SNAPSHOT 64

SNAPSHOT 64 ENHANCEMENT DISK

The perfect companion for your Snapshot 64

CSM SOFTWARE, INC.

SNAPSHOT 64
ENHANCEMENT DISK
MANUAL

Programming and manual by **BILL MELLON.**

INTRODUCTION

Thank you for purchasing the **SNAPSHOT 64 ENHANCEMENT DISK!** This disk includes several utilities for use with programs archived by the **SNAPSHOT 64** cartridge. These utilities are NOT required for the normal use of **SNAPSHOT 64** - they are simply the aids most frequently requested by **SNAPSHOT 64** owners. The utilities on the ENHANCEMENT DISK include:

NEW BOOT PROCESS - The boot process has been updated, allowing **SNAPSHOT 64** to successfully archive more programs than ever! The new boot can be added to previously **SNAPSHOT**'ed programs separately or in combination with the other options below.

FAST BOOT MAKER - Adds a fast loader to programs already archived by **SNAPSHOT 64**. Can also be used to restore the original "slow" boot to the program if desired.

CARTRIDGE MAKER - Creates a cartridge version of virtually any **SNAPSHOT**'ed program! The program will download and run from the cartridge in just a few seconds!

COMPRESSION UTILITIES - For examining the compressed files in a **SNAPSHOT**'ed program. Decompress a file to memory, examine or modify it, then recompress it back to disk.

MEMORY USE DISPLAY - Displays the memory areas used by a **SNAPSHOT**'ed program.

All of these utilities have been carefully designed for convenience and reliability. Even a novice can create a cartridge or install the fast boot! Of course, each utility is thoroughly explained in this manual, just in case any questions arise.

Besides covering the utilities above, this manual will also provide detailed information about the files created by **SNAPSHOT 64**. You can use this information to investigate **SNAPSHOT**'ed programs.

UPDATED BOOT PROCESS

The main boot process has been updated to enhance its abilities. If you have an original program that **SNAPSHOT 64** does not seem to archive correctly, this new boot process may solve the problem. Try it! The new process is available as an option with the fast load or cartridge making utilities, or it can be used separately (see below). The new boot is designed to be 100% compatible with the original boot, so you should use it whenever possible. If you ever experience trouble with the new boot, you can always go back and replace it with the original boot. However, we haven't run across a case yet where this was necessary!

FAST BOOT MAKER

The **FAST BOOT MAKER** utility (FBM) is primarily used to add CSM's own fast loader to previously **SNAPSHOT**'ed programs. It can also be used to restore the "slow" boot if desired. The new UPDATED BOOT PROCESS mentioned above can be included with either fast or slow option, or you may choose the original boot process instead.

In order to familiarize yourself with the **FAST BOOT MAKER** utility, please pick out a previously **SNAPSHOT**'ed program to use as an example in the following discussion. Start by inserting the **SNAPSHOT 64 ENHANCEMENT DISK** in your disk drive. Now type:

```
    LOAD "FAST*",8
    RUN
```

The first question you'll be asked is whether you want the FAST or SLOW load option. Enter either **F** or **S** and hit RETURN (the default is **F** if you just hit RETURN). Next, you'll be asked if you want the NEW (updated) or OLD (original) boot process. Enter either **N** or **O** (the default is **N**). Finally, you'll be asked to enter two characters to identify the program you want to modify (in case your disk contains more than one **SNAPSHOT**'ed program). Enter the same two characters that you supplied when the program was originally **SNAPSHOT**'ed. You'll be instructed to insert the disk containing your archived program. Hit RETURN to continue.

The FBM utility will check the boot program that is currently installed on the disk. For instance, if the two characters you entered were "XX", it will check for the file "XXF". If the file is not found, you'll be informed in case you inserted the wrong disk. Then you'll be asked if you want to continue the process or not. Enter a "Y" and press RETURN to continue. Any other entry will abort the process. If the file was found but is not a **SNAPSHOT 64** boot file, you'll be informed and given the choice of continuing. WARNING - if you continue, the file will be scratched! Finally, the FBM utility will check the free space on the disk (if you switch from the slow boot to the fast boot, you must have at least 5 blocks free on the disk). If everything checks out, FBM will install the boot you've chosen. That's all there is to it!

