ARC 64 for the C-64

For Making Commodore file ARChives
For Dissolving Commodore file ARChives
For Making Commodore file SDArchives
For Dissolving Commodore file SDArchives
For Dissolving IBM style ARChives

PLEASE READ THE IMPORTANT DISTRIBUTION AND LEGAL NOTICES ON PAGE 1
BEFORE USING OR DISTRIBUTING THIS PROGRAM.

SHAREWARE FEE = $20

ARC 64
CONTENTS

==

IMPORTANT LEGAL INFORMATION:

Parsec, Inc. has bought the complete rights to all of Chris Smeets's archive programs, including ARC and CS-DOS. All shareware contributions must now go to Parsec, Inc.

DISTRIBUTION TERMS:
ALL older files relating to ARC and CS-DOS -> MUST <- be removed from every network and disk collection, effective 940228, and be replaced with the newer archives from Parsec, Inc. The archives can only be distributed on networks and by non-profit Commodore approved User Groups. Such User Groups distributing the files on disk can not charge for the disk or its copying.

The files CAN NOT be distributed on "public domain", "shareware", or any other type of magnetic media except by Parsec, Inc. The archives must be distributed COMPLETE in their SFX form and not be renamed. The latest archives can be found on GEnie or are available on disk from Parsec, Inc.

ARC/CSX/SDA/SFX and its routines may not be altered or combined into other programs.

ARC/CSX/SDA/SFX are "shareware" and "by Parsec" and have to be referred to that way in any documentation or ads. This is a legal requirement.

ALL RIGHTS RESERVED

ANYONE INTENTIONALLY VIOLATING THESE DISTRIBUTION TERMS WILL BE HELD LIABLE FOR EVERY COPY TIMES $20 PER COPY, PLUS ALL DAMAGES AND LEGAL FEES.

==

For a free catalog of all our products, send your name and address to:

Parsec, Inc.
PO BOX 111
SALEM, MA
01970-0111
USA

COMMODORE 64 FILE ARCHIVE UTILITY

SHAREWARE

Shareware is an attempt to bypass normal market channels thereby allowing software authors to distribute products without the high cost of conventional advertising and promotion. You benefit by having these savings passed on to you, and we benefit by not having to invest in costly advertising, fancy manuals and packaging. After all, if the software does the job, who needs the fancy package!

If the software does not do the job, then simply do not use it and pay nothing; pass it on to someone else, maybe they will find it useful. At least it did not cost you anything to experiment.

On the other hand, if you find yourself still using the program after a couple of weeks, then obviously it is of some worth to you and you must send in the shareware fee. ARC is distributed as SHAREWARE, but ARC is NOT free software. If you use ARC regularly, then you must send in the contribution. Please be sure to print your name, address, postal zip code, and country.

THE SHAREWARE FEE CONTRIBUTION FOR ARC 64 IS $20.

If you would like a printed manual and disk, then please order from our catalog and just add the regular Shipping and Handling charges like you would on any other order.

Please send your shareware fees to:

PARSEC, INC.
PO BOX 111
SALEM, MA 01970-0111
USA

DISTRIBUTION AND USE

ARC is protected by copyright. This means that the copyright holder (Parsec Inc) is the only person that may legally duplicate ARC or permit someone else to do so. Parsec reserves all rights. Parsec Inc does however allow distribution under these terms:

YOU MAY NOT:

1) Duplicate the PRINTED documentation for ARC either in whole or in part by any means for any purpose.

2) Charge any fee whatsoever for duplicating the machine readable version of ARC, whether done for profit or otherwise.

3) Modify ARC, its documentation, or the SFX file in any way.

4) Include ARC within any program without express written permission and the signing of a license agreement.

YOU MAY:

1) Use ARC on a trial basis only. If you intend to use ARC for more than just a trial then you must send in a contribution.

2) Duplicate the machine readable files associated with ARC (SFX files) provided that this is done free of charge and without modification.

Program development takes time. A lot of it. The more support we get from you in the form of contributions and constructive criticism, the more motivated we will be to spend our time on improving ARC. The amount of time and effort we will be putting into ARC will be in direct proportion to the amount of user support we get.

DISCLAIMER

Parsec, Inc. makes no warranty, either expressed or implied, including but not limited to any implied warranties of merchantability or fitness for a particular purpose regarding these materials and makes ARC available solely on an 'as is' basis.

In no event shall Parsec, Inc. be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of purchase or use of ARC. The sole and exclusive liability of Parsec, Inc, regardless of the form of action, shall not exceed the purchase price of the materials described herein.

IF YOU DO NOT AGREE TO THESE TERMS, THEN DO NOT USE ARC.

UPDATES

The easiest way to get an update is simply to download it by modem from our GEnie library. The most recent version of ARC will always be posted there or available on disk from our catalog.

SUPPORT

We offer support for this product in two forms:

1) on GEnie in the Commodore Bulletin Board
2) through mail
3) in our magazine, TC-128/64.

If you wish to post ARC on your bulletin board for public access, then we would prefer it if you would make the most recent version available and delete previous versions. Please note ALL older versions from Chris Smeets MUST be removed from your disk library and BBS.

When updating ARC, we will try to stick to the following version numbering scheme:

ARC version X.YZ

If Z changes, then a minor change has been made to ARC, and it is probably not worth the effort of updating. If Y changes, then a useful change has been made and it is probably worth the effort to get an update. If X changes, then a major change has been made, and you will have to update if your version of ARC has a lower value of X. X is the number that you see in the column marked "V" when you list an archive's directory.

For example, when we added the /w switch to DIR, the version changed from 1.33 to 1.34. When we added three new data compression techniques, the version changed from 1.34 to 2.00. When we added ARC/F, the version changed from 2.00 to 2.10.

Note that if your version of ARC has the same value of X as the latest version, you will be able to extract files from archives created by the newer version. If X is lower than the most recent ARC, then you may come across some archives which your version of ARC can not handle.

The actual ARC program itself will be distributed as

"ARC###-#.sfx"

where ### is the version number (Example ARC250-2.sfx), the "-#" will be the the number of the file in a series of files that completes the shareware package.

BUGS IN ARC

As we become aware of them, we will try to repair any bugs which manifest their ugly heads.

INTRODUCTION

ARC was inspired by the program of the same name which has been available to MS-DOS users for some time now. It allows you to take several related files and back them up into a single archive file. The archive can later be extracted to obtain exact duplicates of the files that went into it.

Since ARC automatically compresses the files as they are being archived, the resulting archive file takes up less disk storage space than the original files. Savings vary with the types of files being archived and is typically 50% for text and 10-30% for program files.

The primary advantage of ARC is as a backup tool. Word processing files, assembler source code, database files, graphics and music files compress rather well. In fact, due to their heavy use of graphics, pictures, even games, can squeeze down quite well in some instances. Normally a 1541/4040 diskette has a capacity of 664 blocks of storage. Using ARC, you can increase this to the equivalent of about 800 blocks for program disks and 1300 or more for text and graphics disks.

Another big advantage of ARC is the fact that several files can be combined into a single archive. This makes it significantly easier (and faster) to transfer information from one computer to another via the telephone lines. If the bulletin board you are calling uses ARC files, you will spend less time reading directories and trying to figure out which, and how many files you have to download to get the program you want to work! Not only this, but after you extract the archive, all of the files will have the correct name and filetype so renaming files is not nessessary.

ARC is also a useful tool for general file maintenance. Moving files between drives, renaming files, viewing directories, and the editing of small text files is rather simple with ARC.

When Huffman coding is used to squeeze files, the compression ratio (the original length divided by the squeezed length) is about 1.11 for programs, 1.6 for text files, and can be 2.0 or more for graphics screens. Huffman coded files show up as 'squeezed' on an archive directory. Run-length squeezing ('packing') is sometimes more effective, particularly on graphical information.

Version 2.00 of ARC introduced three new data compression methods. 'Squashing' combines run-length coding with Huffman squeezing and is generally superior to Huffman squeezing alone. 'Crunching' is by far the most interesting method used to date in that ARC will choose crunching as the most efficient compressor as often as 80 to 90% of the time. Crunched files are stored using the Lempel-Zev-Welch algorithm and can result in very impressive compression ratios.

