THE SNAPSHOT SHELL

General-Purpose Interrupt-and-Resume Software by Robert Sather

Apple Computer, Inc. makes no warranties, either expressed or implied, regarding the enclosed computer hardware and software package, its merchantability, or its fitness for any particular purpose.

Disclaimer - Dark Star Systems Ltd. makes no representation or warranty with respect to the contents herein contained, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose, and expressly disclaims any and all liability for any special, incidental, or consequential damages resulting from any use of the products contained herein. Furthermore, Dark Star Systems Ltd. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation to notify any person of such changes.

Limited Warranty - To the original purchaser only, Dark Star Systems Ltd. warrants both the magnetic disk on which its software is recorded and the Printed Circuit Board which, together, constitute the system to be free of defects in materials and faulty workmanship under normal service and use for a period of 12 months from the date the program is delivered. If, during this 12 month period, a defect in either constituent of the product should occur, you should contact the company from whom you purchased it in order to obtain a Returned Merchandise Authorization from Dark Star Systems Ltd. Provided that you have returned the enclosed Warranty Registration Card, Dark Star Systems Ltd. will repair or replace products which are covered by the terms of its Warranty free of charge.

Snapshot and Snapshot Shell are Trademarks of Dark Star Systems Ltd.

Apple II, II+ and //e are Trademarks of Apple Computer, Inc.

Contents

1. Overview

Snapshot is the most useful card, after the disk controller, that you will ever have in your computer. When a Snapshot software package has been loaded into the card's 8K of on-board RAM, it is possible to suspend a running program, examine and manipulate it in some way, and resume running it from the point of interruption.

Snapshot software packages are available for various purposes including multi-tasking (the Shuttle), protected program backup (the Copykit), and screen-editing and dumping (the Printerrupt). All these packages have a common element which we call the "Shell". This is the Snapshot card's "house keeper"; a sort of memory manager and mini operating system which allows the Snapshot software package to work within an interrupted program.

The Snapshot Shell is now provided to give software developers and other programmers the ability to store their own machine-code routines in the Snapshot card and make use of its interrupt-and-resume facilities. The Shell can also be used to interface with commercially available software utilities such as the Inspector and Watson.

Using the Shell, you can interrupt whatever your computer is doing and take control of it with a Snapshot package written by yourself. It might be a super debugger, a graphics editor, a comms package, or a machine control program; the only restrictions are space (just over 4K available), and your imagination. When you have finished with your Snapshot program, you can return control of your computer to the interrupted software without it ever knowing it was disturbed.

The Shell comes with menu-building routines which give you the ability to create software packages just like ours. Authors who wish to commercially exploit their own Snapshot software may do so without any licence from - or payment to - Dark Star Systems Ltd. Snapshot cards can be purchased for marketing with your work by arrangement.

You can backup your essential software

Even if you are an experienced computer professional, you cannot be sure you will never accidentally corrupt or erase your original program disks. And, of course, disks have been known to wear out!

If you are lucky, a damaged disk may mean weeks, sometimes months, of waiting for a costly replacement. If you're not so lucky, the company which produced the program you rely on has gone out of business.

The only effective way to safeguard your software investment is to make backups, but software protection makes this difficult. That's why thousands of Apple users around the world, from multi-national corporations to backroom hobbyists, have invested in some protection of their own – the Snapshot Copykit.

The Snapshot Copykit is a powerful, fast and (dare we say it?) "user-friendly" system that enables you to copy memory-resident programs. It takes around 11 seconds to backup a program which uses 64K of RAM, and 25 seconds for one which uses 128K. When the copying process is complete, you have an unprotected, bootable disk containing a working copy of your "protected" software.

Copykit backups are made up of binary files which can be easily transferred to other storage media like hard-disk, 8" disk, 3.5" disks, 80-track disks, and even bubble-memory.

If you just have standard Apple disks, you can use the Copykit's highly efficient compression option to reduce the amount of space your programs take up. That leaves you room to keep several different programs on a single floppy.

But making backups is just the beginning! With creative use of the Copykit, you can:

- *Inspect and modify off-the-peg software to suit your own needs*
- *Save hours of loading and saving data files by suspending one program while you run another*
- *Load and save the largest spreadsheets in 25 seconds*
- *Freeze-frame arcade game action, print out high scores and plan strategy*
- *Save your favourite games at hard-to-reach high levels and return to them again and again*
- *Print out any 40- or 80-column text screen or graphics (if you have a graphics card) and resume running your program instantly.*

darkStar SYSTEMS

2. Getting Started

This manual assumes that you have already installed your Snapshot card in your computer. If you have not, please do so before proceeding further. (If necessary, refer to the installation instructions which accompany the Snapshot card.)