CARTRIDGE MAKER

The **CARTRIDGE MAKER** utility (CM) can be used to convert virtually any **SNAPSHOT**'ed program into a cartridge. The CM utility will create one to three files on disk containing a special version of your program. These files can be directly "burned" onto EPROMs using an EPROM programmer such as the PROMENADE. When the EPROMs are mounted on a standard 4-slot bank-switch cartridge board and plugged into the computer, the program will download itself, switch off the cartridge and execute automatically. NO special knowledge is required to create a cartridge except for burning the EPROMs - and we give you the correct commands for that (PROMENADE only). The PROMENADE, cartridge boards, EPROMs and other EPROM supplies are all available separately from CSM.

Please select a program to use as an example in the following discussion. Insert your **SNAPSHOT 64 ENHANCEMENT DISK** in the drive. Type:

```
LOAD"CART*",8
RUN
```

First you'll be asked for two characters to identify the program you wish to convert to cartridge. Next you'll be asked whether you want the NEW or OLD boot process. Enter **N** or **O** and hit RETURN. The default is the new boot, which should be chosen unless you know it causes a problem with the particular program you're converting (very unlikely). Then you'll be asked to enter a title. The title will be displayed on the screen while the program is downloaded from cartridge (which won't be long!). The title is limited to a maximum of 25 characters. It will be automatically centered on the screen when displayed.

Next you'll be prompted to insert the disk containing your **SNAPSHOT**'ed program. Hit RETURN when ready. The CM utility will check the disk to make sure enough space is left to hold the cartridge version's files. The size of the cartridge files will vary depending on the length of the "D" and "E" files of your program (i.e. the "XXD" and "XXE" files if the two characters you entered were "XX"). These files contain most of the archived program, in compressed format. If there is not enough space left for the cartridge files, CM will tell you and then terminate. No files will be created. This can only happen if you have other programs on the same disk. In this case, you should copy your **SNAPSHOT**'ed program files to a blank disk and re-run the CM utility.

When everything is ready, CM will begin the conversion process. Since the process is rather lengthy, you'll be told approximately how long it will take (based on a 1541 drive). The program will stop and inform you if any disk errors are encountered. When the process is completed, the cartridge version will be stored in 1 - 3 files on the disk. Each file requires a separate EPROM. CM will display each file's name, the type of EPROM required and the correct command to be used when burning the EPROM with the PROMENADE. You can use a different EPROM programmer but you'll have to figure out the proper commands yourself. The names of the files created will be based on the two letters you supplied, e.g. XXCART0, XXCART1, XXCART2. Only two types of EPROMs will ever be required: 27256 (32K) and 2764 (8K). CM will always use 32K chips unless the last file is small enough to fit on an 8K chip (33 disk blocks or less). This insures that the program will fit on a 4-slot board and is also the least expensive configuration (at mid-1986 prices).

Before burning the EPROMs specified by CM, you must determine the correct programming voltage to use. Some chips require 12.5 volts and others require 21 volts. Be sure you know the correct programming voltage for your chips. It's usually written on the chip. If it isn't, it **usually** means 21 volts is required, but to be on the safe side try it at 12.5 volts first. If 12.5 volts is too low, all that should happen is the chip won't program correctly. The yellow PROMENADE light will flash and you'll have to erase the chip before you try it at 21 volts. On the other hand, if you use 21 volts when only 12.5 is required, you'll **permanently damage** the EPROM! That's why you should always start at 12.5 volts when in doubt. With 2764 chips, you can tell the programming voltage even if it isn't written on the chip. If the chip is labeled 2764 **A**, it almost certainly requires only **12.5 volts**. To our knowledge, any other letter after the 2764, or no letter at all, means **21 volts** is required. When in doubt, always check the manufacturers specifications.