A typically text file will compress to about 45% of its original length, and word-pro or paperclip files will crunch down to about one third of their original length. Crunching combines run-length and Lempel-Zev-Welch compression and is unique in that it can be performed without requiring an

initial pass through the data file. It is now possible to override ARC's choice of the compressor, and crunch everything in only one pass! Although this does not necessarily ensure the most efficient use of disk space, significant time savings can be achieved with only a small sacrifice in storage.

In general ARC makes two passes through each source file. On the first pass ARC counts the number of bytes in the file and calculates a two byte checksum for each file being archived. The checksum is stored in the archive and is later used to verify the file's integrity. ARC also prepares a frequency distribution of the bytes used in the file and uses it to generate the Huffman codes. It then calculates what the resultant length would be using each of the three storage methods (store, squeeze, squash, crunch, or pack). Pass two then writes the file to the archive using the most efficient of the five methods available.

In order to achieve some of the flexibility and ease of use of the MS-DOS version we had to do something about Commodore DOS. After all is not:

```
del a:scrapfile1 c:scrapfile2 <return>
```

much easier than:

```
open 15,8,15
print#15,"s0:scrapfile1"
close 15

open 15,9,15
print#15,"s0:scrapfile2"
close 15
```

or how about:

```
move a:arc.exe b:arc.asm c:usq f:
```

That would take some doing! Yet it is the type of thing you want to do quite easily when you are manipulating your disk library.

In general drive letters are both easier and more precise than drive numbers. If you are familiar with MS-DOS or CP/M you probably prefer drive letters. If not, they may take some getting used to. What we have done is assigned each drive a letter according to the table:

drive letter	device number	drive
a	8	0
b	8	1
c	9	0
d	9	1
e	10	0
f	10	1
g	11	0
h	11	1

If you want a program on drive 1 of unit 10, you can do any one of the following:

```
dload "program",d1 on u10    (basic 4.0)
or
load "1:program",10          (basic 2.0)
or
load "f:program"             (with ARC)
```

If you want to load and run the program just type:

```
f:program  <return>
```

ARC will load it and run it (or SYS to it if it is machine language)

In any case we felt that drive letters were sufficiently useful that they would be worth the bother of getting used to. If you are still not convinced think about this example after you have read on.

```
arc/c d:arcfile a:asm.?? b:ed.?? c:*
```

We think you will grow to appreciate the environment in which ARC is available. Most of the commands that you will often require to view directories, move files about, delete files, rename files, or edit simple text files are at your disposal. Future versions of ARC will have even more commands. We felt that this is the environment in which ARC would prove most useful.

Without much further ado we will give you a run down of the commands available.....

[square brackets]	indicate optional parameters
<angle brackets>	indicate required parameters
d:	is a drive letter (usually a destination drive)
s:	is a drive letter (usually a source drive)

1) general DOS commands

```
 a:
```

Selects drive "a" as the default drive. Whenever a command uses a drive letter as an optional parameter, and you do not specify the drive letter, the default drive will be used. Valid drives are the letters A through H

```
 d:filename
```

This will search drive d: for the program specified and if it is found it will be run. If it is a machine language program, then ARC will SYS to its load address. Note that the LOAD is non relocating. That is, it is the same as LOAD with ,8,1

```
 date dd,mm,yyyy
```

This is used to set the date. dd is the day, mm is the month, and yyyy is the year. Since the date is stored internally in the same format as used on the IBM PC, the possible values for yyyy are 1980 through 1980+127.

If you enter an illegal value for one of the above parameters, then the date will not change.

The date is stored in the header to every archive entry, and is a useful way to keep track of which backup is more recent than another. It is also useful when posting programs on bulletin boards for public access, as it indicates how recent a particular version of a program is. Of course, the date will not always be meaningful. In this case we suggest you just leave it set to the default of 01jan1980.

```
 dir[/w] [s:[pattern1]] [s:[pattern2]] ...
```

Lists a directory to the screen. Some examples are:

```
    dir                    - lists default directory

    dir a:                 - list directory of drive a

    dir b:???              - list files on drive b whose
                             names are 3 characters in
                             length.

    dir a*,b*              - list files from the default
                             drive whose names start with
                             'a' or 'b'

    dir a:t* c:t*          - lists all files on drives a:
                             and c: whose names begin
                             with 't'

    dir/w                  - lists the default directory,
                             filenames only.

    dir/w a:a* b:b*        - lists filenames only for files
                             on drive a: whose names begin
                             with "a" and on drive b:
                             with "b".
```

```
 del [d:]<pattern> [[d:pattern] [d:pattern]....]
```

Scratches files from the specified drives. Up to 9 different patterns may be given, but only one is required. Some examples are:

```
     del arc               - delete "arc" from default drive

    del a:temp* b:temp*   - delete all files from drive a:
                            or drive b: whose names start
                            with "temp"
```

```
MOVE <[S:]PATTERN> [[[S:]PATTERN]...] <D:>
```
Copies files between drives. As many as eight patterns may be given as parameters. If either the source or the destination drive is omitted, it is assumed to be the default drive. Some examples are:

```
    move a:arc c:          - move file "arc" on drive 8,0
                             to drive 9,0

    move a:* b:* c:        - move all files from drives a:
                             and b: to drive c:

    move b:test            - moves file "test" on drive b:
                             to the default drive

    move test b:           - moves file "test" on the
                             default drive to drive b:
```

NOTE: MOVE will not handle relative files properly.

REN [D:]OLDNAME NEWNAME
renames a file named 'oldname' to 'newname' on drive d:

TYPE [S:]<PATTERN>
Simply copies the contents of the file named to the screen. Only the first file found will be displayed if wildcards are used.

>disk command

This should be a familiar command. It is the universal wedge, DOS 5.1, DOS wedge, or whatever you want to call it. The only difference is that the command always goes to the default device. Assuming that a: is the default drive, some examples are:

```
    >i1                    - initialize drive 1 (b:)

    >c0=1                  - copy drive 1 to drive 0 on
                             unit 8 (dual drive)

    >r1:test=text          - renames "text" to "test"
                             on drive b:

    >$0                    - lists directory of drive a:

     >$c:                   - lists directory of drive c:
```

SIZE [S:]<FILENAME>
Counts and displays the number of bytes in a file. If the file is a program, you may want to deduct two bytes from the length since the load address is included in the count.

START [S:]<FILENAME>
Prints the load address of a program file.

2) EDITOR COMMANDS.

The editor is primarily intended to edit text files. It was designed to be a text editor for use in editing assembly language source files, although it could be used to edit BASIC programs as well. Find, Change, and Renum, however, will not recognize BASIC tokens or line number references, so you would be better off using POWER or SYSRES or some other editor for BASIC programs.

TEXT
This command puts the editor in text mode. Any upper case characters not in quotes will be forced to lower case, and basic keywords will not be tokenized. When listing a Basic program, you should set BASIC mode, because in text mode, BASIC tokens are not expanded.

BASIC
Puts the editor in BASIC mode. Lower case is not forced, and basic keywords will be tokenized and expanded by LIST.

AUTO [INCREMENT]
Turns on automatic line numbering. Auto with no increment disables auto.

COLD
Same as sys 64738 on the 64. (COLD START)

If you have a reset button on your 64, or if you are using a C-128, then you can re-enable ARC by typing: SYS 12*4096 after a reset.

change/from/to/, range

Changes string from to string to. / can be anything not included in from or to. range is a range of line numbers having the same syntax as LIST.

For example:

change/"//	deletes all quotes from the text in RAM
change/$ffd2/chrout/,2000-	changes all occurences of "$ffd2" to "chrout" in lines numbered 2000 or higher.

DELETE RANGE
Deletes a range of lines from memory. Same syntax as BASIC's LIST.

delete -2000	deletes lines up to and including line 2000
delete 3010-4030	delete lines 3010 to 4030

FIND/STRING/, RANGE
Prints lines that contain string

GET[/CHAR] [D:]FILENAME [,STARTLINE]
Similar to load, except the file is a sequential text file. If startline is given, then the file is appended wiping out any lines numbered equal to startline or higher.

