "DELETE"
 ********************************************************************
2640  FOR I1=GF%(I)+1 TO RT-1:REM "RE-STORE"
2650  FOR J=0 TO FA-1:REM "DATA"
2660  E$(J, I1-1)=E$(J, I1)
2670  NEXT J, I1
2680  Z=Z-1:RT=RT-1:FL=FL OR 4:GOTO 2450
 ********************************************************************
2690  REM "OUTPUT LIST"
 ********************************************************************
2695  IF Z=0 THEN RETURN
2700  POKE C1, 7*C2:PRINT"[BLUE]"
2710  FOR I=0 TO FA-1:SF(I)=0:NEXT
2720  GOSUB 3630
2725  FOR I=0 TO FA-1:PRINT I+1"[LEFT]"M$(I):NEXT
2730  PRINT:PRINT"WHICH FIELDS ARE TO BE"; :IF SP〈40 THEN PRINT-
        :PRINT
2740  PRINT"PRINTED OUT":PRINT:PRINT
2750  GOSUB 1010:IF A=13 THEN 2810
2760  IF A$="Q"THEN RETURN
2770  IF A$="K"THEN 2710
2780  I=VAL(A$):IF I〈1 OR I〉FA THEN 2750
2790  IF SF(I-1)THEN 2750
2800  SF(I-1)=1:PRINT"[RVS]"M$(I-1):PRINT:GOTO 2750
2810  PRINT:INPUT"PRINT OUT";UB$
2820  PRINT:PRINT"HEADING":PRINT :INPUT"FIELD DESCRIPTION[1/
        0]";FM$
2830  FM=VAL(FM$)
2840  PRINT:INPUT"PRINTER=4/BILDS.=0";DV
2850  IF DV THEN OPEN 4, DV, 7:CMD 4
2860  PRINT CHR$(14)UB$CHR$(15):PRINT:PRINT
2870  FOR R=0 TO Z-1
2880  FOR F=0 TO FA-1:IF SF(F)-0 THEN 2910
2890  IF FM THEN PRINT:PRINT"[RVS]"M$(F)" : [RVOFF]"
2900  PRINT E$(F, GF%(R))
```

```
2910  NEXT:IF DV THEN 2930
2920  GET A$:IF A$(  )""THEN POKE 198, 0:WAIT 198, 1:GET A$
2930  PRINT:PRINT:NEXT
2940  IF DV THEN PRINT#4:CLOSE 4:RETURN
2950  PRINT:PRINT" ⟨⟨ RETURN ⟩⟩"
2960  GOSUB 1010:IF A(  )13 THEN 2960
2970  RETURN
 ********************************************************************
2980  REM "SORT"
 ********************************************************************
2990  IF Z=0 THEN RETURN
3000  GOSUB 3640
3010  FOR I=0 TO FA-1:PRINT I+1"[LEFT] "M$(I):NEXT
3020  PRINT"[DOWN]ON WHAT FIELD";:IF SP⟨40 THEN PRINT:PRINT
3030  INPUT"SHOULD SORT TAKE PLACE";SF:SF=SF-1
3040  F1=0:FH=0:FH=PEEK(49)+PEEK(50)*256 :REM "SAVE UPPER LIMIT"
3050  F1=PEEK(47)+PEEK(48)*256: REM "SAVE LOWER LIMIT"
3060  DIM N$(Z), Z(Z+1)
3070  Q=1:FOR I=0 TO Z-1:REM "RE-STORE"
3080  N$(Q)=E$(SF, GF%(I)):REM "SORT FIELD"
3090  Z(Q)=Q:Q=Q+1:NEXT
3100  GOSUB 3240:REM "SORT ROUTINE"
3110  FOR X=0 TO FA-1
3130  FOR Y=0 TO Z-1
3140  N$(Y+1)=E$(X, GF%(Y))
3160  NEXT
3170  FOR Y=0 TO Z-1
3180  E$(X, GF%(Y))=N$(Z(Y+1))
3200  NEXT Y, X
3210  FOR I=0 TO Z-1:N$(I)="":NEXT:REM "DELETE STRINGS FROMN$()
3220  GOSUB 3520:REM "RESTORE OLD POINTER"
3230  RETURN
 ********************************************************************

3235  REM "SORT ROUTINE"
 ********************************************************************
3240  TI$="000000":PRINT"[CLEAR]"
3250  L=INT(Z/2)+1
3260  R=Z:PRINT"[CLEAR]"R"[LEFT, SPACE3]"
3270  IF L)1 THEN 3340
3280  IF R(=1 THEN 3330
3290  H2$=N$(L):H1=Z(L)
```

```
3300  N$(L)=N$(R):Z(L)=Z(R)
3310  N$(R)=H2$:Z(R)=H1
3320  R=R-1
3330  GOTO 3350
3340  L=1-1
3350  J=1
3360  I=2*J
3370  H2$-N$(J):H1=Z(J)
3380  IF I)R THEN 3480
3390  IF I)=R THEN 3420
3400  IF N$(I))=N$(I+1)THEN 3420
3410  I=I+1:PRINT"[CLEAR]"I"[LEFT, SPACE3]"
3420  IF I(R THEN 3480
3430  IF H2$)=N$(I)THEN 3480
3440  N$(J)=N$(I):Z(J)=Z(I)
3450  J=I
3460  I=2*J
3470  GOTO 3390
3480  N$(J)=H2$:Z(J)=H1
3490  IF R<>1 THEN 3270
3500  PRINT"[CLEAR, DOWN2]SORT TIME";TI$
3510  RETURN
 ************************************************************************
3515  REM "RESTORE POINTER"
 ************************************************************************
3520  J=PEEK(47)+PEEK(48)*256+(FH-F1)
3530  POKE 49, J AND 255:POKE 50, INT(J/256)
3540  RETURN
 ************************************************************************
3545 REM "HEADINGS"
 ************************************************************************
3550  PRINT"[CLEAR]";:PRINT TAB(SP/2-10)"*SET UP FILE*":RETURN
3560  PRINT"[CLEAR]";:PRINT TAB(SP/2-10)"[WHITE] * * * M E N U * * *
        ":RETURN
3570  PRINT"[CLEAR]";:PRINT TAB(SP/2-9)"*MAINTAIN FILE*" :RETURN
3580  PRINT"[CLEAR, BLUE]";:PRINT TAB(SP/2-10)"*ENTER
        DATA*":RETURN
3590  PRINT"[CLEAR]";:PRINT TAB(SP/2-8)"*STORE DATA*" :RETURN
3600  PRINT"[CLEAR";:PRINT TAB(SP/2-6)"*LOAD DATA*" :RETURN
3610  PRINT"[CLEAR]";:PRINT TAB(SP/2-8)"* * PAGE * *":RETURN
3620  PRINT"[CLEAR]";:PRINT TAB(SP/2-8)"SEEK/CHANGE" :RETURN
3630  PRINT"[CLEAR]";:PRINT TAB(SP/2-7)"OUTPUT LIST" :RETURN
```

```
3640  PRINT"[CLEAR]";:PRINT TAB (SP/2-7)"SORTLIST":RETURN
 ********************************************************************
3645  REM "$"OUTPUT FIELDS"
 ********************************************************************
3650  PRINT"[HOME, DOWN5]":FOR F=0 TO FA-1 :PRINT"[RVS]"M$(F)" : "
3660  PRINT E$(F, R1):NEXT:RETURN
3670  PRINT"[HOME, DOWN5]":FOR F=0 TO FA-1 :PRINT"[RVS]"M$(F)" : "
3680  PRINT E$(F, R1)"[UP]":A=LEN(E$(F, R1))
3690  IF A>SP THEN FOR X=0 TO A/SP:PRINT"[UP]";:NEXT
3700:  INPUT#1, E$(F, R2):PRINT:NEXT:RETURN
 ********************************************************************
3705  REM "SELECT SEARCH FROM"
 ********************************************************************
3710  PRINT LEFT$(DW$, ZE-1)"[RVS]1:T/2:=/3:( )/4:)/5 :([RVOFF]";
3720  GOSUB 1010:J=VAL(A$):IF J<1 OR J>5 THEN 3720
3730  SF(I-1)=J
3740  PRINT LEFT$(DW$, ZE-1)"  ";:RETURN
 ********************************************************************
3750  REM "HELP PAGE"
 ********************************************************************
3670  PRINT"[CLEAR]* * * * * * * HELP PAGE * * * * * * *"
3770  PRINT:PRINT" IF << RETURN >> APPEARS, YOU HAVE
3780  PRINT:PRINT"OPTIONS:":PRINT:PRINT
3790  PRINT"RETURN = CONTINUE":PRINT
3800  PRINT"K = REPEAT LAST ENTRY":PRINT
3810  PRINT"Q = BREAK OF PROGRAM POINT":PRINT
3820  PRINT"- = ONE RECORD BACK":PRINT
3830  PRINT"A = CHANGE A RECORD":PRINT
3840  PRINT"L = DELETE A RECORD":PRINT
3850  GET A$:IF A$=""THEN 3850
3860  RETURN
5500  "                                                              *
5010  "*FILE ADMINISTRATION WITH FASTTAPE                            *
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                             *
5010  "* SP             = NUMBER OF COLUMNS                         *
```

