D64 ATTACHED
TO THIS PDF

The Ultimate Utility Package

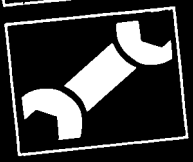
hack-pack

LIGHTNING SPEED RAM-DISK

HIGH POWER PROGRAM COMPRESSOR

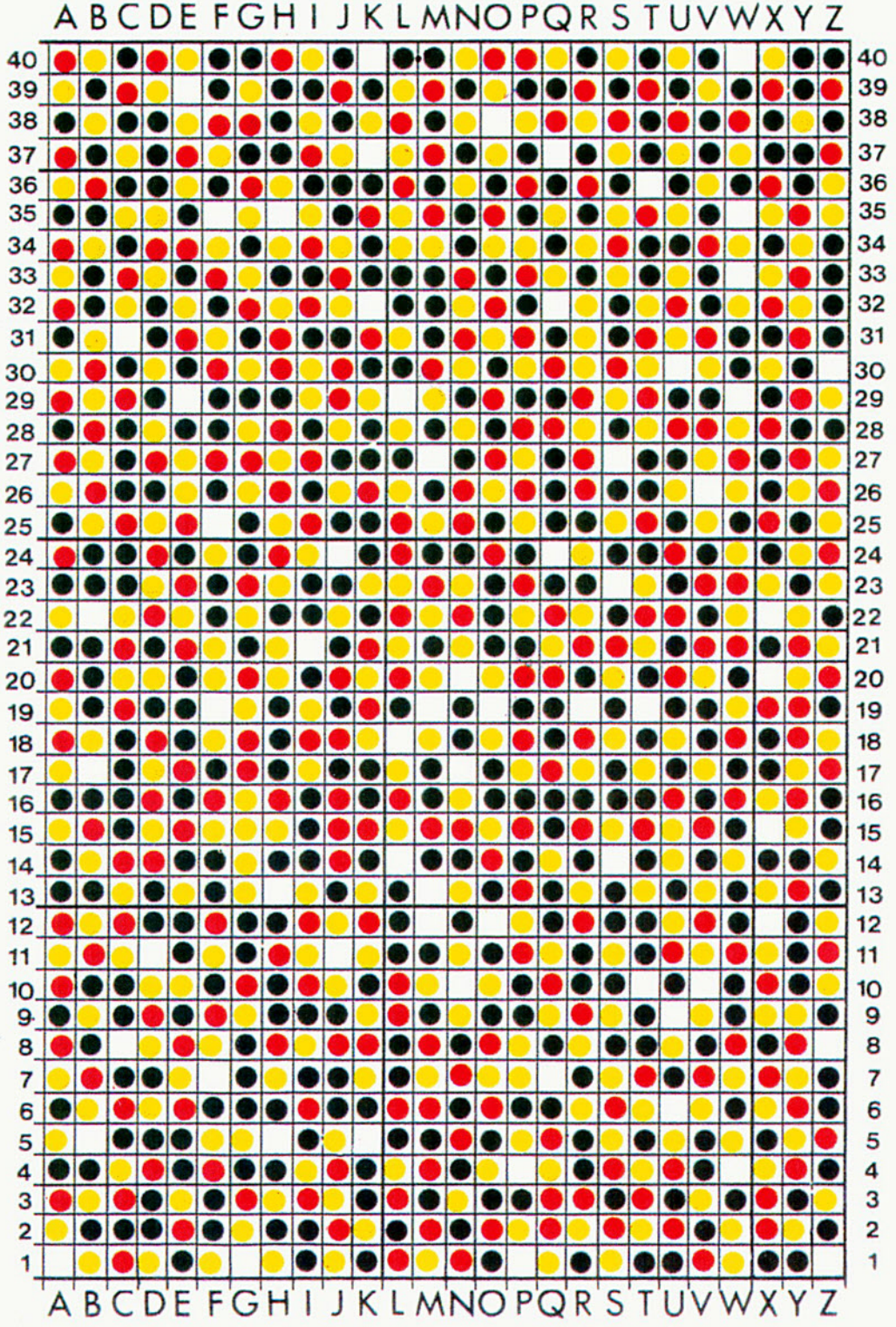
FULL FEATURE TOOLKIT

FIND · CHANGE · MERGE



DUMP · INFO · TYPE · SIZE ..

commodore 128


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

COPYRIGHT

Having said this we hope you will enjoy using HACK-PACK and that you will find it the useful utility that it is intended to be.