Since blank lines and upper-case alphabetics tend to disrupt the scrolling of text, the /" switch is included to allow inclusion of a quotation mark (or any other character) at the beginning of each line. This allows you to manipulate these types of files in RAM.

KILL
Disables the editor, scroll, and restores standard load/save and other vectors.

RENUM <FROM,START,INCREMENT>
Renumbers lines starting with line from. Any lines numbered lower than from are not affected. This command should not be used on BASIC programs, as it does not attempt to update any GOTO or GOSUB references accordingly.

PUT <[D:]FILENAME> [,RANGE]
Stores text from memory to disk. If the file exists it will be be deleted before the new copy is transferred to disk.

REPEAT
Enables program scroll and repeat on all keys.

OFF
Kills repeat and program scroll.

OLD
Recovers from NEW.

HEX NUMBER
HEX $HEXNUMBER
Converts hex to decimal or visa versa.

#
Shorthand for: renum 1,1000,10

3) ARC

Creating an Archive.

```
arc/c      <[d:]arcfile[.arc]> <[s:]pattern> [[s:]pattern]]
arc/cx     <[d:]arcfile[.arc]> <[s:]pattern> [[s:]pattern]]
```

Creates an archive on drive d: named "arcfile.arc". As many as eight patterns may be given. The possible values for x are:

```
n - no compression; store only
p - pack only
s - squeeze only
q - squash only
c - crunch only (two passes)
1 - crunch only (one pass)
```

If you choose option x=1 then part of the information that ARC needs to unarchive the file must be stored at the end of the archive entry rather than the beginning. In particular, this is the files original length and the checksum.

Unfortunately, ARC needs to know a relative file's length when it is at the start of the archive entry so that disk space can be properly allocated. Therefore ARC will not allow the single pass crunch option for relative files. If you do select x=1 when archiving a group of files, any relative files encountered will be processed as if you had chosen x=c.

Some examples:

```
arc/c a:sq b:sq.* b:usq.*
```

This example creates an archive named "sq.arc" on drive a: which includes all files from drive b: whose names begin with "sq." or "usq."

```
arc/cs  a:test.arc f:*
```

Archives all files from drive 10,1 to "test.arc" on drive 8,0 and squeezes all entries.

```
arc/c seq b:*=s
```

Archives all sequential files on drive b: to "seq.arc" on the default drive.

Previous versions of ARC would allow you to encrypt an archive by specifying a password when creating an archive. Unfortunately, this creates a number of problems when crunching files and had to be eliminated. Besides, we never used it anyway. Did anybody else?

Appending to an Existing Archive.

```
ARC/A[X] <[D:]ARCFILE[.ARC]> <[S:]PATTERN> [[S:]PATTERN]]
```
This command is exactly the same as the create archive command and all of the above examples apply. The only difference is that the archive file must exist, and the new entries are appended to it.

If the archive does not exist, a file not found message is issued and the program aborts.

Sometimes the number of blocks displayed on a directory listing does not agree with ARC after an append. This is a bug in Commodore DOS, and should be harmless. Oddly enough, if you try to fix this by validating the disk, you will be wasting your time since the directory block count will not be affected.

Since ARC can handle any archive created by lower versions of ARC, it is ok to append to a version 1 archive.

You should exercise some caution when appending to archives. Every archive must be an integer multiple of 254 bytes in length. When transferring archives via modem using the XMODEM protocol, files are transferred in 128 byte blocks. If the file is not an integer multiple of 128 bytes in length, as the majority of files are not, XMODEM "pads" the file with blanks or nulls. Consequently, an extra disk sector gets added to the end of an archive! It is easy to fix this problem by using a disk doctor program. Simply change the link for the second last sector in the archive to (0,255).

You should have no problem extracting files from XMODEM downloaded archives, because ARC will ignore this XMODEM padding. BEFORE YOU APPEND to an archive, you should fix this problem, or else the files appended to the archive will not be able to be de-archived. If you get a "version error" after extracting all the files from a downloaded ARC, that is normal. That is just the XMODEM padding ARC was trying to read.

We have added a new command to ARC to check an archive for this type of problem.

Simply type:

```
ARC/F ARCHNAME[.ARC]
```

ARC will trace the track,sector links for the archive and get rid of the last sector in the archive. If the archive is ok and the link for the last sector is already (0,255) as it should be, then ARC does nothing and prints the message: archive is ok.

If the archive is bad and the link is anything other than (0,255) then ARC will go to the second last sector in the archive and change its link to (0,255). This drops the extra sector added by the XMODEM padding. ARC will not free up the deleted sector, nor will it adjust the directory block count for the archive. If you need the extra block free, then validate the disk to get it back.

EXTRACTING FILES FROM AN ARCHIVE.

```
ARC/X[D]     <[D:]ARCHFILE[.ARC]> [PATTERN1] [PATTERN2] ...
```

This command is used to extract files from an archive. If no patterns are specified, then ARC will extract all files from the archive. When patterns are given, then only those files in the archive which match one of the patterns will be extracted.

Since ARC no longer supports encryption of data files, it will not be able to handle an encrypted file. You will have to use a previous version of ARC. Since these early versions of ARC were taken out of circulation a long time ago, it is doubtful you will ever encounter them.

Some examples:

ARC/X A:SQ
Extracts all files from "sq.arc" on drive 8,0 and puts them on the default directory.

ARC/X A:SQ *
Same as the above

ARC/XF B:TEMP.ARC
Extracts all files from "temp.arc" on drive 8,1 and puts them on the diskette in drive f: (10,1)

arc/e c:book030386 chapter1
Extracts only the file 'chapter1' from the archive 'book030386' on drive c:

NOTE: arc/x and arc/e are synonyms

VIEWING AN ARCHIVE DIRECTORY.

ARC/L <[D:]ARCHNAME[.ARC]>
This is used to get a listing of the files in an archive. A typical archive directory might look like the following:

ARC/L A:ARCHELP

```
file archive utility version 2.00
08/86 ... chris smeets
directory for archive: a:archelp.arc

======================================================
filename      type blks now  v stowage  date
======================================================
xarc.typeme      p 129  41   2 crunched 24aug86
samples          s 22   9    2 crunched 24aug86
================   ==== ====
2 files            151  50
```

The first two columns give the filename and its type (p,s,u or r). The next two columns give the files length in disk blocks before and after compression. If

the file was relative, then you may not see the same number as you would on the directory, since the relative file's side sectors are not needed by ARC.

The column labeled "v" is the version number of the archive. At present there are three possible versions.

"1" refers to archives created by ARC programs numbered 1.xx and "2" stands for archives created by ARC version 2.xx. In addition, if you are looking at an MS-DOS archive, the version will be "I"(1).

The next column gives the compression method used to store the file. The last column is only displayed if you are using the 80 column version of ARC.

1. ARC/L should handle any MS-DOS archive created by ARC's numbered 5.10 or lower. At present this is the only ARC function that supports MS-DOS archives.

TAKING A LOOK AT WHAT'S IN AN ARCHIVE.

```
ARC/PX/Y  <[D:]ARCHNAME[.ARC]> [[PATTERN1] [PATTERN2]...]
```
This is used to print the specified files in an archive to the screen. Syntax is similar to that for ARC/X, except the output is to screen instead of to disk.

The optional switches x and y can take on the following values:

```
x=p   Converts from standard ASCII to nonstandard
      Commodore ascii (PETscii).

x=a   Converts from PETscii to true ASCII.

y=p   Sends output to the printer rather than
      to the screen. Before running ARC, you must
      set the printers device number and secondary
      address as follows:
          POKE 1027,device
          POKE 1028,secondary address
      If you do not want to do this every time you
      run ARC, then SAVE ARC before running it.
      The default values are device=4 SA=0.
```

```
ARC/V <[D:]ARCHNAME[.ARC]> [[PATTERN1] [PATTERN2]...]
```
This is used to verify the contents of an archive. ARC calculates a two byte checksum for each file in an archive. The checksum is calculated using the bytes of the original file before they are passed onto the compression routines.