Backing up the Shell Disk:

Since any program which you write for use with the Snapshot card will need to be stored on disk with the Shell, it is important to make several backups of the Shell disk before beginning work. Your Shell disk is not copy-protected in any way, so backing it up is simply a matter of using the "COPYA" program on your DOS System Master or a similar copying utility.

If you have 2 drives, you will probably find it convenient to use the copy program supplied with the Shell. Getting into BASIC, inserting the Shell disk in Drive 1 and typing BRUN DISKCOPY <RETURN> will bring up the following display:

```
INSERT SOURCE DISK IN SLOT 6 DRIVE 1

INSERT TARGET DISK IN SLOT 6 DRIVE 2

FORMAT TARGET DISK? (Y, N, Q)
```

Place a blank disk in Drive 2 and press "Y" to start copying. We recommend that you make at least 2 backups of your Shell disk to start with. When you have made your copies, put your original disk away for safe-keeping.

How a Snapshot software package works:

Before proceeding further, we should discuss briefly the basic operating principles of a Snapshot program. It is helpful to think of such a program as comprising two parts: the Shell itself, and the applications routines which do the job the complete package was designed to do. The applications part will be your contribution to the Snapshot program and, for the sake of brevity, we will refer to it from now on as the "Egg".

The Egg and the Shell exist side by side on a single disk which is booted at the start of any session where the Egg's services are going to be required. Both parts of the Snapshot software are loaded from the disk into the 8K of Snapshot RAM and stay hidden there until you need them. In the meantime, you can run any other programs you like.

Now, suppose you are running a program (let's call it "BudgieCalc") and you need to make use of the facilities offered

by the Egg. You simply press the Snapshot trigger to interrupt BudgieCalc and bring up the Menu or whatever other means you have designed to interface with the Egg. Behind the scenes, the following sequence of events takes place:

1. The Snapshot card issues a Non-Maskable Interrupt (NMI) to the computer hardware. This forces the computer to interrupt BudgieCalc and pass control to the Shell.

2. The Shell takes note of the current state of BudgieCalc, memorising the address of the instruction that was executing when the NMI was issued, the contents of the CPU's registers and the condition of the soft-switches which control memory bank switching and the screen.

3. The Shell takes itself and the Egg and swaps places with an 8K portion of BudgieCalc. That bit of BudgieCalc is copied from the computer's main RAM to the Snapshot card's RAM; the resulting empty space is simultaneously occupied by your Snapshot software.

4. The Shell sets the following defaults:

- screen set to Text Page 1 and cleared;
- "Language card" space ($D000-$FFFF) set to ROM;
- alternate character set enabled on Apple //e;
- auxiliary RAM and zero page disabled on Apple //e;
- C800 ROMs on peripheral cards disabled;
- slot ROM space on Apple //e enabled;
- Text window set to full size;
- Page Zero set to normal DOS 3.3 defaults;
- RESET vector points to the beginning of this step. (If <RESET> pressed, all defaults renewed.)

5. The Shell sounds a chime and passes control to the Egg.

6. The Egg runs, doing whatever you designed it to do. It can call routines built into the Shell to provide various useful services:

- send text to the screen;
- set the soft switches to any desired screen mode;
- detect the current state of the soft switches on the //e;
- copy to main RAM that displaced portion of BudgieCalc currently cached in the Snapshot card RAM;
- display BudgieCalc's Text or Graphics screens.

7. When you have finished with the Egg, it will return control of the computer to the Shell which will then copy itself and the Egg back into the Snapshot RAM. At the same time, the Shell will restore the displaced 8K of BudgieCalc, reset the program's soft switches, screen and registers, and resume running it from the exact point at which it was interrupted.

3. Going to Work on an Egg

How to write an Egg:

You can write the Egg using any editor/assembler, being careful to bear in mind the following points:

1. The Egg should be designed to ORG at $B000 and end before $C000.

2. The Shell passes control to an Egg from a JMP $B000 instruction loaded at $AA00.

3. When the Egg has completed its task, it can return control to the Shell with an RTS. Alternatively, it can JMP to an address within the Shell's Jump Table (see below) called "RESUMEPGM".