DISCLAIMER

H A C K - P A C K

C O N T E N T S

Loading The Software

Before using HACK-PACK it is necessary to follow a brief protection sequence to establish you as an authorised user of the software. We hope that you will not find the protection too tiresome; software does have to be protected and this method at least allows you to make backup copies of the system disk. Proceed as follows:

Mount the HACK-PACK System disk and type SHIFT RUN/STOP.

After a short pause, HACK-PACK makes the protection check. On the sheet provided with HACK-PACK will be found a matrix of colored squares. Each square can be identified by means of a simple grid reference. For example to find square E-7, identify column E (marked along the bottom of the matrix) and row 7 (marked along the left hand edge). Square E-7 is where column E and row 7 meet. (see fig 1).

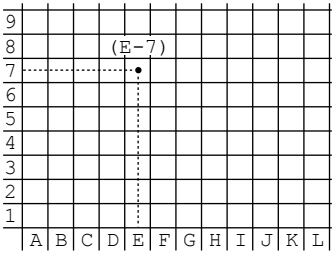


Fig 1.

The protection check is very simple; all you have to do is correctly identify three squares and enter their colors. HACK-PACK will give instructions as follows:

WHAT COLOR IS SQUARE X-XX ?

where X-XX is a grid reference. When you have found the square, enter one of the letters W,Y,G or R depending upon whether the square is white, yellow, green or red. When you have correctly answered three such questions the protection check is complete and the screen will display

Type 1) To Install Toolkit
2) To Install RAM-DISK
3) To Install Toolkit and RAM-DISK
4) To Install Compressor

Some versions of RAM-DISK will load into either bank 0 or bank 1. If you have such a version, you will be asked

Install RAM-DISK in bank (0/1) ?

To which the reply should normally be 0.

When the software is installed into your 128, control of the computer will be returned to you.

TOOLKIT Introduction

The Commodore 128 provides a very powerful and friendly editor for BASIC programs. Unfortunately, certain features which are normally regarded as essential in a programmers' editor have unaccountably been omitted by CBM. In particular, there is no way on the 128 to find a string in a BASIC program much less to search for and replace one string by another.

The HACK-PACK toolkit attempts to fill this need.

Once loaded, the HACK-PACK toolkit fits unobtrusively into an area at the top of RAM in bank 0. From this elevated position it provides the programmer with a number of totally new editing features.

The HACK-PACK toolkit does not waste memory by providing nonsense commands that no one ever uses such as BEEP and FLASH! Instead it adds to BASIC eight really useful commands, all designed to improve programming efficiency and productivity.

Loading the toolkit

To load Toolkit, follow the instructions in the section loading the Software at the beginning of this manual.

When Toolkit has loaded it will sign on with the message

HACK-PACK TOOLKIT (C) 1986 SSOL. READY.

Toolkit is now fully installed and ready for use.

Toolkit Commands

In the following, square brackets are used to denote optional parameters.

FIND **syntax: FIND/search-string/[,line-range]**

The FIND command can be used to search for any text string in a BASIC program. The delimiter (/) may actually be any character not contained in the search string. Line ranges, which specify the part of the program to be searched, conform to the same rules as those pertaining to LIST.

Examples

FIND /GOTO100/

will find all occurrences of the string GOTO100 in the currently loaded program

FIND /A=/,1000-2000

will find all occurrences of the string A= in the line range 1000-2000.

FIND !X/2!,500-

will find all occurrences of the string X/2 from line 500 to the end of the program. Notice that because the search string contains a slash (/), we use shriek (!) as the delimiter.

Tokens and quoted strings

All BASIC keywords such as GOTO, FOR, PRINT and REM are tokenised by the BASIC 7.0 editor, that is, they are held in an encoded form and only decoded into text again by the LIST command. However, if keywords appear in a quoted string, or in a REM or DATA statement, they are left untokenised. When you search for a keyword by typing something like FIND/FOR/, toolkit tokenises the string and then searches for it in tokenised form. FIND/FOR/ would therefore fail to find the FOR in a line like

10 REM PREPARE OUTPUT FOR PRINTING

or

10 PRINT "GONE FOR LUNCH"

To find strings like this we use the special delimiter quote ("). If we use quote as the delimiter and type FIND"FOR", toolkit will find only those lines where the string FOR appears within quotes or in a REM or DATA statement.

Quote always has the effect of switching off tokenisation, even if it appears as part of the search string. For example suppose we wish to find the line

```
10 A$=MID$(B$+"FOR",I)
```

We could type FIND/MID$(B$+"FOR/. Here we don't use quote as a delimiter since if we did, we would fail to find the tokenised MID$. The quote contained in the search string still has the effect of switching off tokenisation for all text following it up until the next quote or delimiter.

