REPLAY II

BY

MICRO ANALYST INC.

V2.Ø

INDEX

1.Ø INTRODUCTION Ø4

2.Ø REPLAY CARD INSTALLATION.....Ø5

3.Ø MAKING A COPY.....Ø5

4.Ø MONITOR.....Ø9

5.Ø EXECUTE REPLAY COPY.....12

6.Ø VIEW SCREEN.....12

7.Ø CREATE DOS BINARY FILE.....13

8.Ø 64K COPIES.....15

9.Ø PACKER PROGRAM.....17

1Ø. PACKING PARAMETERS.....37

11. TIPS ON COPYING AND PACKING.....41

12. PACKING EXAMPLES.....42

13. COMPARE PROGRAM.....47

14. COMMAND FILE CREATE.....48

15. SOFTMOVE.....49

APPENDIX

A. EXTRA USES FOR THE REPLAY CARD.....5Ø

B. MULTI ACCESS PROGRAMS.....53

C. DOS COMPARISONS.....54

D. HEX NUMBER SYSTEM.....54

E. REPLAY TRACK REFERENCE.....55

F. APPLE LANGUAGE CARD OWNERS.....56

G. ASSEMBLY REFERENCE GUIDE.....56

H. RAMSTEP...STEP & TRACE.....61

I. SCREEN PRINT.....63

J. II+ and //e Differences.....64

K. FRANKLIN 1ØØØ USERS.....65

The REPLAY COPY card is guaranteed for 90 days after purchase. All parts and labor are under warranty. If a problem develops in hardware return the card to MICRO ANALYST for repair.

REPLAY COPY is for archival backup and program analysis. MICRO ANALYST takes no responsibility for the actions taken by users of this card.

MICRO ANALYST INC.
P.O. BOX 15003
AUSTIN , TEXAS 78761

(512) 926-4527

1.Ø INTRODUCTION

The REPLAY card can be used for several purposes.

1. Copying programs
   - A) Converting to standard DOS 3.3
   - B) Unprotection for execution on APPLE //e or II
   - c) Transferal of protected programs to hard disk

2. Program development and analysis
   - A) APPLESOFT pointers
   - B) Machine language analysis

With the current Replay eprom you can copy a program in either the lower 48k or the entire 64k of memory. No data compression is used in order to insure maximum reliability of copy and restart.

There are other functions of the Replay card. When the Replay card is used to stop a program all of the memory is conserved. NOT ONE SINGLE BYTE IS CHANGED. The user may examine/change/search memory with a monitor in the Replay eprom. While the user is doing this memory is still preserved. NOTHING is changed unless the user so specifies. You can then restart the program, either 48 or 64k . Applesoft pointers are displayed with their current values.

The program development and analysis functions will be expanded in the future. New eproms dedicated to special analysis are being developed. The current card gives the user some power in that area but specialized eproms will enhance that capability.

Replay can be left in the system and forgotten about until needed. When a copy or analysis is desired simply press the button on the end of an 18" cord outside the computer.

The minimum requirement for Replay is an APPLE II or //e with one disk drive. All standard 16k ram cards are compatible. If a larger ram card emulates a 16k ram card it is also usable. Owners of the APPLE language card with an $F8 Rom on board see appendix F

The following are trademarked or copyrighted names used in this document:

APPLE/APPLESOFT/LOCKSMITH/TURBODOS/NIBBLES AWAY II /REPTON/INSPECTOR/DR. WATSON/VISICALC

2.Ø REPLAY CARD INSTALLATION

Turn off the power to your APPLE. Remove the lid and locate the interface slots in the back of the computer. Replay can be put in any slot. The only requirement is that the disk controller card be in slot 6. Insert Replay into any slot, run the cable out the back of the APPLE, and replace the lid.

***** NEVER PULL HARD ON THE CABLE *****

Pulling on the cable can have disastrous consequences, especially if the power is on.

To trigger the Replay card simply press the button on the cord. If power is on the Replay menu will be displayed immediately. We recommend never pressing the button during disk I/O as this may destroy the data on the disk!!!

3.Ø COPY PROGRAM

3.1 Short Example

Boot a disk with DOS 3.3. When you have the Applesoft prompt type the following lines:

```
NEW
1Ø PRINT"HELLO";:GOTO 1Ø
```

Once these lines have been entered type 'RUN'. The screen should fill up with the word HELLO.

Now press the Replay copy button on the cord. The program will stop and the upper 1Ø lines of the screen will contain a menu. The menu is described fully in the following sections. For now we will use only option C. Type C on the keyboard and the screen will display a new menu. The prompt asks you to remove the DOS 3.3 disk in drive 1 and insert a blank disk. When you have done this hit return. Your original program will then be copied onto the disk in drive 1. This is the Replay quick load copy. When it is completed the main menu will return.

You have now made a copy of the program. You may exit the Replay system by either typing R to restart the interrupted program or by hitting reset to turn off the Replay card.

To execute the copy you have just made press the Replay button; when the menu comes up type E for execute. Make sure the Replay quick load copy disk is in drive one. Try this with the copy you have just made. Once the copy is back in memory the program will restart. For more detailed description proceed to section 3.2.

3.2 Full Copy Procedure

Boot the program you wish to copy. When the program is in and running you are ready to copy it. At any point you can press the Replay button and the program will stop. A good place to copy your program is at a menu.

When you press the button the following menu (shown below between the stars) will appear in the top 10 lines of the video screen. If you are using 80 column card you will have to use the 40 column APPLE video when working with REPLAY. If your monitor or tv cable is run through an 80 column board then you will need to hook directly into the APPELE 40 column video connectors. The program can be restarted in 80 column.

This is the Replay menu showing your current options. If you hit the first letter of any of the commands it will execute.

```
****************************************

          <*> REPLAY II <*>

C) COPY
M) MONITOR
B) BOOT 16 SECTOR
E) BOOT REPLAY DISK
R) RESTART
V) SCREEN (1)  S) RAM/ROM/SLOT (   )
L) VIEW SCREEN

****************************************
```

The //e version will not have the V option as the screen is set by Replay card.

Beneath this menu is blank space or a lot of garbled letters will be present. Some programs put part of their operating system in the video page and you will see the memory of that page being displayed. Replay has extra memory buffered (added) so that nothing is changed by your interruption. However there is only enough memory to buffer 10 lines of text on the video page. Those ten lines are saved and then the Replay menu is displayed.

The first option C) is used to copy the current program. When you press C the following prompt will be displayed:

```
****************************************

SCREEN AND CARDS SET?

REMOVE PROGRAM DISK

PUT BLANK DISK IN DRIVE 1

TO COPY ->HIT RETURN. OR HIT N FOR MENU

****************************************
```

The first line is in reference to the video screen and any interface cards you want turned on when you restart the program. Section 3.2.1 will describe this first step, selecting the correct screen, in more detail. Section 3.2.2 will cover the interface cards in more detail.

3.2.1 Setting restart screen.

When you pushed the button to interupt the program Replay has no means of knowing what video screen was being displayed. Repay will assume text page 1. That is what is meant by the (1) in the option 'V) Screen (1)' of the main Replay card menu. If any other screen is desired upon restart you must tell Replay that. To do so use option V). When you do so the menu below will be displayed:

```
****************************************

VIDEO PAGE:                                  The //e version
                                             is abbreviated.
1 TEXT1   2 HIRES1     3 TEXT2

4 HIRES2 5 HIRES&TXT1 6 LORES1

7 LORES2 8 LORES1&TXT1"

****************************************
```

For instance if a game was being played hires page 1 was probably displayed at time of interrupt. When you interrupt the game type V; then by typing 2 the program will restart in HIRES 1.

When the program restarts hires 1 will be displayed instead of textl (the default choice). Any screen can be selected for display on restart.

3.2.2 Ram/Rom/Slot restart setting

When you selected 'C' for copy the menu said:

```
SCREEN AND CARDS SET?
```

You have just seen how to select the screen. Now the other option is for interface cards. You may wish to have an interface card initialized when the program restarts. To do so select S from the main menu. The following menu will appear:

```
**************************************

ENTER SLOT #1-7 OR....

FOR RAM CARD  A) BANK1 WRITE ENA
              B) BANK2     "
              C) BANK2 WRITE PRO
              D) BANK1     "

**************************************
```

If you want an interface card initialized type the slot number it is in. If you want the RAM card (language card) on when the program restarts select options A,B,C OR D. The 4 alphabet selections are the different modes the ram card can be turned on. Most programs that use the ram card use bank 2 and ram card write enabled. Thus option B would be chosen.

Replay assumes ram card off and no slot initialized. Option S on the main menu can be used to choose the desired restart configuration. The parenthesis next to RAM/ROM/SLOT indicate the restart configuration. For instance if the user chose option B) bank 2 write enabled, and slot 3 initialized then the prompt line would look like:

```
V) SCREEN (1)  S) RAM/ROM/SLOT (B 3)
```

Once the restart screen and the interface slots are correct you can proceed to copy. Press C to copy the program. Remove the program disk and place a blank disk in drive 1. Press return and the copying will start.

When the copying is finished the main Replay menu will return. Memory has been preserved and you can restart the program if you desire. Typing R will restart the program.

The copy you have just made is a quick load copy of the lower 48k of memory. This copy on disk can be reloaded to restart programs in the lower 48k in under 10 seconds. It can be executed by the Replay card.(see section V) If you wish to convert it to standard DOS 3.3 please see section 7.0. To make 64k copies see section 8.0.

Another option of the main Replay menu is the Monitor. It is described next.

4.Ø REPLAY MONITOR

The Replay card has a monitor built into it. When you stop the program by pressing the Replay button the main menu is displayed. By pressing the M key you can enter the monitor. The following menu is displayed.

```
****************************************

<*> REPLAY MONITOR <*>     PC=xxxx

ON STACK=xxxx xxxx  HOOK IN=xxxx OUT=xxxx

A=xx  X=xx  Y=xx  SP=xx
                                SV*BDIZC
$3F2 RESET VECTOR=xxxx STATUS=xxxxxxxx

M,P,L,S,W,Q,#/#,#.#W,#:#,#:'A>

****************************************
```

The x's are replaced with the values found by the Replay card at time of interrupt. The current program counter is displayed PC=xxxx. On the second line are the first 4 bytes on the stack. They are displayed as two return addresses. The first address after the = sign is where the computer would start executing if an RTS was executed. On the same line is the input (KSWL) and output (CSWL) hooks. These are the I/O input and output hooks used by the monitor for keyboard, video..etc. These hooks may or may not be valid. It will depend on the program executing. They are memory locations $38 and $36 respectively.

The third line displays the registers and their value at time of interupt. The fourth line is the reset vector, stored at $3F2. This contains the location in memory where the computer would jump to when the user presses the RESET key. Further along the same line the processor status is expressed in binary. The flags are documented above the binary output. See a 65Ø2 assembly language manual for more information.

The final line is called the command line and it is a list of possible commands with a prompt '>' at the end. The different commands and examples are listed below:

MONITOR COMMANDS

(M)
Display the screen as described above.

(L)
Display memory as hex and ASCII. Eight lines of 8 bytes each will be displayed. Once the first 8 lines are shown the computer will wait for a keypress. If a return is hit the computer displays the command line. Any other keypress will display 8 more lines of memory. For example:

```
200L show memory starting at $200
   L show memory starting at last referenced
address
```

(S)
This will display only 1 line of 8 bytes of memory.

```
4E0S  Show 8 memory bytes at $4E0
   S  Show 8 memory bytes at last referenced
address.
```

(W)
This command will set one entire page of memory to the value stored at $AC. This is useful in finding what areas of memory are used by a program. The user can selectively clear areas of memory and restart the program. If it still runs then that area was not crucial to program operation.

```
100W Clear page at address $100
 05W Clear page zero. Notice the high byte of
the address is the page to clear. If you
enter a number <$100 then zero page is
cleared.
1000.1500W Clear memory pages $10 through $15
   W Clear the memory page of last referenced
address.
```

(:)
This is used to store an entered byte to a specific memory address.

```
100:05 store $05 at address $100
2E0:FF store $FF at address $2E0
4E0:'A store ASCII A at address $4E0
```

(/)
This is used to search memory for a specific value. First you store the value to search for at address $0000. Then you enter the address to start the search, enter a /, then enter the number of pages to search.

```
0:05 store $05 at $0000
0400/1  search starting at $400, search 1
page. Print all addresses containing $05
0800/2  search starting at $800, search 2
pages forward.
```

See the section on extra uses for the Replay card for more info.

(P)

This instruction displays the APPLESOFT pointers. These pointers are only valid if an APPLESOFT program was running at time of interrupt. The following screen is displayed:

```
**************************************

<*> APPLESOFT POINTERS <*>

PROGRAM START($67)=xxxx    END($AF)=xxxx

VARIABLE END=xxxx    STRING END=xxxx

FREE MEM=xxK         CURRENT LINE=xxxx

**************************************
```

All of the numbers displayed are in hex. The program start and end values are shown. The next line is where variables end and strings begin. The space between these two values is free memory and is displayed in Hex on the next line. Also shown in hex is the current line number that the program was executing when you stopped it.

See also section 15.0 for moving APPLESOFT programs to standard DOS 3.3.

When you interrupt the program running the two lowest pages of APPLE memory are moved up to extra buffered memory at $CC00 AND $CD00. Page zero is stored at $CC00 and page 1 at $CD00. So any searches/changes for zero page should be performed at that location. When the program is restarted these pages are moved back down.

To obtain the monitor program counter and register values type M. This completes the Monitor commands.

ENTERING APPLE MONITOR

The user can enter the regular APPLE monitor by the following method.

1. Stop program with replay card.
2. Enter Replay monitor with M command.
3. Note the value of the PC
4. At PC value in memory install following patch=> 4C 59 FF
5. When you restart the program you will enter the APPLE monitor

example:

PC=$2400 when you stop the program.

```
From Replay monitor type:  2400:4C (return)
                           2401:59     "
                           2402:FF     "
                           Q           "   exit to main
                                           menu
From Replay main menu hit R to restart
   and enter APPLE monitor.
```

5.0 EXECUTE REPLAY COPY

When you make a copy with the Replay card the disk you create contains a copy of the lower 48k of memory in a quick load format. This is not a standard format such as DOS 3.3. The Replay card can execute this disk or it can be converted to DOS 3.3 and run without the Replay card. This will be covered in section 6.0.

To restart the program simply turn on the Apple, and press the Replay button. You don't need to have DOS active to restart the program. Once the Replay menu is on the screen insert the disk that contains the quick load copy and press E. The disk will load in and the program will restart.