```
5110  "* ZE          = NUMBER OF LINES                    *
5120  "* A$          = QUESTION STRING                    *
5130  "*FL           =FLAG                                *
5140  "* E$(F, R)    = DATA STRING                        *
5150  "* SF(F)       = SEARCH FIELD FLAG                  *
5160  "*             1 : SEEK PARTIAL STRING              *
5170  "*             2 : ENTIRE'='                        *
5180  "*             3 : ENTIRE' '                        *
5190  "*             4 : ENTIRE'>='                       *
5200  "*             5:ENTIRE'<='                         *
5220  "* S$(F)       = SEARCH STRING                      *
5230  "* U$(F)       = EXAMINATION STRING                 *
5240  "* GF%()       = NUMBER OF DATA RECORDS             *
5250  "*             FOUND                                *
5300  "                                                   *
```

13.1 Program Description.

The program structure is as follows:

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

Mode of Operation

Adaption 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 ColorPOKE instructions, (C1, C2), corresponding to these values.

The main program consists solely of the menu and the branch to the selected subroutine. For data security purposes, a flag FL was introduced; it is always tested at the beginning of a subroutine and set during the subroutine. The individual bits of FL 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 FL is set in line 1770.

Bit zero in FL 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 over written by RETURN.

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

Enter Data (1860—1990).
In line 1865, FL 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 in the output. For safety 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$—"q"), correct (a$ = "k"), 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 by way of the holding loop (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 FL and screen POKE have been checked, the search flags (sf (i)) are set to zero (2150). Starting at line 2190, the computer checks via the holding loop at 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 by d6 by line 2240 and the field description of the required field is printed out by line 2250.

From line 3710, the index for the required searchform is read into the corresponding search flag via subroutine "Select Search Form".

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 53 fields#$(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, in order to optimise the 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 64 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, which 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, analogusly to the subroutine "Paging" (972430—2550).

OUTPUT LIST(2690—2970).

This routine outputs the data records determine 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 enquiry as to whether a list was drawn up, and also after the output of the heading on the screen (2990—3000), the field on which the sort is to be performed is checked and the corresponding index written into sf.

A number of auxillary 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 initialisation. That would use up a great deal of space on the one hand, 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 errors 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 than 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) = Max	z (1) = 1
n$ (2) = Alfred	z (2) = 2
n$ (3) = Peter	z (3) = 3
n$ (4) = Dick	z (4) = 4

After sorting, we have:

n$ (1) = Alfred	z (1) = 2
n$ (2) = Dick	z (2) = 4
n$ (3) = Max	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 fields. 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 field table.

I hope that the working of this program will now be rather clearer to you and you now have some idea as to how and where the program can be adapted to suit your own particular requirements. For example, the following are a few examples of such adaptations:

- printed output: formatted output, corresponding to the file

- help page: this screen output could be written on a separate screen page which could then be exchanged with the actual screen page using, POKE instructions. This adjustment could be of particular interest to C-64 users; the additional screen page could be written below the operating system ROM (from $A000), so that no Basic storage space is lost

14. CASSETTE CONTENTS AND CATALOGUE OF FASTTAPE CASSETTES

In Chapter 7, I described a program which will set up a catalogue 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 ******************************************************
 95 REM*                                                     *
100 REM*       CATALOGUE OF CONTENTS                         *
101 REM*                                                     *
103 REM*       OF FASTAPE CASSETTES                          *
104 REM*                                                     *
105 REM*       (C) DIRK PAULISSEN                            *
106 REM*                                                     *
107 REM* ******************************************************
108 REM V=.7:KL=.54 : REM CONSTANTS FOR COUNTER CALCULA-
    TIONS C60
109 V=.5:KL=.482: REM CONSTANTS FOR COUNTER CALCULATIONS
    C90
110 PA=828: REM START ADDRESS IN CASSETTE BUFFER
```

```
115 SA=171 :REM SECONDARY ADDRESS +1
116 PRINT"[CLEAR, SPACE11]CATALOGUE OF CONTENTS"
117 PRINT:PRINT" FOR FASTTAPE CASSETTES"
120 PRINT"[DOWN2]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":T5$=TA$+"35"
    142 T6$=TA$+"43":T7$=TA$+"61":REM TAB'S
145 IF D$( )"Y"THEN 170
150 OPEN2, 4 :REM OPEN PRINTER
160 PRINT#2, T1$"LFN";T2$"COUNTER";T3$"K-BYTE";T4$"START"T5$"
    END";
162 PRINT#2, T6$" NAME"T7$"SEC.ADDR."
165 PRINT#2
190 SYS 49674:REM VIC20 = 29183
200 A$=":"+RIGHT$("    "+STR$(PEEK(PA)+(256*PEEK(PA+1))-1), 6)
210 B$="C"+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/
    1024*1000+.5)/1000)
235 K$=RIGHT$("      "+STR$(INT(T/1024*1000+.5)/1000), 6)
240 SA$=STR$(PEEK(SA)-1)
250 PRINT Z;K$A$B$:PRINT C$:PRINT
252 FA=1+2↑(Z/4400):REM DEPENDENT FACTOR FOR COUNTER CAL-
    CULATION
255 Z1=Z:Z=INT(Z+V*FA+(T/1024)*(KL*FA)) REM CALCULATE TAPE
    COUNTER
256 PRINT"NEXT COUNTER READING CA."Z
260 N×N+1:N$=RIGHT$("    "+STR$(N), 3)
270 IF D$="Y"THEN PRINT #2,
    T1$;N$;T2$;Z1;T3$;K$;T4$;A$;T5$;B$;T6$;C$;T7$SA$
300 GOTO 190
```

The cataloguing 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 synchronisation is considerably shorter in FastTape programs than in programs stored in the normal way, it is advisable not to store the program immediately following one another, but to leave a 5 sec. 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 disk. It is therefore worth your while, even if you have a disk 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 disk drives. Using the following programs, it will now be much easier to exchange programs between tape and disk.

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 disk.

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!.

Enter the program in your computer and commit it to storage before starting, as it will delete itself automatically after start-up. When you start the program, the machine code program is first written into storage behind the FastTape program, starting at $C3CO ($73CO). You will then be asked for the number of programs to be copied. This value is held in intermediate storage location 767 for the machine code program.

The next thing to do is to insert a formatted disk 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 code 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 RAM to disk. 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 an more programs, you will have to store the number of

programs in question in store location 767, and start the machine code program again by means of SYS 50112 (SYS 28672 in the VIC 20).

```
 85: C3C0                    .PAGE 61
 86: C3C0                    .TIT "BACKUP T=)D"
 90: C3C0                    .OPT P2, O1
100: C3C0                    *= $C3C0 :VIC20=$73C0
110: C3C0
110: C3C0
115:
120:
130:            BACKUP FOR CC ON DISKETTE
135:
140:               WITH FASTAPE
145:
146:
150: C3C0
150: C3C0
150: C3C0
150: C3C0
155:            ; EXPRESSIONS IN BRACKETS APPLY
157:            ;              TO 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   NUMBER   =   $2FF
245: 0803   CSTART   =   $803        ; ($1203) START OF BUFFER
250 C18E    LOADR    =   $C18E       ; ($7188) LOAD FASTTAPE
260: A642   NEW      =   $A642       ; ($C642) BASIC INSTRUCTION
                                                NEW
270: A537   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
 950: C3C0
 950: C3C0
 955: C3C0
 955: C3C0
 960:               ; MAIN PROGRAM
 970: C3C0
 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 NUMBER
1050: C3C3 F0 03                BEQ END
1060: C3D0 4C C0 C3             JMP  START
1070: C3D3 60           END     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      ; OUTPUT 'SAVING
                                                 TIME
1100: C3E0 A9 0D                LDA #CR
1110: C3E2 20 D2 FF             JRS  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
                                              NAME
1200:  C3F4 A9 2C              LDA #", "    ; AND APPEND
1210:  C3F6 9D 41 03           STA CASPUF+5, X
1220:  C3F9 38                 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:  C408 E8                 INX
1320:  C40C 8A                 TXA          ; LENGTH OF FILE
                                              NAME
1330:  C40D A2 41              LDX #$41     ; START ADDRESS OF
                                              FILE NAMES
1340:  C40F A0 03              LDY #$03
1350:  C411 20 BD FF           JSR SETNAM
1360:  C414 A9 08              LDA #$08     ; FILE NUMBER
1370:  C416 A2 08              LDX #$08     ; DEVICE NUMBER
1380:  C418 A0 01              LDY #$01     ; SECONDARY
                                              ADDRESS
1390:  C41A 20 BA FF           JSR SETLFS
1400:  C41D 20 C0 FF           JSR OPEN
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
                                              ADDRESS FROM
1480:  C430 3B                 SEC          ; DIFFERENCE BE-
                                              TWEEN START AND
1490:  C432 ED 3C 03           SBC CASPUF   ; END ADDRESSES
                                              FROM
1550:  C434 08                 PHP            HEADER PLUS
                                              CERTAIN START
                                              ADDRESS
1510:  C435 18                 CLC
```

```
1520:  C436 65 AC                ADC STARTV
1530:  C438 85 AE                STA  ENDVEC
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
1600:  C448 20 D2 FF             JSR  BSOUT    START ADDRESS 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),
1650:  C455 20 D2 FF                JSR  BSOUT
1660:  C458 E6 AC                   INC  STARTV  ;INCREMENT ADDRES
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 ADDRESS
                                                   REACHED
1710:  C462 A5 AD                   LDA STARTV+1
1720:  C464 E5 AF                   SBC  ENCVEC+1
1730:  C466 90 E9                   BCC  PSAVE
1740:  C468 20 CC FF                JSR  CLRCH
1750:  C46B A9 0B                   LDA #$08
1760:  C46D 4C C3 FF                JMP  CLOSE
1762:  C4701
1762:  C4701
1764:  C4701
1764:  C4701
1766:
1768:  C470                ; LOAD FROM CC
1768:  C470
1770:  C470 A2 00          CLOAD    LDX #$00
1780:  C472 A0 03                   LDY #⟨CSTART
1790:  C474 A9 08                   LDA #⟩CSTART
```

```
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 CERTAIN
                                                    ADDRESS
1850:  C480 85 02                  STA  ABSFLAG ; 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 #$00       ; SA
1900:  C48B 20 BA FF               JSR  SETLFS
1910:  C48E 20 BE C1               JSR  LOADR
1922:  C491
1922:  C491
1924:                ; TEST STATUS
1926:  C491
1926:  C491
1930:  C491 20 B7 FF      HOLSTA   JSR  STATUS
1940:  C494 BF                     AND #$BF
1950:  C496 F0 05                  BEQ OK
1960:  C498 A2 1D                  LDX #$1D
1970:  C49A 4C 37 A4               JMP  ERROR
1980:  C49D 60            OK       RTS
] C3CO-C49E
```

```
100  GOSUB 240
110  PRINT"[CLEAR] ***************************** [LEFT2,DOWN]*[UP]*"
120  PRINT"[UP]*BACKUP WITH FASTTAPE **[LEFT2,DOWN]*";
130  PRINT"* FROM CA TO DISK **[LEFT2, DOWN]*";
140  PRINT"*(C) DIRK PAULISSEN **[LEFT2,DOWN]*";
150  PRINT"**********************"
160  PRINT:PRINT:PRINT"HOW MANY PROGRAMS DO YOU WANT"
170  PRINT:INPUT"TO COPY";AZ
180  POKE 767, AZ
190  PRINT:PRINT:PRINT"INSERT SOURCE CASSETTE AND"
200  PRINT"[DOWN] PRESS ⟨RETURN⟩"
210  GET A$:IF A$ ⟨ ⟩CHR$(13)THEN 210
220  SYS 29632
230  END
*********************************************************************
240  REM LOAD PROGRAM FASTCOPY T-D 20
*********************************************************************
```

```
250 E=29853:A=29632:PS=0
260 FOR 1=A TO E:READ X:POKE I, X:PS=PS+X:NEXT
280 IF PS.<>#.27173 THEN PRINT"ERROR IN DATA":END
290 RETURN
300 DATA 169, 0, 32, 66, 198, 32, 112, 116, 32, 212, 115, 206, 255, 2, 240, 3, 76, 192,
    115, 96
310 DATA 169, 16, 162, 65, 160, 3, 32, 189, 255, 32, 44, 247, 169, 13, 32, 210, 255,
    162, 16, 202
320 DATA 189, 65, 201, 32, 201, 208, 4, 224, 0, 208, 244, 232, 169, 44, 157, 65, 3,
    232, 169, 80
330 DATA 157, 65, 3, 232, 169, 44, 157, 65, 3, 232, 169, 87, 157, 65, 3, 232, 138, 162,
    65, 160
340 DATA 3, 32, 189, 255, 169, 8, 162, 8, 160, 1, 32, 186, 255, 32, 192, 255, 162, 8,
    32, 201, 255
350 DATA 169, 3, 162, 18, 133, 172, 134, 173, 173, 62, 3, 56, 237, 60, 3, 8, 24, 101,
    172, 133
360 DATA 174, 173, 63, 3, 101, 173, 40, 237, 61, 3, 133, 175, 173, 60, 3, 32, 210, 255,
    173, 61
370 DATA 3, 32, 210, 255, 160, 0, 177, 172, 32, 210, 255, 230, 172, 208, 2, 230, 173,
    165, 172
380 DATA 197, 174, 165, 173, 229, 175, 144, 233, 32, 204, 255, 169, 8, 76, 195, 255,
    162, 0, 160
390 DATA 3, 169, 18, 134, 10, 134, 147, 132, 172, 133, 173, 169, 5, 133, 2, 169, 0, 32,
    189, 255
400 DATA 162, 1, 160, 0, 32, 186, 255, 32, 136, 113, 32, 183, 255, 41, 191, 240, 5,
    162, 29, 76
410 DATA 55, 196, 96

100 GOSUB 240
110 PRINT"[CLEAR] ****************************** [LEFT,DOWN]*[UP,]*"
120 PRINT"* BACKUP WITH FASTTAPE **[LEFT2,DOWN]*";
130 PRINT"* FROM CASSETTE TO DISKETTE **[LEFT2,DOWN]*";
140 PRINT"* (C) DIRK PAULISSEN **[LEFT2,DOWN]*";
150 PRINT" ************************************************** "
160 PRINT:PRINT:PRINT"HOW MANY PROGRAMS DO YOU WANT
170 PRINT:INPUT"TO COPY        ";AZ
180 POKE 767, AZ
190 PRINT:PRINT:PRINT"INSERT SOURCE CASSETTE AND "
200 PRINT"[DOWN] PRESS <<RETURN>>"
210 GET A$:IF A$<>CHR$(13)THEN 210
220 SYS 50112
230 END
```

```
*************************************************************************
240  REM LOAD PROGRAM FASTCOPY T-D 64
*************************************************************************
250  E=50333:A=50112:PS=0
260  FOR I=A TO E:READ X:POKE I, X:PS=PS+X:NEXT
270  IF PS<>27517 THEN PRINT"ERRORS IN DATA ":END
280  RETURN
290  DATA 169, 0, 32, 66, 166, 32, 112, 196, 32, 212, 195, 20 6, 255, 2, 240, 3, 76,
       192, 195, 96
300  DATA 169, 16, 162, 65, 160, 3, 32, 189, 255, 32, 147, 246, 169, 13, 32, 210, 255,
       162, 16
310  DATA 202, 189, 65, 3, 201, 32, 208, 4, 224, 0, 208, 244, 232, 169, 44, 157, 65, 3,
       232, 169
320  DATA 80, 157, 65, 3, 232, 169, 44, 157, 65, 3, 232, 169, 87, 157, 65, 3, 232, 138,
       162, 65
330  DATA 160, 3, 32, 189, 255, 169, 8, 162, 8, 160, 1, 32, 186, 255, 32, 192, 255, 162,
       8, 32, 201
340  255, 169, 3, 162, 8, 133, 172, 134, 173, 173, 62, 3, 56, 237, 60, 3, 8, 24, 101, 172,
       133
350  DATA 174, 173, 63, 3, 101, 173, 40, 237, 61, 3, 133, 175, 173, 60, 3, 32, 210, 255,
       173, 61
360  DATA 3, 32, 210, 255, 160, 0, 177, 172, 32, 210, 255, 230, 172, 208, 2, 230, 173,
       165, 172
370  DATA 197, 174, 165, 173, 229, 175, 144, 233, 32, 204, 255, 169, 8, 76, 195, 255,
       162, 0, 160
380  DATA 3, 169, 8, 134, 10, 134, 147, 132, 172, 133, 173, 169, 5, 133, 2, 169, 0, 32,
       189, 255
390  DATA 162, 1, 160, 0, 32, 186, 255, 32, 142, 193, 32, 183, 255, 41, 191, 240, 5,
       162, 29, 76
400  DATA 55, 164, 96
```