CHANGE **syntax: CHANGE/search-string/replacement-string/ [,line-range]**

CHANGE functions exactly like FIND except that wherever the search string is found, it is replaced by the replacement string. Again, the delimiter may be any character not appearing in the search or replacement strings.

Examples

```
CHANGE /A$/B$/
```

will change A$ to B$ wherever it occurs in the program.

```
CHANGE /GOTO1000/GOSUB1000/10000-11000
```

will change any occurrence of the statement GOTO1000 between lines 10000 and 11000 to GOSUB1000

The rules governing quotes are the same as those for FIND. For Example, to change the line

```
10 PRINT "GONE FOR LUNCH"
```

to

```
10 PRINT "GONE TO LUNCH"
```

we would type CHANGE "GONE FOR"GONE TO"

DUMP **syntax: DUMP [V][A]**

DUMP entered without parameters causes the names and values of all variables and the names of all functions and arrays to be displayed.

Examples

Suppose we RUN the following program

```
10 A=1:B=2
20 CC$="HACK "+"PACK":D%=-100
30 DEF FN T(X)=SIN(X)/COS(X)
40 DIM A1(2,3),A2%(A,B)
```

Then the DUMP command will produce the following

```
DUMP

A=1                 B=2
CC$="HACK PACK"     D%=-100
T()                 X=0

A1(2,3)             A2(1,2)
READY.
```

The V and A parameters which may optionally follow DUMP allow output to be restricted to variables (V) or arrays (A). For instance, using the same example, the command

```
DUMP A
```

would invoke the following output

```
A1(2,3)             A2(1,2)
READY.
```

INFO **syntax: INFO**

The INFO command displays sane useful information about memory usage in the 128. The information is displayed as follows:

```
         TEXT   VARS   ARRAYS STRINGS
IN USE:  TTTTT  VVVVV  AAAAA  SSSSS
FREE  :  FRE0   .......FRE1..........
```

where

```
TTTTT  is the number of bytes occupied by BASIC text
VVVVV  is the number of bytes occupied by simple variables
AAAAA  is the number of bytes occupied by arrays
SSSSS  is the number of bytes occupied by strings
FRE0   is the number of bytes available for BASIC text
FRE1   is the number of bytes available for variables,
       arrays and strings
```

Examples

If we were to run the program illustrated above under DUMP, the INFO command would display the following

```
INFO
         TEXT   VARS   ARRAYS STRINGS
IN USE:  88     42     90     11
FREE  :  54183  .......64113.........
```

MERGE **syntax: MERGE"filename"**

The MERGE command is very similar to DLOAD. The only difference is that whereas DLOAD loads a BASIC program and destroys any BASIC currently in memory, MERGE appends the new program to the old one.

Before using MERGE you should ensure that the MERGEd program does not have linenumbers which conflict with program in memory.

Examples

Suppose we have two programs A and B. We wish to merge these into a single program; we proceed as follows

```
DLOAD"A"
LOADING 0:A
READY.
MERGE"B"
MERGING B
READY.
```

A quick LIST will reveal that both programs are in memory; A followed by B. This may now be DSAVEd in the ordinary way.

TYPE **syntax: TYPE"filename"**

The TYPE command is used to display ASCII files at the terminal. Any file containing printable characters may be TYPEd. When a complete page of the file has been displayed, the prompt ------ MORE ------ appears at the bottom of the screen. Type RETURN to see the next line, SPACE to see the next page or STOP to quit.

SIZE **syntax: SIZE"filename"**

SIZE is used to determine the size of a file in bytes. It may be used with any file type.

Examples

Suppose we have a sequential file F containing the string "AFFILIATED". To count the number of bytes, we type

```
SIZE"F"
10 BYTES.
READY.
```

QUIT **syntax: QUIT**

QUIT de-activates the toolkit de-allocates the memory assigned to it.

Re-activating the Toolkit

The toolkit may be re-activated after a QUIT command by typing SYS 65369.

Memory usage

The toolkit loads at the top of RAM in bank 0 and occupies about 2.5K. This area of memory is only required by BASIC for programs that exceed about 51K in size. Should a program reach this limit, the system will generate an OUT OF MEMORY message.

RAM-DISK Introduction

One of the best features of the Commodore 128 computer is the large amount of RAM which is available to the user. Unfortunately, in many applications, particularly programming applications, most of that memory lies idle for most of the time.