The programming voltage will determine the proper **CONTROL WORD (CW)** to use in the PROMOS command. That's why we just use "CW" rather than a specific number in the commands we give and why the message appears about checking your manual (see below). The following table gives the correct CW for each type of chip. Also given is the recommended **PROGRAM METHOD WORD (PMW)** for each chip. The PMW is not as critical as the CW; a "wrong" PMW shouldn't damage your chip. However, the PMW affects how long the PROMENADE will take to program the chip, versus how reliable the process will be. The PMW's we've given are conservative in that they are very reliable but take a little longer. You may experiment with other PMW's if you wish; the worst that should happen is that you'll have to erase the chip and try another PMW if the chip doesn't program properly. See your PROMOS manual for alternative PMW's. Even better, the **EPROM PROGRAMMERS HANDBOOK** from CSM discusses CW's and PMW's in detail. It also covers many other topics related to the PROMENADE, EPROMs and cartridges, and comes with a disk containing many useful routines.

PROMOS COMMANDS FOR DIFFERENT CHIPS

VOLTS	2764 (8K)	27256 (32K)
12.5	π4096,12287,0,6,6	π4096,36863,0,230,6
21	π4096,12287,0,5,7	π4096,36863,0,229,6

Once you've selected the right PROMOS commands from the table above for the EPROMs specified by CM, you're ready to begin the programming process. Turn off the computer, plug in the PROMENADE and turn the computer back on. Now you must load the PROMOS (PROMENADE) software from disk. (Note - due to their size, if you are programming 27256 chips you CANNOT use the special HESMON/PROMOS/WEDGE cartridge. Also, you MUST use PROMOS version **1.1** or higher for 2756 chips since version 1.0 has a bug in the 27256 algorithm. If you don't have version 1.1, contact CSM for an update.) Insert your PROMOS 1.1 disk and type **LOAD":*",8**. When PROMOS is loaded in, type **RUN**. You should see the PROMOS startup message on the screen (including the version number) and the lights on the PROMENADE should go out.

To demonstrate how to burn cartridge files onto EPROM using the PROMENADE, suppose CM displays the following information at the end of the conversion process:

FILE NAME	EPROM	PROMOS COMMAND
XXCARTO	27256	π4096,36383,0,CW,PMW
XXCART1	2764	π4096,12287,0,CW,PMW

CHECK YOUR SNAPSHOT 64 ENHANCEMENT DISK MANUAL FOR THE PROPER CONTROL WORD (CW) AND PROGRAM METHOD WORD (PMW) FOR YOUR PARTICULAR CHIPS.

In this example, you would find two files on the disk called XXCARTO and XXCART1. All cartridge files start at location 4096 ($1000) when loaded into memory. Since the "0" file requires a 32K (27256) chip, the file will be up to 130 disk blocks long (up to 32768 or $8000 bytes) and end at or before 36863 ($8FFF) in memory. Regardless of where the file actually ends in memory, we'll burn an entire 32K of memory onto the 27256 EPROM. This way the PROMOS command for all 27256 files can follow the same pattern. Similarly, since the "1" file requires an 8K (2764) chip, the file will be up to 33 blocks long (up to 8192 or $2000 bytes) and will end at or before 12287 ($2FFF) in memory.

Insert the disk containing your cartridge files into the drive. Load the first file into memory using ",8,1". In our example, you would use **LOAD"XXCARTO",8,1**. It's a good idea to zero the PROMENADE socket now - type Z and hit RETURN. Pick up your first EPROM, lift the handle on the PROMENADE socket and set the chip in the socket. The notch by pin 1 of the chip MUST be to the left, matching the diagram on the PROMENADE. Flip the handle down to lock the chip in place. Now type in the proper PROMOS command from above and hit RETURN. Note that the first character in the command is the PI character (shift up-arrow, next to RESTORE). The red and yellow lights on the PROMENADE should come on. Once they go out, the chip is programmed. Remove it from the PROMENADE. It's a good idea to label the chip so you won't get it mixed up with any others. If the yellow light is flashing, the chip did not get programmed properly. Erase the chip and re-check everything before trying again. Any other chips required are programmed the same way: Load the file with ",8,1". Zero the PROMENADE with a "Z" command, then insert the chip with the notch to the left. Type in the proper PROMOS command and hit RETURN. Remove the chip when it's finished and label it.