15.2 Program Description of Disk Backup for Cassette

For those of your 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 $C3C0 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 storage 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 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 disk is a trifle 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 someother 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 prepartory routines SETNAM, SETLFS, OPEN, CHKOUT, required for storage of disk, 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 disk the first two bytes give the address whence the program was transferred to storage. This address is held in the cassette buffer as the start address. It is transferred to the disk 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 compare 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 main loop.

15.3 Cassette Backup for Disk

This program will enable you to transfer one or more programs from a disk 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 transfer it to storage before starting.

This program will also only run if you have previously loaded the FastTape Operating 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 disk and to depress a key.

After a short time, the disk title will appear and the first file entry in the list of the disk'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 up to 48 programs.

If you should press the wrong key during the program, you can terminate the program by pressing RUN/STOP and return 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 disk, 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".

```
60:  C400                    .OPT P1
                ;  **********************************
                ; *                                  *
                ; *                                  *
                ; *              BACKUP              *
                ; * FROM DISKETTE TO CASSETTE        *
                ; *                                  *
                ; *            WITH FASTTAPE         *
                ; *                                  *
                ;  **********************************
                ;
                ;  START WITH SYS 50176 (SYS 29696)
```

```
134:  C400
                ;VALUES AND COMMENTS IN
136:  C400
                ;BRACKETS APPLY TO VIC 20
                ;
142:  C400              *= $C400  ;     ($7400)
145:  C400
145:  C400      START   = *
150:  C400      CLSCRN  = $E544         ;($E55F) CLEAR SCREEN
152:  C400      NUMOUT  = $BDCD         ;($DDCD) OUTPUT POSITIVE
                                          INTEGER NO
154:  C400      MOTAUS  = $FCCA         ;($FD08) SWITCH OFF MOTOR
156:  C400      PTASTE  = $F838         ;($F8B7) CHECK RECORDER
                                          SWITCH
158:  C400      SFINUM  = $F30F         ;($F3CF) SEEK FILE NUMBER
160:  C400      STROUT  = $AB1E         ;($CB1E) OUTPUT STRING
162:  C400      SETPAR  = $F31F         '($F3DE) SET FILE
                                        PARAMETERS
164:  C400      ABSOLUT = $C085         ;($707F) JUMP TO FASTTAPE
169:  C400
169:  C400
169:  C400
170:  C400      GETBYT  = $FFE4         ; INPUT BYTE
180:  C400      SETLFS  = $FFBA         ; SET FILE UP
190:  C400      SETNAM  = $FFBD         ; SET NAME
200:  C400      OPEN    = $FFC0
210:  C400      CLOSE   = $FFC3
220:  C400      TALK    = $FFB4
230:  C400      SETALK  = $FF96         ; SET SECONDARY ADDRESS
                                          TO TALK
240:  C400      IECIN   = $FFA5         ; FETCH BYTE
250:  C400      CLALL   = $FFE7
260:  C400      UNTALK  = $FFAB         ; UNTALK
280:  C400      STOP    = $FFE1         ; CHECK STOP KEY
290:  C400      BASOUT  = $FFD2         : OUTPUT BYTE
300:  C400
300:  C400
300   C400
320:  C400      GA      = $BA           ; DEVICE ADDRESS
330:  C400      SA      = $B9           ; S.A.
340:  C400      MAXBLK  = $98           ; MAX BLOCKS ($5D)
350:  C400      CR      = $0D           ; RETURN
```

```
360:  C400      COPANZ  = $FB           ; LIST LENGTH
370:  C400      TEMP    = $FC           ; BUFFER
380:  C400      FLAG    = $FE           ; FLAG
390:  C400      LISTPI  = $41           ; POINTER TO NAME LIST
400:  C400      FILPUF  = $340          ; BUFFER
410:  C400      LNGTAB  = START+$640    ; FILE NAMES
420:  C400      NAMTAB  = START+$700    ; FILE LENGTHS
430:  C400      LSTMAX  = $30           ; LIST LENGTH
440:  C400      TLSADR  = $09           ; HIBYTE LOAD ADDRESS
450:  C400      TEMP2   = $22           ; AUX POINTER
460:  C400      TPGSTA  = $AC           ; START
470:  C400      TPGEND  = $AE           ; END
480:  C400      PRGSTA  = $A7           ; START
490:  C400      PRGEND  = $A9           ; END
495:  C400      MOFLAG  = $C0           ; FLAG
                  ;
                  ;
560:  C400 A9 01                LDA #1          ; (25)
570:  C402 8D 20 D0             STA $D020       ; (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 6B                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
                        ; INHALT EINLESEN
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       ; INIT. DISK
740:  C42C A9 08                LDA #$08
750:  C42E AA                   TAX
```

```
 760:  C42F A0 00                LDY #$00
 770:  C431 20 BA FF             JSR SETLFS
 780:  C434 A9 01                LDA #$01
 790:  C436 A2 E1                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 SETALK
 870:  C44A A0 04                LDY #$04
 880:  C44C 20 A5 FF    LOOP1    JSR IECIN     ; READ FIRST 4 B
 890:  C44F 88                   DEY
 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 17
 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
                  ; INHALT AUSGEBEN UND FRAGEN
1073:  C47D
1073:  C473
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
```

```
1030:  C489 20 CD BD          JSR  NUMOUT ; OUTPUT BLOCK NO
1040:  C48C 20 B3 C7          JSR  SPACE
1050:  C48F A0 00             LDY #$00      ; READ IN
1060:  C491 20 A5 FF  WRFILE  JSR  IECIN    ; FILE NAMES
1070:  C494 48                PHA ; PUT IN BUFFER
1080:  C495 20 BB C7          JSR  AUSGAB ; OUTPUT THEM
1090:  C498 68                PLA
1200:  C499 99 40 03          STA FILPUF, Y
1210:  C49C F0 03             BEQ NAMEND
1220:  C49E CB                INY
1230:  C49F D0 FO             BNE WRFILE
1240:  C4A1 20 A5 FF  NAMEND  JSR  IECIN
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  FRAGEN     LDA TEMP+1
1300:  C4BO DO 06             BNE TOLONG
1310:  C4B2 A5 FC             LDA TEMP
1320:  C4B4 C9 98             CMP #MAXBLK
1330:  C4B6 90 OA             BCC LNGEOK  : TEXT MAX LENGTH
1340:  C4B8 A9 FC  TOLONG     LDA #⟨SLONG
1350:  C4BA A0 CB             LDY #⟩SLONG
1360:  C4BC 20 IE AB          JSR  STROUT  ; OUTPUT MESSAGE
1370:  C4BF 4C 41 C5  FINISH  JMP NXFILE
1300:  C3C2 A5 FB     LNGEOK  LDA COPANZ
1390:  C4C4 C9 30             CMP #LSTMAX
1400:  C4 C6 90 OA            BCC FANZOK
1410:  C4CB A9 1B             LDA #⟨LILONG
1420:  C4CA A0 C9             LDY #⟩LILONG
1430:  C4CC 20 IE AB          JSR  STROUT  ; OUTPUT MESAGE
1440:  C4CF 1b                CLC
1450:  C4DO 90 ED             BCC FINISH
1460:  C4D2 A9 OO     FANZOK  LDA #$00      ;
1470:  C4D4 B5 00             STA $08
1400:  C4D6 A9 1F             LDA #$1F      ;
1490:  C4D8 85 D3             STA $D3       ;
1500:  C4DA A9 33             LDA #⟨JANEIN
1510:  C4DC A0 C9             LDY #⟩JANEIN
1520:  C3DE 20 IE AB          JSR  STROUT
1530:  C4E1 20 E1 C7  LOOP3   JSR  INPUT
1540:  C4E4 C9 E4             CMP # "N"
```

```
1550:  C4E6 FO 52                BEQ NEIN
1560:  C4E8 C9 4A                CMP # "Y"
1570:  C4EA DO F5                BNE LOOP3
1500:  C4EC A9 45                LDA #<STRJA
1590:  C3EE A0 C9                LDY #>STRJA
1600:  C4FO 20 1E AB             JSR  STROUT
1610:  C4F3 A5 FB                LDA COPANZ
1620:  C3F5 20 F1 C6             JSR  SETPTR
1630:  C4FB A2 00                LDX #$00
1640:  C4FA EB          LOOP4    INX             ; FILENAME START