In several modern large memory micros, notably the Amiga, Atari ST and the Amstrad a RAM-DISK is supplied as part of the operating system so that users can make effective use of all the memory available. Now, without any additional hardware, this is possible on the 128.

In essence RAM-DISK behaves exactly like a much faster 1541 or 1571 floppy disk; the only difference being that all the data on the RAM-DISK is held in RAM and not on a physical disk. All the usual disk commands apply, you use CATOLOG to look at the directory, DLOAD or BLOAD to load a program, OPEN to open a file and so on.

As on the Amiga, the COPY command can be used to transfer programs and data files from floppy disk onto the RAM-DISK and, at the end of a session you can dump all the files from RAM-DISK back onto floppy with a single command.

It is often important for the user to have control over which areas of memory are available to RAM-DISK and which areas are not. RAM-DISK has three modes of operation. In its simplest mode, for pure BASIC applications, RAM-DISK allocates two chunks of memory at the top of banks 0 and 1, and reduces the amount of memory available to BASIC accordingly. With a simple command you can move the BASIC/RAM-DISK boundaries to suit your way of working or to suit a particular application.

If you are using a combination of BASIC and machine code, you can also alter the the upper boundaries of these chunks in order to free up areas of high memory.

Lastly, for complex applications, you can allocate to RAM-DISK any number of separate 256 byte pages anywhere in the machine. In this mode, as in all modes, RAM-DISK will only use those areas of memory that you specify.

Getting started

To load RAM-DISK, follow the instructions in the section **Loading the Software** at the beginning of this manual. When RAM-DISK is installed it will sign on with the message

```
73,RAM-DISK (C) SSOL 1986 DC.X,00,00
XXX Blocks free.
ready.
```

From now on, until you reset the computer, RAM-DISK will respond to all the usual disk commands.

Device numbers

As explained in the 128 System Guide, every device such as the disk drive or printer has a device number. By default, the device number for the floppy disk drive is 8 and all the disk commands make this assumption. If you were to add a second disk drive to the system, you would have to give it a different device number say 9. To look at the directory of the second disk drive you would type CATALOG U9 and to load a program from it you would type DLOAD"prog-name" ON U9 etc. This is exactly how you use RAM-DISK.

Type CATALOG U9. You should get the (so far empty) RAM-DISK directory. Now type in (or load from floppy disk) a short BASIC program and type DSAVE"prog-name" ON U9. If you look at the RAM-DISK directory now, you should see an entry for that file. Type NEW to clear memory and then DLOAD"prog-name" ON U9 to get the program back.

You will probably find in the course of using RAM-DISK that you frequently need to copy files from floppy disk to RAM-DISK and vial versa. A specially extended COPY command is provided for this purpose, details of which may be found in the command summary at the end of this manual.

Error messages

Error messages are issued by RAM-DISK in exactly the same way as they are by a floppy disk, that is by preparing a message for reading when you type PRINT DS$. However, there are certain differences and you are urged to read the summary of error messages at the end of this manual.

Extended RAM-DISK commands

Commands are issued Commodore disk drives by sending a string of text to the drive using a special secondary address of 15, that is by executing a statement like

```
OPEN 1,8,15,"command":CLOSE 1
```

Disk commands such as COPY, RENAME and SCRATCH generate such a string internally and send this to the disk drive automatically. The procedure for sending disk commands in this way is fully covered in Commodore's 1541 and 1571 user's guide, but as an example, the following command will initalize (DCLEAR) the disk drive

```
OPEN 1,8,15,"I0":CLOSE 1
```

Commands may be sent to RAM-DISK in exactly the same way although users will generally use the BASIC 7.0 disk commands provided for this purpose. There are however certain extended RAM-DISK commands which can only be sent via an open statement and these take the form

```
OPEN 1,9,15,"ram-disk-command":CLOSE 1
```

where the second parameter in the open statement is the current device number of RAM-DISK, set by default to 9.

Changing the RAM-DISK device number

This is simple. Type OPEN 1,9,15,"U8":CLOSE1. The RAM-DISK device number is now 8 and the 128 will in future send to RAM-DISK all commands directed to device 8. Since the default device number for all the disk commands is 8, any program which accesses a floppy disk file will now access RAM-DISK, and direct commands such as CATALOG and DLOAD will go to RAM-DISK by default. This can be convenient but of course, if you wish to access the floppy disk you will have to change the RAM-DISK device number to something different, say device 9 as before with the command OPEN 1,8,15,"U9":CLOSE 1. Notice that the second parameter of the open statement is now 8, reflecting the (current) device number of RAM-DISK.