4. Your Snapshot program will displace 8K of the program in main memory and store it in the Snapshot card's RAM, leaving pages $0-7 and $A8-BF free for its own use. Addresses $0300-$03F1 and $A800-$AFFF are occupied by the Shell. The Egg may make free use of $0-$2FF, $3F2-7FF and $B000-$BFFF. The rest of memory contains the undisturbed remainder of the interrupted program (which you mess with at your peril).

Services provided by the Shell:

Let's examine the file on your Shell disk named "LISTING". As its name would suggest, this file contains a listing of portions of the Shell as it was assembled by the Apple ProDOS Toolkit Assembler. You can use it to refer to the addresses of routines and data located within the Shell that your Egg can make use of.

The following is a further description of those routines and data (variables not described here are not used by the Shell):

1. The $A800 data area consists of variables that describe the state of the suspended program at the exact moment of its interruption. If these variables are altered, the interrupted program may not resume properly:

STACKPTR = contents of the stack register.

SLOTROM = slot number of the card with its $C800 enabled (if any).

FFEXSTOW = a byte containing single-bit flags describing the state of the bank-switching soft switches on an Apple //e. (On a II+, this will be set to 0.) A bit is set to 1 if the corresponding switch is on; ie., if a read of the switch gives a negative value. Table 1 (page iii) labels each bit from 0 to 7, reading from right to left, and shows the effect of each when it is on.

MEMSTATUS = the address, when added to $C000, of the soft switch that must be read twice to bank-switch the "language-card" space correctly. For example, a value of $8B means that the address $C08B would be read twice before the interrupted program could be resumed, causing the $D000-$FFFF RAM to be enabled with Bank 1 active.

SOFT = eight single-bit flags that indicate the state of the screen soft-switches. Table 2 (page iv) labels each bit from 0 to 7, reading from right to left, and shows the effect of each when it is on. On the Apple //e, SOFT (like the other variables) is set by the Shell just after the NMI is issued. On the Apple II+, since the hardware necessary to read the state of the switches does not exist, SOFT must be set manually by the user with the DETVID routine (see below) or an equivalent.

ASTOW = the contents of the accumulator.

XSTOW = the contents of the X-register.

YSTOW = the contents of the Y register.

RHSTOW = the contents of the Program Counter (high-order).

RLSTOW = the contents of the Program Counter (low-order).

(Note that RHSTOW and RLSTOW contain the address of the first instruction to be executed when the interrupted program resumes.)

STATSTOW = the contents of the status register.

2. Located at $AA00, the Jump Table provides JMPs to the Shell's service routines. These can be accessed by doing a JSR to the appropriate JMP instruction. Doing this will ensure that the Eggs you write now will be compatible with future versions of the Shell. All routines return with an RTS.

The first JMP is the Shell's entry to your Egg. It points to $B000 unless you change it. If you type, for example, BLOAD SHELL, A$804 <RETURN>, this JMP will be located at $1200. The change can be saved with BSAVE SHELL, A$804, L$1FFB <RETURN>.

Table 1 - FFEXSTOW

Corresponding Address	Bit	Label	Name	Effect when on
-	0		DISABLESLOTS	(Enable Internal ROM) The $C100-$C7FF address space is controlled by the Apple //e internal ROM. The card ROMs are invisible.
$C013	1		RAMRD	Auxiliary RAM can be read.
$C014	2		RAMWRT	Auxiliary RAM can be written to.
$C017	3		SLOT3ENABLE	Rear slot 3 enabled on Apple //e. Set to OFF by presence of 80-column card in auxiliary slot 3 at power-on.
-	4		ENAC800	Enable $C800 internal ROM (may be disabled with $CFFF).
$C016	5		AUXZP	Auxiliary zero page/stack/language card is enabled
-	6		-	Not used by the Shell
-	7		-	Not used by the Shell

Table 2 - SOFT

Corresponding Address	Bit Label	Name	Effect when on
$C018	0	80STORE	Auxiliary 40 columns enabled for writing (on Apple //e only).
-	1	SOFTSET	Shows that SOFT has been given a valid value.
$C01A	2	TEXT	Text mode enabled.
$C01B	3	MIXED	Mixed text and graphics.
$C01C	4	PAGE2	Page 2 enabled.
$C01D	5	HIRES	High-resolution graphics enabled.
$C01E	6	ALTCHARSET	Alternate character set (on Apple //e only) enabled.
$C01F	7	80COL	80-column mode (on Apple //e only) enabled.

