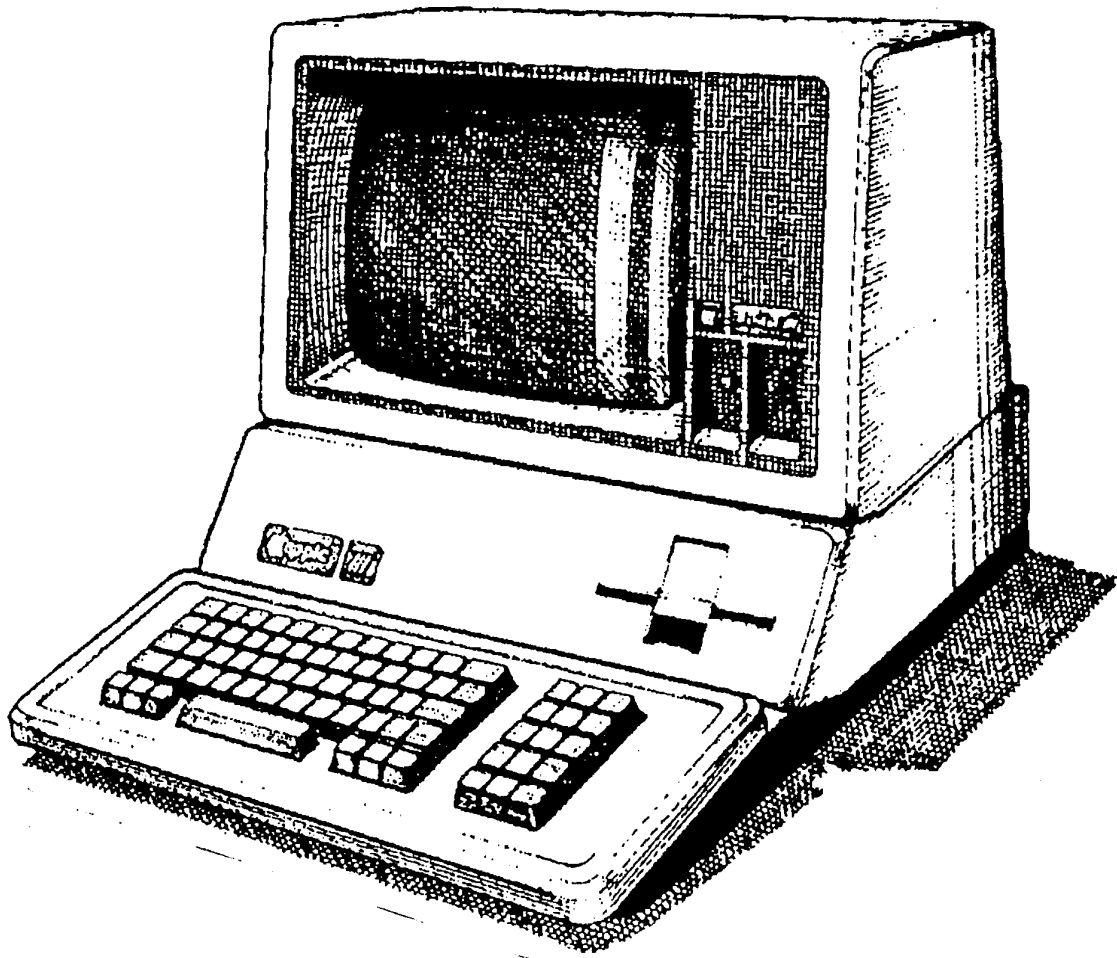# Apple /// Computer Information

DOCUMENT NAME

SOME IDEAS ABOUT AN APPLE ///
COMPUTER EMULATOR /DTC

#
191

Ex Libris David T. Craig

Apple /// Computer
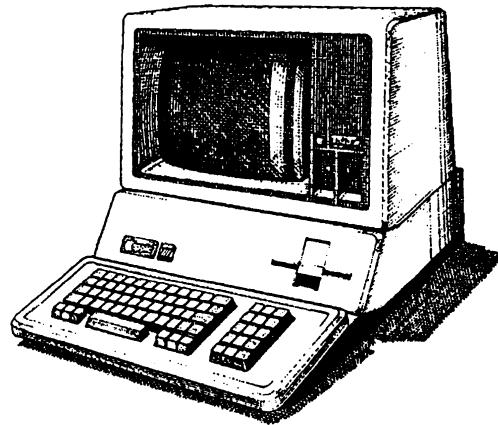
Emulation Program

Ideas

by

David T. Craig

Apple III Computer Information
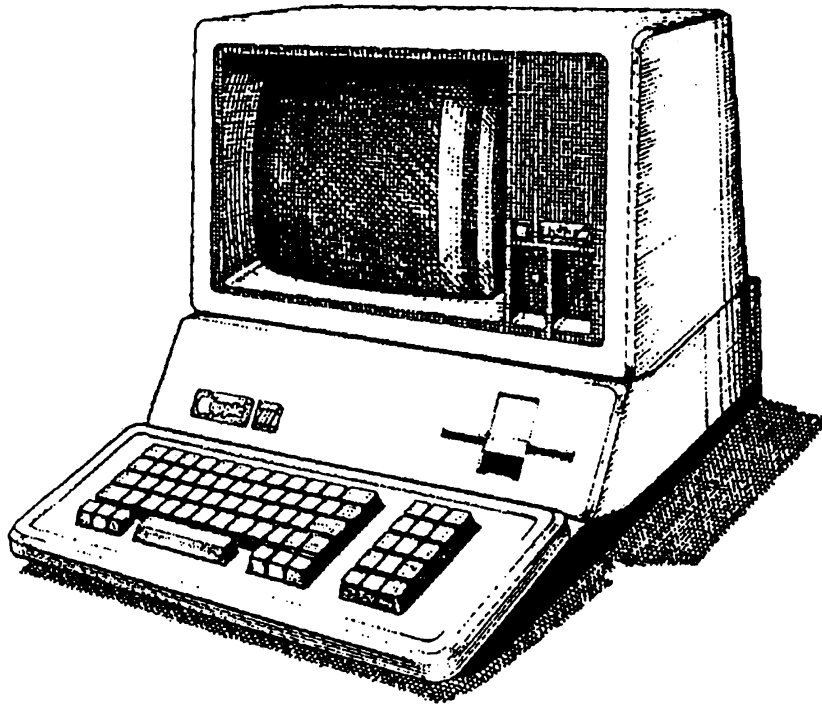
Apple III Emulator Ideas

Document Table of Contents

Version 4   •   12 Dec 1997
Version 3   •   09 Dec 1997
Version 2   •   04 Dec 1997
Version 1   •   28 Nov 1997

Apple III Computer Information

Apple III Emulator Ideas

Version 4   •   12 Dec 1997

SOME IDEAS ABOUT AN ❦ APPLE /// COMPUTER EMULATOR

David T. Craig -- 12 December 1997 -- Version 4

941 Calle Mejia #1006, Santa Fe, NM 87501 USA
e-mail: 71533.606@compuserve.com

TABLE OF CONTENTS

[handwritten annotations in right margin:]
disk:
new: READFILES
read all files from disk to host computer disk

Corrections — by page #

1 - Mod. History = add 2 spaces
9 - "C" char set bank → "K"
10 — _ : add extra space between all bold command line words.