IMPORTANT -- During the remainder of this manual, we shall assume that the device number of RAM-DISK has been changed to 8, and that therefore no ON U9 is required in any of the disk commands. If you decide not to change the device number to 8 (or you decide to change it to something else) you should of course include a Udevice-no. in all disk commands. The same remarks of course apply to the device number in the OPEN statement.

RAM-DISK command formats

For the remainder of this manual (as in the above), all command examples will be given in upper case letters. The purpose of this is to make the text clearer. All commands should be typed in un-shifted characters (without spaces) which will appear on the screen in upper or lower case depending upon which character set is at that moment in force.

BASIC or machine code?

If you are writing programs in pure BASIC and do not use machine code, read the following section but skip the sections on machine code. If you are programing in machine code or a combination of BASIC and machine code, read the following section and then proceed onto the next section: Programming in BASIC and machine code.

RAM-DISK and pure BASIC programming.

The Commodore 128 contains 128Kb of RAM, some 8K of which is used by the operating system. When RAM-DISK initializes, it allocates about half the remaining memory to itself. If your BASIC programs are not longer than about 32K and you are not using enormous arrays, this allocation of memory will not affect you. If you find, either that you want more roan for BASIC or that you would like more blocks free on RAM-DISK, then read on.

Memory allocation on the 128

To get the most out of your RAM-DISK its worth knowing a little about the way memory is handled by the BASIC interpreter.

BASIC needs storage for programs and data (variables, arrays etc). There are two RAM banks in the 128 each containing 64k. Bank 0 is used for BASIC programs (mostly) and bank 1 is used for data. You can always find out how much memory is available in either bank with the statement PRINT FREE(b) where b is the bank number.

RAM-DISK also uses some of the memory in these banks, and there may be times when you want to tell RAM-DISK to use more or less of it.

Pages and blocks

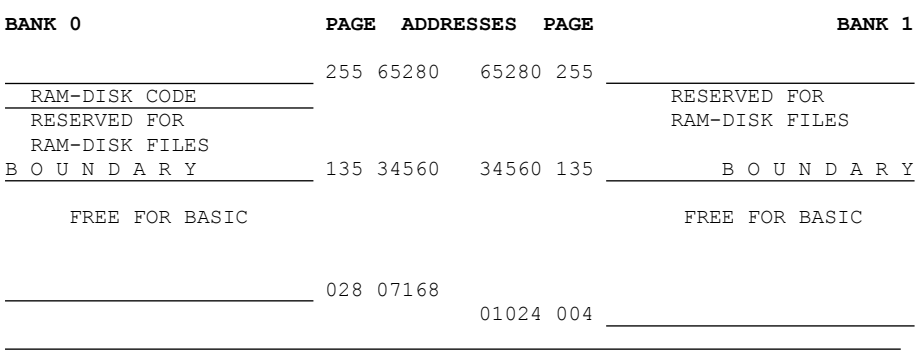
A page is a contiguous piece of memory 256 bytes long. Each RAM bank in the 128 contains 256 pages numbered 0 through 255, making 128K in all.

To find out where a particular page begins, simply multiply the page number by 256.

Page no.	0	1	2	3	4	...	r	...	253	254	255
Address	0	256	512	768	1024	...	r*256	...	64768	65024	65536

Just like the 1541 or 1571 floppy drives, RAM-DISK thinks in blocks; each of which occupies one page. When RAM-DISK first initializes, it allocates 120 pages at the top of each of the RAM banks and reserves some of this memory for its own code. (see fig 2).

Fig 2: A simple 128 memory map



As you can see, there is a boundary in each bank between the area reserved for BASIC and the area reserved for RAM-DISK.

The BOUNDARY commands

The BOUNDARY commands are a set of commands which can be used to look at and change the boundaries in banks 1 and 0.

As you can see from fig 2, to give BASIC more memory you should move a boundary up and to allocate more memory to RAM-DISK you should move a boundary down. To find out where a given boundary is at a particular time, type the following command:

```
OPEN 1,8,15,"Bb":CLOSE 1
```

where b is the bank number. That is, to find where the boundary is in bank 0 type OPEN 1,8,15,"B0":CLOSE1 and to find where it is in bank 1 type OPEN 1,8,15,"Bl":CLOSE 1. When you type one of these commands RAM-DISK prepares a message in its error buffer of the form

```
00,BANK N XXX..YYY,00,00
```

where N is the bank number and XXX is the page number of the boundary. We will deal with the meaning of YYY later. To read this message simply PRINT DS$.

