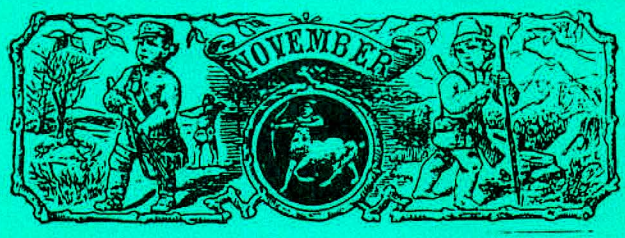
THE lone Newsletter

DEDICATED TO FRUSTRATED DISK USERS



****
***** MUSIC FROM MICRO-W ******
****

Over the past year or so, the music industry has begun to see and feel the presence of a strange, new entity in its midsts. This alien intruder, has the potential to affect the industry in a dramatic, almost revolutionary way. The invader that I speak of is, of course, your friendly home computer. The marriage between music and micros, like so many other unconventional matings, was bound to produce its share of unwanted offspring. However, a number of visionary engineers and musicologists got together and created a standard that hopefully will keep the number of these orphans to a minimum. This standard is now known by the name MIDI, which is an acronym for Musical Instrument Digital Interface. Basically, the MIDI is a hardware device that allows your computer to 'talk' to any MIDI equipted instrument. In the same way that a number of 1541's can be 'daisy-chained' to a Commodore 64, multiple keyboards and synthesizers can be used simultaneously via the MIDI. This would allow a single musician to sound as though he was a full ensemble, or even an entire orchestra. With the appropriate software, the MIDI can turn your computer into a multi-track recording studio- a state-of-the-art digital one no less! The possibilities are almost endless: music transcriptions printed out while playing, composing and orchestration tools, educational software, etc. At Micro-W, our development engineers and programmers are already hard at work with a number of MIDI projects for both the Apple and Commodore computers. Those of you who already enjoy music, either by playing an instrument or just listening, can look forward to some very exciting new things!

****
*** NEW PRODUCTS: MISTER LISTER and THE FONT FACTORY ***
****

MISTER LISTER is a powerful new tool from Micro-W, that will allow you to organize the names and addresses of friends, acquaintances, club members, customers, etc. This program allows the storage of up to 1000 names, any one of which can be recalled within five seconds by a keyed search. Each entry (called a record) contains a space for name, full address, and phone number of the person being listed, as well as a general comments section. In addition, there is a space (field) for a second name or title associated with this person, such as the spouse's name, company name, title, etc. Both the name and this extra field can be used as keys when asking MISTER LISTER to locate an individual record. Further, there are four other fields which can be used to classify an individual, and all of this combines to produce an extremely powerful and flexible system that will give you surprising capabilities. Other special features include a library of 'HELP' screens that may be accessed at any time by pressing a key, and a powerful error module to detect and clear up any errors encountered during processing.

MISTER LISTER will be available soon from Micro-W for the low price of only $29.95.

Are you tired of looking at that same old, dull, boring, unattractive, low quality font that comes out of your Commodore VIC-1525/MPS-801 or compatible graphics printer? Well tire no more! The FONT FACTORY is here to solve your printing problems!

The FONT FACTORY will read any standard Commodore 1541 ascii sequential file, automatically format, and print out the document in any font that is selected. With this ability, the FONT FACTORY will read text files produced by many of the popular Commodore 64 word processors, and produce a more presentable and interesting document. The user has full control over all page formatting, such as page length, line width, left margin, top margin, line spacing, headers, footers, page numbering, justification, etc.

The FONT FACTORY includes an easy-to-use Font Generator, to create and edit your own fonts. Fonts may be as large as 9x7 pixels, and may be printed in normal or double width formats.

The FONT FACTORY is user-friendly and entirely menu driven. Eight preformatted fonts are provided, including one with TRUE LOWER CASE DESCENDERS, when you purchase this software. Additional Font Disks may be purchased separately.

The FONT FACTORY will be available soon retailing at $29.95.