10 - HELP name
10 - change RD cmd to show
(15) P And E bit names (upper case = 1, lower = 0)
12 - ZPAGE - show offset byte line, same for adr1·adr2
13 - S cmd
20 - SCROLL
14 - DISK BUFFER
15 - SS
16 - BPE more general
16-17 del some DP cmds
21 - BPNE font FONTROM

MODIFICATION HISTORY

28 Nov 1997 -- Version 1
Created by David T. Craig.

04 Dec 1997 -- Version 2
New sections: MONITOR SUPPORT, EMULATOR DEBUGGING FACILITIES.
Updated sections: DISK IMAGES, MEMORY BANK SWITCHING EMULATOR, SOS SYSTEM CALL
EMULATION, REFERENCES.
Added several good comments by Chris Smolinski (he's writing a /// emulator called
SARA).

09 Dec 1997 -- Version 3
DISK IMAGES: Updated info about DTCMake3///DiskImage Mac application, made disk image
file an all-text file.
SOS SYSTEM CALL EMULATION: typo Silentypr --> Silentype.
WHAT TARGET MACHINES SHOULD BE SUPPORTED: More pre-68040 Mac comments.
EMULATOR DEBUGGING FACILITIES: typo affects --> affect, added info about
enabling/disabling SOS BRK disassembly, same for ProDOS, added list of emulator
debugging commands.
EMULATOR MEMORY STRUCTURE: New section.

12 Dec 1997 -- Version 4
EMULATOR DEBUGGING FACILITIES: Added examples to every debugging command. Added
commands SNAPSHOTW, SNAPSHOTR, ZPAGE, SPAGE, EPAGE, DRIVERS, macro commands.

"_114.PICT" 663 KB 2001-08-13 dpi: 600h x 600v pix: 4627h x 6118v

1.0   PURPOSE

This document describes some ideas about implementing a software emulator for the Apple /// computer. These ideas are based on my experiences with the Apple /// computer and its software programming. No specific target machine is mentioned in this document since these ideas should be non-target machine specific. These ideas are submitted to stimulate thought about such an emulator and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS), and /// programs (e.g. AppleWriter ///) are based on my extensive collection of /// technical manuals, specification sheets, and many /// technical articles (Dr. John Jeppson's articles are very exhaustive and full of lots of neat /// techoid stuff). I have around 15 Apple manuals, the majority of which were published by Apple, which include user manuals and the technical programming manuals.

For those people seriously interested in implementing an Apple /// emulator program I highly recommend that they have at least the Apple /// Service Reference Manual. This manual, which is almost 500 pages long, is the definitive reference for how the Apple /// computer works. Most of its contents describe theory of operation even though its title suggests service-type information only. The important features of this manual for a /// emulator writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I programmed the /// many moons ago and have worked professionally as an Apple Macintosh computer programmer since 1984.

Note: All comments are welcome. If you have anything to add or correct please let me know and I will update the master copy of this document.

2.0   EMULATOR GOALS

The /// emulator should provide a complete emulation environment for the faithful execution of Apple /// and /// Plus programs. As far as the emulator user is concerned when they run the emulator program their computer should work just like an Apple /// computer and all /// visual fidelity should be maintained. Emulation of the Apple /// Plus computer may also be supported (this means the /// Plus' interlaced screen). If the /// Plus is supported by the emulator you may want to let the user specify if they want to run a /// or a /// Plus.

I think it would be beyond neat if the emulator could run Apple's running horses demo and the other /// demos.

The /// emulator should support an Apple /// computer with at least 256K of memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of memory may also exist since the ///'s operating system (SOS) supports up to 512K of memory. Memory size, if variable, should always be a multiple of 32K. I believe the lowest memory size supported by the /// (ROM?) is 96K. Support for a ProFile disk may also exist (for this disk there would need to be a disk image with a size of 5M). The first floppy disk (.D1) would correspond to the floppy disk drive that is built into the Apple ///. The other disks correspond to external disks and should exist as image files with specific file names (e.g. "Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file should also have a specific file name (e.g. "Apple 3 ProFile").

Image file names should have an extension (e.g. ".D3I") since this is needed by PCs.