To move the boundary up or down, decide where you want to put the new boundary and type the following

```
OPEN 1,8,15,"Bb,p":CLOSE 1
```

where b is the bank number and p is the page number of the new boundary.

You can check that the boundary has moved by looking at the RAM-DISK directory to see how many blocks are free or by using the BASIC FREE(b) function.

You may move the boundary in bank 0 at any time (even from within a program) regardless of whether or not there are files stored on RAM-DISK. Any change to the boundary in bank 1 however **should be preceeded by a CLR**.

Errors

If you try to move the boundary in either bank to a position which would leave insuficient memory to store the files currently on RAM-DISK, the boundary will remain unchanged and you will get a NO BLOCKS error message. If you try to move the boundary in bank 0 to a position which would leave insuficient memory to accommodate the currently resident BASIC program, the boundary will remain unaltered and you will get an OUT OF MEMORY message.
If you move the boundary in bank 1 to a position which would not leave suficient roan for the variables, arrays and strings created by a particular program, you will get an OUT OF MEMORY message when that program is RUN.

You should experiment to find the best boundaries for your working environment.

Programming in BASIC and machine code

If you are calling machine language routines from within a BASIC program you will want to reserve some memory for the machine code and data. In order to give you the freedom to put hunks of code wherever you wish, RAM-DISK provides an extension of the boundary command which can raise or lower the TOP of the area allocated to it as follows:

OPEN1,8,15,"Bb,p1,p2":CLOSE 1

where b and pl are the bank number and boundary page as before and p2 is the page number of the highest page allocated to RAM-DISK in that bank. This command therefore allows you to allocate in either bank, an area anywhere in memory for exclusive use of RAM-DISK. For example, the command OPEN 1,8,15,"B0,100,200" will allocate the area between 100*256=25600 and 200*256+255=51455. The +255 in the second expression arises because p2 is the page number of the **beginning** of the last page allocated to RAM-DISK. It is of course your responsibility to see that your code does not interfere with this area.

If you need to change the upper boundary (p2) and leave the lower boundary unchanged, you can if you wish use the syntax
OPEN1,8,15,"Bn„p2":CLOSE 1.

We can now explain the meaning of YYY (above) that appears in the message produced in response to an OPEN1,8,15,"B0":CLOSE 1 or OPEN1,8,15,"Bl":CLOSE1 command. It is the page number of the upper boundary.

If you wish to prevent RAM-DISK from using any memory in either bank 1 or bank 0 (apart from the RAM-DISK program itself), use an extended boundary command in which the lower boundary is greater than the upper boundary.

NOTE: The boundary commands may not be used to de-allocate the space occupied by the RAM-DISK code itself. Any attempt to do this will result in a PAGE IN USE error message.

Pure machine code programming.

If you are programming in pure machine language or if you are doing weird things with the 128 memory map, this section is for you.

The ALLOCATION commands > and <

At the lowest level, RAM-DISK allows you to allocate pages to it which need not be contiguous, and which may be dotted about anywhere in memory via the allocation commands. OPEN 1,8,15,">bank,page-list":CLOSE 1 allows you to allocate pages and OPEN 1,8,15,"<bank,page-list ":CLOSE 1 allows you to de-allocate pages. Page-list is a comma separated list of the pages that you wish to have allocated (or de-allocated). The list may also contain page ranges expressed p1..p2. As an example, the following command allocates pages 100, 130 and pages 140 through 155 in bank 1 to RAM-DISK.

OPEN 1,8,15,">1,100,130,140..155":CLOSE 1

The actual addresses of the memory allocated will be 100*256=25600 to 100*256+255=15855, 130*256=33280 to 130*256+255=33535 and 140*256=35840 to 155*256+255=39935.

When RAM-DISK first initializes it allocates about half of the 128's RAM automatically and clears these areas. To prevent this happening use the alternative cold start entry point at $FF53.

It is perfectly possible to combine the allocation commands with the boundary commands. In fact the boundary commands actually work by making calls to the lower level allocation routines. Note however that whereas the boundary commands affect the operating system's pointers to the top of memory in banks 0 and 1, the allocation commands leave these unaltered.

Errors

If you try to allocate a page that is already allocated you will get a PAGE ALLOCATED message. If you try to de-allocate a page which is not currently allocated you will get a PAGE NOT ALLOCATED message.

Communicating with RAM-DISK from machine code

RAM-DISK works by re-directing the OPEN, CLOSE, LOAD, SAVE, GET A BYTE and PUT A BYTE vectors in high memory.