To complete the cartridge, you must mount the chips on a **PC4 bank-switch cartridge board** (available from CSM). This board has sockets for 4 EPROMS on it, numbered from 0 to 3. Each EPROM must go in a particular socket, based on the name of the file burned on the EPROM. For example, a chip with a file named "XXCARTO" burned on it would go in socket "0", and so on. Socket "0" is the one closest to the board's edge connector (the part that plugs into the computer). Before you mount the chips on the board, however, you'll have to make a slight change at **each** socket that will receive a 27256 chip. This procedure is covered in the PC4 board's instructions, but we wanted to make sure you have the information handy in this manual.

Turn the board over to the back side (the side without the sockets on it) with the edge connector towards you. Locate **pin 27** of the socket. In our example, it would be socket 0 - the socket closest to you. Pin 27 wouiuld be the second-to-last from the **right** end of the **second** row of socket pins (remember, the socket is upside-down). Near pin 27 is a small solder "dot" with a short, thick solder line leading to the pin. The solder line will usually have a clean break through it, disconnecting the dot from pin 27.

To use a **27256** chip in the socket, you must place a "blob" of solder across the break so the dot is **CONNECTED** to pin 27. Next, there is a small solder line connecting pin 27 with pin 28 (the next pin to the right). You must cut through this line with a sharp knife so pin 27 is **DISCONNECTED** from pin 28. Be careful not to cut any other lines (or yourself)! Once again, on each socket where a 27256 EPROM is to be used, pin 27 must be CONNECTED to the nearby solder dot but DISCONNECTED from pin 28. The exact opposite setup is required to prepare a socket for a 2764 EPROM (which is usually how the board is supplied). For a **2764**, pin 27 must be DISCONNECTED from the solder dot and CONNECTED to pin 28.

For our cartridge example, socket 0 would have to be set up for a 27256 chip, since the file "XXCARTO" is burned on the 27256 chip. Likewise, socket 1 would have to be set up for a 2764. Once the correct changes are made, all that's left is to insert the chips in the sockets. Turn the board right side up with the edge connector towards you. Place the first EPROM over socket 0 (closest to you) with the notch on the chip **to the left**. Make sure all the EPROM's pins are started in their proper holes in the socket. Press the chip straight down into the socket as far as it will go. Double-check the EPROM's pins again to make sure they're in correctly. Now repeat the procedure to install the other EPROM(s).

That's it - your cartridge is ready to use! Turn the computer off and insert the cartridge with the EPROMs face up. Turn the computer back on. If you have a CSM Single Slot or CARDCO 5-Slot expansion board, you don't have to turn off the computer. Just switch off the desired cartridge slot and insert the cartridge with the EPROMs facing you. Switch the cartridge slot back on and press the RESET button. When the computer is powered up or RESET, you'll see your cartridge title appear above the message "ARCHIVED BY SNAPSHOT 64". A couple seconds later your program will be up and running at the exact spot it was **SNAPSHOT**'ed!

DE/COMPRESS UTILITY

The DE/COMPRESS utility (D/C) is designed as an aid to investigating **SNAPSHOT**'ed programs. You may want to modify a program's operation, alter its protection or simply learn more about how it operates. When a program is archived with **SNAPSHOT**, most of the computer's RAM memory is stored in compressed format to save space and load time. The RAM memory from $1000 to $87FF (LOW memory) is compressed and stored in the "D" file (e.g. "XXD"). The RAM memory from $8800 to $FFFF (HIGH memory) is stored in the "E" file. Before you can examine a file, it must "decompressed" into memory. D/C lets you decompress and re-compress these files easily.