To convert the copy into a standard DOS 3.3 copy see section 7.0

6.0 VIEW SCREEN

One last option of the Replay menu on the card is to view any screen. Type L to look at any screen able to be displayed by the APPLEII. Select the screen to view from the option list, when finished hit any key and the screen will revert to the Replay menu.

The screen option list is shown on page 7. The //e version is abbreviated.

7.Ø CREATE DOS 3.3 BINARY FILES

The copy made by the Replay card is in a nonstandard fast load format. If you wish to have a copy in DOS 3.3 this program will convert the Replay copy into standard DOS 3.3.

Two binary DOS 3.3 files will be automatically created for you. These two files together are the copied program. They can be put on Hard disk or moved to any DOS 3.3 disk. A language card must be in the system for the program to restart.

To make the conversion run DOSMAKER supplied on the utility disk. This is option 2 on the Replay utility disk menu. The following menu will appear:

```
**************************************

REPLAY II


ORIGIN DRIVE=>1

TARGET DRIVE=>2

TRANSFER 48K COPY INTO DOS 3.3


OPTIONS:

T) TRANSFER
C) CHANGE SOURCE/DESTINATION DRIVES


(RETURN) TO EXIT

>

**************************************
```

The source (Replay quick load copy) and the target (DOS 3.3 disk) disk drives are shown. Typing C will allow the user to change the default settings.

If you type T the transfer will begin. First some questions are asked. The first asked is what is the name of the file to be created. Enter any name that you desire. Two binary files will be created on the DOS 3.3 disk, one with the name you entered. The other will have the name you entered with a '.REP' extension.

The next question deals with the language (ram) card. You can leave the card on when the program executes or you

can have it turned off. This program only makes a 48k copy of the lower memory. For full 64k copies see section 8.0. Some programs use the language card for buffers, VISICALC for example. In that case you may wish to leave the language card on when you restart. With Visicalc you simply type '/CY' to reinitialize (clear) the ram card while VISICALC is running.

For some programs you may not want the language card on because they require the APPLESOFT in rom. Simply pick the alternative you want.

If you want the language card turned on, then the lower 48k is put back exactly as it was when the copy was made. The language card is left on with write enable and bank 1.

If you want the language card off then there is a small patch made to the program in the lower 48k to turn off the language card. This program (DOSMAKER) will search for a location to put this patch in automatically. You may specify an exact location for the patch if you wish, but it is suggested that the novice allow the program to automatically select a location.

When the option of having the language card off or on has been decided the transfer will start. You will be prompted for the correct disks if you are working on a single drive system.

When the conversion is over you will have two binary files on the DOS 3.3 disk. To execute the copy simply type:

```
BRUN NAME,A$800
```

If you don't want to type the ',A$800' every time perform the two following steps.

```
BLOAD NAME,A$800
BSAVE NAME,A$800,L$3700
```

This will allow you to run the files with the following command:

```
BRUN NAME
```

These two files are standard DOS 3.3 and can be transfered to any disk including Hard disks. It is suggested that users of 5 1/4 floppies put an optimized DOS 3.3 on the disk such as TURBODOS. This speeds up loading as standard DOS 3.3 is very slow.

8.Ø 64K COPIES

You can copy the entire 64k. The entire 64k of ram is copied without compression or packing. This gives a reliable restart. To make a 64k copy perform the following steps:

1. Boot the program you want to copy
2. Press the Replay copy switch
3. Set the screen and slots(see section 3.2)
4. Put a blank disk in drive 1, press C
5. When the copy is done, put the Replay utility disk in drive 1 and press B for boot
6. Choose the option for saveing the 16k ram card (option 5). //e owners have built in upper 16k ram
7. When prompted enter a name for the copy

You now have a 64k copy of the program. It is in two parts. What is suggested is to put the Replay copy on a back of a disk, and on the front put the normal DOS 3.3 file created by RAMSAVE. Transfer the file saved in step 7 above to this disk. To execute the copy do the following:

1. Insert the DOS 3.3 disk with the ramsave file and type 'BRUN NAME'
2. The computer will prompt you to insert the Replay copy
3. With Replay copy in drive 1 press Replay button.
4. Press E for execute. Your copy will now restart

The front of the disk is mostly blank DOS 3.3 and could be used for data file storage, like VISICALC or word processors.

To make 64k copies that do not require the Replay copy takes more skill. Software is being developed to automatically pack reliable 64k copies. For now the only option is to use the PACKER program to condense the lower 48k copy. Then the user could load in the language card saved memory and BRUN a packed copy of the lower 48k. The packed copy of the lower 48k does not require the language card to load in, thus it preserves the language card contents.

To use Packer for 64k copies follow these steps:

1) Stop the program, copy the lower 48k, be sure the ram card restart is set.
2) Boot the Replay utility disk and save the 16k ram card (option 5). Save the ram card as Name1.
3) Pack the lower 48k file and store it as Name2.
4) Run the program 'CONNECT' on the Replay utility disk.

Insert the disk containing the packed 48k copy Name2
and the file Name1.
5) When prompted enter Name1 as the ram card file.

The computer will modify the ram card file slightly and build an 'EXEC' file for you. This is the file used to execute the copied program. Make sure all three files are on the same disk at run time. To execute the copy type:

EXEC EXNAME1

9.Ø PACKER PROGRAM

INTRODUCTION

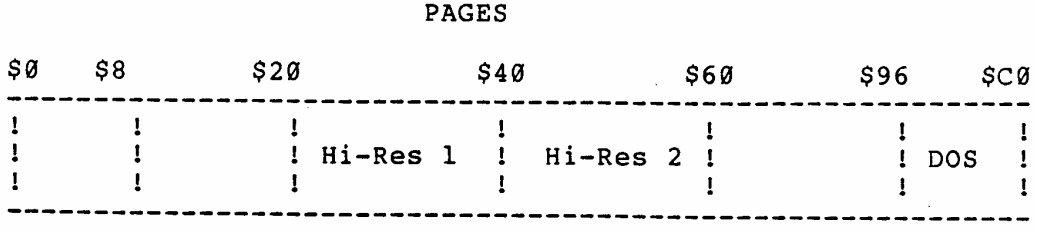
Packer is used to create a binary DOS 3.3 file from your REPLAY copy disk. This will compact your programs and reduce the number of disks necessary to store them on. With Packer you will select parts of the original program (now copied onto Replay quick load disk) and put them into a binary file. Then to run your program you will BRUN this binary file from DOS 3.3.

**** THIS SECTION IS VERY COMPLICATED ****
**** A KNOWLEDGE OF ASSEMBLY LANGUAGE ****
**** IS RECOMMENDED. ****

Basic Theory of Packing

Shown below is a map of the APPLE ram (random access memory). This does not include the 16k upper ram of a ramcard. The Replay quick load copy only contains the lower 48k and this is the program area you will pack. Many programs do not require this upper 16k. For users that have copied a full 64k and now desire to condense the copy into binary files see the section on 64k copies.

(All numbers are in hex)



In this document and in the PACKER program all numbers preceded with a $ symbol are in hex.

IDEA OF PAGES

A convenient way of referring to parts of memory in the APPLE is to use pages. A page is 256 bytes of memory. On the REPLAY disk there are $CØ (192 decimal) pages of memory, these make up the lower 48k memory. $C in base 1Ø is 12. Therefore there are 12*16 decimal pages of memory stored on a REPLAY disk. These are the first $CØ pages of memory on the APPLE.

WHATS ON THE QUICK LOAD DISK

The REPLAY disk contains all $CØ, 192 decimal pages

plus the necessary data to restart the program that was running at time of copy. A reference of what pages are on each track of the REPLAY disk is given by the PACKER program and at the end of this section.

You have made a copy of the original program with the REPLAY card. The REPLAY disk now contains $13 tracks of data. When you used option C on the Replay card menu the REPLAY card copied memory pages $0-$C0 to the REPLAY disk. From tracks 0-$12 there are 10 pages of memory saved on each track. Track $0 stores pages $02-$0B, track 1 contains $0C-$15,.....track $12 contains pages $B6-$BF. Then on track $13 REPLAY put the necessary data to restart the copied program, and pages 0,1.

WHY WE WANT TO PACK

The REPLAY disk can't be booted as a DOS 3.3 disk. It can be booted by the REPLAY card. Although the bootup is very fast ,10 seconds, it requires a dedicated disk. The PACKER program is used to pack the REPLAY disk to a binary file and save it to a DOS 3.3 disk. This binary file can then be executed from normal DOS, and multiple binary files can be put on the same disk.

The program is called PACKER for a definite reason, it takes a larger program and 'packs' it into a smaller program. The largest binary file you can read/write to a DOS disk is $92 (146 decimal) pages long, $7F with normal DOS but $92 with a patch to DOS . There are $C0 pages stored on a REPLAY disk. Obviously you can't store the REPLAY disk as one big binary file. What must be done is to decide what parts of the original $0-$C0 memory pages , stored on the REPLAY disk, to save. You will load in parts of the copied memory into a buffer and look at it with some utilities. When you find a part of memory you want to save you add it to the binary file (store) you are building. There are utilities to aid you with selection and packing. Since the largest file you can store is 146 pages long and the Replay copy contains 192 pages you will need to eliminate 46 pages.

> EXAMPLE: Say you have a program on a REPLAY quick load disk. Let's say also you know the program has parts at $800-$1200, $6000-$8000, and $B000-$BF00. Then there are three modules; one module for each memory section mentioned above. You could say why not make one module from $800 to $BF00. Well the longest binary file you may save/read is $7A00 long ($9200 with language card PACKER). The above file is $B000 bytes long and cannot be loaded by normal DOS, therfore we must pack the modules and make the total length smaller.

The binary file you build will contain a routine (put in automatically by PACKER) to relocate memory modules to their correct running position and restart the program copied by the REPLAY card.(A memory module is a section of

memory you chose to include to the file, it is 1 or more contiguous pages)

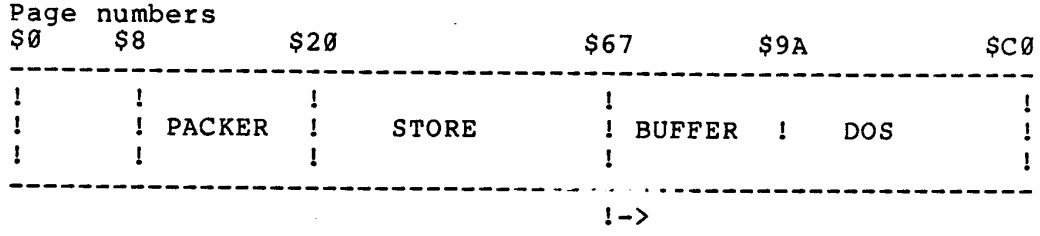
At execution time the relocate routine (RERUN) places these packed modules in their correct locations. This relocate program is called RERUN. It not only puts the modules back to their proper position but reloads processor registers and restarts the program copied by the REPLAY card.

The PACKER program transfers memory modules from the buffer to a another area of memory called the STORE. When all packing is completed this STORE will be save to a DOS 3.3 disk as a binary file. This is the file that you will execute instead of the REPLAY quick load copy or the original disk.

RERUN - a small relocation and restart program that operates at run time.

STORE - the place in memory where the binary file is built during packing.

Below is a map of memory while running the PACKER program.



The PACKER program resides in memory from page $Ø8 through page $1F. At page $2Ø the STORE begins and builds up. The buffer starts at page $67 but as the STORE increases in size the buffer will draw away and shrink, this is automatic. The buffer cannot overwrite DOS as DOS is needed to write the binary file to disk once the file is built. The buffer is used to read in parts of the REPLAY quick load copy disk for analysis. The user decides whether to include part of the buffer into the STORE or not. The section added will be a new memory module.

Another version of Packer (recommended for use) is a language card version. This version has the Packer program loading into the language card thus giving more room in lower memory.

This completes the Basic theory. In review we know the REPLAY disk contains all memory $Ø-$CØ. The PACKER program is used to select parts of the REPLAY disk to be included in

the binary file(STORE). These selected parts are called modules. The binary file is made up of these modules and a program called RERUN. RERUN puts the modules back to their correct place and restarts the program copied by the REPLAY card. The next sections will expand greatly on performing these actions.

OVERALL PROCESS

Starting new; read in part of the Replay quick load copy into the Packer buffer. Examine the buffer for code and ASCII, repeat until all memory has been examined. Option 7 for full mark is helpful. When done the user has a list of pages that the program uses. These are the pages that must be copied (packed) into the file we will create. We put the pages into the file in modules.

A module is a section of memory, contained in the buffer, consisting of 1 or more contiguous memory pages. For example if we decided the program we want to copy uses pages $20-$24 then we would read in the track(s) containing those pages into the Packer buffer, using option 2. Once in the buffer we add them to the file we are building. This is a 'module'.

When finished (all memory on Replay copy examined, and the memory needed added as modules) save the file we created to a DOS3.3 disk. The file save to the DOS3.3 disk is a binary file that you can BRUN. Note the starting address of the binary file when you save it.

Packer Program Operation

We supply two versions of the Packer program. One operates in lower memory without the language card. The other operates with the language card. We suggest using the language card version as it can pack larger files (up to $92 versus $7A for lower memory). Also not all functions described here are available in the lower memory version.

With the PACKER program running the following menu will appear.

```
****************************************

        BUFFER CONTENTS
PHYSICAL $67  $6C
LOGICAL  $0C  $11

PACKER MENU
                     FIRST TRACK          >01
OPTIONS:             NUMBER OF TRACKS     >01

1. INITIALIZE        PHYSICAL BUFFER START>67
2. READ TRACKS       LOGICAL  BUFFER START>0C
3. DISSASSEMBLE      PHYSICAL BUFFER END  >7B
4. ASCII DISPLAY
5. ASCII MARK        PHYSICAL STORE END   >20
6. CODE  MARK        LOGICAL  STORE END   >20
7. FULL  MARK
8. ADD TO STORE
9. PAGES STORED
A. SAVE BINARY FILE
B. EXIT TO BASIC
C. RUN COM FILE
E. EXECUTE CURRENT STORE

(ESC) CATALOG DOS 3.3 DISK
>

****************************************
```

First let us look at the upper right part. This is the status area and will give information needed on the current state of the BUFFER and STORE.

FIRST TRACK

This is the first track stored in the buffer. On the REPLAY disk there are $13 tracks of stored data. During packing you will read in several of these tracks to the buffer. FIRST TRACK tells you which track starts the buffer.

NUMBER OF TRACKS

The number of tracks contained in the buffer is

displayed with this label.