1650:  C4FB BF 40 03             LDA FILPUF, X
1660:  C4FE C9 22                CMP #$22        ; 1ST "," = START
1670:  C500 D0 FB                BNE LOOP4
1680:  c502 86 FD                STX TEMP+1   ; START RETTEN
1690:  C504 E8                   INX
1700:  C505 BD 40 03    UMSETZ   FILPUF, X:
1710:  C50B C9 22                CMP #$22        ; 2ND "1" = END
1720:  C50A F0 06                BEQ UMSEND
1730:  C50C 91 41                STA (LISTPT), Y
1740:  C50E EB                   INX
1750:  C50F CB                   INY
1760:  C510 D0 F3                BEN UMSETZ
1770:  C512 BA          UMSEND   TXA             ; END IN ACC
1780:  C513 A4 FB                LDY COPANZ   ; INDEX IN Y
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:  C5E1 D0 0A                BNE TYPTST
1840:  C520 A9 41       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
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 6B        NEIN     LDA #⟨STRNO
1980:  C53C A0 C9                 LDY #⟩STRNO
1990:  C53E 20 IE AB              JSR  STROUT
2000:  C541 20 B6 C7     NXFILE   JSR  CARET
2010:  C544 4C 7D C4              JMP  LSTART
2020:  C547 A9 OB        INHEND   LDA #$08      ; CLOSE
2030:  C549 20 C3 FF              JSR  CLOSE
2040:  C54C A5 FB                 LDA COPANZ   ;
2050:  C5864E D0 03               BNE COPYST   ;
2060:  C550 4C 00 C4              JMP  START
2062:  C553
2062:  C553
2062:  C553
                    ;COPIEREN VORBEREITEN
2066:  C553
2066:  C553
2066:  C553

2070:  C553 A9 77        COPYST   LDA #⟨FRAGE
2080:  C555 A0 C9                 LDY #⟩FRAGE
2090;  C557 20 IE AB              JSR  STROUT  OUTPUT MESSAGE
2110:  C55D C9 31                 CMP #"1"
2120:  C55F F0 07                 BEQ EINZEL
2130:  C561 C9 32                 CMP #"2"
2140:  C563 D0 F5                 BEN 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
2202:  C571
                2202:  C571       ;COPIER ROUTINE
2206:  C571
2206:  C571
                2206:  C571       ;EINLESEN VON DISK
                                  ; VORBEREITEN
2210:  C571
2210:  C571 A2 00                 LDX #$00     ; TEMP DELETED
2220:  C573 86 FC                 STX TEMP     ; TEMP =⟩PRF
                                                 COUNTER
```

```
2230:  C575 A9 C7      LOLOOP  LDA #⟨SREAD ; "READING"
2240:  C577 A0 C9              LDY #⟩SREAD
2250:  C579 20 IE AB           JSR  STROUT   ; OUTPUT
2260:  C57C A4 FC              LDY TEMP      ; POS. IN TABLE
2270:  C57E BE 40 CA           LDX LNGTAB, Y; LENGTH IN X
2280:  C581 A5 FC              LDA TEMP
2290:  C583 20 F1 C6           JSR  SETPIR
2300:  C586 B1 41      WRTNAM  LDA (LISTPT), Y; FILENAME
2310:  C588 20 D2 FF           JSR  BASOUT
2320:  C588 C8                 INY
2330:  C58C CA                 DEX
2340:  C58D D0 F7              BNE WRTNAM
2350:  C58F A9 02              LDA #$02      ; FILE
2360:  C59 A2 08               LDX #$08      ;DA
2370:  C593 AB                 TAY           ;SA
2380:  C594 20 BA FF           JSR  SETLFS
2390:  C597 A6 FC              LDX TEMP
2400:  C599 BD 40 CA           LDA LNGTAB, X
2410:  C59X 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
                                          BUFFER
2460:  C5A7 9D 40 03           STA  FILPUF, X:
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 D2 C9   LOOP7   LDA PRGRE, Y; ", P, R" APPEND
2530:  C5B5 9D 40 03           STA  FILPUF, X
2540:  C5B8 C8                 INY
2550:  C589 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
```

```
2640:  C5CB A0 00          LDY #$00
2650:  C5CD 8 4 22         STY TEMP2
2660:  C5CF 85 23          STA TEMP2+1
2670:  C5D1 AD 11 D0       LDA #D011     ;(LINES2670-2690
                                           DROPPED
2680:  C5D4 29 EF          AND #$EF      ;OPEN SCREEN FROM
2690:  C5D6 8D 11 D0       STA #D011     ;  CHANNEL 15
2700:  C5C9 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-"DISK
2750:  C5E3 A2 D7          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          ;PROGRAM IN FFER L
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     ;(DRGPPED)
2960:  C617 09 10           ORA #$10      ;(DROPPED)
2970:  C619 8D 11 D0        STA #D011     ;(DROPPED)
2980:  C61C 28              PLP
2990:  C61D 90 03           BCC NOFEHL
```

```
3000:  C61F 20 37 C7            JSR  FEMELD
3002:  C622
3002:  C622             :STORE ON CC
3006:  C622
3006:  C622
3010:  C622 A5 FC       NOFEHL  LDX TEMP
3020:  C624 A5 22               LDA TEMP2
3030:  C626 85 AE               STA  TPGEND
3040:  C528 A5 23               LDA TEMP2+1
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 4E               LDA #⟨ZICASS ;"(NSERTCASSETTE"
3090:  C633 A0 C8               LDY #⟩ZICASS
3100:  C638 20 1E A8            JSR  STROUT
3105:  C63B 20 E1 C7            JSR  INPUT
3110:  C63B 20 25 C8            JSR  TASTE      ; CHECK RECORDER
                                                   KEY
3115:  C53E A9 E3       NOMELD  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       WRNAM2  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:  C65C 20 BA FF            JSR  SETLFS
3250:  C65D A6 FC LDX           LDX TEMP
3260:  C660 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       ULOOP   LDA (LISTPT), Y; FILENAME IN
```

```
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          ;FILENAME
                                       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;PRFSTARTADR
3450:  C68B 85 A7       STA  PRGSTA  ; IN PRGSTART
                                       VECTOR
3460:  C68D CB          INY
3470:  C68E B1 AC       LDA (TPGSTA), Y
3480:  C690 85 AB       STA  PRGSTA+1
3490:  C692 C8          INY
3500:  C693 84 AC       STY  TPGSTA
3520:  C695 A5 AE       LDA TPGEND   ; END ADDR
3530:  C697 38          SEC
3540:  C698 E5 AC       SBC  TPGSTA
3550:  C69A  08         PHP
3555:  C69B 18          CLC
3560:  C69C 65 A7       ADC PRGSTA
3570:  C693 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             ;PRG
3630:  C6A9 A2 05       LDX #5       ; SYNCHRONISATION
3640:  C6AB 86 AB       STX $AB      ; REPEATED
3560:  C6AD 20 85 C0    JSR  ABSOLUT ; FASTTAPE
3690:  C6B0 20 E5 C6    JSR  SCRON   ; (DROPPED)
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:  C F6C2 A9 ED    CENDEN    LDA #⟨EDCOP ;"END OF COPYING"
3760:  C6C4 A0 C9                LDY #⟩EDCOP
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   EINZE2    JSR NNAMAU
3820:  C6D2 A9 4E                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:  C6D6 20 B6 C7   NXTFIL    JSR CARET
3880:  C6R2 4C 75 C5             JMP LOLOOP  ; NEXT FILE
3882:  C6E5
3882:  C6E5
3882:  C6E5            :SUBROUTINES
3886:  C6E5
3886:  C6E5
3886:  C6E5            :SCREEN ON
3888:  C645
3888:  C645
3890:  C6E5 20 74 C7   SCRON     JSR SEUIP     ;(LINES 3890-3930
                                                 DROPPED
3900:  C6EB AD 11 D0             LDA $D011
3910:  C6EB 09 10                ORA #$10
3920:  C6ED 8D 11 D0             STA $DO11
3930:  C5F0 60                   RTS
3932:  C6F1
3932:  C6F1
;
                                 ;
3936:  C6F1
3936:  C6F1
3930:  C5F1 A0 00      SETPIR    LDY #$00
3950:  C6F3 0A                   ASL A
3960:  C6F4 0A                   ASL A
3690:  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  LIST PT +1
4070: C706 A0 00              LDY #$00
4080:  C70B 60                RTS
4082:  C709
4082:  C709                 : BYTE IN STORAGE
4806:  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
                          :ASCII-BYTE  CONVERT
4146:  C713
4146:  C713
4150:  C713 20 A5 FF        ASC HEX 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:  C71C 29 0F                   AND #$0F
4240:  C723 05 57                   ORA $57
4250:  C725 60                      RTS
4252:  C726
4252:  C726
                          ;HEX    IN  ASCII-BYTE
                             ;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:  C7NL34 4C D2 FF          JMP  BASOUT  ; OUTPUT
4362:  C737
4362:  C737
                    ;FEHLERMELDUNG AUSGEBEN