BUGS/FIXES

We have recently discovered a problem that affects a significant number SUPERCLONE diskettes, particularly its most recent incarnation--version 2.0 with the TOUGH-NUTS UTILITY. The problem is that when a track has error #21 on every sector, SUPERCLONE (or FASTCLONE) will not copy these errors onto the destination disk. Fortunately, there aren't many popular commercial programs around today that still use such a simple copy-protection approach. However, this new information now sheds light on why a number of the popular 'golden oldies' (FLIGHT SIMULATOR, COLOR 80, etc.) were confounding a lot of SUPERCLONE users. This situation would have been cleared up sooner, however the bug didn't appear on all versions of TCM and so was misleading.

The good news is that there are a number of simple ways to handle this problem. First of all, determine if your SUPERCLONE disk has this bug. If you have the most recent version including the TNU and FMB (Load the directory and see; there should also be a 'V2' as the disk ID) then it undoubtably has the problem. Those of you with earlier versions may not have the bug (however you should return them to us anyway to receive the latest version) and therefore, the following information does not apply to you.

There are three ways of handling this. First, is to realize what an error #21 means. It is the absence of any sync marks. This is exactly how a virgin disk (unformatted) right out of its box comes. Therefore, if you use a brand new disk, you can 'copy' an error #21 on a whole track by merely skipping that track and using SUPERCLONE to copy everything else. What if you don't have a new unformatted disk? Well, there's a way around that too. Use UNGUARD to create the error (use a timing constant=99). In order for UNGUARD to work, however, the track where the error is to go on the destination disk must be formatted and free from any errors. So, use UNGUARD to format the disk, then create the error #21, and finally go to SUPERCLONE and copy all the other tracks.

A second way to deal with the bug is to eliminate it altogether. This is not for the squeamish (they can use the third approach), however what needs to be done is relatively simple.

You're going to perform a little surgery on your SUPERCLONE disk. First, carefully remove the write-protect tape from the disk. Then load the CLONE MACHINE and select option #2 (the track/block editor) from the menu. Go to track 19, block 8 and locate byte number $14 (this is the 21st byte counting from the top left of the screen. This byte should read as an 'A3'. Change it to 'DD' and hit the return key. Next go to track 20, block 15. Again locate byte number $14. This time it should read as a '0C'. Change it to '72' and hit the return key. That's it, so put the write-protect tape back on and test it out - If all has gone well then you've just busted a bug!

As a third alternative, if you'd like, you may send your SUPERCLONE disk back to us for a replacement. Be sure to call first, to get a return authorization number. Then include that plus $3.50 for postage and handling costs in your package.

HOW TO DESIGN AN AUTO-START PROGRAM BOOT

Over the past few months, we have received a number of questions concerning how one may put copy-protection onto their own disks. In the first two issues of the CLONE NEWSLETTER, the technique for doing this was demonstrated both in BASIC and machine language. However, in a couple of letters a very good point was raised: If someone can list your program after it is loaded, then it may be quite easy to locate the code that checks for the error and then change it so as to eliminate the protection. For this reason alone, an auto-start boot is the perfect addition to any program that you may write. However, even if you're not looking for increased security, adding an auto-start to your software affords it an extra measure of convenience, not to mention an obvious air of professionalism.

In this article, we will explore in detail a number of ways to create an auto-start boot for your own programs. However, because of the very nature of this type of program, it can only be done in assembly code. Therefore, at the end of this article, I have included instructions for creating your own auto-start boot that requires no knowledge of machine code whatsoever.

To begin with, it might be helpful to those completely in the dark about this subject, as to what exactly is a 'boot' program. A boot or bootstrap program is one whose principle job is to start the execution of some main program. The term comes from the idea of 'pulling oneself up by their own bootstraps'. An auto-start boot on the C64 is always loaded the same way:

```
LOAD 'bootname',8,1
```

The '1' at the end of this statement causes the program to be loaded into the exact same memory location as it was originally saved from. In the case of a boot, this location is always somewhere in low memory beneath the normal BASIC program space(2048-). In order for an auto-start boot to work, the normal turn of events that follows a load must somehow be changed. But just what is the normal turn of events after a load? Well, a load initiated at the BASIC level (as illustrated above) causes the Kernal LOAD subroutine to be executed. After it is completed, the BASIC operating system then calls the Kernal CLALL subroutine to close all open files. Finally, a jump is made to the BASIC warm-start vector at $0302,$0303.