PHYSICAL BUFFER START

This is the page number of the physical start of the buffer in memory. During packing this number will change as the STORE increases in size. This is where the memory data from the REPLAY quick load disk is stored for your analysis and selection.

LOGICAL BUFFER START

Let's digress to introduce two new concepts. They are physical and logical addresses. Data in the buffer has a physical address in memory when you run the PACKER program, the physical address is the data's address at that instant. But the buffer contains data read from the REPLAY quick load disk. That data from the REPLAY disk has an address called a logical address, the address the data occupied when it was captured by our REPLAY card.

> EXAMPLE: PACKER is now being run and the buffer start is $6700. When data is read into the computer from the REPLAY disk it will be stored starting at $6700. Then the data's physical address is $6700. Now say we read in track 1 from a REPLAY disk. In appendix B we see that track 1 contains data copied from $0C00-up. The data was at $0C00 during original program execution. The logical buffer start is then $0C00. That is where it was copied from at execution time. That is where the data should actually go. We cant put it there because it interferes with the PACKER program and the STORE, so we put it in the buffer.

PHYSICAL BUFFER END

This is the position in memory where the buffer ends. Beyond that point there is no more data from the Replay quick load disk. So in between PHYSICAL BUFFER START and PHYSICAL BUFFER END is the data read in from the REPLAY disk.

PHYSICAL STORE END

The PACKER program builds a binary file from the REPLAY disk with your help. The binary file in memory is called the STORE. To give you an idea how large a binary file you have built so far the physical STORE end is given. The STORE always begins at $2000($0800 for language card version).

LOGICAL STORE END

The binary file you are building will execute somewhere

in memory. PACKER allows you to build it with a starting address anywhere between $800 and $A000. The file is built from $2000 ($0800 for 1c version) up but it is modified to be able to run in the above range. You will need to specify the starting address when you 'initialize' the STORE. More on this later. With LOGICAL FILE END you can monitor where your binary file ends, you don't want to overwrite DOS at execution time. If when your program (packed binary file) is loading in it overwrites DOS then the system will crash.

BUFFER CONTENTS

Across the top of the screen are two lines. One line is labelled PHYSICAL and the other is LOGICAL. This will give the user a translation from physical address to logical address. When the buffer contains data the top of the screen will tell the user the physical address of the data for dissasembly and analysis. It will also tell the you the data's logical address.

Looking at our example menu we can see from the status area on the right the FIRST TRACK is 1, the number of tracks is 1, the buffer starts at $6700 and ends at $7B00. The LOGICAL BUFFER START is $0C00. The more you use these the easier will become reading them.

PACKER MENU

9.1 INITIALIZE STORE

In INITIALIZE you will prepare the groundwork for building the BINARY FILE. Thinking in reverse for a moment, if you had the BINARY FILE finished and on a disk the computer would need to know where you wanted it loaded. We will call this the DESTination address, or DEST.

INITIALIZE needs to know this before the file is built as it is a very basic piece of information. This address is different from the address where you will build the file. The address where you build the file is the Physical address (where it is built at this moment) while the address DESTination would be the Logical address (where it will execute at run time).

The other thing INITIALIZE needs to know is the placement of the restart program RERUN, but we will postpone that discussion for a moment.

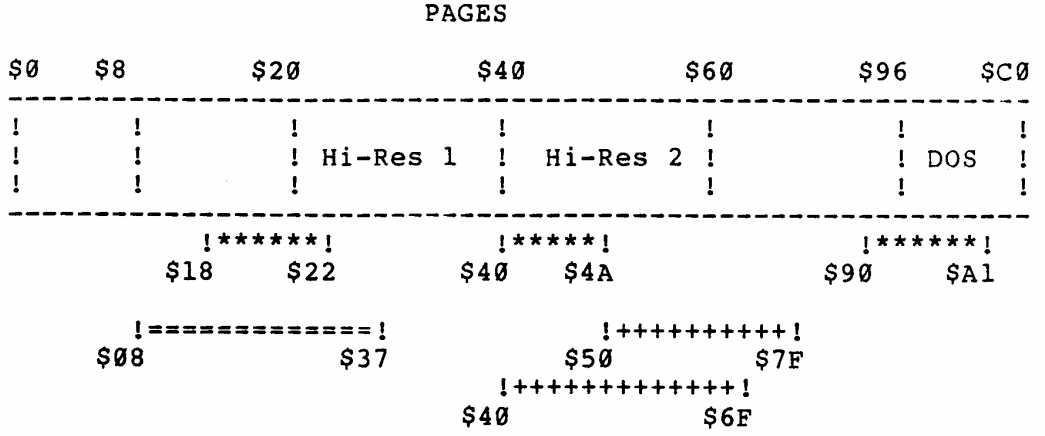
How to Choose DESTination

1. It can be between $0800-$A000.
   The lower 8 pages of memory are always included for you, they are used extensively. It cannont be above $A000 as the largest binary file is $92 hex long.
2. Choose a lower value, $08 (page 8) is a good point.

Selection logic for DESTination:

You are going to build a binary file made up of memory modules. RERUN will relocate these modules to their correct addresses and then RERUN will start the program that we packed. The memory modules are sections of memory you chose to include in the STORE(binary file). The best way to explain this is with an example.

The memory map below shows the APPLE memory with a program marked in 3 sections of stars, called modules.



The total length of the three starred areas is $28 pages. Don't forget about the lower 8 pages that are included automatically. With their inclusion the length of the binary file is now $30. What is needed is the ability to take their compacted length of $30 and fit it into memory somewhere. There are several criteria for this placement.

1. First of course is the limit set by the program of starting between $08 and $A0.
2. Second is to avoid causing the program to write over itself.

As an example three alternative packing locations are given, more could be suggested. The two locations marked with plus signs are OK. You could pack the binary file between $50 and $7F. This is best since this spot does not interfere as RERUN takes the packed file apart and places it back in the starred (*) areas. Since the $50 area is not overlapping any stars there is no chance for a conflict. Another alternative would be to pack it between $40 and $6F. Even though it resides where one of the modules goes if you check the sequence of placement you will see there will be no conflict. By the time the module going to $40-$4A is placed the binary file between $40-$6F has allready been moved to $18-$22, so this area ($40-$4A) can be written into. i.e. the lower 8 pages have been moved out and the range of $18-$22 has been moved.

The last alternative is a bad selection. This attempts to put the binary file starting at $0800. This leads to

trouble. The first 8 pages are relocated without any problems. But now we wish to move the first set of stars out to their proper location at $1800 to $2200. When we do this we conflict with a portion of the binary file we have not moved out yet. An error will occurr!!. The module's destination is $1800. In other words if RERUN puts the module out to its correct location it will overwrite the end of our packed binary file.

The map below should help visualize the movements for the bad selection.

BINARY FILE

```
    !    $0-$7   !  $18-$22 !   $40-$4A  !   $90-A1    !
Logical
    !=============================================!
   $08          $10         $1B           $26            $37
Physical
```

One benefit that PACKER provides is error checking for overwrites. When you try to add a module to the STORE the program PACKER will check for conflicts in overwritting. Also at the time of saving to a DOS 3.3 disk the file is checked again. This second check will become obvious later.

So for this example the input to the first question of starting address could be $50. The program wants only the page number of the starting location. You cannot start the binary file in the middle of a page. Later in the text we will refer to DESTination, or DEST for short.

The second question asked in this initialization is placement of the restart program RERUN. All this time we have been talking about moving the memory modules around. The RERUN program is in control of this moving. It is a one page restart program built into your binary file. It will be modified to fit wherever you want it to go. As with the first question there are limits and placement logic. The limits for RERUN placement are:

1. It must be after the DESTination
2. not more than $7A ($90 for lc version) pages after DESTination
3. it cannot be in the first 8 pages

The logic for selecting RERUN's location is not too hard. RERUN must be able to run until all of the memory modules are relocated to their correct locations. It is always in control and therefore can never be overwritten. Pick a single page of memory that is not used by your copied program and put RERUN there. Remember the aforementioned limits. There are utilities and text to help you decide where to put the 1 page of RERUN program. See option 6 code search in the documentation.

In the above example a good choice could be $70. Enter the page number only. This entry in later text will be

referred to as REDEST. This page was not used by the program we copied. Ah.. you say it is in the middle of one of the memory modules. Not so.. the values of DEST and REDEST are set before anything else. They define the building points of the binary file, the STORE. Later when you begin to add memory modules the PACKER program will automatically split modules to fit snugly around the RERUN program. In other words if you want to add a module to your STORE, and it will try to overwrite RERUN the Packer program will split the module and make two modules with the RERUN program between them. This is automatic and will be taken care of for you.

After the two questions are answered the disk drive will spin shortly. This is Packer loading the first 8 pages to the STORE and loading the restart data into RERUN.

Next you pick the screen that is to be shown at execution time. It could be hires 1,2 text 1,2 etc... You will be asked four questions on screen control. Packer needs to know what screen to build into RERUN. This is the screen to show when the program restarts. There are 4 questions with two options for each. Option 1 is default, if you hit any key but 2 for the alternate, option 1 will be chosen. They are self explaining.

9.2 READ TRACKS TO BUFFER

You will be asked to enter the range of tracks you wish to load into the buffer to work with. Please note one added feature. Just above the first query of 'START TRACK' is a status line of 'TRACKS /LOAD '. This is the maximum number of tracks you may load.

9.3 DISASSEMBLE CORE

This option allows you to use the APPLE disassembler on any area of memory. This will be one of your major tools to determine what parts of the REPLAY quick load disk to include in your STORE.

You could read in several tracks from the Replay copy of the copied program. Then use this option to disassemble the buffer area. The limits of the buffer and the logical memory area are all given in the status area.

> EXAMPLE: You have made a copy of a program. Now you wish to pack it. Using option 2 you read in a section of memory. Now you can use option 3 to disasemble the buffer. On entry to this option the status areas will be displayed and the computer will ask you for the starting address ,physical address, to begin decoding. Enter this as a full address, not as a page number. If you have a specific range of memory you wish to decode you can use the translation on the top of the screen to calculate the physical address.

The computer , through calls to the mini disassembler in rom, will show you 20 lines of code. At that time it will halt and wait for a keypress. If Return is hit the computer will exit to the MAIN menu. If anything else is pressed the next 20 lines of code will be shown. This will continue to loop until a Return is hit.

The benefit is the ability to look at sections of the REPLAY disk and see if a program is there. The disassembler will list the address, the instruction codes and the assembly mnemonics. For the novice at assembly language appendix C will help you in spotting possible program areas.

9.4 DISPLAY MEMORY AS ASCII

The program you copied will not only have instruction codes, the machine code that actually 'runs' the program, it will also have some parts of memory set aside for ASCII storage. ASCII is a term used to represent the codes of letters,numbers,symbols...etc. Every letter, number and symbol..... all are represented by a unique hex number. Programs use these codes to print something on the screen. The Packer program has several pages of nothing but ASCII codes.

An example would be the MAIN menu. That menu is put on the screen by a driving program. The driving program has a certain place in memory where a copy of the menu is stored in ASCII form. Most programs use areas of storage for ASCII. You will need to look for these also and include them in your STORE. You couldn't run the program if no menu or prompts were put on the screen. Also the program will expect the ASCII to be there, if not then the program could blowup or go off into never never land.

On entry to this section the usual status will be displayed. Then the program will prompt you for the starting address to begin showing memory. Enter a complete address, not a page number.

The program will then begin displaying memory as if it was ASCII stored. It is easy to spot the areas you want. You know what menus, prompts etc are part of the program you copied. Look for them. You might be surprised what else you can see. The display on the screen will give you 32 characters per line with an address on the left. As in the dissasembler section if you hit Return the program goes to the MAIN menu. If any other key is pressed the next series of memory is displayed.

9.5 ASCII SEARCH AND MARK

Here is a search utility to help you. This section will search the buffer for occurrences of ASCII. The program will start with the first page of the buffer and count the number

of stored bytes with values between $A0 to $D0. This is the normal range of ASCII for letters and numbers. At the end of the page there is a count. The program compares this count to a stored value. If the count from the page it just checked is greater than the stored value it 'marks' that page as being a page to include in the STORE. All pages of the buffer are checked. At the end of the buffer all the 'marked' pages are shown to you. Say the buffer starts at $6700 and has one track in it. If you use option 5 all 10 pages of the buffer are checked. At the end the program shows you which pages in the buffer were marked.

At the first of this section we talked of the stored value. This is the value the count must be equal to or greater than to 'mark' the page. This value is set in the program but can be changed by the user. On entry to this option the program displays the current value and requests a new one. If you like the current value just hit return and the program continues with analysis. If you wish to change the value enter the new one, in hex. The program will then use this as the default value until changed.

After the buffer is checked the marked pages are shown. They are listed in the format shown below.

$67...$71 $79...$82 $90...$90

There will be from 0 to ?? pairs of numbers listed. The first number of a pair is the first page marked in a continuous series. The second number after three dots is the last page marked in a series. All numbers in between the values were marked also.

9.6 CODE SEARCH AND MARK

This section is very similiar to the previous. It is a utility to search the buffer memory for possible areas of instruction codes.

The buffer is searched page by page as in the previous section. This time the program counts the number of assembly lines it can find in a page. The APPLE has a dissasembler built into the monitor roms. One assembly line is one line on the screen that contains an opcode in hex and some assembly mnemonic describing what that opcode is. When the APPLE disassembler is called an address is passed to start the disassembly. What this section does is pass the disassembler a page of memory and counts the number of lines of assembly code it generates. For each page it gets a count. It compares this count against another stored value different from the ASCII value. If the count for the checked page is lower than the stored count the program 'marks' that page as being a page to include in the STORE. Therefore the lower the found or calculated count the more likely the page contains valid instruction codes. This is because the fewer lines generated mean more opcodes per line. If a page contains nothing but hex $FF then the dissasembler will

generate 256 decimal lines. Hex $FF is not a valid opcode.

The count for each page is displayed on the screen before checking. When you run this section you will see a block of numbers displayed on the screen. These are the counts for the pages the program is checking. At the end of the buffer all marked pages are displayed just as in the previous section.

On entry to this section the current stored value, called sensitivity, is displayed. The user may change it the same as in the previous section. Now a few words on the sensitivity. The fewer lines of assembly code the disassembler generates the more likely the page being checked is a valid page made up of instruction codes. Therefore to make the program very sensitive to selecting pages , and finding some borderline or trash code, make the sensitivity value high. To find areas of very packed good code then set the sensitivity to a small value such as $60 or $70. A good overall value is around $98. Assuming an average of two bytes per line of assembly code comes out to a sensitivity of $80. Try different values and see what results you get. See appendix G for more information.