The normal decompression process in **SNAPSHOT**'s main boot starts by loading the compressed file into memory. The compressed memory itself starts at $1000, but it's preceded by COMPRESSION TABLES at $0F10-0FFF. This applies to both "D" and "E" files. Next, the file is decompressed into the full $1000-87FF area. When the "D" file is decompressed, it is left in this area. The "E" file, however, is transferred up to $8800-FFFF after decompression. This means that the "E" file must be decompressed FIRST; otherwise, it would wipe out the "D" file when it was loaded in. Compression is just the reverse; first the $1000-87FF area is compressed and stored in the "D" file, then the $8800-FFFF area is moved down to $1000, compressed and saved in the "E" file.

The D/C utility can be executed from BASIC or from a machine language monitor. The monitor commands given below are compatible with HIMON (from the disks supplied with CSM PROGRAM PROTECTION MANUALS VOL. I & II). Other monitors will use the same or very similar commands. Note - the start address for executing D/C from a monitor is different than the SYS address used when starting D/C from BASIC. This is because the routine uses an RTS instruction to return to BASIC from a SYS command, while it uses a BRK command to return to the monitor. To execute the D/C utility from BASIC, type:

```
    LOAD"DE*",8
    RUN
```

To execute D/C from a monitor:

```
    L"DE*",08
    G 0814
```

D/C will ask whether you want to COMPRESS or DECOMPRESS a file. Enter **C** or **D** as desired. Next, you'll be asked the question "LOW or HIGH memory ?". When decompressing, your answer indicates whether you want the memory left in the LOW area ($1000-87FF) after decompression or transferred to the HIGH area ($8800-FFFF). Likewise, when compressing, your answer indicates whether the HIGH memory should be transferred down to the LOW area first. Enter L or H to indicate your choice. Since the I/O devices and the BASIC and KERNAL ROMs lie "on top" of the HIGH area, it is usually more convenient to leave files in the LOW area after decompression, where they're more accessible to a monitor. You would also choose L when you want to compress a file you've been working on in the LOW area. **Remember, the LOW/HIGH question really indicates where you want to WORK with the memory rather than where it's normally located.** Its normal location is indicated by the original file name (see below).

The final question D/C asks is for a FILENAME. When decompressing, this file will be loaded at $0F10 first, as explained above. Likewise, after compressing, the compressed memory will be stored on disk under the name you give. You may use the "@0:" prefix in the name to replace an existing file (if you're daring). The original compressed files follow the convention of two characters plus "D" or "E". Your file names **DO NOT** have to follow this convention. This gives you more flexibility if you want to experiment with several versions of the same original file. However, we suggest that you retain the "D" or "E" in the name somewhere to remind you where the file resides in memory. If you want **SNAPSHOT**'s main boot to load your modified files, you'll have to rename them to follow the normal convention. Be sure to save a copy of the original files under other names, e.g. "ORIGINAL XXD".

Unlike the previous questions, answering the filename question is optional. If you just hit RETURN, no disk access will be done. This is more useful than it sounds. For instance, suppose you've been working on a file in LOW memory and you want to save a backup copy every once in a while as you go. When you save a copy, the memory will be left in the LOW area in COMPRESSED form. Before you can go back to working on it, it must be decompressed. Of course, you could re-load it from disk and decompress, but with large files this takes a couple minutes. Instead, just choose the decompress option WITHOUT giving a filename, and you'll be back in business in a second or two.

MEMORY USE DISPLAY

The MEMORY USE (MU) utility displays the main areas of memory that are used by a **SNAPSHOT**'ed program. The utility can be started from BASIC or a machine language monitor.