4362:  C737
4362:  C737
4370:  C737 AD 11 D0    FEMELD  LDA $DO11     ;(LINES 4370-4390
                                                 DROPPED)
4380:  C73A 09 10               ORA #$10     ;
4390:  C73C 8D 11 D0            STA #D011
4400:  C73F A9 07               LDA #⟨WMACH;"CONTINUE?"
4410:  C741 A0 CA               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:  C73E 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 1A       NNAMAU LDA #⟨SNFILE ;"NEXT FILE"
4510:  C757 A0 CA               LDY #⟩SNFILE
4520:  C759 20 1E AB            JSR STROUT ;
4530:  C75C A5 FC               LDA TEMP ;
4540:  C754 0A          ASL     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    WARNAM3LDA NAMTAB, X ;FILENAME OUTPUT
4620:  C76B 20 D2 FF             JSR  BASOUT
4630:  C76E E8            INX
4640:  C76F 88          DEY
4650:  C770 D0 F6                BNE WRNAM3
4660:  C772 60                   RTS
4662:  C773
4662:  C773
                        ;DISKETTE
                                 ;
4566:  C773
4566:  C773
4670:  C773 AD 11 D0 SEUIP        LDA $D011      ;(LINES 4680-4820
                                                 DROPPED)
4680:  C776 29 EF                AND #$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 A3 DB                LDY#)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
                                 ;
4826:  C799
4826:  C799
4830:  C799 20 E4 FF     WAIT    JSR  GETBYT
4840:  C79C F0 FB                BEQ WAIT
4850:  C79E 60                   RTS
                        ;DISK INITIALISE
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 BA FF              JSR SETLFS

JSR
SETLFS

4900:  C7A9 A9 01              LDA #$01      ; "I"
4910:  C7A9 A2 DF              LDX #⟨STRI
4920:  C7AB A0 C9              LDY #⟩STRI
4930:  C7AD 20 BD FF           JSR SETNAM
4940:  C7B0 4C C0 FF           JMP OPEN
                ;SPACE OR CARRIAGE RETURN
4946:  C7B3
4946:  C7B3
4950:  C7B3 A9 20      SPACE   LDA #$20      ; SPACE
4960:  C7B5 2C                 .BYT $2C
4970:  C7B6 A9 0D      CARET   LDA #CR       ; CARRIAGE RTN
4980:  C7B8 4C D2 FF           JMP BASOUT    ; SEND
4982:  C7BB
4982:  C7BB
                        ;OUTPUT
4986:  C7BB
4986:  C7BB
4990:  C7BB 20 D2 FF   AUSGAB  JSR BASOUT
5000:  C7BE 8A         STASTE  TXA           ;A, X, Y, REGISTER

5010:  C7BF 48                 PHA
5020:  C7C0 98                 TYA
5030:  C7C1 48                 PHA
5040:  C7C2 20 E1 FF           JSR STOP      ; STOP KEY?
5050:  C7C5 18                 CLC
5060:  C7C5 D0 01              BNE WEITER
5070:  C7C8 38                 SEC
5080:  C7C9 68         WEITER  PLA             ; REGISTER
                                               RESTORE
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
                    ;
5136:  C7D0
5136:  C7D0
5140:  C7D0 A2 F6   ENDE        LDX $F6
5150:  C7D2 9A                  TXS
5160:  C7D3 20 E5 C6            JSR SCRON       ; (DROPPED)
5170:  C7D6 A9 0F               LDA $0F
5180:  C7D8 20 C3 FF            JSR CLOSE
5190:  C7DB 20 EF FF            JSR CLALL
5200:  C7DE 4C 00 C4            JMP START
5202:  C7E1
5202:  C7E1
                    ;           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:  CB00 C9 20               CMP #$20
5340:  CB02 08                  PHP
5350:  CB03 90 0B               BCC MELAUS ;
5360:  CB05 48                  PHA
5370:  CB06 20 B6 C7            JSR CARET
5380:  CB09 20 B6 C7            JSR CARET
5390:  CB0C 68                  PLA
5400:  CB0D 20 26 C7            JSR HEXASC
5310:  CB10 20 A5 FF MELAUS     JSR IECIN
5420:  CB13 C9 OD               CMP #CR
5430:  CB15 F0 09               BEQ MELEND
5440:  C818 28                  PLP
5450:  C818 08                  PHP
5460:  C819 90 F5               BCC MELAUS
```

```
5470:  C818 20 D2 FF              JSR BASOUT
5480:  C81E:90 F0 BCC MELAUS
5490:  C81E 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 MOTAUS
5511:  C834
5511:  C834
5511:  C834
                    ; STRINGS
5513:  C834
5513:  C834
5513:  C834
5520:  C834 0D 0D 0DQUEINS        .BYT 13, 13, 13
5530:  C837 12 51 55              .ASC "[GRN]       To CASSETTE
                                   TAPE"
5540:  C84D 00                    .BYT 0
5540:  C84E 0D 0D   ZICASS        .BYT 13, 13
5560:  C850 12 5A 49              .ASC "[RED][RVS]  NO  [ROFF]
                                   [BLU]"
5570:  C869 0D 00                 .BYT 13, 0
5580:  C86B 93 11 1CTITEL         .ASC "PLEASE INSERT A [RVS]DISK
                                  [ROFF]"
5590:  C88C 0D 0D                 .BYT 13, 13
5600:  C88E 1F 20 20              .ASC "PLEASE INSERT [RVS]
                                  CASSETTE[ROFF]"
5602:  C8A5 0D 0D                 .BYT 13, 13
5604:  C8A7 20 20 20              .ASC "[CLR][CUD][RED]          BA
                                  CKUP COPY FROM DISK"
5606:  C8C6 0D 0D 0D              .BYT 13, 13, 13
5608:  C8C9 20 20 20              .ASC "      WITH [RVS][ORA]F A S '
                                   T A P E [ROFF]"
5610:  C8E3 0D 0D 0D              .BYT 13, 13, 13
5620:  C8E6 1E 20 20              .ASC "  (C)  DIRK PAULISSEN ."
5630:  C8FB 00                    .BYT 0
5640:  C8FC 0D       SLONG        .BYT 13
5650:  C8FD 1C 12 21              .ASC "[GRN]       E = END [BLU]]"
```

```
5660:  C91A 00                    .BYT 0
5670:  C91B 0D       LILONG       .BYT 13
5680:  C91C 1C 12 21              .ASC "[RED][RVS]!! FILE TOO
                                  LARGE TO COPY [BLU]"
5690:  C932 00                    .BYT 0
5700:  C933 12 4A    JANEIN       .ASC "[RED][RVS]!! THE LIST IS
                                  FULL[BLU]"
5710:  C944 00                    .BYT 0
5720:  C945 12 1E 20 STRJA        .ASC "[RVS] YES/NO[CUL][CUL][CUL]
                                  [CUL][CUL][CUL][CUL][ROFF][BLU]"
5740:  C951 00                    .BYT 0
5750:  C952 1C 12 21              .ASC "[RVS][GRN] YES [ROFF] [BLU]
5760:  C96A 00                    .BYT 0
5770:  C96B IC 12 20 STRNO        .ASC "[RED][RVS]!! ILLEGAL FILET
                                  YPE [BLU]"
5780:  C976 00                    .BYT 0
5790:  C977 0D 0D 0DFRAGE         .BYT 13, 13, 13:
5800:  C992 OD OD                 .ASC "WANT TO COPY PROGRAMS:)"
5810:  C992 0D 0D                 .BYT 13, 13
5820:  C994 20 20 20              .ASC "1)CONTINUOUSLY"
5830:  C9AB 0D 0D                 .BYT 13, 13
5840:  C9AA 20 20 20              .ASC "2)INDIVIDUALLY"
5850:  C9BA 0D 0D                 .BYT 13, 13
5860:  C9BC 43 4F 50              .ASC "WHICH ONE?"
5870:  C9C6 00                    .BYT 0
5880:  C9C7 0D 0D    SREAD       .BYT 13, 13
5890:  C9C9 52 45 41              .ASC "READING"
5900:  C9D1 00                    .BYT 0
5910:  C9D2 2C 50 2CPRGRE         .ASC ", P, R"
5920:  C9D6 00                    .BYT 0
5930:  C9D7 55 49 2DUIMIN         .ASC "UI-"
5940:  C9DA 00                    .BYT 0
5950:  C9DB 55 49 2BUIPLU         .ASC "UI+"
596-:  C9DE 00                    .BYT 0
5970:  C9DF 49         STRI       .ASC "I"
5980:  C9E0 00                    .BYT 0
5985:  C9E1 24         CATALO     .ASC "$"
5986:  C9E2 00                    .BYT 0
5990:  C9E3 0D 0D    WRITI       .BYT 13, 13
6000:  C9EF 53 41 56              .ASC "SAVING"
6010:  C9EC 00                    .BYT 0
6020:  C9ED 0D 0D    EDCOP       .BYT 13, 13
```

```
6030:  C9EF 1C 12 45          .ASC "[WHT][RVS] COPYING
                              COMPLETED[BLU]"
6040:  CA06 00                .BYT 0
6050:  CA08 20 57 45          ..ASC "WANT TO CONTINUE?"
6060:  CA19 00                .BYT 0
6070:  CA1A OD ODSNFILE       .BYT 13, 13
6080:  CA1C 4E 41 45          .ASC "NEXT FILE:"
6090:  CA2D 00                .BYT 0
```

```
**********************************************************************
100 REM BACKUP DISKETTE =) CASSETTE FOR C-64
**********************************************************************

110 E=51575:A=50176:PS=0
120 FOR I=A TO E:READ X:POKE I, X:PS=PS+X:NEXT
130 IF PS<>171785 THEN PRINT"ERROR IN DATA":END
140 SYS A:NEW