When you verify an archive, ARC actually de-compresses each archive entry and calculates a new checksum using the bytes passed to it by the decompressor. Both checksums should be the same. If they do not match, an error message is displayed indicating that the archive may have been corrupted due to a disk error or a transmission error during upload or downloading.

Again the syntax is the same as that for ARC/X.

By now you must have noticed that ARC/X ARC/P and ARC/V are all minor variations of the same thing.

4) MEMORY MAP

```
     $033c-$03ff        - cassette buffer. used by ARC
     $0801-$0fff        - not used
     $1000-$4fff        - work space for ARC/C and ARC/X
                          CRUNCH string table is stored here
     $5000-$7fff        - workspace for ARC and MOVE
                          commands only. All other commands
                          leave this area alone.
     $8000-$8fff        - used in 80 column version. (ROM)
     $9000-$97ff        - not used
     $9800-$9fff        - 80 column screen.
                          Not used in 40 column ARC
     $a000-$bfff        - program area. ARC, MOVE, and DIR
     $c000-$cfff        - program area. Editor commands.
                          (sys 12*4096 to enable ARC after
                           a KILL)
     $d000-$ffff        - work space for ARC/C and ARC/X
```

You may notice that there is a rather significant jump in the amount of workspace that ARC needs to do its job from previous versions of ARC (28K to be exact). This is due to the CRUNCH routines, which are rather demanding in terms of memory. If you have a program in memory when archiving or dearchiving a file, then it will almost certainly be clobbered by ARC.

When invoking the ARC command, BASICs pointers are not changed in any way by ARC. You may have to type NEW before running a program after using the ARC command (otherwise you may get an ?out of memory error).

THEORY OF OPERATION

Why do you use compression programs? Because of the reocurrance of certain values in data, graphics, and programs. By keeping track of the reoccurring values and replacing them with shorter values, you can compress a file. All methods of data compression take advantage of redundancy of one form or another.

For example, a graphical image stored in RAM may look something like the following if viewed with the machine language monitor:

```
.:2000 00 00 00 00 00 00 00 00
.:2008 00 00 ff ff ff ff ff 00
.:2010 00 00 00 00 00 00 00 00
```

```
.:2018 a0 0b ff ff ff ff ff ff     and so on....
```

Database programs often sacrifice disk space in order to gain speed. Relative files, for instance, store their data at the beginning of each record, and pad the record with zeros. Since every record is the same length, the DOS can easily calculate where each record starts and thus randomly skip to any record in the file. THE MANAGER, and other database programs pad their records with spaces. In either case there is a great deal of space to be gained when packing this type of file. We have seen some DBASE III files in excess of 1 megabyte 'crunch' down to only 50,000 characters or so! Most of this is due to packing.

For those of you that are interested in statistics, we have included a small utility program with ARC that analyzes the frequency distribution of the bytes in a file and graphically displays the results. On the top portion of the screen you will see the frequency distribution of the bytes in the file. On the bottom portion is a bar graph representing the lengths of the Huffman codes generated by the squeeze algorithm. A huffman code can be anywhere from 0 to 24 bits in length. Each bit in the Huffman code is represented by two pixels on the graphics screen. To run the utility you must have ARC in memory and type:

```
a:analyze [d:]filename
```

The program will then read through 'd:filename' and display a frequency distribution for the file.

Archive File Format
Each archive entry consists of a short header followed by the compressed file. If the file is squeezed or squashed, the Huffman encoding table appears immediately following the header and before the file data. The archive header consists of the following bytes:

```
offset  length     description
------  ------     -----------
  0       1        version  1 for ARC 1.xx
                            2 for ARC 2.xx
  1       1        storage. 0=store  1=pack
                            2=squeez 3=crunch
                            4=squash 5=1 pass crunch
  2       2        checksum lo,hi
  4       3        original length-(bytes) lo,mid,hi
  7       2        squeezed length-(blocks) lo,hi
```

```
offset  length     description
------  ------     -----------
  9       1        file type. s,p,u or r
 10       1        length of filename
 11       n        filename
```

The following additional bytes occur if version is 2 or higher.

```
11+n     1        record length if relative file.
                         (254 otherwise)
11+n+1   2        date in MS-DOS format.

                    bits: 0-4 = day
                             5-8 = month
                             9-15= year-1980
```

The file data follows starting at offset 11+n+2.

ARC TRICKS AND TIPS.

1) MANIPULATING ARCHIVES.

If you were to use a disk doctor to look at an archive, you would notice that every file in the archive begins at the beginning of a CBM disk sector and ends at the end of a sector. This fact can be used to advantage, and will be in future ARCs, in a number of ways.

First of all, it makes it possible to break an archive up into a number of smaller archives simply by manipulating the directory and a few track and sector links. Since it is not necessary to move any data, this should be a relatively fast procedure. You could split an archive in two with a disk doctor as follows:

1) create a dummy file on the same disk as the archive.
2) scratch the dummy file.
3) locate the archive and follow the sector links until you see the filename of the archive entry which is to be the first filename in the second archive.
4) jot down the track and sector you are looking at.
5) backup to the previous sector and change the link to 0,255
6) get back to the directory and locate the dummy file from (1)
7) change the filetype from (6) to program (130)
8) change the track,sector to the ones you jotted down in (4)

You now have two separate archives. The block count for these files will not be correct in the directory, and should be fixed, but no errors should arise if they are not. A similar proceedure could be used to delete an archive entry. Simply manipulate the sector links to point around the entry to be deleted and then validate the disk.

You could also re-arrange the order of the archive entries in an archive, append one archive to another, transfer an archive entry to another archive and so on and so forth.   All of this could be done without having to move a single byte of data!

2) RANDOM ACCESS ARCHIVES.

An archive could be made random access in each of two ways.

The first method would be to follow the track and sector links and read the number of CBM disk blocks in each archive entry. This information could then be used to calculate where each archive entry begins.

A second method would be to convert the archive to a relative file. Simply create the archive by normal procedures, and then use ARC to archive the archive you just created. Now load and run your favorite disk sector editor and view the first sector of the archive of the archive. Change the file's type to "R" for RELative in the archive header. (You will see the filetype as a "P" two bytes before the filename) Now all you have to do is unarchive it and it will become a relative file with record length 254 and can be accessed randomly.

But why would you want a random access archive anyway???

We plan to write a simple un-squash routine that can be accessed as a subroutine by a BASIC or Machine Language programmer. Unfortunately the CRUNCH routines require a great deal of workspace (about 28K) and would not be of much use to most applications. However, a small routine could reside inside about 2K bytes which would un-pack, un- squeeze, or un-squash an archive.

If your application requires a large number of graphics screens, or a great deal of text you could cram at least twice as much information on a disk using random access archives!

CHOOSING THE COMPRESSOR MANUALLY

ARC now lets you choose the method of storage yourself. The most useful option is when you opt to crunch all files in a single pass. If disk storage is a consideration, then the following rules of thumb can be used to decide when to choose this option and when to avoid it.

If you let ARC choose the most efficient compressor, then you will find that:

1) Text files almost always CRUNCH.
2) Word Pro or Paperclip files almost always CRUNCH.
3) BASIC programs almost always CRUNCH.
4) Machine language programs less than 40 blocks will usually CRUNCH and sometimes SQUASH.
5) Larger ML programs usually SQUASH and occasionally SQUEEZE or PACK.
6) Graphics images either CRUNCH or SQUASH (50:50) and occasionally PACK

QUICK START DOCS

1) Disable fastload cartridges
2) When no drive designator is given, ARC uses the current default drive.

ARC disk drive designation characters:

DEV = device # DRV = drive #

ARC	DEV	DRV	ARC	DEV	DRV	ARC	DEV	DRV	ARC	DEV	DRV
a:	8	0	c:	9	0	e:	10	0	g:	11	0
b:	8	1	d:	9	1	f:	10	1	h:	11	1

Extracting (dissolving) arc files:

```
arc/x a:filename
```

Extracts all files from the ARC file called "filename" on drive a: (device 8/drive 0 like a 1541/1571 would be.)

Creating ARChive files:

```
arc/c a:arcname.arc a:*
```

This creates an ARChive file on drive a: and puts the files being archived into a file named "arcname.arc" on drive a:. This is the way "I" suggest you archive:

Put all files you want to ARChive ALONE on a disk and then ARChive them all by using the pattern match as exampled above by the end-of-line "a:*". This is MUCH easier than trying to type all the file names you wish archived.