There is another searching and marking algorithm used in this section. This is transparent to the user but is done automatically. Any page that is marked as having possible code is checked again. The next check is for 16 bit indexing. This is a popular means of moving data, doing table lookups and many other objectives. PACKER cannot spot the data or table areas easily, because the data or table can be made up of anything. One way to try and mark those pages and include them into your STORE is to look for spots where the program uses them. That is done in this part. PACKER will take any marked page with a count of less than $80 and do a search for 16 bit indexed used. Any pages indexed into by the program will be marked for you, also any pages jumped to by a JMP or JSR are marked.

** That is why strange pages may appear in the marked display. Your buffer may only contain between $02-$33 but the marked pages when shown may have other values marked. These were obtained by looking for these indexing opcodes.

This is a good place to pick a value for REDEST. Before INITIALIZING come to this option and do code search. Choose an unmarked page and disassemble it with option 3, if it looks like trash or unused then use it as REDEST.

See appendix C for more detail on assembly code searching.

9.7 FULL CHECK OF REPLAY DISK

This section will do a full search and mark on the entire REPLAY disk. All memory pages will be loaded and searched. Both the code and ASCII search will be used. On exit the program will display the marked pages the same as in options 5 and 6.

There is one important thing to stress here. Use of this option after you have started building the STORE is not recomended. The original full size buffer will be used for speed. If your STORE has a physical STORE end past $67 then you will lose that data. Remember as the STORE gets larger the buffer shrinks to draw away from the end of the STORE. This section will not use the small buffer size but use the original full buffer. If you have started building and the physical end is not at or beyond $67 then you can use this option safely.

The use of this full check can greatly help spotting the usefull areas of memory. You may obtain a hardcopy of the marked pages the same as in the previous sections by entering the printer address when requested. If a carriage return is hit the output will go to the screen.

Since this section calls the code and ASCII mark options the current sensitivities before entering are used. If you want a more optimistic marking make both sections 5 and 6 more sensitive.

9.8 ADD TO FILE

This section is used to take portions of memory in the buffer and move then to the STORE. You must have previously INITIALIZED. You have looked at the buffer with options 3-7 and now you are ready to move a portion from the buffer toiinary file, STORE, you are building.

On entry the usual status areas are displayed. The program will prompt you for the beginning physical page number to add. If you hit return the program goes back to the MAIN menu. Two things to emphasize here!! The first is you will enter the **Physical** page number, not the logical. The translation table at the top of the page will help you recognize what logical parts of memory you are adding. The second thing is to stress that you are to enter the page number only, not the full address. All additions to the STORE are in full page increments. No half or partial pages are included.

The program will prompt you for the ending physical page number. Again this is the physical, and you must use page numbers. The number you enter will be included also. So if you enter $67 first and then $6B the program will take pages $67.68.69.6A.6B and move them into the STORE as a memory module.

Before moving the pages to the STORE the program will first do an error check. The first error check done is to ensure that repositioning this module at run time by RERUN would not cause it to overwrite RERUN itself. You cannot save a module which will attempt to relocate over RERUN. Remember, you must pick a page for RERUN that will not be used by the program you copied. If you do attempt to overwrite RERUN the program will print an error message and ask for the starting address again. The program knows where RERUN will be located by the address you entered in

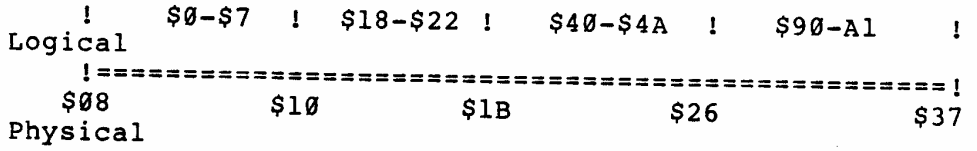
INITIALIZE for REDEST.

The next error check is for an attempt to overwrite itself. This is the same error that we covered in the initialize section. Any attempt by the module to overwrite itself or another unplaced module is not allowed. An error message is printed and you are prompted for the starting physical page again. At this point you have two options. The first is to start over again and change the location of DEST. By doing that you may be able to move the binary file around enough to stop any overwrite errors. This is not the best solution but you may have to resort to it sometimes.

The other alternative is to delay adding this section. This is going to be tricky but let's try with an example. Lets go back to the example used in INITIALIZE. Below is the map again.

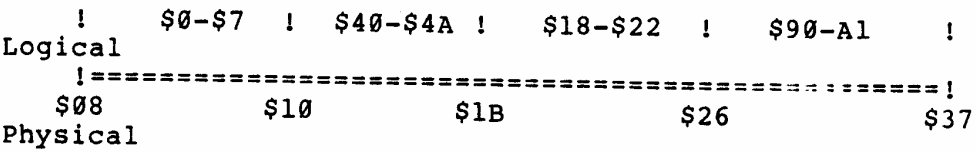
This map is a representation of our binary file (after packing) as it is loaded into memory ready to be relocated by RERUN.

BINARY FILE



This setup causes an error when the module stored at $10-$1A moves out to its correct position at $18-$22. A solution to this would be to delay including this module until later. If you added the module for $40-$4A before the $18 module then there would not be a problem. The new map shows the memory setup.

BINARY FILE



The module placement above would be correct. The modules are always moved one page at a time starting with the lowest page in a module. Thus the page at $1A physical has a logical address of $18 and can be moved down. The module below has allready been moved out so the space can be written on.

This brings up the order of placement. The modules that you add to the STORE are placed back into their correct memory locations. They are done one module at a time starting with lowest module in memory and moving up. Thus any memory below the current module can be written to if

desired. The only exception to this is when the RERUN program is below the current module. You can write anywhere but that particular page. In the initialize section we said you must pick a page not used by the copied program.

RERUN does not care in what order the modules are packed into the binary file. It will relocate them out one by one at execution time.

9.9 Show Current Pages Stored

This option will allow you to determine what pages you have allready moved into the STORE. On entry to this section the computer will ask for your printer slot number. If you want a hardcopy of the data enter the slot number the printer interface card is in. If you only want to see the data on the screen then hit return.

The usual status areas are displayed then DEST and REDEST with labels. Below this is a printout similiar to the marked pages output from code or ASCII search and mark. The computer will print several pairs of numbers. Each pair of numbers is a module. The first number is the logical start of the module. The next number, after three dots, is the last page included in the module. With this data you can keep a record of the areas of memory that were used to build the binary file. It is a good idea to use this option with the printer just before saving the binary file. That way you can keep a record of the pages used.

****

If the program doesn't fully execute try adding more pages, or use more sensitive values in the search and mark routines.

****

9.A Put File to DOS Binary

This is the final step of using this program to build a file. In this section two important things occur. The first is a final error check of the stored modules. This is the second error checking mentioned in the INITIALIZE section.

The purpose of this check is to stop any overwriting of the binary file by itself. You might think the error checks in ADD section did all of this. Well they did to the best of their capability at that time. The problem then was that the program had no means of knowing how large the binary file would become. Neither did you on the first try. The best way to explain this is with another example.

EXAMPLE: The user has initialized the STORE and added two sections. They are shown in the map below.

BINARY FILE

```
    ! $0-$8 !  $08-$28       !$50-$52!
Logical
    !================================!
   $08     $10                $30     $32
Physical
```

The user has added the $08-$28 module and the $50-$52 module. Neither of these cause an error at this time. I am assuming RERUN is placed somewhere out of harms way. The error happens later when the user adds the section from $A0-$C2. The map below now shows the status of the binary file when it tries to execute. The lower 8 page module relocates ok, the next module of $08-$28 also has no problems. The problem of overwrite occurs when the module $50-$52 attempts to relocate. It will overwrite part of the last module which has not been moved yet.

BINARY FILE

```
    ! $0-$8 ! $08-$28        !$50-$52!   $A0-$C2        !
Logical
    !=================================================!
   $08     $10                $30     $32                $54
Physical
```

This error can only be caught when the entire file has been built. When you enter option 9 the program checks for these errors. If found it will tell you which module and where the overwrite occurred. The solution is to use option 9 to get a copy of the pages you included. Then use option 1 to re-initialize the STORE.

With the knowledge of where the problem of overwrite occurred, you can move the value of DEST to try and avoid this. If you increase dest by $04 this would solve the problem. You can't decrease it anymore because it is allready at the minimum. Another solution would be to delay including that module as in the previous example. Use the same DEST but this time add the module $A0-$C2 before the module $50-$52. With so much ability to move the program start, change order of modules..etc there should be a solution to almost all packing problems.

Once the error checking has occurred, assuming no errors, the program does the final touchups on the binary file. It will display the command you need to use to execute the binary program. An example is shown below.

BRUN REPLAY,A$1000

One note is for very large files that have a LOGICAL STORE END at or near $9A00 there might be a problem. Issue the command 'MAXFILES 1' before the BRUN command and there should be no problems.

Packer will prompt you for a file name to save the binary file under.

The file is stored from the range of $2000 ($0800 for 1c version) -->$??00 . This is where the file resides in memory as it is being built. To execute the file you use the brun command, but you must use an address extension as shown in the above example. The PACKER program will give you the full command to use. If you don't want to use the address extension all the time the file can be changed. Instead of giving a BRUN command you could BLOAD the REPLAY file. In the example above you would 'BLOAD REPLAY,A$1000'. Next you would save the file again at its proper location. This way all you need to do to run the file is to type BRUN REPLAY. To save the file you need the starting address $1000 and the length. PACKER gives you the length of the file at the same time that it saves it. The length shown is the hex value. Let's say for the example above the length is $40. That is $40 pages of length.

You would use PACKER to build the file and then save it to a binary disk. Then exit PACKER and issue the command:

BLOAD REPLAY,A$1000

Then you would give the command:

BSAVE FILENAME,A$1000,L$4000.

The length of the file goes after the L$ and put two 0's after the length.

The only exception to this is if the file needs to do a master relocate. The modules will be relocated by RERUN. A master relocate is when the entire binary file must be moved. This happens when the file you built wants to load where DOS is. You use DOS to load the file so you cant overwrite that. The solution is to load the file in lower memory and then relocate up to higher memory. This is automatically done for you. If you choose DEST such that the binary file will overwrite DOS when it loads the PACKER program adjusts for this. It builds the file at $2000 ($0800 for 1c version) and modifies it to relocate to higher memory once loaded. You can tell if the binary file will do a master relocate by looking at the LOGEND value at time of saving. If LOGICAL STORE END is greater than $9A00 then the binary file you save will do a master relocate.

The exact sequence for a master relocate is as follows. When option A is hit PACKER checks LOGEND. If LOGEND is >$9A

then a flag is set in RERUN. Some parameters are set in RERUN that give the desired destination of the binary file. The binary file is always loaded in initially at $0800 (for 1c) or $2000 for lower memory PACKER. Once in memory the first 3 bytes of the binary file jump to RERUN, and the flag in RERUN is checked. If set RERUN relocates itself up into memory. It can overwrite DOS at that time because it is not needed any more. Once relocated the operation is the same, all modules are placed and program restart occurrs. All of this action is transparent to the user.

This takes you back to the main menu.

9.B Exit to Basic

This will exit the user to APPLESOFT.

9.C Run Command File

This option will pack a binary file for you if a command file is available for the program you have. A command file is a short binary program containing all the information necessary to pack a program copied by a REPLAY card. There are several command files on the disk with the packing program. They all start with C.name. For example the command file for lower memory PACKER is called C.LOWPACK.

If a user has made a copy of the packing program with the REPLAY card then he could use the command file to pack it into a binary file. This is only for example as the packing programs are given to you unprotected. The user would run the packing program and choose option C. On entry the program will ask you for the command file name. Put the disk with the command file in drive one and enter the name C.LOWPACK. The computer will load the command file and ask you for the REPLAY disk that contains a copy of low memory packer. Insert that disk in drive one and hit return. The program will then pack that disk into a binary file in memory.

On exit the program will tell you to use option A to save the binary file. Hit A for saving. The program will ask for a DOS3.3 disk and request a program name to save the file with. It will also give you the command to use to execute the stored binary program.

That is all there is to command file packing. If you wish to create new command files for other programs see section 7.0 of this manual.

9.E EXECUTE CURRENT STORE

This will execute the current store you have built in memory. The packer program will move it to where it would be loaded in by DOS and then jump to the starting location.(DEST)

It is recommended that you use option A to save the store first as using E will kill packer and your STORE. This

allows a user to build a store, save it and immediately test it without exiting to reload...etc.

10.0 PARAMETERS FOR REPLAY PACKER PROGRAM

This is the parameter list for REPLAY disks. Programs will be listed by name. Each program will have the necessary pages to copy given. Use the PACKER program to access a REPLAY copy disk and pack the given pages.

The * after a program name means supplied by alternate and not tested yet.

The ! after a program means the moudules given are in correct sequence for entering into command file create.
Some prorams will also have other information given on useful memory locations. There will be labels for some locations such as 'men' or just a memory location in parenthesis for a level or whatever.

NOTE: The following program titles are copyrighted by software companies.