Knowing what actually happens after a load is completed now gives us the power to take fine control over our machine. There are three possible ways to do this. The first should be fairly obvious. Since after everthing is done the operating system blindly jumps to the address contained in the BASIC warm-start vector, the simplest approach would be to change these two bytes so that they point to the start of your boot. But how do you change these two bytes? You could do it by poking the correct values in before initiating the load. However, this would hardly be any better than having to type 'RUN' after a normal program is loaded, and certainly wouldn't be worth the effort at all. If you look at page 318 in the C64's Programmer's Reference Guide, you'll notice that the memory from $02A7 to $02FF is unused. Also notice that the BASIC warm-start vector is only two bytes away at $0302 and $0303. Why not create a program that loads into memory starting at $02A7 and ends at $0303. This way your program automatically changes the warm-start vector for you just by being loaded in. On the following page is an example illustrating exactly how this type of auto-start boot can be implemented.

```
$02A7   LDA #00
$02A9   JSR $FF90 ; no printing of loading message
$02AC   JSR $FFE7 ; always close all files before starting
$02AF   LDA #$02  ; logical file number
$02B1   TAY       ; secondary address
$02B2   LDX $BA   ; current device number
$02B4   JSR $FFBA ; Kernal SETLFS routine
$02B7   LDA #$04  ; number of characters in your program's name
$02B9   LDX #$D5  ; low-order address of name
$02BB   LDY #$02  ; high-order address of name
$02BD   JSR $FFBD ; Kernal SETNAM routine
$02C0   LDA #$00
$02C2   JSR $FFD5 ; load main program
```

At this point a number of decisions have to be made. We changed the BASIC warm-start vector to point to the start of this code. What happens if we don't change it back to its normal address? Well, whenever our main program gets interrupted(either by the RUN/STOP key or a normal END, STOP, or last line termination), the BASIC operating system will jump to the warm-start vector which still points to our code. Then the program will start loading in again. You probably don't want that to happen, and so we'll change the vector.

```
$02C5   JSR $FF8A ; restore default vectors
```

Now at this point, if your main program is also in assembly code, you should just add a jump instruction to its beginning. However, if your program is in BASIC, we'll need a few more lines of code.

```
$02C8   STX $2D
$02CA   STY $2E   ; set start of BASIC variables
$02CC   JSR $A68E ; set start of BASIC work space
$02CF   JSR $A660 ; initialize BASIC parameters
$02D2   JMP $A7AE ; go to BASIC interpreter loop
$02D5-  'NAME'    ; this is where your main program's name goes
```

Finally, don't forget to change the warm-start vector...

```
$0302   $A7          $0303   $02
```

Now save this program from $02A7 to $0304, and name it anything you'd like.

A second and less obvious way of making an auto-start boot is to change the vector at $032C and $032D so that it points to your program's starting address. This is the Kernal CLALL vector and it's called by the operating system to close all files after any type of activity that requires the opening of files. The loading of your auto-start boot is an activity that requires the opening of files, and so you can successfully divert the normal turn of events by using this approach. Here's how it's done:

First, look at pages 319 and 320 of the Programmer's Reference Guide. You'll notice that the CLALL vector is at $032C-$032D and that seven bytes ahead is an unused area of memory stretching from $0334 to $03FF (the tape I/O buffer is only used by the datacorder). This is where our auto-start boot will go. Start by stuffing the beginning address of the boot into the CLALL vector-

```
$032C  $34           $032D  $03
```

Now begin writing the boot at $0334:

```
$0334   JSR $FF8A ; restore default vectors
```

This is extremely important to do right away. If instead you began by first calling the Kernal CLALL routine, can you imagine what would happen?

```
$0337   JSR $FFE7 ; NOW close all files
$033A   LDA #$02  ; logical file number
$033C   LDX $BA   ; current device number
$033E   TAY       ; secondary address
$033F   JSR $FFBA ; Kernal SETLFS routine
$0342   LDA #$04  ; number of characters in your program's name
$0344   LDX #$5D  ; low-order address of name
$0346   LDY #$03  ; high-order address of name
$0348   JSR $FFBD ; Kernal SETNAM routine
$034B   LDA #$00
$034D   JSR $FFD5 ; Kernal LOAD routine
```

Once again, as in the previous example, we have two different routes to take depending on whether our main program is in assembler or BASIC. For assembly code, just jump to its starting address. For BASIC, continue on:

```
$0350   STX $2D
$0352   STY $2E   ; set start of BASIC variables
$0354   JSR $A68E ; set start of BASIC workspace
$0357   JSR $A660 ; initialize BASIC parameters
$035A   JMP $A7AE ; jump to BASIC interpreter loop
```

Now you must stuff the name of your main program into its correct address-

```
$035D-  'NAME'
```

Finally, save this program from $032C to $035D plus the length of your program name.