Routines in the Jump Table:

NAME:	RESUME
FUNCTION:	Resumes running the interrupted program, undoing everything which was done at NMI time. It packs the Egg, the Shell and the lowest 8 pages into the Snapshot card RAM, sets the soft-switches and registers, restores and restarts the interrupted program, and makes the Snapshot card invisible to any software in main memory. The resumed program will not be altered in any way unless as a result of the Egg's operation.
INPUT:	None
OUTPUT:	Does not return to caller
USES:	Many zero-page locations, the stack, etc. It is not safe to assume that the Egg will find its zero-page data intact after the next NMI.

NAME:	DETVID
FUNCTION:	Permits viewing of the interrupted program's screens and the setting of SOFT (see above). DETVID waits for the user to press the left- and right-arrow keys. Each key-press cycles the display to a different video-mode. When the user presses <ESC>, the current mode is stored in SOFT, the default mode is set, and control returns to the caller. The value in SOFT will be used to set the screen mode when the interrupted program is resumed. Note that DETVID is usually unnecessary on the Apple //e (where SOFT is set by the Shell), but it's essential on the II+ (where SOFT can only be set by the operator). SOFT remains set on the II+ until BOOTADISK (see below) is called, or until reset with DETVID.
INPUT:	None
OUTPUT:	Sets SOFT
USES:	All registers

NAME:	SETSOFT
FUNCTION:	Sets the screen mode soft switches to any desired setting, according to the values of the bits in the accumulator (see Table 2, page iv, for a description of the significance of each bit). SETSOFT neither uses nor modifies SOFT itself. For example: a value of $64 in the accumulator will make SETSOFT set the screen to 40-column Text Page 1, with the alternate character set enabled (a default setting).
INPUT:	Accumulator
OUTPUT:	None
USES:	All registers

NAME:	GETSOFT
FUNCTION:	The reverse of SETSOFT, it works only on the Apple //e. GETSOFT checks the current condition of the soft switches and puts their settings into the accumulator.
INPUT:	None
OUTPUT:	Accumulator
USES:	All registers

NAME:	ISIT128K2E
FUNCTION:	Determines if the host computer is an Apple //e with an extended 80-column card in auxiliary slot 3. If so, the "=" flag is set.
INPUT:	None
OUTPUT:	"=" flag set if true.
USES:	Accumulator

NAME:	HAY80COL
FUNCTION:	Determines if the host computer is an Apple //e with a standard- or an extended- 80-column card in auxiliary slot 3. If so, the "=" flag is set.
INPUT:	None
OUTPUT:	"=" flag set if true
USES:	Accmulator

NAME:	ISITA2E
FUNCTION:	Determines if the host computer is an Apple //e. If so, the "=" flag is set.
INPUT:	None
OUTPUT:	"=" flag set if true
USES:	Accumulator, Y-register

NAME:	MSGOUT
FUNCTION:	Sends a message to the screen. A string of ASCII characters, followed by a zero, should be assembled immediately after a JSR to MSGOUT. The characters will be sent to the screen (using COUT1), starting from the current cursor position. On the Apple II+, any lower-case characters entered will be converted to upper-case. Execution will continue immediately after the zero following the ASCII characters (see the source code on your Shell disk).
INPUT:	None
OUTPUT:	Sends messages to the screen
USES:	All registers, $FE, $FF and monitor zero-page locations

NAME:	RWTSE
FUNCTION:	Read/Write a disk sector. RWTSE is a condensed version of RWTS (see the LISTING file on your Shell disk for a complete comparison of the two). RWTSE is present in the Shell from $BB00 to $BFFF. If your Egg extends beyond $BB00, you must not attempt to use RWTSE. (Note that RWTSE's entry point is not at $BB00.)
INPUT:	I/O block parameters
OUTPUT:	Carry set/clear, error code
USES:	All registers, DOS zero-page locations

NAME:	TEXT1WAY
FUNCTION:	Copies the interrupted program's Text Page 1 from the Snapshot card RAM to $0400-$07FF. This copy may be altered or destroyed without adversely effecting the interrupted program. The page in Snapshot is left intact.
INPUT:	None
OUTPUT:	None
USES:	All registers

NAME:	MSGOUT1
FUNCTION:	Sends a message to the screen, but differs from MSGOUT (see above) in that the horizontal starting position is given by the X-register and the vertical starting position (from the top of the screen) is given by the Y-register.
INPUT:	X and Y registers
OUTPUT:	Message to screen
USES:	As MSGOUT