150 DATA 169, 1, 141, 32, 208, 141, 33, 208, 169, 6, 141, 134, 2, 169, 107, 160, 200,
    32, 30, 171
160 DATA 169, 52, 160, 200, 32, 30, 171, 32, 153, 199, 201, 69, 208, 1, 96, 32, 182,
    199, 32, 182
170 DATA 199, 32, 159, 199, 169, 8, 170, 160, 0, 32, 186, 255, 169, 1, 162, 225, 160,
    201, 32
180 DATA 189, 255, 32, 192, 255, 169, 8, 32, 180, 255, 169, 0, 32, 150, 255, 160, 4,
    32, 165, 255
190 DATA 136, 208, 250, 32, 165, 255, 133, 34, 32, 165, 255, 166, 34, 32, 205, 189,
    32, 179, 199
200 DATA 32, 165, 255, 240, 6, 32, 210, 255, 24, 144, 245, 32, 182, 199, 32, 182, 199,
    32, 165
210 DATA 255, 32, 165, 255, 160, 0, 132, 251, 32, 165, 255, 133, 252, 32, 165, 255,
    133, 253
220 DATA 166, 252, 32, 205, 189, 32, 179, 199, 160, 0, 32, 165, 255, 72, 32, 187, 199,
    104, 153
230 DATA 64, 3, 240, 3, 200, 208, 240, 32, 165, 255, 32, 165, 255, 165, 144, 240, 3,
    76, 71, 197
240 DATA 165, 253, 208, 6, 165, 252, 201, 152, 144, 10, 169, 252, 160, 200, 32, 30,
    171, 76, 65
250 DATA 197, 165, 251, 201, 48, 144, 10, 169, 27, 160, 201, 32, 30, 171, 24, 144,
    237, 169, 0
260 DATA 133, 8, 169, 31, 133, 211, 169, 51, 160, 201, 32, 30, 171, 32, 225, 199, 201,
    78, 240
```

```
270 DATA 82, 201, 74, 208, 245, 169, 69, 160, 201, 32, 30, 171, 165, 251, 32, 241,
    198, 162, 0
280 DATA 232, 189, 64, 3, 201, 34, 208, 248, 134, 253, 232, 189, 64, 3, 201, 34, 240,
    6, 145, 65
290 DATA 232, 200, 208, 243, 138, 164, 251, 24, 229, 253, 153, 64, 202, 189, 64, 3,
    208, 10, 169
300 DATA 82, 160, 201, 32, 30, 171, 76, 65, 197, 201, 83, 240, 242, 201, 80, 240, 3,
    232, 208
310 DATA 230, 230, 251, 24, 144, 7, 169, 107, 160, 201, 32, 30, 171, 32, 182, 199, 76,
    125, 196
320 DATA 169, 8, 32, 195, 255, 165, 251, 208, 3, 76, 0, 196, 169, 119, 160, 201, 32,
    30, 171, 32
330 DATA 225, 199, 201, 49, 240, 7, 201, 50, 208, 245, 169, 0, 44, 169, 255, 133, 254,
    169, 15
340 DATA 32, 195, 255, 162, 0, 134, 252, 169, 199, 160, 201, 32, 30, 171, 164, 252,
    190, 64, 202
350 DATA 165, 252, 32, 241, 198, 177, 65, 32, 210, 255, 200, 202, 208, 247, 169, 2,
    162, 8, 168
360 DATA 32, 186, 255, 166, 252, 189, 64, 202, 133, 34, 165, 252, 32, 241, 198, 162,
    0, 177, 65
370 DATA 157, 64, 3, 200, 232, 198, 34, 208, 245, 160, 0, 185, 210, 201, 157, 64, 3,
    200, 232
380 DATA 192, 4, 144, 244, 138, 162, 64, 160, 3, 32, 189, 255, 32, 192, 255, 169, 9,
    160, 0, 132
390 DATA 34, 133, 35, 173, 17, 208, 41, 239, 141, 17, 208, 169, 15, 162, 8, 168, 32,
    186, 255
400 DATA 169, 3, 162, 215, 160, 201, 32, 189, 255, 32, 192, 255, 162, 2, 32, 15, 243,
    32, 31, 243
410 DATA 165, 186, 32, 180, 255, 165, 185, 32, 150, 255, 160, 0, 32, 165, 255, 32, 9,
    199, 166
420 DATA 144, 240, 246, 32, 235, 199, 8, 169, 2, 32, 195, 255, 173, 17, 208, 9, 16,
    141, 18, 208
430 DATA 40, 144, 3, 32, 55, 199, 166, 252, 165, 34, 133, 174, 165, 35, 133, 175, 232,
    165, 252
440 DATA 208, 13, 169, 78, 160, 200, 32, 30, 171, 32, 225, 199, 32, 37, 200, 169, 227,
    160, 201
450 DATA 32, 30, 171, 164, 252, 190, 64, 202, 165, 252, 32, 241, 198, 177, 65, 32,
    210, 255, 200
460 DATA 202, 208, 247, 162, 1, 169, 0, 168, 32, 186, 255, 166, 252, 189, 64, 202,
    133, 34, 165
470 DATA 252, 31, 241, 198, 162, 0, 177, 65, 157, 64, 3, 232, 200, 198, 34, 208, 245,
    138, 162
```

```
480 DATA 64, 160, 3, 32, 189, 255, 160, 0, 169, 0, 132, 172, 133, 173, 177, 172, 133,
    167, 200
490 DATA 177, 172, 133, 168, 200, 132, 172, 165, 174, 56, 229, 172, 8, 24, 101, 167,
    133, 169
500 DATA 165, 175, 101, 168, 40, 229, 173, 133, 170, 162, 5, 134, 171, 32, 133, 192,
    32, 229
510 DATA 198, 230, 252, 166, 252, 228, 251, 176, 7, 36, 254, 16, 16, 76, 223, 198,
    169, 237, 160
520 DATA 201, 32, 30, 171, 32, 225, 199, 76, 0, 196, 32, 85, 199, 169, 78, 160, 200,
    32, 30, 171
530 DATA 32, 255, 199, 32, 37, 200, 32, 182, 199, 76, 117, 197, 32, 115, 199, 173, 17,
    208, 9
540 DATA 16, 141, 17, 208, 96, 160, 0, 10, 10, 132, 66, 10, 38, 66, 10, 38, 66, 133, 65,
    165, 66
550 DATA 24, 105, 203, 133, 66, 160, 0, 96, 145, 34, 230, 34, 208, 2, 230, 35, 88, 96,
    32, 165
560 DATA 255, 41, 15, 10, 10, 10, 10, 133, 87, 32, 165, 255, 41, 15, 5, 87, 96, 72, 74,
    74, 74, 74
570 DATA 32, 47, 199, 104, 41, 15, 24, 105, 48, 76, 210, 255, 173, 17, 208, 9, 16, 141,
    17, 208
580 DATA 169, 7, 160, 202, 32, 30, 171, 32, 225, 199, 201, 89, 208, 1, 96, 201, 78,
    208, 244, 76
590 DATA 208, 199, 169, 26, 160, 202, 32, 30, 171, 165, 252, 10, 10, 10, 10, 166, 252,
    188, 64
600 DATA 202, 170, 189, 0, 203, 32, 210, 255, 232, 136, 208, 246, 96, 173, 17, 208,
    41, 239, 141
610 DATA 17, 208, 169, 15, 32, 195, 255, 169, 15, 162, 8, 168, 32, 186, 255, 169, 3,
    162, 219
620 DATA 160, 201, 32, 189, 255, 32, 192, 255, 169, 15, 76, 195, 255, 32, 228, 255,
    240, 251
630 DATA 96, 169, 15, 162, 8, 168, 32, 186, 255, 169, 1, 162, 223, 160, 201, 32, 189,
    255, 76
640 DATA 192, 255, 169, 32, 44, 169, 13, 76, 210, 255, 32, 210, 255, 138, 72, 152, 72,
    32, 225
650 DATA 255, 24, 208, 1, 56, 104, 168, 104, 170, 176, 1, 96, 162, 246, 154, 32, 229,
    198, 169
660 DATA 15, 32, 195, 255, 32, 231, 255, 76, 0, 196, 32, 228, 255, 208, 233, 32, 190,
    199, 144
670 DATA 246, 162, 15, 32, 15, 243, 32, 31, 243, 165, 186, 32, 180, 255, 165, 185, 32,
    150, 255
680 DATA 32, 19, 199, 201, 32, 8, 144, 11, 72, 32, 182, 199, 32, 182, 199, 104, 32, 38,
    199, 32
```

```
690 DATA 165, 255, 201, 13, 240, 9, 40, 8, 144, 245, 32, 210, 255, 144, 240, 33, 171,
    255, 40
700 DATA 96, 32, 56, 248, 32, 205, 199, 208, 248, 169, 7, 133, 192, 76, 202, 252, 13,
    13, 13
710 DATA 80, 76, 69, 65, 83, 69, 32, 73
720 DATA 78, 83, 69, 82, 84, 32, 65, 32
730 DATA 18, 68, 73, 83, 75, 146, 0, 13
740 DATA 13, 80, 76, 69, 65, 83, 69, 32
750 DATA 73, 78, 83, 69, 82, 84, 32, 18
760 DATA 67, 65, 83, 83, 69, 84, 84, 69
770 DATA 146, 33, 13, 0, 147, 17, 28, 32
780 DATA 32, 32, 32, 32, 32, 32, 32, 32
790 DATA 66, 65, 67, 75, 85, 80, 32, 67
800 DATA 79, 80, 89, 32, 70, 82, 79, 77
810 DATA 32, 68, 73, 83, 75, 13, 13, 31
820 DATA 32, 32, 32, 32, 32, 32, 84, 79
830 DATA 32, 67, 65, 83, 83, 69, 84, 84
840 DATA 69, 32, 84, 65, 80, 69, 13, 13
850 DATA 32, 32, 32, 32, 32, 32, 32, 87
860 DATA 73, 84, 72, 32, 18, 129, 70, 32
870 DATA 65, 32, 83, 32, 84, 32, 84, 32
880 DATA 65, 32, 80, 32, 69, 32, 146, 13
890 DATA 13, 13, 32, 32, 32, 40, 67, 41
900 DATA 32, 32, 32, 32, 68, 73, 82, 75
910 DATA 32, 80, 65, 85, 76, 73, 83, 83
920 DATA 69, 78, 32, 46, 13, 13, 13, 30
930 DATA 32, 32, 32, 32, 32, 32, 32, 32
940 DATA 32, 32, 32, 69, 32, 61, 32, 69
950 DATA 78, 68, 32, 31, 0, 13, 28, 18
960 DATA 33, 33, 18, 70, 73, 76, 69, 32
970 DATA 84, 79, 79, 32, 76, 65, 82, 71
980 DATA 69, 32, 84, 79, 32, 67, 79, 80
990 DATA 89, 32, 31, 0, 13, 28, 18, 33
1000 DATA 33, 18, 84, 72, 69, 32, 76, 73
1010 DATA 83, 84, 32, 73, 83, 32, 70, 85
1020 DATA 76, 76, 31, 0, 18, 32, 89, 69
1030 DATA 83, 47, 78, 79, 157, 157, 157, 157
1040 DATA 157, 157, 157, 146, 31, 0, 18, 30
1050 DATA 32, 89, 69, 83, 32, 32, 146, 32
1060 DATA 31, 0, 13, 28, 18, 33, 33, 18
1070 DATA 32, 73, 76, 76, 69, 71, 65, 76
1080 DATA 32, 70, 73, 76, 69, 84, 89, 80
```

```
1090  DATA 69, 32, 31, 0, 28, 18, 32, 32
1100  DATA 78, 79, 32, 32, 146, 32, 31, 0
1110  DATA 13, 13, 13, 87, 65, 78, 84, 32
1120  DATA 84, 79, 32, 67, 79, 80, 89, 32
1130  DATA 80, 82, 79, 71, 82, 65, 77, 83
1140  DATA 32, 58, 62, 13, 13, 32, 49, 41
1150  DATA 32, 67, 79, 78, 84, 73, 78, 85
1160  DATA 79, 85, 83, 76, 89, 32, 32, 32
1170  DATA 32, 13, 13, 32, 50, 41, 32, 73
1180  DATA 78, 68, 73, 86, 73, 68, 85, 65
1190  DATA 76, 76, 89, 13, 13, 87, 72, 73
1200  DATA 67, 72, 32, 79, 78, 69, 63, 0
1210  DATA 13, 13, 82, 69, 65, 68, 73, 78
1220  DATA 71, 32, 0, 44, 80, 44, 82, 0
1230  DATA 85, 73, 45, 0, 85, 73, 43, 0
1240  DATA 73, 0, 36, 0, 13, 13, 83, 65
1250  DATA 86, 73, 78, 71, 32, 0, 13, 13
1260  DATA 28, 18, 32, 67, 79, 80, 89, 73
1270  DATA 78, 71, 32, 67, 79, 77, 80, 76
1280  DATA 69, 84, 69, 68, 32, 146, 31, 0
1290  DATA 13, 87, 65, 78, 84, 32, 84, 79
1300  DATA 32, 67, 79, 78, 84, 73, 78, 85
1310  DATA 69, 63, 0, 13, 13, 32, 32, 32