To start MU from BASIC, use:

```
    LOAD"MEM*",8
    RUN
```

To use MU from most monitors, use:

```
    L"MEM*",08
    G 0814
```

The only information MU needs is the two characters that identify the program you want to display. MU will display a grid to summarize memory usage in the $1000-FFFF area (the same area covered by the compressed "D" and "E" files). Each square on the grid represents one "page" of memory, 256 bytes. Each horizontal line of squares displays one "block" of memory, 4096 bytes. The numbers along the sides and top identify the address of the pages displayed. For instance, the first line displays each page of memory from $1000 to $1F00. The contents of each square on the grid tell you how the corresponding page of memory is affected by the **SNAPSHOT**'ed program. For an accurate picture of memory usage, you MUST use the CLEAR MEMORY function (F3 key) before **SNAPSHOT**'ing the program (you should be doing this anyway). The meaning of the symbols used in the display are:

" " (space) - The page is not used by the program; it still contains **SNAPSHOT**'s fill character ($BB).

"*" (asterisk) - The contents of the page vary; the page is not all filled with the same value.

Any other character - The page is completely filled with the ASCII value of that character. Characters whose ASCII value is not normally displayable, e.g. screen color codes, are displayed in reverse according to the following table:

True character	Displayed as reverse of
$00-1F (0-31)	$20-3F (32-63)
$80-9F (128-159)	$40-5F (64-95)
$C0-DF (192-223)	$60-7F (96-127)
$E0-FF (224-255)	$A0-BF (160-191)

In particular this means that a $00 byte is displayed as a solid reverse block, and an $FF as a reverse checkerboard character. These are the most common fill characters.

SNAPSHOT FILES

The following section covers the various files created when you **SNAPSHOT** a program. You DO NOT have to know any of this to use **SNAPSHOT.** This information is for the hacker types who want to experiment with their **SNAPSHOT**'ed programs. Although some of the following information is summarized in the normal **SNAPSHOT** manual, and the CODE INSPECTOR displays many of the key values, the detailed description here should open up many new possibilities for investigating **SNAPSHOT**'ed programs. We'll cover the "pre-boot" file first, and then the other files in alphabetical order. Note that this is NOT the order in which they are loaded into memory; the load order is:

PREBOOT, F, E, D, A, B, G, C

PRE-BOOT

The file that you load directly when you boot up a **SNAPSHOT**'ed program is called the "pre-boot" file. It's the first file listed and the only one whose name can be more than 3 letters long. All other files for this **SNAPSHOT**'ed program will have 3-letter file names. The first two letters will always be the same (for example, "XX") followed by a letter from A to G (e.g. XXA, XXB, etc.). The preboot file loads in at $02A7-0303. Like most autoboots, it uses the BASIC warm start vector at $0302 to begin execution at $02A7. The preboot sets up the screen colors and prints the file name that you used to load the preboot. Then it loads in the main boot file, which is file "F" (e.g. XXF). The "F" file's name is stored at $02FD-FF. The preboot file is identical in all **SNAPSHOT**'ed programs except for the two identifying characters at $02FD-FE. After the "F" file is loaded, these two characters are copied into it at $081E-1F. Then the "F" file is executed at $0820.

FILE A

The "A" file holds the memory from $0400 to $07FF, which is normally the screen memory. However, the last 24 bytes ($07E8-FF) are not part of the screen display and could contain important data. Also, some programs relocate screen memory to another area, so the $0400-07FF area may contain part of the program code or data.

FILE B

The "B" file contains the color RAM located at $D800-DBFF. Remember that only the least significant 4 bits (lower nybble) of each byte of color RAM is meaningful; the other 4 bits are totally random. Like screen memory, the last 24 bytes of color RAM ($DBE8-FF) are not used for display purposes and can hold useful values.

FILE C

The "C" file contains the program code from $0800-0FFF. It is the last file loaded in.