I have noticed that sometimes ARC will try to read the actual ARC file it has created and end with a "write file open" error. This does not effect the archive in any way.

ARC also has a built in wedge. Standard DOS wedge commands can be sent. Here are a few examples:

```
>$*              ;disk directory     (typing "dir" works also.)
>$$              ;free blocks
>*=p             ;displays all PRG files.
>$a*             ;displays all files starting with the letter "a"
>s0:f1           ;scratchs file named "f1"
>i0              ;initialize disk (reads bam/dir into drive memory).
>n0:diskname,01  ;formats disk w/id of "01".
```

I suggest you refer to your drive manual for more help on disk commands.

ARC date setting:

When creating an ARChive it is a good idea to set the date so you and other users will know the date the file was actually archived. Here is an example:

```
date 15,11,1986    ;this sets the ARChive date to 15nov1986
date               ;displays the current date of ARC.
```

Hardcopying those "doc xxx.arc" doc's:

1st you must set your printer device number and secondary address by POKEing values into ARC in one of two possible ways:

1) load arc. do NOT run. Then:
poke2051,(printer device #/defaults to 4)
poke2052,(secondary address/defaults to 4/I suggest CBM printers use 7)

```
2) load/run ARC. Then:
poke13*4096-1,(printer device)
poke13*4096-2,(secondary address)and96
```

If you use method one you can resave ARC so your values will be the default whenever you load your version of ARC.

Other misc ARC commands:

```
arc/l arcfile    ;lists files/info that are in an ARChived file.
arc/p filename   ;types contents of "filename" to screen w/out extracting.
arc/e f1 f2      ;extracts ONLY the "f2" file from the archive file "f1".
load "c:filename ;loads "filename" from drive 9/0.
a:program        ;load/run "program" from drive 8/0.
type filename    ;types "filename" contents to screen.
a:arc aid*       ;boots an "arc menu" that allows easier use of ARC.
```

Changes in ARC version 2.50

With ARC 2.30, it was nessessary to reset the 1541 disk drive by turning it off and then on again before dissolving archives that contained relative files. In order to prevent this, ARC2.40 checked to see what type of drive it was dealing with, and if it found a 1541 or 1571 it would reset the drive with the UI command before proceeding to create or dissolve the archive. I have also heard stories of weird things happening when using the append feature of CBM DOS (openla,dv,sa,"filename,a") like ARC does when creating SDA's. Again, resetting the drive was found to alleviate the problem.

Note that these problems are not ARC problems. They are just a couple of the many bugs that come as standard equipment when you buy a 1541 or 1571.

This means that REL files will still be a problem for people that have only one 1541 or one 1571. You will have to reset your drive manually by turning it off and then back on before dissolving any archives that contain REL files.

The second problem was a little more obscure. If you think the 1541 is slow, thats nothing, try writing to a 1571 double sided disk. Now that is slow!

Well, this problem seemed to be alleviated somewhat by using a secondary address of 1 when writing a file. In general, it seems to work and things are sped up somewhat. So ARC tried to use an SA of 1 whenever possible. This meant an archive being created (but not being appended to) and PRG or SEQ files being extracted from an archive.

Enter the 1581.
ARC 2.50 does not use this sa=1 trick any more. It was found to cause serious difficulty with the 1581 disk drive. See the short basic program "81bug.bas" accompanying this text file for more details on this. So it looks like you 1571 owners are going to have to settle for 1541 mode, at least with ARC64.

Note that these problems were due to bad handling of the drives, and did not relfect any errors in the logic of creating and dissolving archives. (ARC 2.50 is pretty much the same as ARC 2.30 in this regard).

2) ARC/M
It is now possible to "move" files to an archive. This is identical to creating a new archive, so refer to ARC/C in the ARC documentation for details. The difference between ARC/C and ARC/M is that ARC/M will delete (scratch) the source file after it has been archived. This should make it possible to create large archives on single 1541 systems.

You should exercise some caution when using this option. First of all, the resulting archive may be too large to be dissolved if you have only one 1541, and secondly if you do this without first making a backup and something should go wrong, you may lose your source files.

3) C64 Link
ARC was also experiencing difficulty with the C64 Link IEEE interface. The Link requires a unit 15 and ARC only allowed drives up to 11. ARC 2.50 allows for units 8 to 15. (a-p) now.

The Link and ARC were also in conflict over the cassette buffer. ARC no longer uses the cassette buffer, so these conflicts should no longer be apparent. ARC now does a cleaner job of changing the ILOAD, ISAVE and IOPEN vectors, which should also fix a problem or two.

4) Help
Due to popular demand, and a bit of reorganization, ARC will now give you a brief help screen if you type ARC with no parameters.

ARC 2.50 also displays brief help when entering several commands with no parameters. This does not apply to those commands which are valid with no parameters. ARC 2.50 also adds a "help" command to list a summary of available commands.

5) New Memory Map
ARC now sits in memory from $8000-$C0FF. If you use ARC's text editor, watch out! This will reduce the maximum size of a file that can be loaded into RAM with the GET command.

This does, however, free up $C100-CFFF which is a popular place to put small utility programs like 1541 fast loaders and the like. The loader portion of ARC has also been modified to simplify patching in a fast load or save utility. If you have a fastload program that is somewhere between $C100 and CFFF that you would like to boot at the same time as ARC, proceed as follows...

First install your fast loader or whatever else you want to run at $C100-$CFFF.

Next attach it to ARC250.4 with a machine language monitor. If you are using Micromon, the procedure would be ...

```
L 0801 "arc250.4" 08            fetch the unmodified version

A C000 JMP $C100                SYS to enable your stuff

A C100 JSR $XXXX                JSR to initialize your stuff
A C103 JMP $8XXX                whatever used to be at $C000

T C100 CFFF 5200                tag it on to ARC
S 0801 6000 "MY ARC250.4" 08    save it to disk
        :
        --- Replace with 5200+size of your code
```

Next time you LOAD and RUN ARC, the "attached" code will be moved to $C100-$CFFF and initialized before intializing ARC. Do not try to run ARC now, it will not work since BASIC's program pointers do not correctly reflect the size of the program in RAM. LOADing it from disk is required.

If you want to play around a bit, there is also some unused RAM beneath the BASIC roms at $B300-BFFF (approximately). To find out exactly what is free, use the monitor to fill this area with FF's before running ARC, and then browse it again with the monitor after ARC is run. This area will get smaller and smaller with subsequent releases of ARC. Do not count on it being there later on.

6) "write file open"
ARC 2.30 would incorrectly abort creation of an archive when it came across the open archive file in the directory. This has been corrected in ARC 2.50

7) "file exists"
If you inadvertently included a file more than once in an archive, ARC 2.30 would abort dissolution of the archive with a "63, file exists, 00, 00" disk error when it came across the second copy of the file. ARC 2.50 will not do this. It displays a "file exists!" message and then goes on to the next file in the archive.

8) Drives "2"-"9"
Some types of hard disks allow for drives numbered other than "0" and "1". ARC now fetches the drive and device number corresponding to a particular drive letter from a table in RAM. If you are using a hard drive, or simply want to re-assign drive letters, you can modify this table to suit your needs.

There are a couple of other things there that you may want to alter. See the ARC.JUMP file for details. This table sits at $C100 after ARC is run, and at $5100 before ARC is run.

```
$514F (20815) background color
$5154 (20820) border color
$5159 (20825) character color (ASC"char to print")
$515D (20829) drive numbers for device "a" through "p"

$516D (20845) device numbers for devices "a" through "p"

$517D (20861) printer device number (defaults to 4)
$517E (20862) printer secondary address (OR'ed with 96) (defaults $67)
$517F (20863) default drive at start up ( "a" )
$5180 (20864) default unit ( 8 )
$5181 (20865) default date MS-DOS format (2 bytes)
```

Note: best to rummage these areas with micromon in case there is a typo in there somewhere.