Current date 05/23/83

NORAD ! $8...$1A $20...$33 $34...$65 $90...$9A
DEST=08 REDEST=1C

SARGON ! $8...$2E $2F...$2F $36...$36 $38...$3F
DEST=08 REDEST=37

MARS CARS ($800B)

REPTON MEM LOCATIONS: BOMBS=$005E SHIPS=$0061

CANYON CLIMBER MEN=$82

SWASHBUCKLER #GUYS=$9C2

BUG ATTACK ($4CA2)

HORIZON V SHIPS=$84

SNAKE BYTE SNAKES=$725E LEVEL=$7265

TUBEWAY ($A3)

SERPENTINE ($D8)

GOBBLER #GOBBLERS=$6046

NIGHT CRAWLER #SHIPS=340A

```
NEPTUNE SHIPS=$87  BOMBS=$88

CRISIS MOUNTAIN MEM LOCATIONS: MEN=$37B

A.E. ($1ABC)

CONGO  ($461A)

TORAX * $40...$60  $0F...$20

INVASION FORCE ! $8...$33  $34..$40
     DEST=08  REDEST=42

BUG ATTACK * $09...$15 $40...$84

SPACE RAIDERS $20...$84

TWERPS * $5F...$BF $1F...$3B

APPLE PANIC * $8...$20  $60...$C0

FIREBIRD ! use C.NO $20 $50

HORIZON V * $8...$20 $90...$BF

BEER RUN * $8..$20 $60...$8F $A0...$BF

DUET ! $8...$33  $34...$40 $9D...$BF  $4C...$4E
     DEST=08  REDEST=42

STARBLAZER $8...$1F $40...$9F LCMOD $A0...$BF   FUEL=$4800
BOMBS=$4980  SHIPS=$4A80

CHOPLIFTER  $8...$1E  $60...$BF

JAWBREAKER  $8..$1F  $60..$83  $8E..$92  $96..$BF

ROCKET COMMAND  $8...$40 $5E..$62

LOCKSMITH 4.1 ! $8..$1F  $80..$BF
 & NYBBLES       DEST=08  REDEST=29
   AWAY II

ECHO 1.0 ! $08...$17  $BA...$BF
   DEST=08  REDEST=18

COPY II+ V4.1 UTILITY $08...$49 $A0...$A1  $B0...$BF
   DEST=08  REDEST=4A  Display text 1 on restart

COPY II+ V4.1 BIT COPIER $08...$30
   DEST=08  REDEST=31  Display text 1 on restart

SENSIBLE SPELLER ! $8...$33 $34...$3F $60...$8D $8E...$96
     DEST=1F  REDEST=42
```

```
SCREENWRITER ! $Ø8...$1F $AA $B3..$B5 $B7 $BF $96...$9C
     $8E...$95  $3F...$6F  $7Ø..$83  $84...$8D  CLEAR HIRES
    DEST=Ø8   REDEST=2Ø  Display Text 1 on restart

SCREENWRITER RUNOFF    Same as Screenwriter execpt
    display hiresl on restart.


APPLE LINK ! $8...$33 $98...$AØ $6Ø...$8D $8E...$96
    . DEST=22  REDEST=42

MONTY PLAYS MONOPOLY  $Ø8...$3D
     DEST=Ø8  REDEST=3E  Display graphics 1 on restart

LASERSILK  ! $8...$33 $34...$3F  $89...$9F  $6Ø...$6F
             $7Ø...$83  $84...$88
     DEST=8  REDEST=42

CANNONBALL BLITZ ! $8...$1F $98...$BF  $51...$79  $7A...$83
                   $84...$8D  $8E...$96
     DEST=8  REDEST=2Ø    ($6F12)

SEAFOX * ! $Ø8...$1F $96...$BF $8E...$94 $59...$5F $6Ø...$8D
     DEST=8  REDEST=95   #subs $6D69

ULTRA CHECKERS V2.Ø  $Ø8...$33 $34 $3F
     DEST=Ø8 REDEST=4Ø   Display hires 1 on restart

SCANNER V1.6 * $Ø8...$28

DISK ORGANIZER V2.6 * $Ø8...$24 $3Ø...$31 $37...$3A
$8Ø...$8F
     DEST=1Ø REDEST=25

BACK IT UP II+ V2.4 $9F...$BF $8Ø...$8F
     DEST=2Ø REDEST=28

SNOGGLE  ! USE C.NO $2Ø $5Ø

APPLE STELLAR INVADERS $Ø8...$ØF $98...$BF  $41...$98
     DEST=Ø8 REDEST=1Ø

MARS CARS  Use C.no $2Ø-$5Ø

RASTER BLASTER * $ØA..$2Ø  $4Ø...$8BØØ

APPLE OIDS *  $Ø8...$1F  $4Ø...$8Ø

VISICALC ! $Ø8...$33 $34...$65 $66...$79  $7A...$7F
     DEST=Ø8  REDEST=8Ø

STAR CRUISER * $Ø8...$1F  $4Ø...$8Ø

VISIDEX * $Ø8...$1F  $97...$CØ  $6Ø..$61
```

```
MAGIC WINDOW  * $Ø8...$48  $96...$CØ

THE ELIMINATOR ! $9Ø...$AF  $Ø8...$1F  $4Ø...$6F  $7Ø...$83
      $84...$86    DEST=Ø8  REDEST=2Ø

CROSSFIRE ! $Ø8...$33  $34...$65  $66...$79  $7A...$7F
      DEST=Ø8 REDEST=88

SNACK ATTACK ! $Ø8...$1F  $9A...$9A  $29...$51  $52...$83
      $84...$8D  $8E...$97  DEST=Ø8  REDEST=2Ø   FISH=$EA

TAXMAN LEVEL=$55
```

Thanks to Replay users for some of the parms and mem locations.

Pirates Bay (modem 415-775-2384) also contributed many

Zaxxon	Ships	$BF2B
Thief	men	$31A

11.Ø TIPS ON COPYING AND PACKING BINARY FILES

The REPLAY card will allow you to copy many different programs for backup, quick restart, and packing to binary files. There are a couple of tips for operation of the card. We will go over some of these here.

Remember that the REPLAY card will copy total load programs only. This does not hinder you from copying programs like the word processors. You could copy the editor program which will let you build and edit files. You could then copy the printer program which will access your text file for printing. Most word processors store your files in text files on normal DOS 3.3 disks. Keep examples like this in mind.

There are several coniderations to keep in mind when packing a binary file. First is the REPLAY copy itself. You want to make the REPLAY copy at the best time for later restart.

The largest binary file you can load in from a disk is about $9A hex pages long. This is because you can only load into memory not taken up by DOS. To pack programs that are extremely long you may not be able to pack the HIRES page in with the code and data. This means that when the program starts the HIRES page is blank or full of garbage. A nice fix for this is to copy the original program in the act of 'refreshing' or 'redrawing' the HIRES page. Most all programs at some time or another will erase and redraw the HIRES page. Each program is different but find the sequence or command that will force this action. Then right as you issue the command hit the switch for copy. Timing is tricky, it might take a couple of tries.

The benefit here is that when you pack the binary file you don't need to include the HIRES page. It will be redrawn for you by the program when it restarts.

Another idea involves the ram card and its use. When you make a copy of a program with the Replay card the quick load copy is a copy of the lower 48k ram and the data necessary to restart the copied program. Some programs advertise or write in their documentation that they don't use 64k, they only use 48k. Well....take that with a grain of salt. Many programs only use 48k for their program code but if they see a ram card or in the case of the //e they see the upper 16k ram they will use it. The reason is for protection. Since the Replay card copies 48k initially they can tell if a copy has been made by writing some byte sequence in the ram card and checking it every so often. If a 48k copy is being run the bytes in the ram card will be changed.

There are several solutions to this problem.

1. remove the ram card check (hard but best)

2. copy without ram card in system. If they don't see a ramcard they don't use that protection system.
3. copy all 64k. Troublesome for size of file. wasted space.

To remove the protection is the best of course. Some of the programs we have looked at can be modified by deleting the reference to address $C080 which turns on the ram card.

Copying the program without the ram card is an easy solution except if they do a status check of the system. Some programs will take an inventory of the Apple when they boot in. If that inventory changes, i.e. a new card in a slot or one missing, then they crash.

12.0 PACKING EXAMPLES

Here is an example for packing a binary file. We will go through the steps for packing the program LOCKSMITH. There is a command file for this program but we will go through the entire sequence of making a copy, analyzing code, packing a binary copy, and creating a command file.

The first thing you would do is to make a copy with the REPLAY card. This gives you a copy in REPLAY format. You could execute this copy with the REPLAY card but you would like to put it into a binary file for easier use and storage. You could use DOSMAKER on the utility disk but you want to make a copy that does not require the language card. Boot the PACKER program and put in the copy disk. Now for a little thinking.

This program is a bit copier. It has large sections of code for analysis and operation. It also has large buffers for reading in data from disk for analysis and storage back out. The buffers do not need to be copied. They are not necessary. Programs that have buffers are bit copiers, communications programs (for modems), word processors, spelling correctors....etc. You do not need to copy the buffers of these programs, only the code. Other areas that need not be copied are the hires pages of APPLE MEMORY for programs that use the hires pages such as plotters, games etc.... As mentioned above if the user copies the program at the correct time the hires pages need not be copied. If you did not copy a program at the proper time then the hires pages need to be copied because the program is executing on restart and the hires page is full of garbage. The hires pages take up a large block of memory and if you copy them your binary file is growing rapidly in length. If the program you copied has a large code and data segment you may not be able to fit in everything.

For now we want to find all the code areas of the LOCKSMITH program. The first thing to do when starting analysis on a program is to use option 7. This will access all the program stored on disk and do a code and ASCII search and mark. For more description of this option see the

documentation on PACKER. When this option completes the computer will show you a list of all page it considers valid program or data. It is recommended to enter a printer slot number and obtain a hardcopy of this list. With this list you will proceed to analyze the memory and select the pages to store to a binary file. The PACKER program shows code from Ø-$2Ø and $8Ø-$CØ. The printout is shown below after running opion 7 for full mark.

```
POSSIBLE PAGES:
START...END
ØØ...Ø2 Ø4...2Ø 8Ø...A9 AE...B6 CØ...CØ F9...F9 FB...FF
```

This is not readily evident from the list output but let's look at it. There are a couple of broken blocks above $8Ø but a large amount of code is shown. The same can be said about the section between $Ø-$2Ø. The broken sections are simply code pages with a little different number of assembly lines than other pages or small unused areas or small buffers for that programs use. The means of finding this out is to use options 2 and 3. Option 2 will load any section of memory you desire into the buffer. Option 3 will dissasemble any part of memory. Use this to analyze the Replay buffer containing parts f te copied program. Note the memory marked above $BFØØ. The Replay copy does not contain this memory but is shown marked because Locksmith uses that part of memory. The $CØØØ is for keyboard i/o and the $F8ØØ ... is for monitor rouines. Your don't have to worry about copying these sections as all APPLES have them built in.

Now let's look at the marked memory. Load a section of memory of your Locksmith copy. To do this use option 2 of the packer program and load tracks $C to $1Ø. This corresponds to the memory from $7A to $AB. Now use option 3 to look at memory starting at logical $7FØØ. To do this you need to know where the copied memory is stored. You have just loaded tracks $C to $1Ø into the Packer program buffer. The logical pages loaded and their current physical locations are given across the top of the screen. For this case the logical page $7A is at physical page $67. Logical page $7F is at physical page $6C. So to look at (dissasemble) logical page $7F we would type 3 (to disassemble) then enter $6CØØ for the address to start at. The memory will be displayed with mnemonics. Hitting a space bar will continue. Hitting return will exit. If your continue on to where $6DØØ starts to be displayed (logical $8ØØØ) you will see that as logical $8ØØØ is displayed valid program code is shown. From there up to $BFØØ needs to be copied. The same can be done for lower memory.

The hardest part of a program to find and include is the data areas. There is no valid code in these areas and to dissasemble them will do little good. The PACKER program will try to detect these areas by analyzing the valid program code found. The data areas will be accessed by the

program code. When PACKER finds a valid program code area it looks for any 16 bit indexed addressing and marks any pages found (this addressing mode is used to access data). There are other addressing modes that access data and the largest used is called zero page indexed. The PACKER program cannot easily spot this type, read an assembly manual and use option 3 for this. PACKER will also mark any pages accessed by a JSR or JMP, subroutine call or goto statement. Almost all code and data areas are found. These pages are marked. This indexed search is where stray pages will show up in the marked display.

The best procedure to take is to obtain a working packed file and then if desired try to reduce it further by taking out the pages not needed. Therefore pack all the pages you can that are marked. If there are a couple of skipped pages between 20 or 30 marked pages include them also.

The program COMPARE can be used to look for code area, see the documentation on COMPARE. Also the Replay card can be used. Make a copy of the program, then using the W command of the monitor clear selected areas of the memory. Restart the program. If the program restarts and runs completely then that area was not crucial to program operation. If the program crashes you can re-execute the copied program. Since it takes under 10 seconds to restart you can iterate very quickly.

For LOCKSMITH the hard code areas were packed into a binary file. This file was executed and worked. Options 2 and 3 were used to look at the memory and a decision was made to pack $0-$1F and $80-$BF. The procedure for packing is below.

In PACKER use option 1 to initialize the STORE. Set DEST to $08, this value gives us the most room to pack a file. Set REDEST to $21, looking at te full mark listing we see that page 21 is not used ,marked, by Packer. Page $21 does not need to be copied so put the RERUN program there. For the screen setup use text, page 1, all text. Do not clear the hires screen on startup. The program will return to the menu. That finishes initialize.

Now we are ready to start packing the sections of memory we decided to include. For our example we said we wanted $0-$20 and $80-$BF.

Use option 2 to read in tracks 0-4. Then use option 8 to add to the store. Hit 8 and then $6D for start and $85 for end physical addresses. This corresponds to adding the section $08-$20. We don't include $0-$07 as that is always done for your, they are important and are packed automatically. Now hit option 2 and read tracks $0C-$10. Use option 8 again. Enter $6D for start and $98 for end physical addresses. This corresponds to adding the section $8D to $AB. Notice we cannot make one large module of $80-$BF. We cannot load that large a section into the Packer buffer all at once. So we make two smaller loads, making it into two modules. To load the rest hit option 2 and read in tracks $11 and $12. Use option 8 and enter $67 for start and $7A

for ending physical addresses. You have just added the $AC-$BF section. That completes the packing!

Some short comments on modules and placement. For the example we just did there was no problem with module placement. This problem occurrs with programs that are very large or very segmented. The problem is to place he modules in sequence such that they don't overwrite one another when they are being loaded to there correct position in memory. The Packer program can detect these types of errors and will inform you of them. The solution is to change the order of packing a module and to create different size modules. When we packed Locksmith we packed from lower memory up. We could just as well pack from higher memory down. The order of module placement is only important for eliminating problems with placement. This is covered in more detail in the Packer program.

Now that we have a packed file use options 9 to list the packing and parameters. The output would look like:

```
RUN DESTINATION......$08
RERUN DESTINATION....$21

CURRENT PAGES IN STORE.
START...END
$00...$07 $08...$18 $19...$20 $80...$AB $AC...$BF
```

To end the session use option A to save the packed file to disk. PACKER will prompt for a name to store the file under. Also given is the command to execute this stored file. ***NOTE THIS COMMAND. YOU MUST USE THE ADDRESS EXTENSION.*** The command will be :

```
BRUN 'NAME',A$XXXX
```

Where the XXXX is the address to start loading in the binary file.

After you have saved your file you could type E to execute the current STORE. This will test your packing. See option E in PACKER documentation.

Test the file out by execution. If it works you are ready to build a command file. Run COMMAND FILE CREATE.

The first question asked is for the name of the command file. To identify these it is recommended to use a C. prefix. It is not necessary though.

Then the program proceeds to ask you the questions needed to build the command file. All that is needed can come off a hardcopy listing if you use option 9 in PACKER once you have packed the file. Option 9 shows the values of DEST, REDEST and all the modules in proper order of inclusion.