NAME:	BOOTADISK
FUNCTION:	Boots a new disk, clearing main memory. SOFT and FFEXSTOW are reset, the I/O vectors are initialized, the Page 3 vectors are reset, language-card RAM is disabled, the Shell and the Egg are moved to the Snapshot RAM, and control is given to the disk controller card.
INPUT:	The slot to be booted should be in the accumulator. (If the value is zero, the computer will cold-start.)
OUTPUT:	None
USES:	As RESUME

NAME:	SWAPALL
FUNCTION:	Swaps those portions of the interrupted program cached in the Snapshot RAM at NMI time back into main memory, so that they can be accessed without alteration. Other parts of the interrupted program are simultaneously swapped into the Snapshot RAM. (See memory maps.) Swapped areas must be restored by running SWAPALL a second time (or an even number of times) before resuming the interrupted program.
INPUT:	None
OUTPUT:	None
USES:	All registers

Putting the Egg in the Shell:

Once you have written your Egg, use FID to transfer it to one of your copies of the Shell disk. Now, rename it (if you haven't already done so) "Egg". Type: EXEC PUT EGG INTO SHELL <RETURN>. This will run a short EXEC file program that BLOADs SHELL into memory, BLOADs EGG into SHELL and BSAVEs the expanded SHELL file.

When the resulting disk is booted, SHELL is quick-loaded into memory at $804, asks you what slot the Snapshot card is in, loads itself into the card's RAM when you respond, and asks you to press the Snapshot trigger. At this point, pressing the trigger will activate your Egg; pressing <RESET> (or <CTRL> <RESET>) will leave you at the BASIC prompt.

The Shell disk:

The Shell is supplied on a fairly ordinary DOS 3.3 disk. When it is booted, it loads the SHELL file to $804 and starts it running. The DOS is modified to speed up BLOADing and to let you TLIST (display) a text file. Listings are stopped and restarted by depressing the <SPACE> bar rather than the usual <CTRL>S. INIT is gone, and the RWTS cannot format a disk.

4. The Menu Driver Routines

The Shell's menu driver routines are intended to offer you a simple method of incorporating Snapshot-type cursor bar menus into your Egg. These routines are entirely table driven and may be called recursively, allowing several sub-menu levels to be active simultaneously. Each menu or sub-menu requires one table but only one copy of these routines is needed to handle all menus.

Source code for the menu driver is contained in the text file named "DOMENU" on the Shell disk. This file is in a format compatible with the Apple DOS toolkit assembler, Apple //e assembler, Big Mac and Merlin.

The basic procedure for using menus from within your Egg is as follows:

1. Ensure that the computer is displaying the right video mode (i.e., Text Page 1) with a full text window. (This will only be necessary if your Egg has changed the video mode since it received control from the Shell.) The text screen should be cleared before displaying the menu.

```
e.g., JSR HOME   ;CALL MONITOR AT $FC58
```

With sub-menus, you can clear just part of the screen if you prefer. If you want to display a title on the screen, this should be done before calling the menu itself.

2. Set up the Y and A registers to point to the appropriate menu table.

```
e.g., LDA #MENUTABLE   ;LSB TO A REGISTER
      LDY #MENUTABLE/258   ;MSB TO Y REGISTER
```

3. Call the PRMENU routine to display the menu on the screen. (This will not put the cursor on-screen.)

```
e.g., JSR PRMENU   ;PRINT MENU
```

4. Call the DOMENU routine to operate the cursor bar for selection of menu options. This routine accepts the left-arrow key to move the cursor bar up and the right-arrow key to move it down. A menu option is selected with the carriage return key.

5. If DOMENU exits with the carry flag set (because an unrecognized key has been pressed), you may check the value in the accumulator to see if the pressed key is pertinent to the operation of the Egg. If not, you should introduce a loop back to the DOMENU call.

```
e.g.,    JSR DOMENU
         BCC GOTONE       ;IF <RETURN> WAS PRESSED
         CMP #ESC
         BNE LOOP
       ;FALLS THROUGH HERE IF <ESC> PRESSED
```

6. If DOMENU exits with the carry flag clear, the the option # selected is in the A and X registers and may be used to access data in tables, or to branch to the required command handler (see the demonstration program "GOMONITOR").

The Menu Subroutines (see GOMONITOR for menu table format)

NAME:	PRMENU
FUNCTION:	Displays menu from the menu table. Options may be displayed in lower case, but all options will automatically be converted to upper case if the host computer is an Apple II+. PRMENU will not display de-activated menu options.
INPUT:	A = LSB of menu table's address; Y = MSB of menu table's address.
OUTPUT:	Leaves (MENP) pointing to start of menu table.
USES:	A, X, and Y registers; zero page $06 and $07.