The final example to be given for writing an auto-start boot is not the most eloquent, but it is an interesting example of how you can 'trick' a computer into doing your bidding.

Remember that the Kernal LOAD routine that's used to load in your boot is actually a subroutine (it's executed with a JSR instruction). When a subroutine is finished with the 'RTS' instruction, the return address is popped off of the stack and put into the program address counter. If none of this makes any sense to you, don't worry. All you really need to understand is that you can alter the normal turn of events by putting the starting address of your boot into the stack. Now, the stack in the C64 resides from $0100 to $01FF, so just where should the address go? Well, we really can't be certain what value the stack pointer might have when the load of our boot is done. So, what we have to do is fill up the entire stack with our starting address. And since it's a two byte address, and we won't know which byte will be popped off first, both bytes had better be the same. This limits us in our choice of starting addresses for our boot. It will have to be like one of these: $0101, $0202, $0303, $0404, $0505, $0606, $0707. Because our boot program will have to start loading in at $0100 (in order to fill the stack), the most appropriate place for our starting address to be is at $0203, just after the stack. Notice that the actual starting address is one byte higher than what gets popped off the stack. This is because the program counter always gets incremented after being loaded.

Ok, so we've determined that we need to fill the stack with $02's and have our code start at $0203. However, unlike the two previous examples, we'll have to write the assembler code at some location other than where it will eventually load into. This is because the job of filling the stack with all $02's is going to be difficult, if not impossible with most machine language monitors. Thus, for convenience, I found it best to start by writing at $C100, an offset of exactly $C000 from where the code will actually reside. So, first fill locations $C100 to $C1FF with $02's. Then proceed as follows:

```
$C203   JSR $FFE7 ; close all files
$C206   LDA #$00
$C208   JSR $FF90 ; prevent printing of loading message
$C20B   LDA #$02  ; logical file number
$C20D   TAY       ; secondary address
$C20E   LDX $BA   ; current device number
$C210   JSR $FFBA ; Kernal SETLFS routine
$C213   LDA #$04  ; number of characters in program name
$C215   LDX #$2E  ; low-order address of name
$C217   LDY #$02  ; high-order address of name
$C219   JSR $FFBD ; Kernal SETNAM routine
$C21C   LDA #$00
$C21E   JSR $FFD5 ; Kernal LOAD routine
$C221   STX $2D
$C223   STY $2E   ; set beginning of BASIC variables
$C225   JSR $A68E ; set beginning of BASIC workspace
$C228   JSR $A660 ; initialize BASIC parameters
$C22B   JMP $A7AE ; go to interpreter loop
$C22E-  'NAME'    ; put the name of your main program here
```

Now save this code from $C100 to $C22E plus the length of your program's name. Load TCM's track/block editor and go to the directory at track 18, block 1. Locate this program (the one that you just saved), and note the starting track and block numbers for it (they are the two bytes that just precede the program name. Go to this track and block and note that the third and fourth bytes are $00 and $C1, respectively. This is the loading address of the program in the standard low-byte, high-byte format. Change the $C1 to $01 and then hit the return key. Your auto-start boot should now load into $0100 and stuff the stack with $02's, effectively creating the detour that was sought after.

Finally, as was promised earlier, here is a way to create an auto-start boot for those of you without any knowledge of assembly language. Use the CLONE MACHINE'S file copy utility to copy either the 'V2NOTES' or 'SUPERCLONE' file onto the disk that has the program you wish to auto-start. If you put it onto a newly formatted diskette, then your program can be loaded by simply typing: LOAD'*',8,1 . Otherwise, you will have to type whatever name that you specified for it when making the copy. Before you can use it though, you will have to modify it so that your main program will be loaded by it. Use TCM's track/block editor and examine track 18, block 1 of the disk that your program is on. This is the start of the directory of the disk. Find the name of the file that you just copied. If it's not on this page of the directory, then hit the 'f5' key until the program name does appear. Position the cursor on the second byte BEFORE the name and press the 'f5' key. This should bring you to the starting track and block of the file. Press the 'f7' key and you should see the original program name that was loaded by this boot (either 'CPYT1' or 'NOTES') at $54. Just type right over this, the name of your main program. If it's exactly five letters long then you're done, so hit the return key. Otherwise, you'll have to locate byte $29, and change it (it's a '$05') to reflect the number of characters that are in your main program's name. That's it!