The create program asks for these values. The only other questions are for screen setup. Remember text page 1

was used. Once you have entered these parameters insert a DOS 3.3 disk to store the command file to.

One note....when entering the modules exclude the lowest module of $Ø-$Ø8. This is always entered by the program.

Now you have a command file to pack the program at any time with minimal user action. See section 14.Ø for command file create.

13.0 COMPARE

This program is used for analysis of machine language programs. It requires two disk drives, each drive will contain one copy of a Replay disk. This is the fast load format disk created by the Replay card. The disk in drive 1 is considered the primary copy. The program has two modes of analysis.

13.1

The first mode will compare two Replay copies of a program. This is useful for locating areas of program code and defining the areas of memory usage by the copied program. If the two memory copies compare the same in an area then it is likely that area is used by the copied program.

The area of memory compared between the copies is $0000-$BFFF. For each page of memory the number of differences between the copies is counted and stored. The user may see a listing of these counts on the screen or have it sent to the printer. To operate this analysis run the program COMPARE. When the menu appears put the 2 Replay copies into the disk drives. Select option 1 to compare the disks. When the program is finished the menu will come back. Now select option 3 to display the counts. You may send the output to the printer by entering the printer slot when prompted, or hit return for the output to go to the screen.

The output will consist of 12 lines of 16 bytes each. The page numbers are listed on the left. The count for each page is listed.

For maximum use the two copies should be seperate. Don't make both copies from one Run time. This will randomize the memory contents somewhat so that only the areas of memory used by the program have zero counts. A zero count indicates that page was the same for each copy.

13.2

The second mode of analysis is for page by page comparison. This mode, option 4 on the COMPARE program menu, will ask the user for a page to compare between the copies. Have the Replay copies in drives 1 and 2. The two copies are compared for the requested page in a byte by byte sequence. Any differences are marked by the program.

The computer result is a hex display of the requested page. The primary copy page is displayed. If there is a difference between the primary copy and the secondary copy there will be a star before the hex number is displayed. An example is given below of the output:

```
$0C> 05 FA 3E*20 31 25*FF*EF....12 BYTES...
```

The display will show all 256 ($FF) bytes of the page.

They are shown 12 bytes to a line. The example above is for memory starting at relative byte $ØC in a page. Therefore $xxØC contains $Ø5, $xxØD contains $FA....etc. The value at $xxØF is $2Ø on the primary copy but is different on the secondary copy. If the user wants to see what that value is he could swap disks and perform the analysis for this page again. Remember the primary copy (the disk in drivel) is the page displayed.

For other uses of this program see the section on 'EXTRA USES FOR REPLAY'.

14.Ø COMMAND FILE CREATE

There is a program called command file create on your program disk supplied with the REPLAY card. This program will create a command file to be used by the PACKING programs. This file contains all the necessary data for the PACKER program to access a REPLAY copy disk and pack it into a binary file. Each copied program is unique and needs a command file.

When you run command file create the program will prompt you for the needed data. It will then store the command file on a DOS3.3 disk.

The first question asked is for the name of the command file. This is the name that it will be stored under on a DOS disk. Next it will ask you for DEST. This is the beginning page address of the binary file you will pack. Enter a two digit hex number for the page number where the binary file will begin.

Next the program requests REDEST. This is the address of the RERUN miniprogram in your binary file. Enter a two digit hex number representing the page where RERUN will go.

For more information on DEST and REDEST see option 1 on PACKER program.

Now the program requests which screen setup to build into the program. The next question asks how many modules there will be. This is the number of memory modules ***excluding*** the lower 8 pages memory module. That module is always present.

The program then goes into a loop depending on the above entered number of modules. It requests the LOGICAL starting and ending page address of each module. Enter the addresses as two digit hex numbers.

You would want to use this program command file create when you have successfuly packed a binary file and know what pages to pack. A good idea is to use option 9 in PACKER once you have a file packed. This option gives you DEST, REDEST, and all modules you packed, and it gives the modules in the

correct order.

If you are building a command file from a parameter list you will need to pay close attention to the sequence of modules. Option 9 will give the correct sequence of modules to load. The parametere lists may only list areas of memory and not a good sequence of modules.

For instance the parameter list of a program that has 0..1A $20..65 $90..9A cannot be entered directly into a command file. You must use packer to find a workable sequence of modules. In this case.....

8..1A 20..33 34..65 90..9A

The Packer buffer can only hold so much at one time. Some parameter lists may be in a good form for entry into command file create. Look for comments on the parameter lists.

Once the last module has been entered the program will prompt for a command file disk and save the file.

When running PACKER this command file can be used to pack any copy of that particular program.

15.0 SOFTMOVE

This is a utility program on the Replay utility disk. With this program you can take an APPLESOFT program under a protected DOS and move it to standard DOS 3.3. This does not mean that program will still run as there may be other imbedded protection mechanisms in the APPLESOFT program itself.

To use this program execute it from the options menu of the Replay utility disk or from a prompt with the utility disk in drive one type:

BRUN SOFTMOVE

It will ask you for the Replay quick load copy disk that contains the copy of the APPLESOFT program to move to standard DOS. When the transfer is done the utility SOFTMOVE will exit to basic. The APPLESOFT program is in memory and can be saved with the command :

SAVE FILENAME

APPENDICES

APPENDIX A
EXTRA USES FOR THE REPLAY CARD

Besides being a program copy system the REPLAY card can also be used for program analysis. The ability to halt program execution at any time and have all the restart data saved for you is a nice utility.

If you wonder how a program works or where the microprocessor is currently executing code at any time you can use the REPLAY card. Simply boot the program you wish to analyze, let it proceed to the point you are interested in and interupt at that time for copy or analysis.

Most programs on the market these days are in assembly language. There are some programs that are in APPLESOFT for sell. To find out where the APPLESOFT program is executing use the monitor command P for APPLESOFT pointers.

The zero page is used by many programs for variable storage or for pointers. Machine language programs especially use the zero page for variable storage. An extra use for the Replay card is for gameing or variable search.

For an example let us take the game REPTON. This is a game that uses starships and bombs. Usually there is a variable somewhere in memory that reflects the current number of each. Wouldn't it be nice to know where that variable is stored? You could then set any number you like using the Replay card and the monitor change commands.

To find out its location is not too much trouble. It is a machine language game and therefore its most used variable are on the zero page for fast access..etc. First make a copy of the game as it first begins. The number of ships is 5. Then using the Replay monitor type the following commands.

```
Ø:Ø5  put value 5 at location Ø

CCØØ/1    search page Ø for value 5. Remember that the
          program copy of pages Ø,1 are buffered at $CCØØ
```

When the computer finishes listing all addresses that contain 5 write them down. There won't be too many. Now restart the copy and intentionally loose one ship. Stop the program and type the following monitor commands.

```
Ø:Ø4   put value 4 at Ø

CCØØ/1 search page Ø for value 4
```

Now write down any addresses that match with the previous addresses. If there is more than one then you have two options. One is to change each location and see the effect (put FF and see if you have 256 ships) or you can

iterate one more time and search for 3 this time. You should only have to do this one more time at most. For us it fell out the second iteration. The address for ships is at $0061 and bombs is $005E. Remember to change the values at $CC61 and $CC5E as this is where the zero page is stored for restart.

The COMPARE program's ability to do comparisons between two copies is useful for this type application.

PROTECTION ANALYSIS

The example below will demonstrate analysis of a protection mechanism.

EXAMPLE:

Some programs use a set track for copy protection. This track may not contain sector data, it could contain only a set sequence of bytes from 1 to ?? that the program looks for. Bit copiers have problems copying tracks like this without help from users. Nybbles Away II has a track/bit editor for working on a track but without knowing what sequence of bytes the program expects the user may have a hard time. The REPLAY card can be used to solve this problem.

Make a copy of the disk using a bit copier. Then boot the copied disk. If the disk boots part ways but then hangs on a given track the program may be looking for a sequence of bytes and cant find them. This could be a synchronized copy also. Either way the program does not like this particular track for some reason. Well....let's find out why!!!

This time boot the copied disk and right when the program moves to the track it does not like make a copy with the REPLAY card. This may necessitate removing the cover of the disk drive to watch the disk booting a couple of times. A good calibrated eyeball helps. Watch the read/write arm and copy the program right when it stops on the track giving problems.

Now you have a copy of the program as it is reading that track. Make a note of the program counter and the first and second return addresses on the stack. To proceed from here you must run the PACKER program. If the PC value is above $F800 then the program was using a monitor routine. Then the address on the stack is probably where the program will return to. Once you have the location in lower memory where the calling routine is you want to look at that area.

Use option 2 to load the buffer with memory from around the above value. Then use option 3 to dissasemble and analyze the code in that area. Find out where the disk is being read. The code will look something like below.

```
0000 LDA $C088,X
0003 BPL $0000
```

This is the code to look at the disk input data latch. After this code is either storage of the loaded data or comparison to a set value (such as $D5). This will look something like:

```
0003 BPL $0000
0005 CMP #$D5
0007 BEQ $0020
0009 (PRINT ERROR MESSAGE AND REBOOT)
       "     "
0020 (ALL OK PROCEED)
```

Usually the code will look for a sequence of bytes so there may be several sets of the above code. If the loaded data does not match then the program will usually print an error message and reboot or crash. You have two options at this point.

One is to use the track and bit editor of NYBBLES AWAY II and make the track look like the program wants. The other is to modify the code of the program. The first is easy if not many bytes are checked, if the track contains a significant amount of data then forget it!!!

The second is more appealing as it does away with the protection scheme. To do that you need to modify the code as it resides on disk. This can get mighty tricky and time consuming. Utilities such as the INSPECTOR and Dr. WATSON come in handy here. They allow access to the disk on sector level. The procedure is sometimes not easy and may require a couple of iterations. Make the copy of the program as described above. Now what you must do is analyze the code and find out where an error is detected. A code to look for is:

```
JMP $C600
```

This reboots the disk in drive 1. So any code that executes that is error code. Some programs as mentioned above will look for a sequence of bytes. If any one of the bytes is not correct it will branch to an error code area, look for this. Also look for any code that prints 'error' or other error indicators such as a letter in the upper left corner. Find all entries to such code.

Once you think you have located all the error code and entries to them you must change that code. The procedure to try first is to put nop's ($EA) where the program branches to that code. In other words change the program so that there is no entry to the error code. You can change the program in memory but that won't do you much good. What you must do is to change the code on the copied disk. Change the diks that was copied by the bit copiers, not the disk made by REPLAY. We only used the REPLAY disk to find the code areas for analysis.

Here is the fun part, sometimes it can be quick but other times it can take some time. A printer is a definite help. What you must do is to find out where the code you have analyzed is stored on the bit copied disk. It could be on almost any track. Use logic!!! If the disk booted track zero and then checked track one for a sequence of bytes then the code to find must have come off track zero. But....if the program loaded several tracks before checking one track the code you want to find can come off any one or several of the loaded tracks.

Find that code!!! Load sectors and dissasemble, compare against the code you want to find. Nybbles Away II has a nice utility for reading a sector and then disassembling that sector in memory. When you have a match enter your changes and save the sector back to disk. Of course if they are using different address markers you may have to modify DOS some. Find a source listing of the RWTS DOS code and there are a few bytes to change that allow different address markers to be used. That goes into a much deeper realm of analysis.

So the basic procedure is as follows:

Make a copy of disk with bit copier
Run copy and find track it hangs on
Boot copy and make REPLAY copy when
the trouble track reached.
Analyze the code reading the track
Find all error code and entries to such

Options:
A) Make track fit desired with track editor
B) change code on bit copied disk to ignore all
all error conditions.

APPENDIX B
MULTI ACCESS PROGRAMS

One method to obtain a working backup copy is as follows. Make a bit copy of the disk. If that copy boots and runs then you have a backup.

If not you may still be able to have a backup. On the back of the bit copied disk put a REPLAY copy of the main or only program. Boot that copy and flip the disk when it is running. Then when it goes back to disk it goes to the bit copied disk that won't boot. The format may have been copied good enough to allow access by the main program.

If this doesn't work try finding the code that accesses the disk repeatedly. Some programs only access the master disk to check for a copy, no data is loaded. Find the code

using techniques described above and branch around it. Even if a few bytes are loaded you might want to simply modify the code around that area to reset those bytes and then continue. See the example above for removing copy protection from a disk. This requires a good knowledge of assembly language.

APPENDIX C
DOS COMPARISONS

Many programs use a patched copy of DOS 3.3 for their protection mechanism. Their verison of DOS is very close to normal but has changes to it so that normal DOS 3.3 cannot be used to access their disk.

Compare is a good program to analyze the difference between standard DOS 3.3 and the new patched DOS. First boot the disk with the non-standard DOS and make a copy with the Replay card. Now boot a standard DOS 3.3 disk such as the system master and make another copy with the Replay card.

Now use the Compare program on the two copies. The Compare program will point out the differences betwee the two versions.

A use for this analysis is for Packing. If you want to pack a program that uses a patched version of DOS then all you need to copy (for DOS, *NOT* the whole program) would be the patched areas. When the packed program is executed a standard version of DOS is in the machine. The patched areas could simply be overlayed onto the copy of DOS present now. This saves copying the entire DOS which is $28 pages long.

APPENDIX D
HEX NUMBER SYSTEM

The number system everyone uses normally is base 10. The computer uses binary numbers, which is base 2. The binary numbers can be read easier in base 16 which is hex. For an example of each see below.

```
BASE 10
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
BASE 16
 0 1 2 3 4 5 6 7 8 9 0A 0B 0C 0D 0E 0F
```

In hex you use alphabet letters to count once you get past 9.

BASE 10 decimal	BASE2 binary	BASE 16 hex
09	0000 1001	09
14	0000 1110	0E
24	0001 1000	18
32	0010 0000	20

In base 16 you count up to 15 before going to a second digit, Ø 1...E F 1Ø. In base 1Ø you count up to 9 before going to a second digit , Ø 1...8 9 1Ø.
In base 1Ø every digit position is a power of ten. Starting with 1Ø to the Ø=1.

Take the number 123.

3 times 1Ø to the Ø = 3
2 times 1Ø to the 1 =2Ø
1 times 1Ø to the 2 =1ØØ

total=123

It is the same with hex. every digit is a power of 16.

Take the number 123 in hex. Translate to decimal.

3 times 16 to the Ø = 3
2 times 16 to the 1 =32
1 times 16 to the 2 =256

total=291 decimal

In the document all numbers preceded by a $ are given in hex.