All the RAM-DISK commands discussed above can be issued from machine language in exactly the same way as commands are issued to a physical disk drive, details of which are to be found in Commodore's own documentation. To send a command to RAM-DISK (or floppy disk) or to open a file proceed as follows:

1) Set the system variables for the current logical address ($B8), the device number ($BA), the secondary address ($B9), filename length ($B7) filename address ($B3), filename bank ($C7) and the filename string itself to appropriate values.

2) Call OPEN ($FFC0) and then CLOSE ($FFC3).

The procedure is the same for opening a file as it is for sending a command (e.g. COPY) the only difference being that to send a command, the secondary address should be set to 15 and the filename string to the text for the command.

Getting and putting bytes

This is done in exactly the same way as bytes are got or put to any other device, that is by sending a talk or listen, writing or reading the data and sending an abort i/o.

Memory occupied by RAM-DISK.

RAM-DISK is a machine code program occupying a little less than 8K which loads at the top of memory in bank 0 (or in some RAM-DISK issues, optionally in bank 1). Other areas used by RAM-DISK are $03F0 to $0400 and $1300 to $1400 in common RAM.

Cold start and Warm start

There are three initalization entry points provided by RAM-DISK which re-start the system in different ways. These are as follows

1) Start RAM-DISK from cold
2) Start RAM-DISK from cold without allocating any pages
3) Warm start RAM-DISK.

There are circumstances, such as after a hardware reset, when RAM-DISK will lose its links into the operating system. On these occasions you can use the warm start entry point without loosing data. This procedure re-establishes RAM-DISK's links with the operating system and closes any open files. Provided the areas used by RAM-DISK have not been corrupted no files will have been lost.

The addresses of the entry points may be found in the section **BASIC and Machine Code Entry Points**

Command summary

The following standard disk commands have the same effect on RAM-DISK as they do on the 1541 or 1571 floppy disk drive:

OPEN	CLOSE	DOPEN	DCLOSE	APPEND	CATALOG
DIRECTORY	CONCAT	DCLEAR	COPY	SCRATCH	COLLECT
LOAD	SAVE	DLOAD	DSAVE	BLOAD	BSAVE
RENAME	HEADER	DVERIFY			

pattern matching with ? and * is supported.

RAM-DISK will respond to a range of extended commands. These must be issued by opening a file to the error channel and writing the command to it as follows

```
OPEN 1,device,15,"command":CLOSE 1
```

where device is the current RAM-DISK device number

The following is a list of the allowable commands

CMD TYPE	COMMAND	MEANING
UNIT	Un	Change RAM-DISK device number to n
BOUNDARY	Bb	Prepare an error channel message giving upper & lower boundaries for bank b
	Bb,pl	Change lower boundary in bank b to pl
	Bb„p2	Change upper boundary in bank b to p2
	Bb,pl,p2	Change lower boundary to pl and upper boundary to p2 in bank b
ALLOCATION	>bank,list	Allocate the pages in "list" to RAM-DISK
	<bank,list	De-allocate the pages in "list" from RAM-DISK
QUIT	q	Quit RAM-DISK
DEBUG	d	Put RAM-DISK into debugging mode
	x	Restore RAM-DISK to normal operation

"list" is a comma separated list of page numbers and page ranges. A page range is a pair of page numbers separated by two dots. See the example under The Allocation Commands.

The extended COPY command

The COPY command has been extended to allow copying between RAM-DISK and floppy disk. Any string which begins with an ampersand (&) is taken (without the &) as being the name of a floppy disk file on device 8 as follows:

COMMAND	MEANING
COPY"file1" TO "file2"	Copy file1 on RAM-DISK to file2 on RAM-DISK
COPY"&file1,type" TO "file2"	Copy file1 on floppy disk to file2 on RAM-DISK (see note)
COPY"file1" TO "&file2"	Copy file1 on RAM-DISK to file2 on floppy disk
COPY"*" TO "&"	Copy all files on RAM-DISK to floppy disk

NOTE: The command for copying a floppy disk file to RAM-DISK requires the filetype to be appended to the disk filename as shown above. Possible filetypes are **p** (prg), **s** (seq) and **u** (usr). For example, to copy the program file "prog" to RAM-DISK we would use the command COPY "&prog,p" TO "prog" and to copy the data file "data" to RAM-DISK we would use the command COPY "&data,s" TO "data".