FILES D & E

The "D" and "E" files contain the program's RAM memory from the $1000-87FF and $8800-FFFF areas, respectively, in compressed format. **BOTH** files load in at $0F10 and follow the same format. The only difference is that after being decompressed in the $1000-8700 area, the "E" file is transferred up in memory to the $8800-FFFF area. Other than that, the following discussion applies equally to both files. The compressed memory is stored starting at $1000 and is preceded by two COMPRESSION TABLES. The two tables are located at $0F10-87 and $0F88-FF respectively. Each table contains one byte for every "page" of memory (256 bytes) in the $1000-$8700 area ($78 pages total). The memory below $1000 in the original program is not compressed; see the "A", "C" and "G" files.

The first compression table, at $0F10, indicates which pages have been compressed. The byte at $0F10 refers to the page at $1000, $0F11 refers to $1100, $0F12 refers to $1200, and so on up to $0F87, which refers to $8700. A page can only be compressed if it contains the same byte value throughout the page. This doesn't have to be **SNAPSHOT**'s fill character ($BB) from the CLEAR function (F3 key); it can be any value as long as the page is filled with it. When a page has been **compressed**, the corresponding byte in the first table will be a **$00**. Any **NON**-zero value will indicate that the page is **NOT** compressed. A non-zero byte actually represents the location within the page of the first byte that didn't match the previous byte(s). For instance, a $06 byte would indicate that the first 6 bytes ($00-05 within the page) were the same but the seventh byte was different. No compression would be done in this case.

The second compression table, at $0F88-FF, has a dual meaning depending on whether the page is compressed or not. If a page is compressed, the corresponding byte in the second table is the **FILL VALUE**, i.e. the value that the page was originally filled with. If the page at $1400, for example, was originally filled with the value $FF, then the byte at $0F14 in the first table would be a $00 (meaning compressed) and the byte at $0F8C in the second table would be the fill value $FF. No other bytes are saved to represent that page. On the other hand, if a page is NOT compressed, then the entire page will be stored in the area past $1000. However, the page will not usually be stored in its original location in this area. It will usually be stored lower down in memory than originally, because previous compressed pages have been omitted entirely. The page must be moved up in memory to reach its original location. The second compression table tells **where** in the $1000 area the page has been stored.

Let's look at an example to help clarify things. The decompression routine works backwards in memory (we'll explain why in a minute) so suppose the LAST couple bytes of the compression tables are:

```
FIRST TABLE   $0F10-0F87:  .... 03 00
SECOND TABLE  $0F88-0FFF:  .... 41 BB
```

The decompression routine starts at the end of the FIRST table, $0F87, representing the page at $8700. Since the first table contains a $00 byte, page $8700 has been compressed. Consulting the last byte of the SECOND table, $0FFF, tells the routine that the page was filled with the value $BB (**SNAPSHOT**'s default fill value). The routine will then fill page $8700 with $BB's to restore it.

Next, the routine moves back one byte in the tables. The byte at $0F86 in the first table represents the page at $8600. The fact that the byte is $03 in particular is not significant to the routine, just the fact that it's NON-zero. Since the byte is non-zero, page $8600 was NOT compressed. Therefore, the entire page will be stored somewhere past $1000. Where? Well, the corresponding byte in the second table at $0FFE is a $41, which means the page is stored at $4100 currently. The decompression routine will copy the page at $4100-41FF up to $8600-86FF. Since pages are copied UP in memory, this explains why the decompression routine works backwards from the end of memory.

The routine continues backwards through the table until it reaches a NON-compressed page which is already stored at its original location, without having to be moved. This indicates that none of the pages lower down were compressed either - if some had been compressed, this page would have been moved down in memory. since this page is in its original location, all the lower pages are too, and the decompression is complete. This point will always be reached, even if it's all the way back at $1000.