APPENDIX E
REPLAY TRACK REFERENCE

The tracks on a REPLAY disk contain the memory of the APPLE at time of copy. The reference table below will show you what pages are stored where.

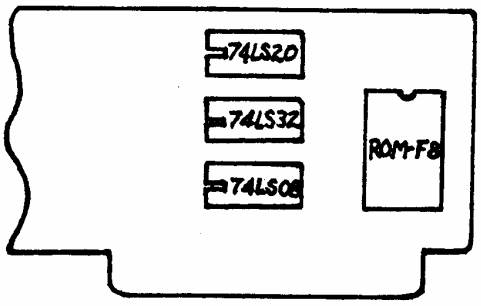
Track storage all numbers in hex.

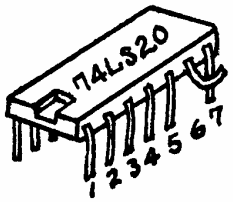
TRK	PAGES	TRK	PAGES
ØØ	Ø2-ØB	ØB	7Ø-79
Ø1	ØC-15	ØC	7A-83
Ø2	16-1F	ØD	84-8D
Ø3	2Ø-29	ØE	8E-97
Ø4	2A-33	ØF	98-21
Ø5	34-3D	1Ø	A2-AB
Ø6	3E-47	11	AC-B5
Ø7	48-51	12	B6-BF
Ø8	52-5B	13	Ø,1,RESTART
Ø9	5C-65		
ØA	66-6F		

APPENDIX F
APPLE LANGUAGE CARD OWNERS

Owners of Replay that have the APPLE language (ram) card must make a slight modification to the language card. Follow the procedure below:

1) With power off remove the ram card.
2) Locate the integrated circuit labelled 741s20. The diagram below shows its location:





3) Remove the chip from its socket carefully. Please note the direction of the notch on the chip. When you reinsert the chip later, make sure the notch on the chip is in the same direction.
4) Bend pin 6 out from the chip so that when you reinsert the chip that pin will not go into the socket.
5) Reinsert the chip and plug the ram card back into the APPLE.

If your APPLE does not have an autostart rom (rom is autostart if disk turns on when you turn power on), then you need to swap out the two $F8 roms. To do this simply unplug the rom $F8 on the mother board and exchange it with the rom on the ram board.

Your APPLE will operate exactly the same as before, execpt now it will not interfere with the Replay card.

APPENDIX G
ASSEMBLY EXAMPLES

The next few pages give examples of assembly language code listed using the APPLE disassembler. They are commented as to what to look for.

```
1000L

1000-   8D 2C 17    STA   $172C
1003-   20 3E 14    JSR   $143E
1006-   B0 E3       BCS   $0FEB
1008-   20 6C 12    JSR   $126C
100B-   AE 22 17    LDX   $1722
100E-   4C 5B 0F    JMP   $0F5B
1011-   20 58 FC    JSR   $FC58
1014-   A9 1F       LDA   #$1F
1016-   A2 22       LDX   #$22
1018-   20 FD 16    JSR   $16FD
101B-   20 E1 16    JSR   $16E1
101E-   4C 3C 0C    JMP   $0C3C
1021-   20 58 FC    JSR   $FC58
1024-   A9 67       LDA   #$67
1026-   8D 19 17    STA   $1719
1029-   A9 05       LDA   #$05
102B-   8D 16 17    STA   $1716
102E-   A9 20       LDA   #$20
1030-   8D 17 17    STA   $1717
1033-   A9 8D       LDA   #$8D
*
```

Here is an example of normal program code. Notice no ?? , every line has an assembly mnemonic on the right. 53 decimal bytes were used for 20 lines giving roughly 100 lines of code per 256 byte page. That is $64 bytes of code for $FF byte page. With the sensitivity set above $64 hex this page would be marked if the rest of the page was similiar.

```
*5000L

5000-   00          BRK
5001-   00          BRK
5002-   00          BRK
5003-   00          BRK
5004-   00          BRK
5005-   00          BRK
5006-   00          BRK
5007-   00          BRK
5008-   00          BRK
5009-   00          BRK
500A-   00          BRK
500B-   00          BRK
500C-   00          BRK
500D-   00          BRK
500E-   00          BRK
500F-   00          BRK
5010-   00          BRK
5011-   00          BRK
5012-   00          BRK
5013-   00          BRK
*
```

Example of memory all zeroes. Every line is BRK which is the assembly mnemonic for 00 hex byte. Not marked.

```
6000L

6000-   FF          ???
6001-   FF          ???
6002-   FF          ???
6003-   FF          ???
6004-   FF          ???
6005-   FF          ???
6006-   FF          ???
6007-   FF          ???
6008-   FF          ???
6009-   FF          ???
600A-   FF          ???
600B-   FF          ???
600C-   FF          ???
600D-   FF          ???
600E-   FF          ???
600F-   FF          ???
6010-   FF          ???
6011-   FF          ???
6012-   FF          ???
6013-   FF          ???
*
```

Example of memory all hex $FF. Notice the large number of ?? meaning the APPLE disassembler cannot decode this section. $FF is not a valid opcode instruction so the disassembler prints ??

```
]CALL -151

*A900L

A900-   55 CE       EOR   $CE,X
A902-   56 45       LSR   $45,X
A904-   52          ???
A905-   49 46       EOR   #$46
A907-   D9 00 21    CMP   $2100,Y
A90A-   70 A0       BVS   $A8AC
A90C-   70 A1       BVS   $A8AF
A90E-   70 A0       BVS   $A8B0
A910-   70 20       BVS   $A932
A912-   70 20       BVS   $A934
A914-   70 20       BVS   $A936
A916-   70 20       BVS   $A938
A918-   70 60       BVS   $A97A
A91A-   00          BRK
A91B-   22          ???
A91C-   06 20       ASL   $20
A91E-   74          ???
A91F-   22          ???
A920-   06 22       ASL   $22
A922-   04          ???
*
```

Not program code. Note large amount of repeating BVS instruction. Several ??? are present. If a lot of ??? are present then the section is probably not valid code. Use the ASCII search or display to see if there are menus or prompts here. Also several BRK are present then code is probably not valid.

A950L

```
A950-   02          ???
A951-   01 C0       ORA   ($C0,X)
A953-   A0 90       LDY   #$90
A955-   00          BRK
A956-   00          BRK
A957-   FE 00 01    INC   $0100,X
A95A-   00          BRK
A95B-   02          ???
A95C-   00          BRK
A95D-   01 00       ORA   ($00,X)
A95F-   07          ???
A960-   00          BRK
A961-   01 00       ORA   ($00,X)
A963-   FF          ???
A964-   7F          ???
A965-   00          BRK
A966-   00          BRK
A967-   FF          ???
A968-   7F          ???
A969-   00          BRK
*
```

Not program code. Note several ??? and many BRK

BD00L

```
BD00-   84 48       STY   $48
BD02-   85 49       STA   $49
BD04-   A0 02       LDY   #$02
BD06-   8C F8 06    STY   $06F8
BD09-   A0 04       LDY   #$04
BD0B-   8C F8 04    STY   $04F8
BD0E-   A0 01       LDY   #$01
BD10-   B1 48       LDA   ($48),Y
BD12-   AA          TAX
BD13-   A0 0F       LDY   #$0F
BD15-   D1 48       CMP   ($48),Y
BD17-   F0 1B       BEQ   $BD34
BD19-   8A          TXA
BD1A-   48          PHA
BD1B-   B1 48       LDA   ($48),Y
BD1D-   AA          TAX
BD1E-   68          PLA
BD1F-   48          PHA
BD20-   91 48       STA   ($48),Y
BD22-   BD 8E C0    LDA   $C08E,X
*
```

Good program area, 35 decimal bytes gives 20 lines of code. This comes out to 146 lines of code per 256 byte page. With sensitivity set at $92 or above this page would be marked if the rest of the page was similiar to this.

```
1393L

1393-   20 6C 8A    JSR   $8A6C
1396-   4C 44 14    JMP   $1444
1399-   20 6F 8F    JSR   $8F6F
139C-   97          ???
139D-   0F          ???
139E-   20 72 8C    JSR   $8C72
13A1-   20 DF 8A    JSR   $8ADF
13A4-   20 9B 8F    JSR   $8F9B
13A7-   F5 0A       SBC   $0A,X
13A9-   20 79 8A    JSR   $8A79
13AC-   A9 01       LDA   #$01
13AE-   8D A9 0A    STA   $0AA9
13B1-   4C 44 14    JMP   $1444
13B4-   20 6F 8F    JSR   $8F6F
13B7-   E8          INX
13BA-   0F          ???
13B9-   20 72 8C    JSR   $8C72
13BC-   20 DF 8A    JSR   $8ADF
13BF-   20 6F 8F    JSR   $8F6F
13C2-   AB          ???
*
```

Tricky example of good code. There are many ??? in here but the rest of the printout looks good. This is an example of data passing information to a subroutine. The address of the data is on the stack. The data passed is not valid instruction opcodes. The subroutine uses the data and then returns to the calling routine with an adjustment for the data.

There are 48 bytes used giving 108 lines of code per 256 byte page. Any sensitivity above $68 would mark this page.

APPENDIX H

ANALYSIS PROGRAM FOR REPLAY II CARD

There is a new program on the Replay utility disk. It is called RamStep. This program will allow machine language programs to be traced by the user. Some uses are:

Analyze program operation/flow
Locate areas of program by tracing
Detect bugs in assembled code
Dissasemble memory with forwards or backwords scroll
Locate variables in memory by tracing program

Only quick load copies of programs can be used with RamStep. To obtain a quick load copy boot the program you want to analyze, then using the Replay card make a copy. This is the quick load copy.

To execute the analysis program choose the option RamStep from the Replay utility menu. You must have a Ram card in the Apple to use this program. Once the program starts the upper ten lines of the screen will have the menu.

On the top of the screen is one line giving the important register contents and stack contents.

To load a program for analysis use option L. You must use quick load copy disks.

Once the program is loaded in using the L option the user can dissasemble, make pathces with the mini-assembler or do a step and trace.

The assembler is similiar to the APPLE mini-assembler. To exit the assembler type:

$EB00G

You will return to the main menu.

The dissasembler will prompt you for an address to start at. Enter the address in Hex. The top ten lines will then contain a disassembly of code at that address. At that time the following keys will function:

(space)	will disassemble next 10 lines
->	will disassemble next 5 lines and scroll up
<-	will back up about 5 lines and dissasemble forward
(return)	exit disasemble routine

The M option will take you to the Replay monitor. It is a monitor very similiar to Apple monitor. The entry points to the important routines are the same as in the Apple monitor. There are some improvements though.

Now a memory list such as 100.200 will display hex and

ascii.

The step and trace routines have been rewritten to allow transparency. You can now step or trace a program and maintain memory. The new commands are:

#S Step a program, show instruction mnemonics and registers
#T Trace a program continuously
#R Trace a program and restore the video

The difference between T and R is the video text page. When you are working with RamStep the upper ten lines of text page are remove to safety. Some programs use that memory for code. If you trace a program and it attempts to execute in the video memory you must put back that memory as at time of interupt. Therefore the R command will do this. You cannot see a continuous mnemonic disassembly this way, as with the T command, but you can tell where the program is executing (more in a minute).

Once the you put the program in T or R mode you can view any one of 3 screens, textl/hiresl/hires2. There are three keys that control this.

If you hit a (;) the hiresl page will display, hit a (<-) for hires2 and hit a (->) for text 1.
That way you can watch the disassembly mnemonics on textl and watch graphics on hires in continuous operation.

To stop the execution type a

(ctrl) E control E

This will exit you to the Replay monitor. If you were in R mode the memory locatons $3A $3B contain the PC at time of exit. The pages Ø,1 are stored at $DCØØ and $DDØØ for the ram card version of ramstep.

There is an eprom version of this analysis. The eprom can replace the copy eprom that comes with the Replay card. Then the user can stop a program at any point, analyze as discussed above, and restart to look at another time. The user can stop/analyze/restart continuously.

If you wish to order this eprom send $2Ø and request the:

Replay ML utility eprom.

The analysis eprom has the same commands as the step and trace ram card program. Its reentry point from the assembler or monitor is $FØ8l. The pages Ø,1 are stored at $CCØØ and $CDØØ for the eprom version.

APPENDIX I

Screen Print

This program will allow users to load in and print any of the screens that the Apple displays. The procedure is as follows.

1. Make a quick load copy at the point you want to dump the screens.
2. Run Packer, option 4 on the Replay utility disk.
3. Once Packer is running hit C for command file pack
4. For textl or 2 enter:
   C.TEXT1&2.CAPTURE
   For Hires 1 or 2 enter:
   C.HIRES1&2.CAPTURE
5. The auto pack file will be loaded from the Replay utility disk.
6. Insert the quick load copy you made of your program and hit return.
7. When finished packing hit return and use option A to save the packed file
8. Now choose option B on the Replay utility disk and it will prompt you.

The dump program is in basic and easily modifiable. It assumes a Dumpling type interface. It can be changed to any type interface or driver easily.

APPENDIX J

Apple II and //e Monitor Differences

The Apple II and //e are very similiar so that programs written for the II can be run on the //e and to a limited extent the other way around.

There are some differences though that need to be realized. The major areas are shown below.

//e and II monitor rom differences ($F800 rom)

//e expansion roms at $Cxxx

80 column software

//e softswitches

Because of these differences programs copied on one machine may not run on the other. The biggest problem is generally the monitor roms. The //e has an expanded monitor rom that uses the $Cxxx range and the main monitor routines are different. The entry points for the main routines have been kept the same but from there on the code could be changed.

This brings a problem in when a program is interupted during access to the monitor rom bios routines. If the program is in the middle of a routine and the user attempts to move it from say a II to a //e it is not guaranteed the program will restart. In the //e the code is most likely different or shifted.

The solution to this is to only copy the program when it is executing in lower memory. That way it will most likely transfer to the other machine, since the main bios routine entry points are the same. To do this, stop the program and look at the PC in the Replay monitor. Only copy the program if the PC is below $F800.

One of the other problem areas is the expansion rom in the $Cxxx range. This is related to the above discussion. If the PC is in the range of $Cxxx then don't copy it if you are moving from the //e to the II. The II does not have the expansion rom.

The //e softswitches are many, some programs depend heavily on their presence and use. Others don't use any of the new softswitches. There is no clear cut solution for this as it is hardware dependent.

If there is 80 column software being used the transfer from machine to machine is a problem. Especially so if the user has an 80 column card in the II and is moving to the extended 80 column Apple video card. They are not functionally the same. The softswitches again come into the picture for control.