3.0   EMULATOR USER INTERFACE

When the user runs the Apple /// emulator program the user should see on their computer screen a screen (or a window representing the screen on GUI systems) corresponding to the ///'s screen which the user would see if they were in front of a real Apple /// computer. All /// text and graphic modes should be supported by the

"_115.PICT" 936 KB 2001-08-13 dpi: 600h x 600v pix: 4415h x 6083v

/// emulator (this includes the special modes supported by the /// Plus and its interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes the current /// screen to either a text file (for text modes) or to a graphic file (for graphic modes) or always just creates a graphic file. The screen dump graphic file should be a standard graphic file for whatever target machine your support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple Macintosh produce PICT files). Since the /// supports custom character sets dumping the screen to a PICT file (or to the target computer's clipboard) may be the best solution.

The emulator screen if implemented in a GUI window may also display a status area at the bottom of the window. This status area would display at least two lines of text and would keep the user informed of what the emulator was doing internally.

4.0   DISK IMAGES

The /// emulator should read disk image files which correspond directly to real /// 140K disks. When the /// emulator starts it should look in its folder and if there exists a /// disk image file the emulator should boot this image. If there are multiple disk image files then the emulator may want to display a list of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of these disks there should exist an utility program that runs on the /// computer and which outputs disk block data to the /// serial port (I plan to make this utility and call it DTCDumpIt). This utility's output should be a hex/ascii dump that specifies block numbers and has a checksum for each line of data. This utility should ask the user if it should dump a file or a disk.

On the target machine there should exist a similar utility that inputs the disk block data and creates a disk image file. I recommend that the transmitted disk block data consist of a hex dump with block number and checksum information in a human readable fashion. The receiving program (on the target computer) would read this human readable information, verify that the data was sent correctly, and produce binary disk image file images (I plan to create this utility for the Apple Macintosh and call it DTCMake///DiskImage).

There should also exist a disk image file for the ///'s Boot ROM (recommended file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image. This ROM should be the Revision 1 ROM (not Revision 0) since this was the last ROM produced and SOS 1.3 (the last SOS) requires this ROM.

Users should also be able to format a disk image by specifying the disk drive device name (e.g. .D2). Users should then be able to name the disk image so that they can use it later. Users should be able to assign specific disk images to specific disk drives.

I recommend that all disk image files have a very specific internal format. This format should support the verification of disk image files so that if a disk image file becomes corrupted in some fashion the /// emulator can detect this corruption, not use the image, and alert the user.

Note: Support for existing Apple][disk image files may be feasible but I recommend against this since the format of these images could change.

The proposed image format:

The disk image file contains two parts, a header part and a data part. The header part appears first followed by the data part. The header part contains identification and verification information. The data part contains the actual disk blocks for the /// disk. This file contains only text, no binary data appears here in any fashion. The only non-text information that can appear in these files is the Carriage Return (CR) and the Line Feed (LF) characters. The emulator should ignore

LFs if appropriate. All information appears in lines with a maximum length of 255 characters. Character case is immaterial. Blank lines are ignored. The reason for this format is so these image files can be transferred over the internet without the need for any binary-to-text conversion. Also, text-only files can easily be viewed by people using a word processor.

The header part contains:

```
Line                    Comments
----------------        ----------------------------------------------------------
Signature               "APPLE /// DISK IMAGE"
Version                 "VERSION" version number (e.g. "1")
Image Name              "IMAGE NAME" name of image, anything the user wants,
                        most likely the name of the interpreter on the disk,
                        e.g. "Apple Writer ///"
Creation Date           "CREATED" date image file created, "YYYY-MM-DD"
Created by Name         "CREATED BY" name of person or company who created this image
Comment                 "COMMENT" comment for anything user wants
Data Size               "DATA SIZE" size of data part (decimal, e.g. "143360")
Data Checksum           "DATA CHECKSUM" hexadecimal checksum (e.g. "FA7C3188")
Reserved 1              "RESERVED"
Reserved 2              "RESERVED"
Reserved 3              "RESERVED"
Reserved 4              "RESERVED"
Tech Comment            "TECH COMMENT" name of program that this is for
Header Checksum         "HEADER CHECKSUM" hexadecimal checksum (e.g. "B97C31D5")
```

Notes:

The checksum should be calculated as the exclusive-OR of each byte followed by a left rotation of 1 bit. Checksum starts with zero. Checksums should always be 4 bytes in size and be stored in the header as an 8 character string.