1320  DATA 32, 32, 78, 69, 88, 84, 32, 70
1330  DATA 73, 76, 69, 32, 58, 32, 0, 0

READY.
 ********************************************************************
100  REM BACKUP DISKETTE =⟩ CASSETTE FOR VIC20
 ********************************************************************
110  E=31156:A=29696:PS=0
120  FOR I=A to E:READ X:POKE I, X:PS=PS+X:NEXT
130  IF PS⟨⟩145927 THE PRINT "ERROR IN DATA":END
140  SYS A:NEW
150  DATA 169, 25, 141, 15, 144, 169, 6, 141, 134, 2, 169, 19, 160, 120, 32, 30, 203,
       169, 221
160  DATA 160, 119, 32, 30, 203, 32, 69, 119, 201, 69, 208, 1, 96, 32, 98, 119, 32, 98,
       119, 32
170  DATA 75, 119, 169, 8, 170, 160, 0, 32, 186, 255, 169, 1, 162, 103, 160, 121, 32,
       189, 255
180  DATA 32, 192, 255, 169, 8, 32, 180, 255, 169, 0, 32, 150, 255, 160, 4, 32, 165,
       255, 136, 208
```

```
190 DATA 250, 32, 165, 255, 133, 34, 32, 165, 255, 166, 34, 32, 205, 221, 32, 95, 119,
    32, 165
200 DATA 255, 240, 6, 32, 210, 255, 24, 144, 245, 32, 98, 119, 32, 98, 119, 32, 165,
    255, 32, 165
210 DATA 255, 160, 0, 132, 3, 32, 165, 255, 133, 4, 32, 165, 255, 133, 5, 166, 4, 32,
    205, 221
220 DATA 32, 95, 119, 160, 0, 32, 165, 255, 72, 32, 103, 119, 104, 153, 64, 3, 240, 3,
    200, 208
230 DATA 240, 32, 165, 255, 32, 165, 255, 165, 144, 240, 3, 76, 64, 117, 165, 5, 208,
    6, 165, 4
240 DATA 201, 93, 144, 10, 169, 133, 160, 120, 32, 30, 203, 76, 58, 117, 165, 3, 201,
    48, 144
250 DATA 10, 169, 164, 160, 120, 32, 30, 203, 24, 144, 237, 169, 0, 133, 8, 169, 188,
    160, 120
260 DATA 32, 30, 203, 32, 138, 119, 201, 78, 240, 82, 201, 74, 208, 245, 169, 206,
    160, 120, 32
270 DATA 30, 203, 165, 3, 32, 203, 118, 162, 0, 232, 189, 64, 3, 201, 34, 208, 248,
    134, 5, 232
280 DATA 189, 64, 3, 201, 34, 240, 6, 145, 65, 232, 200, 208, 243, 138, 164, 3, 24,
    229, 5, 153
290 DATA 64, 122, 189, 64, 3, 208, 10, 169, 218, 160, 120, 32, 30, 203, 76, 58, 117,
    201, 83, 240
300 DATA 242, 201, 80, 240, 3, 232, 208, 230, 230, 3, 24, 144, 7, 169, 244, 160, 120,
    32, 30, 203
310 DATA 32, 98, 119, 76, 122, 116, 169, 8, 32, 195, 255, 165, 3, 208, 3, 76, 0, 116,
    169, 0, 160
320 DATA 121, 32, 30, 203, 32, 138, 119, 201, 49, 240, 7, 201, 50, 208, 245, 169, 0,
    44, 169, 255
330 DATA 133, 5, 169, 15, 32, 195, 255, 162, 0, 134, 4, 169, 77, 160, 121, 32, 30, 203,
    164, 4
340 DATA 190, 64, 122, 165, 4, 32, 203, 118, 177, 65, 32, 210, 255, 200, 202, 247,
    169, 2
350 DATA 162, 8, 168, 32, 186, 255, 166, 4, 189, 64, 122, 133, 34, 165, 4, 32, 203,
    118, 162, 0
360 DATA 177, 65, 157, 64, 3, 200, 232, 198, 34, 208, 245, 160, 0, 185, 88, 121, 157,
    64, 3, 200
370 DATA 232, 192, 4, 144, 244, 138, 162, 64, 160, 3, 32, 189, 255, 32, 192, 255, 169,
    19, 160
380 DATA 0, 132, 34, 133, 35, 169, 15, 162, 8, 168, 32, 186, 255, 169, 3, 162, 93, 160,
    121, 32
390 DATA 189, 255, 32, 192, 255, 162, 2, 32, 207, 243, 32, 223, 243, 165, 186, 32,
    180, 255, 165
```

```
400 DATA 185, 32, 150, 255, 160, 0, 32, 165, 255, 32, 227, 118, 166, 144, 240, 246,
    32, 148, 119
410 DATA 8, 169, 2, 32, 195, 255, 40, 144, 3, 32, 17, 119, 166, 4, 165, 34, 133, 174,
    165, 35, 133
420 DATA 175, 232, 165, 4, 208, 13, 169, 247, 160, 119, 32, 30, 203, 32, 138, 119, 32,
    206, 119
430 DATA 169, 105, 160, 121, 32, 30, 203, 164, 4, 190, 64, 122, 165, 4, 32, 203, 118,
    177, 65
440 DATA 32, 210, 255, 200, 202, 208, 247, 161, 1, 169, 0, 168, 32, 186, 255, 166, 4,
    189, 64
450 DATA 122, 133, 34, 165, 4, 32, 203, 118, 162, 0, 177, 65, 157, 64, 3, 232, 200,
    198, 34, 208
460 DATA 245, 138, 162, 64, 160, 3, 32, 189, 255, 160, 0, 169, 19, 132, 172, 133, 173,
    177, 172
470 DATA 133, 167, 200, 177, 172, 133, 168, 200, 132, 172, 165, 174, 56, 229, 172, 8,
    24, 101
480 DATA 167, 133, 168, 165, 175, 101, 168, 40, 229, 173, 133, 170, 162, 5, 134, 171,
    32, 127
490 DATA 112, 230, 4, 166, 4, 228, 3, 176, 7, 36, 5, 16, 16, 76, 197, 118, 169, 115,
    160, 121, 32
500 DATA 30, 203, 32, 138, 119, 76, 0, 116, 32, 39, 119, 169, 247, 160, 119, 32, 30,
    203, 32, 138
510 DATA 119, 32, 206, 119, 32, 98, 119, 76, 110, 117, 160, 0, 10, 10, 132, 66, 10, 38,
    66, 10
520 DATA 38, 66, 133, 65, 165, 66, 24, 105, 123, 133, 66, 160, 0, 96, 145, 34, 230, 34,
    208, 3
530 DATA 230, 35, 88, 96, 32, 165, 255, 41, 15, 10, 10, 10, 10, 133, 87, 32, 165, 255,
    41, 15, 5
540 DATA 87, 96, 72, 74, 74, 74, 74, 32, 9, 119, 104, 41, 15, 24, 105, 48, 76, 210, 255,
    169, 141
550 DATA 160, 121, 32, 30, 203, 32, 138, 119, 201, 89, 208, 1, 96, 201, 78, 208, 244,
    76, 124
560 DATA 119, 169, 160, 160, 121, 32, 30, 203, 165, 4, 10, 10, 10, 10, 166, 4, 188, 64,
    122, 170
570 DATA 189, 0, 123, 32, 210, 255, 232, 136, 208, 246, 96, 32, 228, 255, 240, 251,
    96, 169, 15
580 DATA 162, 8, 168, 32, 186, 255, 169, 1, 162, 101, 160, 121, 32, 189, 255, 76, 192,
    255, 169
590 DATA 32, 44, 169, 13, 76, 210, 255, 32, 210, 255, 138, 72, 152, 72, 32, 225, 255,
    24, 208
600 DATA 1, 56, 104, 168, 104, 170, 176, 1, 96, 162, 246, 154, 169, 15, 32, 195, 255,
    32, 231
```

```
610 DATA 255, 76, 0, 116, 32, 228, 255, 208, 236, 32, 106, 119, 144, 246, 162, 15, 32,
    207, 243
620 DATA 32, 223, 243, 165, 186, 32, 180, 255, 165, 185, 32, 150, 255, 32, 237, 118,
    201, 32
630 DATA 8, 144, 11, 73, 32, 98, 119, 32, 98, 119, 104, 32, 0, 119, 32, 165, 255, 201,
    13, 240
640 DATA 9, 40, 8, 144, 245, 32, 210, 255, 144, 240, 32, 171, 255, 40, 96, 32, 183,
    248, 32, 121
650 DATA 119, 208, 248, 169, 7, 133, 192, 76, 8, 253, 13, 13, 13
710 DATA 80, 76, 69, 65, 83, 69, 32, 73
720 DATA 78, 83, 69, 82, 84, 32, 65, 32
730 DATA 18, 68, 73, 83, 75, 146, 0, 13
740 DATA 13, 80, 76, 69, 65, 83, 69, 32
750 DATA 73, 78, 83, 69, 82, 84, 32, 18
760 DATA 67, 65, 83, 83, 69, 84, 84, 69
770 DATA 146, 33, 13, 0, 147, 17, 28, 32
780 DATA 32, 32, 32, 32, 32, 32, 32, 32
790 DATA 66, 65, 67, 75, 85, 80, 32, 67
800 DATA 79, 80, 89, 32, 70, 82, 79, 77
810 DATA 32, 68, 73, 83, 75, 13, 13, 31
820 DATA 32, 32, 32, 32, 32, 32, 84, 79
830 DATA 32, 67, 65, 83, 83, 69, 84, 84
840 DATA 69, 32, 84, 65, 80, 69, 13, 13
850 DATA 32, 32, 32, 32, 32, 32, 32, 87
860 DATA 73, 84, 72, 32, 18, 129, 70, 32
870 DATA 65, 32, 83, 32, 84, 32, 84, 32
880 DATA 65, 32, 80, 32, 69, 32, 146, 13
890 DATA 13, 13, 32, 32, 32, 40, 67, 41
900 DATA 32, 32, 32, 32, 68, 73, 82, 75
910 DATA 32, 80, 65, 85, 76, 73, 83, 83
920 DATA 69, 78, 32, 46, 13, 13, 13, 30
930 DATA 32, 32, 32, 32, 32, 32, 32, 32
940 DATA 32, 32, 32, 69, 32, 61, 32, 69
950 DATA 78, 68, 32, 31, 0, 13, 28, 18
960 DATA 33, 33, 18, 70, 73, 76, 69, 32
970 DATA 84, 79, 79, 32, 76, 65, 82, 71
980 DATA 69, 32, 84, 79, 32, 67, 79, 80
990 DATA 89, 32, 31, 0, 13, 28, 18, 33
1000 DATA 33, 18, 84, 72, 69, 32, 76, 73
1010 DATA 83, 84, 32, 73, 83, 32, 70, 85
1020 DATA 76, 76, 31, 0, 18, 32, 89, 69
1030 DATA 83, 47, 78, 79, 157, 157, 157, 157
```

```
1040 DATA 157, 157, 157, 146, 31, 0, 18, 30
1050 DATA 32, 89, 69, 83, 32, 32, 146, 32
1060 DATA 31, 0, 13, 28, 18, 33, 33, 18
1070 DATA 32, 73, 76, 76, 69, 71, 65, 76
1080 DATA 32, 70, 73, 76, 69, 84, 89, 80
1090 DATA 69, 32, 31, 0, 28, 18, 32, 32
1100 DATA 78, 79, 32, 32, 146, 32, 31, 0
1110 DATA 13, 13, 13, 87, 65, 78, 84, 32
1120 DATA 84, 79, 32, 67, 79, 80, 89, 32
1130 DATA 80, 82, 79, 71, 82, 65, 77, 83
1140 DATA 32, 58, 62, 13, 13, 32, 49, 41
1150 DATA 32, 67, 79, 78, 84, 73, 78, 85
1160 DATA 79, 85, 83, 76, 89, 32, 32, 32
1170 DATA 32, 13, 13, 32, 50, 41, 32, 73
1180 DATA 78, 68, 73, 86, 73, 68, 85, 65
1190 DATA 76, 76, 89, 13, 13, 87, 72, 73
1200 DATA 67, 72, 32, 79, 78, 69, 63, 0
1210 DATA 13, 13, 82, 69, 65, 68, 73, 78
1220 DATA 71, 32, 0, 44, 80, 44, 82, 0
1230 DATA 85, 73, 45, 0, 85, 73, 43, 0
1240 DATA 73, 0, 36, 0, 13, 13, 83, 65
1250 DATA 86, 73, 78, 71, 32, 0, 13, 13
1260 DATA 28, 18, 32, 67, 79, 80, 89, 73
1270 DATA 78, 71, 32, 67, 79, 77, 80, 76
1280 DATA 69, 84, 69, 68, 32, 146, 31, 0
1290 DATA 13, 87, 65, 78, 84, 32, 84, 79
1300 DATA 32, 67, 79, 78, 84, 73, 78, 85
1310 DATA 69, 63, 0, 13, 13, 32, 32, 32
1320 DATA 32, 32, 78, 69, 88, 84, 32, 70
1330 DATA 73, 76, 69, 32, 58, 32, 0, 0

READY.
```