BASIC and machine code entry points

These entry points may be called from BASIC as follows

BANK b:SYS(decimal-entry-point)

where b is the bank (usually 0) where the RAM-DISK code is loaded. Entry points should be called from machine code using a JSR.

HEX	DECIMAL	ROUTINE
FF50	65360	Cold start RAM-DISK. (This is where RAM-DISK enters when it is started from the HACK-PACK menu)
FF53	65363	Alternative cold start. Initialise RAM-DISK without initialising any blocks
FF56	65366	Warm start. Restarts RAM-DISK preserving all files

Summary of error messages

Error messages are generated by RAM-DISK in exactly the same way as they are by a floppy disk drive. All the BASIC 7.0 disk commands cause the disk (or RAM-DISK) error channel to be read and assigned to the reserved variable DS$. To see the error message, PRINT DS$.

Debugging mode

A possible source of problems is that RAM-DISK does not have an error indicator as does the 1541/1571. It is therefore not possible to see when things are going wrong by looking at the disk drive. Of course programs should always check DS or DS$, after disk operations, however, to aid program development, RAM-DISK can be put into debugging mode by sending the character d to the error channel. When RAM-DISK is in debugging mode, BASIC programs will stop on any RAM-DISK error condition. Debugging mode may be switched off by sending an x to the error channel.

The following is a complete list of RAM-DISK error messages. Where appropriate the error numbers are the same as those illicited by the 1541/1571. RAM-DISK specific error messages have numbers in the range 80 to 86.

MESSAGE	MEANING
00, OK.	All is well
00,BANK N XXX..YYY.	The response to a BOUNDARY information request.
00,BYE.	The last message left by RAM-DISK when it quits.
23,FILE CORRUPT.	A page allocated to RAM-DISK has been corrupted.
30,SYNTAX ERROR.	A command is syntactically erroneous.
61,FILE NOT OPEN.	An attempt has been made to access an un-opened file.
62,FILE NOT FOUND.	RAM-DISK can find no trace of the file requested.
63,FILE EXISTS.	An attempt has been made to write a file which has the same name as one already stored on RAM-DISK.
64,FILETYPE MISMATCH.	The file type specified or implied by a command does not match the file of that name on RAM-DISK.
72,NO PAGES.	A command requests more pages than are available.
72,DIRECTORY FULL.	The directory can accommodate no further files.
81,OUCH.	An attempt has been made to allocate a page occupied by the RAM-DISK code.
82,PAGE NOT ALLOCATED.	An attempt has been made to de-allocate an already un-allocated page.
83,FILE OPEN.	An attempt has been made to open an already open file.
84,READ FILE ONLY.	An attempt has been made to write to a read file.
85,WRITE FILE ONLY.	An attempt has been made to read from a write file.
86,PAGE NOT RAM.	An attempt has been made to allocate a non RAM page (should not occur).

There are occations when it is usefull to be able to tell the source of an error message, that is, whether it came from RAM-DISK or from floppy disk. RAM-DISK error messages can easily be identified by the period (full stop) which always terminates the text of the message.

Error messages are annunciated by RAM-DISK, just as they are by Commodore floppy disks, with a trailing ,XX,XX. Floppy disk drives use these positions to indicate any track and sector address associated with the error. RAM-DISK uses them to provide certain diagnostic information for use by the manufactures. This information need not concern the user.

The power up message

The issue number of your RAM-DISK may be found by reading the error chchannel before any commands have been issued. The message will be as follows

73,RAM-DISK (C) 1986 SSOL IX.X,00,00

where X.X is the issue number.

What is the COMPRESSOR

Compressor is a utility for squeezing BASIC programs dawn into their smallest possible size. First, Compressor removes all "dead wood" such as spaces and REM statements from your program. Next it reduces the number of lines in the program by attempting to pack as many statements onto each line as possible.

Compressor acts on a disk file and writes its output to disk (or RAM-DISK).

Using Compressor

1) Prepare a work disk and save a copy of the program you want to compress onto it. Make sure that there are enough blocks free to accomodate the compressed program.

2) Mount the HACK-PACK system disk and type

RUN "compressor"

After a second or two Compressor will respond with

COMPRESSOR: MOUNT WORK DISK - HIT RETURN

Obey, now you will get the message

PROGRAM NAME ?

Type it in and hit RETURN

Compressor runs through two passes. When it has finished it will display some statistics about your program such as the number of GOTOs and GOSUBs and the total number of lines in both versions of the program.

If you look at the disk directory you will see an extra file with the same name as your original program, but with the characters .c appended. This is the compressed program. DLOAD it just as you would any other BASIC program.