If nothing else, you should at least set your printer SA and device number as follows..

```
LOAD"arc250.4",8
POKE 20861,printer device number
POKE 20862,printer secondary address  OR 96
SAVE"new arc240.4",8
```

10) almost 1764 RAM disk support
This is experimental at this point. If you install your RAMdisk as unit 14 with the default interface page at $CF00, most things will work. ARC 2.50 treats unit 14 differently from other drives, so unit 14 cannot be assigned to a real drive.

Install the RAM drive first. Then run ARC.

Most commands, move, ren, del, arc, size, get, put, type worked ok for me (most of the time). The problem areas were...

arc/x	can not have the ram drive as both the source and the destination drive. If either the source or the destination is a floppy, it seems to work ok. I do not know why this is so, but I suspect it is the RAM disk and not ARC that is at fault here. arc/c seems to work ok when the ram drive is both the source and the destination drive and there is not much difference between the two as far as the drive is concerned.
arc/f	do not use it. This requires block read commands which the RAM drive does not support.

ARC uses block read commands when dealing with relative files, so you will not be able to ARC a REL file on the RAM disk, nor un-ARC a rel file to the ram disk.

For general purpose i/o (LOAD and SAVE excluded) this RAM disk is pretty slow. If you are using an IEEE drive, it is quicker and safer to just ARC to/from disk. However, because of the speedy LOADs and SAVEs to RAM disk, it is pretty handy to have around.

In general, I would approach the RAM disk with some caution. It is a great place to LOAD and SAVE from/to and may get you out of a bind if you have only one 1541 and want to dissolve a large archive. However, I would recommend that you avoid using it when creating archives.

ARC tries to be compatible with the RAM drive, but I am not too sure about it. I suspect there are some unresolvable conflicts remaining which may introduce errors if you try to create archives with it. In any case, experiment a bit and be careful!

11) Version Error!
When dissolving archives, the very first byte ARC fetches from the archive is the version number (1 or 2) of ARC which created it. IF ARC found something other than 1 or 2 it would report a version error. This was causing some confusion and making people think there were more versions of ARC around than there actually are.

In reality, a version error could be caused by several things. It could be an archive that has uploaded to CIS with their B-protocol, but downloaded to the C64 with XMODEM. This causes a few extraneous bytes which are quite bothersome to remove to be attached to the start of the file. It also causes a version error.

If you were to try to dissolve an MS-DOS archive, ARC would find a version of $1A and also report a version error. It would do the same if you were to pick a file at random from your library and rename it to "something.arc" and try to dissolve it. For these reasons, ARC now states that such files are not archives, or if they start with $1A, that they are IBM archives.

ARC JUMP TABLE

```
;arc.jmp
;==============
;
zc000  jmp entry    ;enable editor
zc003  jmp kill     ;kill editor
zc006  jmp getlen   ;get length of parameter .x
zc009  jmp movn     ;move parameter .x to (fnadr),y
zc00c  jmp setup    ;setup parameter .x as a filename (sets $B7,$BA,$BB)
zc00f  jmp gtpcn    ;get number of parameters into .a
zc012  jmp prtchr   ;screen print ($E716)
zc015  jmp getdfl   ;default drive=.a device=.x
zc018  jmp getdev   ;.a=device letter. returns drive in .a device in .y
zc01b  jmp pme      ;print immediate
zc01e  jmp delpcx   ;scratch file (parameter number .x)
zc021  jmp zaa71    ;read ds$. abort if error
zc024  ldx #scrsiz  ;get screen line length
 rts
zc027  jmp grdy     ;switch BASIC back in and go ready
zc02a  jmp getdef   ;get default drive letter into .a
```

```
zc02d  jmp setdef   ;set default drive from letter in .a (sec if error)
zc030  jmp color    ;set screen colors
zc033  rts
       nop
       nop
second jmp jsecnd   ;these are used to read the disk error channel, and
tksa   jmp jtksa    ;to send disk commands. They have been modified
acptr  jmp jacptr   ;slightly to intercept unit 14 (m:) and treat it
ciout  jmp jciout   ;a little differently in order to accommodate CBM's
untlk  jmp juntlk   ;1764 RAM disk. Testing was done with RAMDOS111286
unlsn  jmp junlsn   ;version 3.2   If you're having trouble with some
listen jmp jlistn   ;non-standard (if there is such a thing) CBM drive
talk   jmp jtalk    ;try fiddling with these
;
;
color lda #0
 sta $d020
 lda #0
 sta $d021
 lda #""
 jmp chrout
;
drives  .asc "0101010101010101"
units   .byt 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 13, 13
        .byt 14, 14, 15, 15
;
ptrdv   .byt 4     ;printer device number
ptrsa   .byt $67   ;printer SA or'd with $60
defalt  .byt "a"   ;default drive
unit    .byt 9     ;default unit number
date    .wor $0021 ;default date (010180) MS-DOS format
;
```

```
diskq cmp #"a"     ;check for valid drive letter
 bcc dqs           ;too low
 cmp #"q"
 rts
dqs sec            ;carry means bad drive letter
 rts
;
getdfl lda defalt
getdev jsr diskq   ;abort if invalid drive letter
 bcs dqs
 sec               ;convert drive letter to dr,dv
 sbc #"a"
 stx t03+1
 tax
 lda units,x
 tay
 lda drives,x
t03 ldx #0
 clc
t033 rts
;
 .end
```

ARC LOG

```
;arclog
;-----------------------------------------------------------------------
;arc200   aug86
;         Versn changes to 2
;         Files are now stored, packed, packed+squeezed,or packed+crunched
;         Added arc/l for ibm archives
;         Added relative file routines
;         Added single pass crunching.
;         Improved checksum calculation, detects added zeros or
;                                        transposed bytes
;         Added date function
;         Checksum error with packed files has disappeared
;           (pack routines have been completely re-written)
;         DEMO version uploaded to compuserve, delphi
;-----------------------------------------------------------------------
;arc210   sep86
;         Added arc/f - strips xmodem padding from an archive
;         Patched arc/x and arc/l to ignore xmodem padding
;         Removed encryption routines ( they interfere with crunch )
;         Changed signon prompt
;         Fixed scroll-won't scroll if quote or insert mode
;               or while running program
;         Removed 'F1' key - will add it back later using vector at $028f
;         Added 'iqplop' vector to freeze list on shift and ignore BASIC
;               keywords during text mode.
;
;-----------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;arc220   14sept86
;         Fixed bug. ARC210 would ignore the last file in an archive
;                    thinking that it was xmodem padding if it was only
;                    one block long.
;         Patched un-pack routine to detect improperly packed files from
;                    ARC 1.xx (ARC 1.xx would pack n bytes as n+1 bytes
;                    if n was > 255)
;         Added ASCII conversion routines to arc/p as well as output to
;                    printer.
;
;                    arc/p[x]/[y] archfile file-to-type
;
;                    x=a  convert petscii to ascii
;                    x=p  convert ascii to petscii
;                    y=p  send output to printer
;
;                    user must define the printer as follows:
;                    poke 1027, printer device number
;                    poke 1028, printer secondary address
;                    before running arc
;
```

```
;                    default is 4,0
;
;-----------------------------------------------------------------------
;ARC221   02oct86
;         Fixed editor so it won't force lower case in TEXT mode.
;         SCROLL won't expand BASIC tokens in TEXT mode.
;-----------------------------------------------------------------------
;ARC230   20oct86
;         Added ARC/CZ   - disables crunch.
;         Separated ARC extract routines into a separate module which
;             can be used as a subroutine by external applications. (SDA)
;             (ARC/CZ was required since this subroutine can't un-crunch)
;-----------------------------------------------------------------------
;ARC240   21may87
;         Added arc/m (add with delete)
;         Added open 15,8,15,"ui" if 1541 to fix rel file problem
;                   and problems with append.
;         Added drives i-p to allow C64 Link drive 15
;         Added syntax diagram for arc with no parameters
;         Relocated to $8000-$c1ff to eliminate conflicts with programs
;                   that use the cassette buffer (c64 link)
;         Fixed directory read routine so that it will ignore the open
;                   archive file being created...'write file open'
;                   shouldn't be a problem anymore.
;         Added code to bypass existing files, so arc/x won't abort if
;                   the file its attempting to extract already exists
;         Use SA=1 whenever possible to avoid second side slow down with
;                   the 1571 (Does this REALLY work???)
;         Added to jump table $c030 sets screen colors
;                            $c033 initialize drive (ui if 1541/1571)
```

```
;         Put jumptable and drive letter-->unit,drive conversion
;                   routines in a place where they can easily be patched
;                   ARC now uses tables which can be altered by the user
;                   if they want drives "2" through "9" or units other
;                   than 8-15
;         ARC doesn't use IERROR vector anymore and is more careful
;                   when installing new ISAVE, ILOAD, IOPEN. This should
;                   make it compatible with some software fast loaders
;                   and fast savers.
;         ARC doesn't use $C100-$CFFF. This area can be used for turbo
;                   load and save type stuff.
;--------------------------------------------------------------------
;ARC250   20Aug87
;         ARC 2.40 went bezerk with a 1541/71 due to the UI. This code
;         is no longer used.
;         ARC 2.40 did it once too often, ARC 2.50 doesn't do
;         it at all, so REL files are still going to be a problem.
;
;         Added to the JMP table at $C000. ARC now JMPs to the low level
;         KERNEL routines (TALK,LISTEN etc) through this table. It never
;         JMPs or JSRs them directly. Also patched these low level kernel
;         routines in an attempt to achieve some compatibility with CBM's
;          1764 RAM disk. Didn't (and can't) meet with 100% success here,
;          but it should be useful nontheless.
;
;          Added lots of help for those that hate to read docs.
;
;          ARC 2.40 also set the top of RAM for BASIC pointer (MEMSIZ)
;          incorrectly. This has been fixed.
;
;          Added code to intercept an MS-DOS archive during ARC/X and
;          display an appropriate message. This used to result in a
;          "version error" which was causing some confusion.
;
;          deleted any opens with sa=1 due to 1581 incompatibility.
;          ARC 2.50 never uses an sa of 1.
```