THE DISK EXAMINER -- An Investigative Tool

As most of you now know, the TOUGH-NUTS UTILITY is a new program on the latest version of the SUPERCLONE disk. Through the proper use of this utility, it is possible to make backups of programs employing the very latest and most sophisticated protection schemes that exist today. However, as most of you also know, this program requires the user to input a unique set of parameters for a given piece of software. Without the correct parameters, the TNU cannot be used effectively.

In the last issue of the CLONE NEWSLETTER, we were able to publish the TNU parameters for a number of popular programs, thanks to the efforts of some early users. We certainly want to encourage these contributions, and we will continue to offer both cash and/or merchandise rewards for those submissions deemed helpful.

In this issue, we are going to print the listing of a software tool that has been of immeasurable help in determining these parameters. It's called the EXAMINER, and what it does is show you exactly what any track looks like in terms of it's sector headers. Every sector on a track has its own header which is composed of five bytes of information - a two byte ID, a track number, a sector number, and a checksum. This program will sequentially read every header on a track, regardless what is on it. For example, if you were to use the EXAMINER to study track 35 of any ELECTRONIC ARTS program, you would see that the track is normal except that each header has number 34 as the track number. The fact that it would read as an error 21 (no sync mark) when trying to access it through the normal DOS, reveals how misleading these errors sometimes are.
Probably the most useful function that the EXAMINER can be used for, is to locate tracks with extra sectors and determine the parameters needed for option #1 of the TNU. An extra sector can only occur between tracks 18 and 24 inclusive, and so that limits your search a bit. When it does occur, it will usually be very obvious by scanning the readout of the sector numbers. Normally the sector numbers will be in sequential order, starting randomly with any of the legitimate values. When an extra sector is present, you will usually see an out of place number. For example, check out track 24 of SENTINEL, if you have it. You will see the following order of sector numbers:

0,1,2,3,4,5,6,7,8,9,10,11,12,18,13,14,15,16,17,18

Notice that there are two sector 18's and 20 sectors in all instead of the normal 19. Note that the EXAMINER will only display the standard number of sectors on a track at any one time. Therefore, to have seen the ordering of all TWENTY sectors in this previous example would have required displaying track 24 at least twice (the odds are good that the starting sector will be different from one time to the next). From the information that this example reveals, all of the required parameters for option #1 of the TNU can be determined. Obviously, the track number is 24 and the range (number of sectors) is 20. The reference sector can be any one other than 18, since in this case it is not unique. Finally, the question of density bits. If the EXAMINER should print out any graphics characters or other garbage for all the headers on a track (or just plain lock up), then its likely that the density bits HAVE been altered. However, if the readout is clean up to some sector, and then becomes wierd, its more likely that there are some extra sync marks stuffed in. In this case, use the first sector where the garbage starts as the reference. Generally you should respond 'NO' to the density bit question unless the TNU continually locks up while trying to copy with the default value.

Another good example to study using the EXAMINER is the game WIZARD.

Check out either track 1, 2, or 3. The result should be interesting. For those of you without this disk, this is the way the sectors read:

0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,9

In this case, the TNU isn't needed to create an extra sector (you can't have an extra sector at these tracks) or change the density bits. However, what IS important here, is that the only unique sector is number 10. Thus we set the reference sector to 10, specify the normal range (for these tracks) of 21, answer 'NO' to the density bit question and we're in business. By the way, the checksum numbers on these tracks are incorrect and that is indicated on the screen by asterisks.

I know that some of this may be a little confusing to read, but I urge you to stay with it and try the EXAMINER out. I think that you will find it much easier to understand by actually using it than by reading about it.