The Tech Comment's purpose is to allow people who obtain an image file to be able to contact someone about the file's purpose.

The data part contains lines representing 16 bytes from the original disk. Each line has a specific format which begins with the starting disk address for the line, 16 bytes, the ASCII equivalent of the 16 bytes, and a checksum for the bytes of the line with the format:

[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [1234567890123456] 12345678

The last line of the file must be the word "FINIS".

Sample disk image file:

```
   APPLE /// DISK IMAGE
   VERSION 1
   IMAGE NAME Apple Writer ///
   CREATED 1997-10-11
   CREATED BY David T. Craig
   COMMENT Thanks to Paul Lutus
   DATA SIZE 16
   DATA CHECKSUM FA7C3188
   RESERVED
   RESERVED
   RESERVED
   RESERVED
   TECH COMMENT For David Craig's /// Emulator - 71533.606@compuserve.com
   HEADER CHECKSUM B97C31D5

   [00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [Apple.///.Emul..] FA7C3188
```

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 4 / 23

FINIS

5.0   6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The heart may
be referred to as the "6502 engine." The emulator should support all of the 6502
instructions, the 6502 registers, and the special Apple /// registers (e.g. the bank
switch register, the environment register, and the zero-page register). Special
register descriptions and usage can be found in the Apple /// SOS Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and
environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the ///
emulator should work much more easily.

Support for special /// features may also exist at this level of the /// emulator.
For example, the /// emulator may not want to emulate all of the ///'s memory-mapped
I/O features, but instead intercept access to special areas or routines and call the
target machine's operating system to handle these features. See sections ROM
EMULATION and MEMORY-MAPPED I/O EMULATION for more details.

6.0   ROM EMULATION

The /// emulator should also support as much as possible the ///'s Boot ROM. This
means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation
discussion.

For the Boot ROM's floppy disk I/O support I recommend that all the gory details here
not be supported directly at the memory-mapped I/O level but instead the /// emulator
should emulate this I/O. Specifically, the /// emulator should intercept any access
to the Boot ROM routines which read or write disk blocks and use the appropriate
target machine operating system routines to accomplish this feature.

The /// emulator should also initialize the ROM's character set which the ROM
normally loads into a special RAM chip that is not accessible to the ///'s 6502
processor. See section MEMORY BANK SWITCHING EMULATION for more details.

7.0   MEMORY-MAPPED I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world need to
be handled by the /// emulator. These areas include such addresses as the speaker
addresses. The Apple /// Service Reference Manual provides detailed information
about these addresses.

All accesses to memory by the /// emulator must respect the bank switch and
environment register settings so that the emulator does not try to access a memory-
mapped address when that address is not mapped into the 6502 address space.

Programs which access low-level I/O locations such as the disk I/O addresses should
not be supported. I assume most /// programs will access hardware components using
SOS or device drivers.

Note: Chris Smolinski says that emulating the low-level stuff on a Power PC-based
Macintosh is not very difficult and works rather fast (he's implemented in his SARA
emulator the ///'s floppy disk I/O).

8.0   MEMORY BANK SWITCHING EMULATION

The /// emulator must also fully support the ///'s bank switched and enhanced
indirect addressing memory architecture. Detailed descriptions and usage of ///
memory handling can be found in the Apple /// SOS Reference Manual.