SELF DISSOLVING ARCHIVES - SDA
SDA is part of Parsec's file archive utility, and as such is shareware. If you find that you are creating Self-Dissolving-Archives, then obviously you are using ARC and find it useful and are obliged to send in a shareware contribution for the use of ARC.

If you are reading this text file, then you have probably already gone through the process of dissolving the archive that was used to distribute it. Pretty easy, huh? Well, that is what self-dissolving archives (SDarc's) and SFX files are all about. It takes the pain (or confusion) out of dissolving archives.

Self-dissolving-archives, on the other hand, necessary take on the same status as whatever is contained within them. This may mean that they are public domain, or that some restrictions apply. Read the documentation within each individual SDA file and abide by the copyright notices and restrictions that the author places on the file.

Not only are SDarc's easy to use, but since the archives created are self dissolving, it is no longer necessary for people receiving such files to have ARC installed and running before they can dissolve them. All you have to do is LOAD and RUN an SDarc. This is probably what most people do to downloaded programs anyway, so instructions are not even necessary. For those of you that have only a single 1541 or 1571, you will appreciate the convenience of just LOADing the SDarc into RAM, switching disks, and depositing the un-ARCed files on a fresh disk.

SDarc's are also very easy to create. All you have to do is attach the short 4 block SDA program to the beginning of the archive as outlined below.

But, convenience has a price. In order to keep the SDA program as short as possible, we have made a few simplifying assumptions about the archives they are to dissolve. After all, one of the nice things about ARC is that it saves you disk space and up/download time. If the SDA header were too large, we would lose that advantage. The limitations imposed by these assumptions are as follows:

1) The SDarc must be created using an ARC program numbered 2.xx That is, it must be a version 2 archive.

2) The SDarc must NOT contain any crunched files. It may contain any combination of stored, packed, squeezed, or squashed files. Crunched files require about 28K of RAM as workspace, and this would impose too much of a restriction on the maximum size of an SDarc. It would also double (or more) the size of the SDA header.

3) The SDarc must not contain any RELative files.

4) SDA is machine sensitive. The 64 version will not work in 128 mode on the 128, and the PET version only works on the PET and so on. Incidentally, the PET version is for BASIC 4.0 only.

5) You have to watch out how big the SDarc is. If you try to load a program bigger than about 200 blocks into the 64, you will wipe out the memory mapped I/O and crash the 64. SDarc's can be larger than 200 blocks on the C-64. The maximum size is about 243 disk blocks, ($0801-$F9FF) but you will have to write

your own boot program so that it gets loaded into RAM properly. See the notes below for size restrictions for the various versions of SDA.

CREATION OF SDAs

Creation of SDarc's is quite simple.
First, select the correct version of the SDA header program. If you are ARCing 64 files, use the 64 version; for PET files, use the PET version. You will need to make a copy of the SDA program, because it will be 'used up' by the SD-archive.

Next rename the copy of SDA to the name of the archive you want to create. The name you choose will have to end with the filename extension '.arc'.

Next use ARC 2.50 to append to the SDA program using ARC/AZ

Next, rename the SDA to whatever you feel like. It is not really an archive, so avoid the '.arc' filename extension. We suggest filename.SDA

Finally, LOAD and RUN it, and make sure it works before uploading it to a bunch of bulletin boards. Make sure you do not neglect this step! You never know what might go wrong.

A sample session, with ARC active, might go something like this. We will assume all the needed files are on drive a:

```
>c0:newfile.arc=0:sda230.64                 (make a copy)
arc/az newfile whatever you want in it      (make the archive)
ren newfile.arc newfile.sda                 (rename it)
dload"newfile.sda"                          (check it out)
run
```

An alternate method would be to first create the archive using ARC/CZ, and then use DOS to append the archive to the SDA program.

When you RUN an SDarc, it assumes that the archive follows immediately after the SDA program. Make sure that you avoid using XMODEM to transfer the SDA program(in un-ARCed form) because this will change its file size and it will think that the XMODEM padding is the first archive entry, and will abort since the header will no doubt be invalid.

The very first file in the SDarchive is not extracted to disk, but is typed to the screen instead. (I get a little nervous when a program starts right off writing to disk without any explanation.) This first file can, of course be squeezed or squashed, and should contain instructions for the user, copyright notices or whatever you think is appropriate.

After displaying this first file, SDA waits for the user to hit a key. If the key is the RUN/STOP key, then SDA just goes to the READY prompt and will not extract the SDarchive. If any other key is pressed, it goes ahead and extracts the remaining files to the disk in drive 0 of unit 8.

As the file is extracted, you will see the names of the files being created displayed one at a time. If, after extraction, the checksum is ok SDA prints "ok" after the filename. If there is a checksum error, SDA prints (?).

SIZE RESTRICTIONS:

```
$0200-$0230   is used by SDA. (All versions)
$0100-$0116   is used by SDA. (All versions)
$FA00-$FEFF   is used by SDA in the C=64 and C=128 versions.
$7B00-$7FFF   is used by SDA in the PET version.

$0401-$07F6   SDA header in the PET. Max size of an SDarchive = 119 blocks.
$0801-$0BF6   SDA header in the 64.    ''      ''      ''       201   ''
$1C01-$1FF6   SDA header in the 128.   ''      ''      ''       223   ''
```

This means that in the PET, your SDarchive can be as big as $7AFF-$0401, or a maximum of 119 blocks when stored on disk. In the 64, it can be $D000-$0801 or about 201 blocks on disk. If you want to construct your own BOOT program, then it could be as large as $FA00-$0801, or about 243 disk blocks. In the C-128 it can be $FA00-$1C00, or about 223 CBM disk blocks.

Note that we have padded the length of the SDA header to an exact multiple of 254 bytes. This way it will take up exactly four blocks on disk. If, for some reason you wish to convert it to a regular archive, then all you have to do is boot up your favorite disk doctor and change the (track,sector) pointer to the first block in the file so that it points to the fifth block in the file, and the job is done.

There are several versions of SDA

```
SDA230.pet is for the 4032/8032
SDA230.64  is for the adorable 64
SDA230.128 is for the C=128
```

SDA230 is the original version of SDA and does not support crunched files. You will have to use ARC/AZ to create these SDA's.