The EXAMINER is almost one hundred lines of code, and much of it is critical, so be careful when typing it in. To make it easier to locate any errors in copying the large number of data statements, I included a check of these numbers as part of the program. Any mistake here will cause the program to abort with the offending line number displayed. To be on the safe side, don't run the program until you have first saved it to disk.

```
1000 REM ************************
1001 REM ***** THE EXAMINER *****
1002 REM **** COPYRIGHT 1984 ****
1004 REM ** MICRO-W DIST. INC. **
1006 REM ************************
1008 GOSUB1340
1010 T(0)=2:T(1)=3:T(2)=8:T(3)=15:T(4)=25
1020 XT=49408:FX=49471
1030 OPEN15,8,15,"I0":GOSUB1320
1040 PRINT"≡▫▫▫▫▫▫▫▫▫▫▫■■■■■■■■■■ENTER TRACK (1-36):";:GOSUB1240
1050 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(2)CHR$(TR)CHR$(0)
1060 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(176)
1070 PRINT#15,"M-R"CHR$(0)CHR$(0):GET#15,A$:IFASC(A$+CHR$(0))>127THEN1070
1080 SYSFX
1090 POKE251,0:POKE252,192:SYSXT:REM XMTPG
1100 PRINT#15,"U3":REM EXEC $0500
1110 PRINT#15,"M-R"CHR$(0)CHR$(3):GET#15,A$
1120 IFASC(A$+CHR$(0))<>0THEN1140
1130 PRINT"◙▫▫▫▫▫▫▫▫▫▫▫■■■■■■■■■■SYNC ERROR ON TRACK";TR:GOSUB1280:CLOSE15:GOTO103
0
1140 PRINT"◙ ID  TRACK  SECTOR  CHECKSUM (*=BAD)":PRINT
1150 PRINT"▫──────────────────────────────────"
1160 SH=21+(TR>17)*2+(TR>24)+(TR>30):FORC=0TOSH-1:CH=0
1170 PRINT#15,"M-E"CHR$(158)CHR$(5)
1180 PRINT#15,"M-R"CHR$(22)CHR$(0)CHR$(0):FORI=0TO4:GET#15,A$:GOSUB1310
1190 IFI<2THENPRINTTAB(T(I));A$;:NEXT
1200 PRINTTAB(T(I));ASC(A$+CHR$(0));:NEXT
1210 IFCH<>0THENPRINT"*";
1220 PRINT:NEXT:GOSUB1280
1230 CLOSE15:GOTO1030
1240 OPEN1,0:INPUT#1,TR$:CLOSE1:TR=VAL(TR$)
1250 IFTR<1ORTR>36THEN1240
1260 RETURN
1270 OPEN15,8,15:INPUT#15,A,B$:PRINTA,B$:CLOSE15
1280 PRINT"≡▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫■■■■■■■HIT ANY KEY TO CONTINUE";
1290 GETA$:IFA$=""THEN1290
```

```
1300 RETURN
1310 A=ASC(A$+CHR$(0)):CH=(NOT(CHANDA))AND(CHORA):RETURN
1320 PRINT"▒▒▒▒▒▒***▒▒SUPERCLONE DISK EXAMINER  ***"
1330 RETURN
1340 DIML(45):DIMCS(45):D=49152
1350 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒CHECKING DATA STATEMENTS - PLEASE WAIT."
1360 FORI=1TO45:READA:CS(I)=A:NEXT
1370 FORN=1TO45:X=0:FORJ=1TO8:READA:POKED,A:D=D+1:X=X+A:NEXT:L(N)=XAND255:NEXT
1380 FORI=1TO45:IFCS(I)<>L(I)THENGOSUB1410
1390 NEXT:IFE=1THENEND
1400 RETURN
1410 PRINT"ERROR IN LINE NUMBER ";1470+I*10:E=1:RETURN
1420 DATA239, 76,137, 91,223, 95,108,207
1430 DATA 50,239,192, 22,249,169,157,213
1440 DATA 45,210,109,187, 6, 4, 103, 245
1450 DATA175, 27,132,105,235,188, 92,108
1460 DATA193, 49,108, 93, 11, 88, 76, 47
1470 DATA179, 33,213, 91, 98
1480 DATA120, 32,  0,254, 32,148,  5,160
1490 DATA 72,162,  0, 44,  0, 28, 16, 10
1500 DATA232,208,248,136,208,245,140,  0
1510 DATA  3, 96,160,  0, 44,  0, 28, 16
1520 DATA251, 44,  0, 28, 16,244,200,208
1530 DATA248, 32, 47,  5, 76, 74,  5,120
1540 DATA169,208,141,  5, 24, 44,  5, 24
1550 DATA 48,  6,169,  0,141,  0,  3, 96
1560 DATA 44,  0, 28, 48,240,173,  1, 28
1570 DATA184, 96, 32,  0,254, 32,148,  5
1580 DATA120,160,  0,162,  0, 80,254,184
1590 DATA173,  1, 28,153,  0,  3,200,232
1600 DATA224, 10,208,241,192,210,176, 12
1610 DATA 32, 47,  5, 80,254,184, 32, 47
1620 DATA  5, 76, 83,  5, 32, 47,  5,160
1630 DATA  0,185,  0,  3,201, 82,240, 14
1640 DATA152, 24,105, 10,168,192,210,208
1650 DATA240,169,  0,141,  0,  3,169,  0
1660 DATA141,157,  5, 96,173, 12, 28,  9
1670 DATA 14,141, 12, 28, 96,  0,120, 32
1680 DATA  0,254, 32,148,  5,173,157,  5
1690 DATA170, 24,105, 10,141,157,  5,160
1700 DATA  0,234,189,  0,  3,153, 36,  0
1710 DATA200,232,192, 10,208,244, 32,151
1720 DATA244,120, 96,120, 32, 54,245, 32
1730 DATA 86,245, 80,254,184,173,  1, 28
1740 DATA153,  0,  5,200,208,244,160,186
1750 DATA 80,254,184,173,  1, 28,153,  0
1760 DATA  1,200,208,244,169,  0,133, 48
1770 DATA169,  5,133, 49, 32,224,248, 96
1780 DATA247,247,128,128,247,247,  0,128
1790 DATA247,247,136,128,247,247,  0,136
1800 DATA169,  2,168,162,  8,234, 32,186
1810 DATA255,169,  2,162,100,160,193, 32
1820 DATA189,255, 32,192,255,162, 15, 32
1830 DATA201,255,160,  8,185, 83,193, 32
1840 DATA210,255,136,208,247, 32,204,255
1850 DATA162,  2, 32,201,255,177,251, 32
1860 DATA210,255,200,208,248, 32,204,255
1870 DATA234,234,169,  2, 76,195,255,162
1880 DATA 15, 32,201,255,160,  7,185, 92
1890 DATA193, 32,210,255,136,208,247, 32
1900 DATA204,255, 96,  8, 32, 48, 32, 50
1910 DATA 58, 80, 45, 66, 96,  1,  1,  0
1920 DATA104, 87, 45, 77, 35, 50,234,234
```