These are all problem areas and possible help when transferring from II <---> //e. If you have any further questions please call for technical.

APPENDIX K

FRANKLIN USERS

The Replay card can be used on the Franklin computer but there are some limitations. First and foremost is that only 48k programs can be copied. This is due to the way the Franklin has the upper 16k disabled.

To use the Replay card the user must disable the upper 16k of memory so the Replay card can function. To do this a jumper must be switched from one pin to another on the Franklin mothrboard. This jumper is documented in the Franklin manual for disable of upper 16k. It is located in the middle of the top row of ram chips on the Franklin motherboard.

The page in the Franklin manual is 5-9. The location on the motherboard is the third rom from the left. There is a ro row of chips on the motherboard that are much larger than all the other chips, these are the rom chips.

ADDENDUM TO REPLAY DOCUMENTATION

COMPRESSION OF 48/64K PROGRAMS

****** TO USE THE COMPRESS PROGRAM YOU NEED A QUICK LOAD
****** COPY OF THE PROGRAM YOU WISH TO PACK. WE RECOMMEND
****** EXECUTING CLEARMEM BEFORE BOOTING THE ORIGINAL DISK
****** AND MAKING A QUICK LOAD COPY. CLEARMEM WILL SET
****** ALL MEMORY TO ZERO WHICH IMPROVES THE ANALYSIS
****** ABILITY TO REJECT UNEEDED MEMORY.

There is a program on your disk called 'COMPRESS'. This program is an automatic compression program for 48k or 64k programs. It will read the quick load copy that you make , along with the Dos binary 3.3 file created for 64k copies, and analyze the copy to compress its size. Not all programs can be compressed but many can. Examples of some programs which are difficult to compress are Choplifter and Repton. These programs are 48k programs that use an immense amount of memory. The shape tables for the different objects take up a lot room.

Examples of programs that can be compressed are word processors, modem programs, bit copiers ....., these programs may operate all over memory but they have large buffers which can be eliminated.

To operate the program you will need these items:

Disk in drive 1: Quick load copy of lower 48k made by Replay card.

Disk in drive 2: Dos 3.3 disk with 260 sectors free
If 64k copy:
The FILE.RAM created by Replay utility disk of upper 16k ram.

Boot a Dos 3.3 disk and arrange the disks and drives as shown below. The disk in drive 2 should be a Dos 3.3 disk with at least 260 sectors free.

To create the disk used in drive 2 a user could:

Initialize a new disk.

If 64k copy use FID , or other, to move the FILE.RAM over to the new disk.
Remember the FILE.RAM is the binary file created by Replay Utility disk option 5 when making a 64k copy. It is the upper 16k ram.

HOW TO USE COMPRESS

Boot the Replay utility disk and choose the option for the 'COMPRESS' program. The program will be loaded in and executed. There is a section at the beginning which allows user override or user input. The default settings are in place. If you don't want to change anything hit return. The different selections are documented following this brief example.

Two parameters that should be thought about each time are the QUICK LOAD COPY (48k) and USE APPLESOFT. The default setting is for 48k, if you have a 64k copy then type Q and COMPRESS will work with 64k. Applesoft pointers are not used with the default setting. If you think you have an Applesoft program then hit A to turn on use of Applesoft pointers.

When you hit Return to continue, the program will prompt for the correct disk setup. If it is a 64k copy the program will prompt you for the name of the 16k ram file created by option 5. Enter the full name of the file with the .RAM extension. After analysis the program will prompt you for a file name to save the compressed file under. Enter any name.

When the compressed file has been saved to disk the packing program will tell you if a ram card is needed to execute or not. To execute the file simply type :

BRUN FILENAME

Two files are created, the second with a .REP extension. They are standard Dos 3.3 and can be put on a hard disk.

PARAMETER EXPLANATION

A brief explanation of each setting is given here:

R) STOP AFTER ANALYSIS

This is to allow user viewing of the pages marked for copying and the different marking results. If this is Y for Yes then the program will halt after analysis to allow user input. See the following section on Viewing Marks. If this is N the program will continue straight in to pack the marked pages.

Q) QUICK LOAD COPY
FULL COPY

This indicates that the copy to be packed is a 48k quick load copy only or that it is a full 64k copy. If Y only is selected 48k will be analyzed and packed, Full copy will show N. If Quick load copy shows N and the Full copy shows Y then 64k will be analyzed and packed. The FILE.RAM is then needed on the disk in drive 2.

HOW TO USE COMPRESS

Boot the Replay utility disk and choose the option for the 'COMPRESS' program. The program will be loaded in and executed. There is a section at the beginning which allows user override or user input. The default settings are in place. If you don't want to change anything hit return. The different selections are documented following this brief example.

Two parameters that should be thought about each time are the QUICK LOAD COPY (48k) and USE APPLESOFT. The default setting is for 48k, if you have a 64k copy then type Q and COMPRESS will work with 64k. Applesoft pointers are not used with the default setting. If you think you have an Applesoft program then hit A to turn on use of Applesoft pointers.

When you hit Return to continue, the program will prompt for the correct disk setup. If it is a 64k copy the program will prompt you for the name of the 16k ram file created by option 5. Enter the full name of the file with the .RAM extension. After analysis the program will prompt you for a file name to save the compressed file under. Enter any name.

When the compressed file has been saved to disk the packing program will tell you if a ram card is needed to execute or not. To execute the file simply type :

BRUN FILENAME

Two files are created, the second with a .REP extension. They are standard Dos 3.3 and can be put on a hard disk.

PARAMETER EXPLANATION

A brief explanation of each setting is given here:

R) STOP AFTER ANALYSIS

This is to allow user viewing of the pages marked for copying and the different marking results. If this is Y for Yes then the program will halt after analysis to allow user input. See the following section on Viewing Marks. If this is N the program will continue straight in to pack the marked pages.

Q) QUICK LOAD COPY
FULL COPY

This indicates that the copy to be packed is a 48k quick load copy only or that it is a full 64k copy. If Y only is selected 48k will be analyzed and packed, Full copy will show N. If Quick load copy shows N and the Full copy shows Y then 64k will be analyzed and packed. The FILE.RAM is then needed on the disk in drive 2.

P) SAVE HIRES 1 PRIMARY
This allows the user to specify that hires 1 is to always be packed.

S) SAVE HIRES 2 SECONDARY
This allows the user to specify that hires 2 is to always be packed.

D) USE DOS IN MEMORY
This is used with programs that use a patched version of Dos for their operating system. The Dos that is in memory when the compressed file is Brunned is preserved and patched with any differences to the copy. The benefit is that a smaller copy is needed. Only the patches are copied. The Dos in memory at Analysis time is used as the normal Dos, be sure to use that Dos on the disk that contains the compressed file if the D option is set to Y.

A) USE APPLESOFT POINTERS
If you know or suspect the program is in Applesoft set this to Y. Then the Applesoft pointers are used to mark memory. Don't forget about the SOFTMOVE program on the Replay Utility disk. Softmove will move an Applesoft program from a copy into normal Dos 3.3. It will not move binary programs called by Applesoft.

The next three lines are for marking sensitivities.

M) CODE MARK SENSITIVITY >9Ø

This is used to find program code areas. See option 6 of the Packer program for more detail.

I) INDEX MARK SENSITIVITY >8Ø

This is the sensitivity level where indexed marking will occur. The higher the number the more pages will be analyzed for indexing, JSR JMP etc.

T) TEXT MARK SENSITIVITY >BØ

This is used to find areas containing ASCII which is used in menu's or prompts. See option 3 of the Packer program for more detail.

E) EXTRA PAGES/SECTION >Ø2

This value is the number of pages that are added after a code segment ends. The program in memory can be considered as several code segments spaced around memory. After each segment this number of extra pages will be added to allow for spillover of program code or data.

VIEWING MARKS

One very useful tool for the user is the override or user input area. If the Stop after analysis is set to Y then the user can view the analysis results and have editing capability.

Once analysis is done the results are displayed. There are several informative tools. First displayed is the Memory Map containing the mark counts. These are the dissasembly marks found by the code search section.

All inverse pages are marked and will be packed. The total number of free pages is shown below the marked pages. This is called the composite mark page. It is made up from several sources. The first is dissasembly analysis, then indexing , JSR JMP Page referencing...etc.

Other analysis can be viewed by typing the numbers 1-9. The menu below will appear:

```
(   MARKED SHOWN )
(   MARKED SHOWN )
(   MARKED SHOWN )

P> PACK  R> RESTART  D> DISSASEMBLY
1 ASSMBLY   2 ONEBYTE   3 DOS COMPARE
4 INDEXED   5 ENTER #   6 NEW SENSE LVL
7 COMPOSITE 8 APLSOFT   9 USE DOS  ?>
```

Let's take them one at a time.

1 ASSMBLY

Show in inverse all pages found containing assembly code. This is dependent on Mark sensitivity. The current Mark level is shown.

2 ONEBYTE

Show in inverse all pages that have a single byte value all through the page.

3 DOS COMPARE

In the memory range $9A to $BF show in inverse all pages different from the Dos currently in memory.

4 INDEXED

Show all pages in inverse that are referenced to by indexed registers, JSR's JMP's.

5 ENTER #

This provides user override capability. The user is prompted for a page #, not a full address. Flip the status of the page entered. If it was marked, unmark it, if it wasn't marked then mark it. Continue asking for more pages until a carriage return is entered with no page #.

6 NEW SENSE LVL

Prompt user for new sense level. This will change the Assembly mark and thus the Composit mark but *not* the indexed mark. The indexed mark is done at disk read time.

7 COMPOSIT MARK

Show the Composit mark page.

8 APLSOFT

Using the Applesoft pointers show where the program and variables reside in memory. This does *not* alter the composite mark, it only shows where the Applesoft pointers are showing a program. The user must R> restart and change the Applesoft parm at the start to alter a composite mark.

9 USE DOS

Flip the status of the USE DOS parm.

Above the 1-9 menu options are 3 letter options. The second letter R will take the user back to the beginning of the program where he can alter the setup parameters and re-analyze the copy.

The first letter P is to continue with the Packing. The program will proceed using the composite mark to pack the program.

The final letter D is to view sections of the copied program. The user will be prompted for the FULL address to dissasemble. That section of memory will be loaded into a buffer and the dissasembly done. A full screen is shown. The A key will take the user back one page, the S key will go forward one page. The buffer only has 20 pages at a time valid, it starts at $2000 hex.

PROBLEMS

File does not work on compression.

This is due to not enough memory being marked and copied. Go back and run Compress again. This time alter the setup values. To make the Compression analysis mark more pages ,and thus include them in the packed file, change all or some of the values below:

CODE	MARK SENSITIVITY	>	RAISE VALUE
INDEX	MARK SENSITIVITY	>	RAISE VALUE
TEXT	MARK SENSITIVITY	>	LOWER VALUE
EXTRA	PAGES/SECTION	>	RAISE VALUE

The limit will be the number of free pages left after marking. The Compress program needs 50 decimal ($32 hex) pages free to build the Compressed files. The number of free pages are given in the VIEWING MARKS area when the composite mark is shown.

Games have shapes fuzzy or not there.

The shapes in games are stored as data tables. The locations in memory of the data tables are difficult to completely locate. The Compress program finds most of these by the index mark. To try and mark more shape or data areas raise the index mark sensitivity and/or the extra pages/section values.

Compress program cannot locate a space for restart program

The restart program built into the compressed file needs to be in the lower 48k memory space. If all of that area is marked this error will occurr. Use less sensitive values in analysis to mark fewer pages, or... with the user override in the viewing mark section free some pages manually. Don't forget the dissasembly option can be used to view the memory being compressed.

COMPRESS.UTIL PROGRAM

When the Compress program is run it generates the compressed program binary files that are used to replace the quick load copy. One other file is generated called PTEST. This contains the packing information derived from analysis.

The program Compress.Util can be used to load in this short file and bypass the analysis going directly to Viewing Mark. The benefit of this is that a user could analyze a program, then pack using the results and then execute the packed file. If the packed file did not execute fully he could run Compress.Util and load in the Ptest file created. Then change the mark sensitivity or use the override to manually mark. The program is then packed again based on the new marking. Ptest is saved out with the new modifications.

The user could make several passes this way without the necessity of analyzing the 48k copy each time. The Ptest file could be saved to another disk and renamed for later work.

APPLE //E USERS WITH EXTENDED 80 COLUMN CARD
(64K ON 80 COLUMN CARD)

A new program on the Replay utility disk is called AUXMEM. It will copy
the first part of the auxiliary memory. This memory is on the 80 column board. Users with this board and who want to copy programs like Visicalc //e or Applewriter //e can use this program.

The user can copy the most used part of the extra 64k memory with this program. To run this program boot the Replay utility disk and type Q to obtain an Applesoft prompt. Then type

BRUN AUXMEM

Insert a Dos 3.3 disk to save the binary file to. You will name it. The program will then put a binary file on the disk contaning a copy of the extra memory.

As an example of use let us look at Applewriter //e (A program by Apple Computer Co.). To copy this program the user would follow the procedures for 64k copy as in the Replay manual, making the lower 48k quick load copy then the upper 16k ram copy. After the user has made the 16k upper ram copy and has an Applesoft ( ]) prompt he would type:

BRUN AUXMEM

Make sure the Replay utiltity disk is in the drive. The Auxmem program will prompt you to put in the Dos 3.3 disk , use the same one the upper 16k is saved on. Next the user names the file to be saved on the disk, let's call it Aux//e. When finished you have 3 seperate pieces of memory. The quick load 48k, the upper 16k ram and the Extram memory file just created. You can execute the copy in this order:

Brun (file created with auxmem) Aux//e
This puts back the aux memory and returns.

Brun (file created with option 5, upper 16k ram)
This loads up the ram card space.

Next the user is prompted to put in the quick load copy, hit replay button and type E. The program should restart.

Alternatively the user can use the compress program to combine the quick load copy and the upper 16k ram into one binary file execution. See the section on Compress. This eliminates the need for the Replay Card to execute.

Let's say the user has used compress on the Applewriter //e and has a binary file (2) called App//e.

Then to execute the program he needs to load in the aux memory and then Brun App//e. This can be done with one command using an exec file command. There is a program called EXECMAKER that will do this. For our example you would :

BRUN EXECMAKER

type	AUX//E	(return)	causes auxmem to load
type	APP//E	(return)	loads normal mem
(return)			no more files
type	Applewriter //E		Name of exec file

Then to execute Applewriter //E without Replay card the user types:

EXEC Applewriter //E.