The maximum size of an SDA when stored on disk, for SDA230 is as follows:

```
PET version ... 119 blocks (29K)
 64 version ... 201 blocks (49K)
128 version ... 223 blocks (55K)
```

You must include an introductory title file when using SDA230 or SDA231.

The very first file in the SDA is not extracted to disk, instead it is displayed on the screen. This gives you a chance to describe what is in the SDA and tell the user to either hit RUN/STOP to abort, or any other key to start dissolving the SDA to drive zero of unit 8 (note that some of the C=128 versions below use the ESC key instead of RUN/STOP)

See the enclosed file "title.bas" You can use it to easily create colorful title screens. Simply edit the print statements until you like what you see and then change the OPEN statement to send the output to disk instead of to the screen. Or, if you are using ARC128, then:

```
RUN: >d:title
```

```
SDA231.64  is for the adorable 64
SDA231.128 is for the C=128
```
Sorry, PET users, this is where you get left behind.

SDA231 is essentially the same as SDA230, except that it can handle crunched files as well. Anything that ARC 2.xx can create (with the exception of relative files) SDA231 can un-ARC.

You can use ARC/A, ARC/A1, ARC/AZ or whatever you feel like using.

The maximum size of an SDA when stored on disk, for SDA231 is as follows:

```
  64 version ... 196 blocks (48K)
 128 version ... 223 blocks (55K)
```

These maximum sizes also apply to SDA232 and SDA233. You could, get a few more blocks out of the 64 version of SDA230 by making your own boot file to take care of loading on top of the Kernel and I/O. You can not do this with SDA 231 or higher.

SDA232 is for lazy people like me

It is the same as SDA231, except it will display a directory of the SDA (like

ARC/L) so you do not have to include a title screen. If you want to include a title screen, then use SDA231 instead.

Note that SDA232 and 233 do not work with files that have been archived with ARC/A1.

SDA233 is for people that do not trust other people.

It is the same as SDA232, except that it allows you to protect the contents of the SDA by encrypting it using a simple password.

To "password protect" an SDA, just create it using SDA233 just as you would any other SDA. Then LOAD it into memory and:

```
RUN:password
```

where "password" is the password that you will need to dissolve the SDA. The password can be as long as you like.

When SDA has finished encrypting the SDA, just hit RUN/STOP, or ESC to abort and SAVE the SDA back to disk.

The SDA is now garbled, and can only be un-garbled if the same password is used when dissolving it. If you use the wrong password or no password, the SDA will not dissolve properly, so do not forget your password!

Changing screen colors.
Some, not all, SDA's will set the 40 column screen's background and border colors to black. If you want, you can select your own colors with the following POKE's.

Just LOAD the appropriate SDA, do the POKE's and then SAVE it to disk again.

SDA231.128
```
poke 7199,border
poke 7199+5,background
```

SDA231.64, SDA232.64 and SDA233.64
```
poke 2064,border
poke 2064+5,background
```

SUGGESTED NAMING of ARCHIVES

```
                      For files just ARCed = filename.arc
           For ARC files with a SDA header = filename.sda
           For LZW files done on the C-128 = filename.lha
For self dissolving LZW files on the C-128 = filename.sfx
```

CSX - The MS-DOS archive dissolver for the Commodore 64.

(C) 1988 - Parsec, Inc. All commercial rights reserved.

CSX is part of Parsec's file archive utility, and as such is shareware. If you find that you are dissolving IBM type ARChives, obviously you are using CSX and find it useful and are obliged to send in a shareware contribution for the use of ARC. The same copyright restrictions that apply to ARC apply to CSX.

CSX is straight forward. Just RUN "csx.bas" and follow the prompts. The machine language is accurate to the best of our knowledge and should be able to handle any IBM, Amiga or Atari ST archive you might come across. Please let us know if this is not so.

CSX consists of two files:

```
csx.bas - The BASIC driver.  This will fetch the machine language module from
          the disk drive.
csx64.ml  The C-64 mode machine language module.
```

All About GEnie

GEnie, founded in 1985, is an online information service with 100s of RoundTables (RTs) available around the world. Each RoundTable is focused on a specific area of interest such as Commodore computers, IBM computers, cars, movies, family issues, medical, law, military and banking etc.

Parsec, Inc. manages The Commodore RoundTable.

Commodore users from all over the world participate in the many exciting features GEnie has to offer, including nightly Real Time Conferences (RTCs), monthly multi-player games, an active message area to post questions and get answers, exciting guests, prizes, and games such as Trivia, Hangman, Jumble, and Mastermind run on C-128s during our RTCs. Users can also gain access to over 14,000 files available in our libraries.

It is the OFFICIAL GEOS SUPPORT network.

The Commodore 64/128 RT is also the official support area for CMD, GEOS, COLOR64 BBS, Parsec, and others. The Commodore 64/128 RT is staffed by some of the best and brightest in the Commodore business.

The monthly subscription fee is $8.95 per month which includes up to four hours of standard connect time usage at 300-1200-2400 baud.

Each additional hour is only $3.00 per hour.

For more information concerning rates, local access numbers (including 1-800 numbers for those in rural areas), additional services and other charges, and to join GEnie, call with a modem:

```
1-800-638-8369  (8-N-1 300/1200/2400)
Upon connection type: HHH     <return>
At the U#: prompt type: xtx99018,commrt
```

Please have a major credit card or your check book handy.

Once you have connected and signed up, type the word "Commodore" to move to the Commodore RoundTable on GEnie, page 625. See you online!

ORDER SHEET CATALOG V3.03 - 940125

```
TC128/64 Magazine with Disk
  Mailed by either 1st class
  or 2nd class mail. (check one)
     USA -> 12 issues for             ($48.00)
  Canada -> 12 issues for             ($50.00)
Overseas -> 12 issues for             ($60.00)=$   .

     Please note that there are NO S&H
     charges if you are ordering just
     a magazine subscription.

ARC/ARC128/CS-DOS shareware disk    ($  3.50) =$    .
Register ARC/ARC128/CS-DOS          ($ 20.00) =$    .
Geos Font Guide                     ($  9.95) =$    .
Clip Art Guide                      ($  4.00) =$    .
TC-128's * Stuff#01 book            ($ 19.95) =$    .
TC-128's * Stuff#02 book            ($ 19.95) =$    .
TC-128's 500 Q & Answers            ($  8.95) =$    .
RamDos128 - Case Study              ($ 29.95) =$    .
Enhanced SIDPlayer Book             ($ 29.95) =$    .
TC-128/64 Back issues
  #34 with the disk                 ($  9.95) =$    .
TC-128/64 Companion Disk (disk only)
```

```
  #32 or #33                   (each $  9.95) =$    .
Catalog disks w/ PD disk lists      ($  3.00) =$    .

Number of P.D. disks ordered on the
other side of this order sheet X ($1.00 each) =$    .

                                              ---------
********************************   Sub Total    $
* Pick a Shipping & Handling   *      S&H    =  $    .
* charge                       * Sales tax   =  $    .
* USA     ->       $ 4.25      * (5%-MA only)   -------
* Canada  ->       $ 7.50      *       Total = $    .
* Overseas Surface $18 surface *
* Overseas Airmail $30+        *
********************************
```

LEGAL NOTICES

C-128, C-128D, C-64, CBM, C-1541, C-1571, C-1581, C-1764 and other names of Commodore equipment are trademarks of Commodore Business Machines. All other trademarks or servicemarks mentioned in this book belong to their respective owners and are mentioned for their benefit or for editorial purposes.

Liteweir, Lweir, RUR U2, Software Light Years Ahead of the Rest, Twin Cities 128/64, TC-128/64, ARC 64, ARC 128, SDA, and SFX are trademarks of Parsec,Inc.

FILES IN THIS SFX ARCHIVE

arc250.4	prg	75
arc250.4.doc	prg	1
----------------	prg	1
sda230.64	prg	4
sda231.64	prg	7
sda232.64	prg	9
sda233.64	prg	10
title.bas	prg	6
-- for ibm arc -	prg	1
csx.bas	prg	21
csx64.ml	prg	13

```
-- misc files --  prg    1
81bug.bas         prg    5
```