SOLUTIONS

TRIVIA FEVER:

1) Copy with SUPERCLONE.
2) TNU - Option #3 - Reformat track 17.
3) TNU - Option #1 - track 17, range 21.
4) TNU - Option #1 - tracks 18-22, range 20.

Submitted individually by:
Randy Goldstein, and John Cincotta

BEACH-HEAD:

1) Copy with FMB.
2) Load 'BEACH-HEAD',8 from copy.
3) In immediate mode, POKE4847,169:POKE4848,245
   POKE4851,169:POKE4852,40
4) SAVE'@0:BEACH-HEAD',8

No more banging, and make all the backups you want.

Tom Poulin
Jacksonville, N.C.

GHOSTBUSTERS:

1) Copy with FMB.
2) Recopy track 18 with SUPERCLONE.
3) Put on a write-protect tape - very important!

COLOR 80:

1) Copy tracks 17, 18, and 25 to 27 using SUPERCLONE.
2) Use TNU to format tracks 2 and 34 if disk was unformatted
3) Use UNGUARD, error #21, TC=99, on tracks 2 and 34.

John Mallory
Ed. Note: After fixing the SUPERCLONE bug as described in this issue, you can copy tracks 2 and 34 with SUPERCLONE and skip steps 2 and 3.

NEUTRAL ZONE:

1) Copy with FMB.
2) TRACK/BLOCK EDITOR- Track 14, sector 19.
   Change byte $EC from $94 to $96.

Because of SUPERCLONE, a backup of NIGHT MISSION PINBALL from Sublogic will run. But, it will have the head chatter when looking for the error. The following will eliminate the head chatter from the backup:

NIGHT MISSION PINBALL:

1) Copy the diskette using SUPERCLONE or TCM.
2) Load TCM: Load'CLONE',8,1
3) Take menu option 2 (edit track/block)
4) Insert backup copy of NIGHT MISSION PINBALL.
5) Change byte $3A from $21 to $00 (Hex mode).