NAME:	DOMENU
FUNCTION:	Handles menu option selection. De-activated options are skipped automatically. The carry flag is set if an unrecognized key (i.e., neither the left-arrow, the right-arrow nor <RETURN>) is pressed and that unrecognized key is in the A register (with top bit set).
INPUT:	Requires (MENP) to be pointed to the start of the menu table.
OUTPUT:	Exit when <RETURN> is pressed with the selected option number (0 to MAX) in the A and X registers and when the carry flag is clear.
USES:	A, X, and Y registers.

NAME:	CLRWINDOW
FUNCTION:	Clears a (lower) portion of the screen to display a sub-menu and homes the cursor to the top left of the sub-menu "window". The size of the text window is not set.
INPUT:	None
OUTPUT:	None
USES:	A and Y registers.

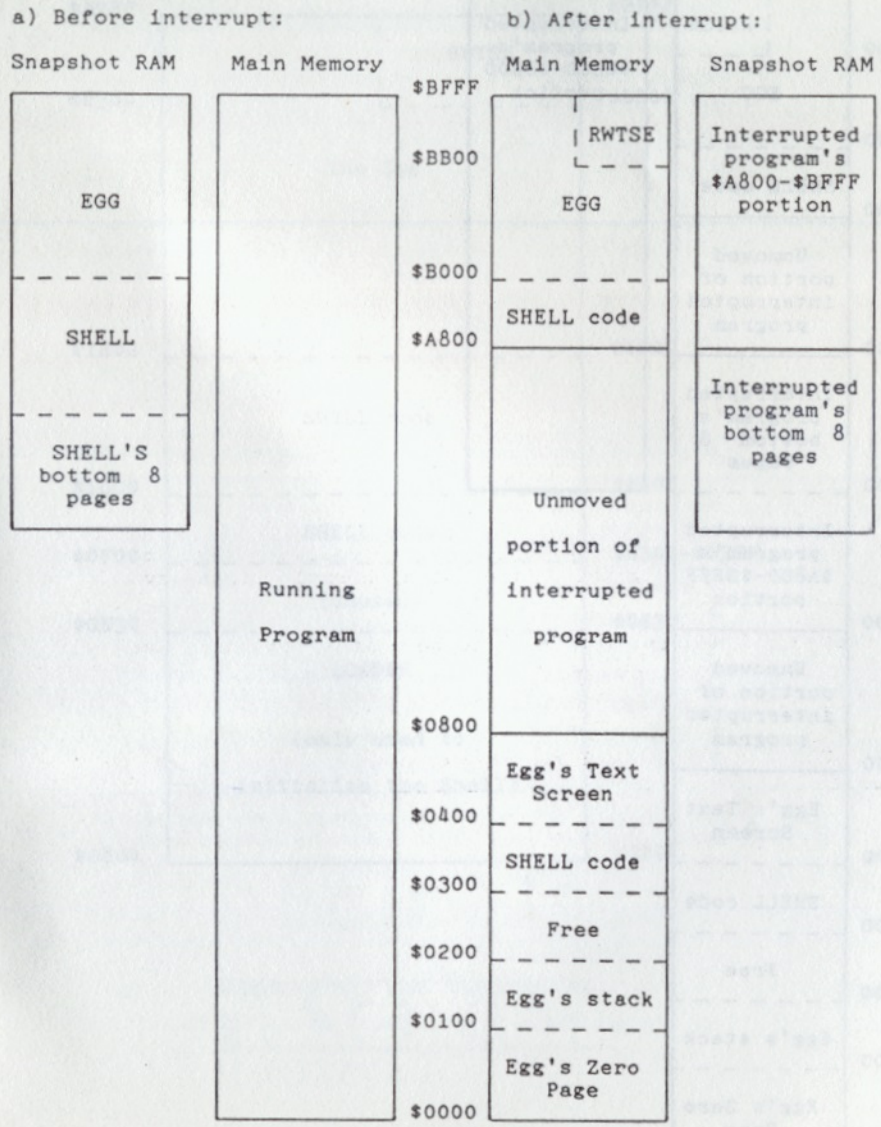
NAME:	CLRWINDOW1
FUNCTION:	Provides an alternative enty point to CLRWINDOW, clearing the screen from line A to EOP.
INPUT:	The A register contains the line from which to begin clearing.
OUTPUT:	None
USES:	A and Y registers.