15.4 Program Description of Cassette Back-up for Disk

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

Premliminary Remark:

This program operates with two different loading and storage starting addresses. 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 initialisation of the disk, the file names are first read in and held in 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 test for file type of the file to be copied. If it is not a program file, then an error message is output.

If the entire list of 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 (TLSADR). If no error has ocurred 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. Finally a fundamental point.

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 on account 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 disk at the higher transfer rate. For this reason, the screen is switched off during the loading process and the disk changed over 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

A Important Storage Locations

Hex	*Decimal*	*Meaning*
+0001	1	CPU output register Bit3 writing line for datassette Bit4 CASS-SENS, recorder key checking 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 fields
0031-0032	49-50	Pointer for end of fields
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
00BE	190	Pass counter when reading and writing
00BD	191	Input/ouput 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 checking

*9120	37152	Bit3 Writing Line

+ applies only to C-64
* applies only to VIC 20

Books

Trricks and Tips for the 64

A collection of easy-to-use programming techniques for the 64. A perfect companion for those who have run up against those hard-to-solve programming problems. Covers advanced graphics, easy data input, BASIC enhancement, CP/M cartridge on the 64, POKES, user-defined character sets, joystick/mouse simulation, transferring data between computers, and much more. A real treasure chest!

ISBN 948015 020 **Price £8.95**

Anatomy of the 64

An insider's guide to the lesser known qualities of the 64. Includes sections on graphics, sound synthesis, input/output control, sample programs using kernel routines, and much more. For those who want to know, it includes complete disassembled and document ROM listings.

ISBN 948015 004 **Price £8.95**

The Commodore 64 Machine Language Book

Aimed at those who want to progress beyond BASIC. Write faster, more memory-efficient programs in machine language as specifically geared to the Commodore 64. Learn how to use all of the 6510 Instructions, to program high resolution graphics, to perform input and output operations with plenty of easy to understand examples. Included are listings for a working Assembler, Disassembler, and 6510 Single Step Simulator.

ISBN 948015 039 **Price £8.95**

Advanced Machine Language for the Commodore 64

Gives you an intensive treatment of the powerful 64 features. The author delves into areas such as interrupts, the video controller, the timer, the real time clock, parallel and serial I/O, extending BASIC, and tips and tricks from machine language.

ISBN 948015 055 **Price £8.95**

The Commodore 64 Idea Book

For those who wonder what to do with their 64. It is written for the novince and presents dozens of program listings, presenting a huge number of possibilities for you on your machine. Themes include car expenses, electronic calculator, recipe file, share lists, construction cost estimater, personal health record, diet planner, computer poetry, party planning, and more.

ISBN 948015 08X **Price £8.95**

Commodore 64 Science and Engineering

An introduction to your computer in the world of science and engineering. Describes variable types, computational accuracy, and various sorts of algorithms. Topics include linear and nonlinear regression. Chi-square distribution. Fourier analysis, matrix calculations, and more. Programs are for the disciplines of chemistry, physics, biology, astronomy, and electronics. Includes many program listings.

ISBN 948015 101 **Price £8.95**

Anatomy of the 1541 Disk Drive

Unravels the mysteries of using the misunderstood disk drive. Details the use of program sequential and relative files with lots of working examples. Also includes several useful utilities—Disk Monitor, File Protector, Backup, Merge, and more. Discusses the intervals of the DOS with the complete fully commented ROM listings.

ISBN 948015 012 **Price £8.95**

Graphics Book For Your 64

Takes you from the fundamentals of graphic design to the advanced topics such as computer-aided design. Shows you how to program new character sets, move sprites, draw in HIRES and MULTICOLOUR, use a lightpen, handle IRQ's, do 3D graphics, projections, curves, and animation. Dozens of samples.

ISBN 948015 047 **Price £8.95**

Software

PASCAL 64

This full compiler produces fast 6502 machine code. Supports major data types: REAL, INTEGER, BOOLEAN, CHARACTER, multiple dimension arrays, RECORD FILE SET, and pointer. Offers easy string handling procedures for sequential and relative data management and ability to write interrupt routines in PASCAL Extensions included for high resolution and sprite graphics. Link to ASSEM/MON machine language.

ISBN 948015 071

Disk only **Price £34.99**

Assembler/Monitor 64

Features macro assembler and extended monitor. The macro assembler offers free form input, complete assembler listings with symbol table (label), and conditional assembly. Has all the commands plus single step quick trace and breakpoint, bank switching, and more.

ISBN 948015 063

Disk only **Price £19.99**

FirstBASE 64

This powerful data manager handles up to 2000 records per disk. You select the screen format using up to 50 fields per record. FirstBASE 64 can sort on multiple fields in any combination. Complete report writing capabilities to all Commodore or ASCII printers. Very easy to use.

ISBN 948015 152

Disk only **Price £35.99**

BASIC 64

The compiler that won't break you! Is compatible with Commodore BASIC. Compiles to fast machine code. Protect your source and learn, too.

ISBN 948015 25X

Disk only **Price £34.99**

ADA Training Course

An introduction to ADA—the programming language of the future. Includes syntax checker/compiler, editor, and a 110-page step-by-step manual describing the language.

ISBN 948015 306

Disk only **Price £49.99**

FirstWORD 64

Complete word processor displays 80 columns using horizontal scrolling. Can edit in memory up to 24,000 characters, plus chain longer documents. Complete text formatting, block operations, standard letters, on screen prompts. So easy to start and use—you cannot be without this!

ISBN 948015 209

Disk only **Price £35.99**

PowerPLAN 64

A state-of-the-art, fully integrated, graphics and spread sheet progam using the latest options. Comprehensive manual with step-by-step guide. **At last you can forecast your sales results, create professional presentations!**

ISBN 948015 608

Price £44.99

This book is a must for every cassette owner. It will explain in detail the internal workings and how you can get much faster loading, better storage and much more. We hope you enjoy it.





This was brought to you

from the archives of

http://retro-commodore.eu