NIGHT MISSION PINBALL is like FLIGHT SIMULATOR II in that a look-up table is used by the program to look for errors. The above changes the look-up table.

Michael L. Brown

LETTERS TO THE EDITOR

Dear Editor,

I just received the latest issue of the CLONE NEWSLETTER. There is a comment regarding the dongle for M'FILE. I have found a much simpler way to defeat the required key. The results of my tests led me to try a variable resistor in the required port. So, I plugged in a pair of Commodore paddles. Sure enough, when both paddles are set about mid-range, the system operates. Try it; I have on several machines, with several sets of paddles and it worked in each case. There is no need for an IC etc., and most Commodore owners already have the paddles.

Try my method, and when I am shown correct, just send money. Thanks for a good product, and good service.

Michael Sussman
Upper Black Eddy, Penn.

Ed. Note: Sounds interesting Michael. Maybe someone out there who owns M'FILE can verify this for us.

Dear Editor,

Here are some cloning experiences I have had.

Commodore Business Accounting System Payroll Module:

This system consists of two double-sided disks. SUPERCLONE works just fine on side 1 of both disks but produces faulty copies of both side 2's. Strangely enough, a FMB copy of the side 2's will work.
To summarize:

To clone Payroll

1) Disk A side 1 - Use SUPERCLONE
2) Disk A side 2 - Use FMB
3) Disk B side 1 - Use SUPERCLONE
4) Disk B side 2 - Use FMB

MUSICALC-1

The troublesome track here is #30. There is a problem in using SUPERCLONE to copy this program which produces a discrepency between the original and copy on track 30, block 0 (error 20), and track 30, block 17 (error 23). I used TNU with the following parameters to produce a working copy:

Non-Standard Sectors
Track #30
Alter Density Bits - YES, to 2.
Reference Sector - 9
Range - 9

All other MUSICALC modules and templates can be SUPERCLONED normally.

If this information merits a reward, please send me some software, preferably FANTASTIC FILER PROFESSIONAL , in lieu of cash.

Nicholas J. Manzoli
Brookline, Mass.

Ed. Note: Great work, Nick. You've earned yourself both our respect, and a FANTASTIC FILER PROFESSIONAL.

FANTASTIC FILER™

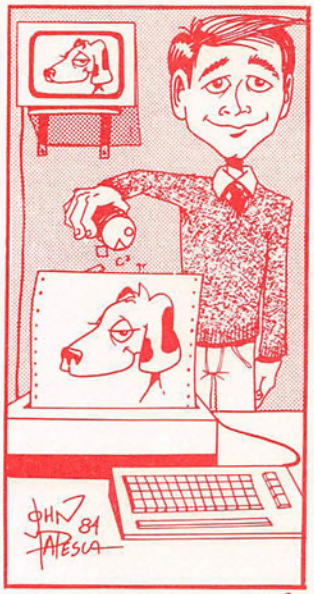
"A COMPLETE DATA BASE FOR THE COMMODORE 64™"

"SOLVES YOUR FILING PROBLEMS"

SO MUCH SOFTWARE FOR ONLY
$29.95

SCREEN DUMPER 64™

How would you like to have a copy of all of the text or graphics that appear on your monitor screen? Well SCREEN DUMPER 64™ may be what you are looking for. This program will transfer to your printer* a copy of what you see on your monitor screen including hi-resolution graphics, text, and multicolor sprite, etc. It even works with the KOALA PAD™. You can load this program into your computer in a hidden location so that it shouldn't interfere with your programs. This means that you can use your Commodore 64 normally and then call up this routine to dump what is on the screen. Colors are represented by 16 shades of gray for faithful reproductions.



ALL THIS FOR ONLY
$29.95

Call: (201) 838-9027 To Order

MasterCard VISA

Micro-W. DISTRIBUTING, INC.
1342 B Route 23
Butler, N.J. 07405

Micro-W. DISTRIBUTING, INC.
P.O. Box 198
Butler, NJ 07405

Bulk Rate
U.S. Postage
PAID
Permit No.
66
Butler, NJ
07405