NAME:	GETCURAX
FUNCTION:	Gets the current menu option number.
INPUT:	Requires (MENP) to be pointed to the start of the menu table.
OUTPUT:	The A and X registers contain the current option number. (Remember the first option number is 0, not 1.)
USES:	A and X registers.

NAME:	ISITACTIVEX
FUNCTION:	Checks to see if the menu option number in X is legal and active. This function is automatically executed by both PRMENU and DOMENU, so you will rarely need to use it.
INPUT:	The menu option number to be checked in the X register.
OUTPUT:	The V flag is set if option X is not in range for that menu; the carry flag is set if the option has been de-activated.
USES:	A, X, and Y registers.

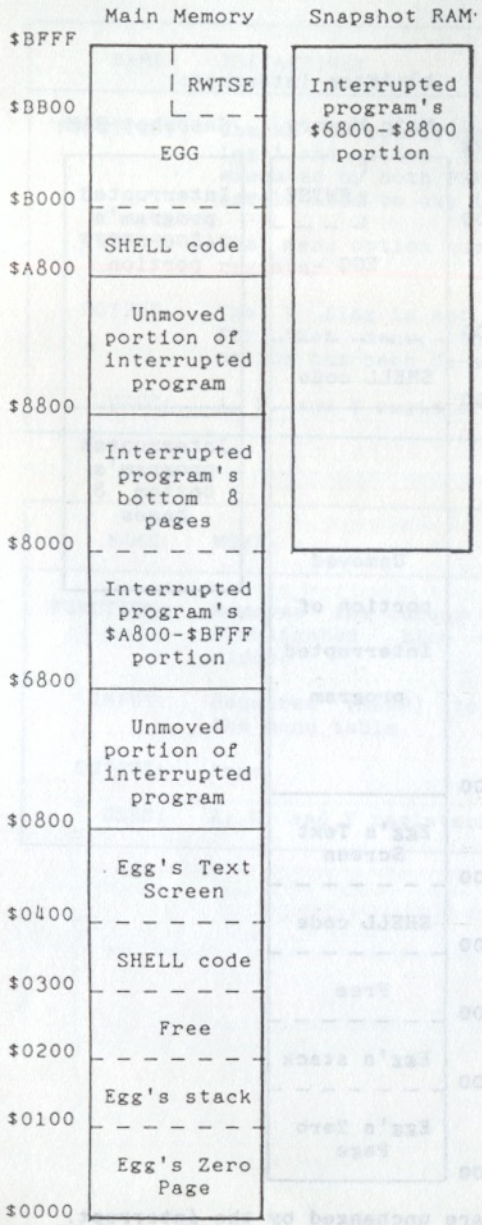
NAME:	MOFF
FUNCTION:	Removes the cursor bar by replacing the menu's highlighted line with normal (non-inverse) video.
INPUT:	Requires (MENP) to be pointed to the start of the menu table.
OUTPUT:	None
USES:	A, X, and Y registers.