FILE F

File "F" is the main boot program, and it's loaded in by the preboot. "F" then loads the rest of the files. The "F" file is IDENTICAL in all **SNAPSHOT**'ed programs, at least until the **ENHANCEMENT DISK** was released. The fast loader, the cartridge routines and the updated boot all required modifications to the "F" file (and only the "F" file). On the disk, you'll find four versions of the "F" file. These files contain the same four options that are available with the FAST BOOT MAKER utility. In fact, you can install a new set of options by simply deleting the original "F" file, copying the desired file to the disk and renaming it. The names of the files indicate the options they provide:

FNF	-	Fast New (Updated) Boot
FOF	-	Fast Old Boot
SNF	-	Slow New (Updated) Boot
SOF	-	Slow Old Boot

FILE G

The "G" file contains four different memory areas. The file is initially loaded into memory at $0BB3-$0FFF. The four parts are moved to their original locations at different times in the boot process. The memory areas contained in this file are described below.

$0BB3-0BCF: This area holds the values for the SID (sound) chip at $D400-D41C. However, these values don't really indicate what the original contents of the SID chip were because the SID chip is mostly WRITE-ONLY - reading the chip does not return the values that were written there. This is why sound is a major limitation of ALL memory-dumping cartridges. Fortunately, almost all programs with sound will rewrite the SID values many times. Even if the initial screen does not have the correct sound, subsequent screens usually will. Also, most programs can be restarted once they're loaded in, which usually restores the sound. By altering the SID values in the "G" file yourself, you can sometimes correct the sound too.

$0BD0-0BFE: This area holds the values for the VIC (video) chip at $D000-D02E. Not all of these values were obtained by just reading the VIC chip, since some of its registers return a different value when read than what was written there. The most important examples of this are the raster register at $D012 (and bit 7 of $D011) and the interrupt mask register at $D01A. Special routines are used to determine these values indirectly.

$0BFF: This byte is the 6510 processor's STACK POINTER (SP). It is NOT the value that was in effect when the program was **SNAPSHOT**'ed, for two reasons. First, the NMI interrupt used to start the **SNAPSHOT** process (through the button on the cartridge) automatically pushes 3 bytes on the stack: the processor status byte (P) and the two-byte program counter (PC). Second, **SNAPSHOT** itself stores 26 bytes on the stack. This means that to find the true stack pointer value, you must **add $1D (29)** to the value in $0BFF. The PC value on the stack tells you where the program was executing when you interrupted it; the values past it on the stack often indicate subroutine calls that led the program to that point. The stack itself is at $0100-$01FF in memory (see below for its location in file G). The stack pointer SP is relative to $0100 - if SP is $23, the **next AVAILABLE byte** of the stack is at $0123. Since the stack grows DOWNWARD in memory (i.e. the stack pointer was DECREMENTED after the last byte was pushed on the stack), the **last USED byte** of the stack would be at $0124.

The bytes pushed onto the stack by **SNAPSHOT** and the NMI interrupt are listed below in REVERSE order. The contents of location $0094, for instance, is the LAST byte pushed on the stack. We've broken the values into four groups, with the start of each group indicated. The contents of location $0094 will be found at $0100 + SP + $01, for instance, and the PC HI byte will be found at $0100 + SP + $1D. Only the **low** byte of such a calculation is significant - the true high byte of the location on the stack is **ALWAYS $01** regardless of the calculated result. In other words, the low byte is always relative to $0100 and so can never indicate a location past $01FF. This is because the stack automatically "wraps-around" when either end is reached ($0100 or $01FF).

SP+$01:	$0094, 0095, 0096, 0097 (Key serial & tape values)
SP+$05:	$DD04, DC04, DD05, DC05, DD06, DC06, DD07, DC07 (CIA Timers)
SP+$0D:	$DD0D, DC0D, DD00, DC0E, DC0F, DD02, DD03, DD0E, DD0F, DC03, DC02 (Other CIA values)
SP+$18:	Y, X, A, P, PC lo, PC hi (6510 Registers)

$0C00-0FFF: This area of file G holds the original contents of locations $0000-03FF in memory. Obviously, many important values are stored here. The stack located at $0100-01FF in memory is stored at $0D00-0DFF in file G. Also, the location $0000 and $0001 values, which control the memory configuration, are stored at $0C00-0C01.

This concludes our look at **SNAPSHOT**'s inner workings. Happy hunting!