4. Memory Maps

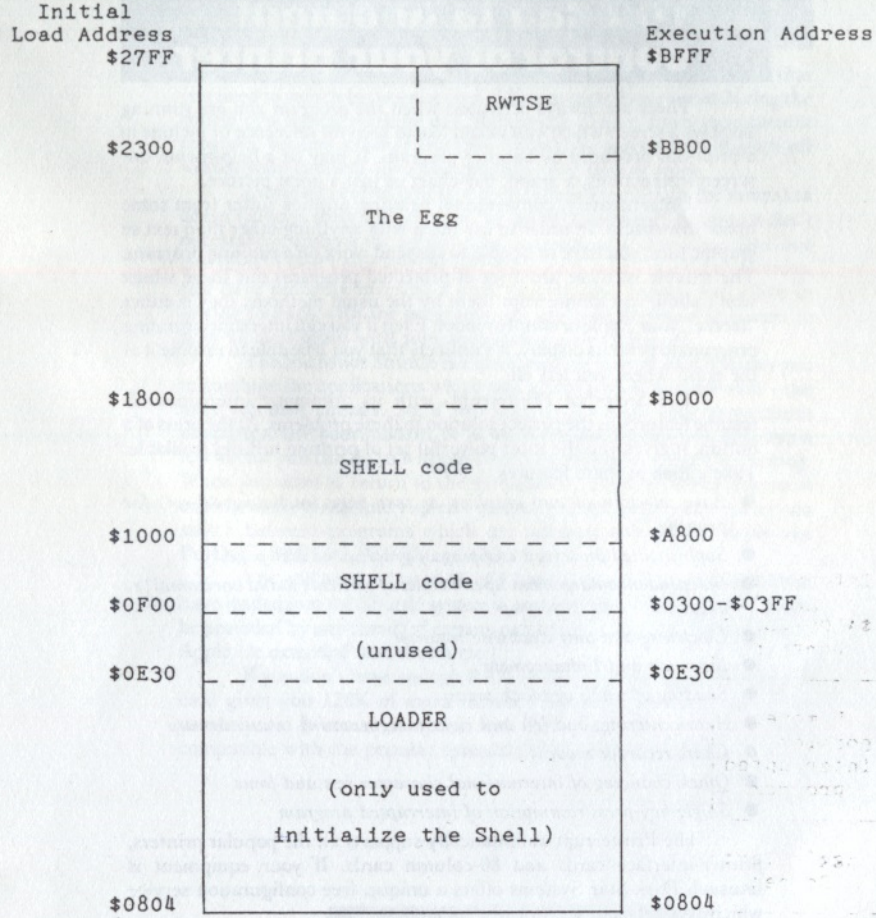


Note: all other areas of memory are unchanged by the interrupt.

Effects of SWAPALL:



SHELL file structure:



You can take complete control of your printer

There are always occasions when the program you are running displays a screen which you would like to keep for reference or include in a print-out produced by another program. It may be a help-menu, on-screen instructions, a graph, bar-chart or just a great picture.

Unfortunately, conventional printing utilities suffer from some major drawbacks. In order to use them with anything other than text or graphic files, you have to be able to suspend work on a running program. The trouble is, there are a lot of protected programs out there which won't allow you to interrupt them by the usual methods; they'll either "freeze" your Apple or simply reboot. Even if you can interrupt a running program to print its display, it's unlikely that you'll be able to resume it at the point where you left off.

The Snapshot Printerrupt, with its automatic interrupt-and-resume features, is the perfect solution to these problems. At the press of a button, it gives you the most powerful set of printing utilities available. Take a look at these features . . .

- *Easy selection of any graphics or text page (including 80-col) for printing*
- *Sophisticated on-screen cropping of graphics or text pages*
- *Independent enlargement up to 8 times of vertical (y) and horizontal (x) axes*
- *Clockwise and anti-clockwise rotation*
- *Inversion and Enhancement*
- *Shading of white or black areas*
- *Auto-centering, and left and right margin setting in any density*
- *Chart recorder mode*
- *Quick changing of international character sets and fonts*
- *Single key-press resumption of interrupted program*

The Printerrupt automatically supports all the popular printers, printer-interface cards and 80-column cards. If your equipment is unusual, Dark Star Systems offers a unique, free configuration service which will get your Printerrupt up and running.

darkStar SYSTEMS

Your Apple can do more than one job at a time

No matter what you use your computer for, the chances are that you need to switch between several different tasks many times during the course of a typical working day. Repeatedly closing down your current program, booting another and then finding the place where you left off wastes your valuable time and disrupts your flow of work.

Lumping several different applications together on the same disk doesn't always solve the problem. So called "integrated" programs don't necessarily combine the applications you want and, even if you find one that does, it won't give you the sort of power that you're used to. Besides, you have probably invested a great deal of time, money and effort in getting to grips with the programs you use now; do you really want to start all over again with something completely different?

The Snapshot Shuttle is a multi-tasking system which allows you to combine the applications which you actually want to work with – the ones you own already. So, if you want to interrupt your spreadsheet program to use your modem, or to word-process a letter, or just to zap a few aliens, you can do so without swapping program disks or re-booting. When you want to return to the spreadsheet, the Shuttle can resume it exactly where it was interrupted – instantly. The Shuttle will even let you switch between programs which use differing operating systems like ProDOS, PASCAL, CP/M and DOS 3.3.

You will need at least 64K of RAM for every program you wish to have loaded into the Shuttle system at any one time. This extra RAM can be provided by any standard expansion card (e.g., Ramex, Saturn, Titan, Apple //e extended 80-column etc.)

If you don't have enough RAM at the moment, our RAMrod 128 card gives you 128K of extra memory for about half the cost of its competitors. It comes complete with RAMdisk software and is fully compatible with the popular spreadsheet expansion packages.

darkStar SYSTEMS