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 5 / 23

The /// emulator should also support the ///'s character set RAM chip. This holds the bitmap descriptions of each of the 128 characters in the /// character. This RAM area, which is not accessible to the ///'s 6502 CPU, holds 1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the storage of the character set in the SOS.DRIVER file. I believe the ROM character set has bits that are reversed compared to the SOS.DRIVER character set.

The storage of text and graphics in memory should be supported also. This should happen automatically when a /// program writes to the text/graphic memory buffers. The emulator needs to detect such writes and update its screen as appropriate.

9.0 SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be intercepted by the /// emulator. But certain calls may need to be intercepted unless a lower level of the /// emulator intercepts these feature already. System calls to SOS or drivers that may need intercepting by the /// emulator could be:

```
o  Disk I/O            (.D[1-4] and .PROFILE drivers)
o  Keyboard I/O        (.CONSOLE driver)
o  Screen I/O          (.CONSOLE and .GRAPHIC drivers)
o  Sound generation    (.AUDIO driver)
o  Serial port I/O     (.RS232 driver)
o  Silentype Printer   (.SILENTYPE)  [I'm not sure about support for this]
o  Clock I/O           (Y2K dates may be a problem)
```

I recommend that the /// emulator intercept all activity dealing with the above and have the target machine perform the equivalent features. For example, to read or write a disk block the /// emulator should have a routine that accesses the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so that the user can specify specific properties of some of the above drivers. For example, if the target machine supports several output ports the emulator may let the user specify which port to use (e.g. for the .PRINTER driver the user could assign it to a specific serial or parallel port on the target machine).

Note: The ///'s clock does not support the year 2000 or greater. I think the emulator should support Y2K dates but I'm not sure if SOS's file system date stamps will support this easily.

10.0 DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM CALL EMULATION. I suspect the programming within the /// emulator for this area could be the most work since there are lots of device drivers that make up a simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is interrupt handling. Since the emulator does not have physical devices connected to it in any direct fashion I don't think interrupts exist as far as the emulator is concerned. Interrupts dealing with disks or the keyboard can be handled at a lower level by having the /// emulator call the appropriate system call in the target machine. These low-level I/O handlers should set up the appropriate driver data areas so that the rest of the ///'s software (SOS and the interpreter) will work correctly. For example, keyboard I/O should be setup in the /// emulator so that when the keyboard input memory-mapped I/O location is accessed the target machine OS really reads the keyboard and sets up the memory-mapped location as appropriate.

11.0 KEYBOARD SUPPORT

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 6 / 23

11.1   User interface support

The /// computer's keyboard layout is basically compatible with modern keyboards.
The /// keyboard does have two extra keys, Open Apple and Closed Apple which are
positioned to the left of the Apple /// keyboard. Also present on the keyboard are
four arrow keys. The emulator should support these keys either directly (i.e., the
target machine has similar keys) or associate other keys with the ///'s special keys
(e.g., the Macintosh computer's two Option keys could be used to simulate the special
Open and Closed Apple keys). The emulator's associated keys need not physically be
in the same location as the ///'s special keys but having them in the general area
will be beneficial.

Note: The /// Plus keyboard contains an extra key, Delete, compared to the ///
keyboard.

11.2   Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped I/O
locations as detailed in section DEVICE DRIVER EMULATION.

12.0   MONITOR SUPPORT

The emulator should support the Apple's built-in ROM Monitor. Entry to the Monitor
should be similar to how this is done on a real /// (at startup if Open Apple and
Control keys are pressed). The code in the ROM which tests for Monitor entry should
work.

13.0   APPLE] [EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple] [Emulation Disk. I'm not
sure of what would be involved here but suspect that if the ///'s 6502 CPU and the
memory-mapped I/O locations are robustly supported that the] [emulation should work
also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do not
exist in the Apple] [world.] [emulation details can be found in the Apple ///
Owner's Guide and the Apple /// Service Reference Manual.

Note: I have a disassembled listing of the Apple] [Emulation Disk ROM source listing
which could prove useful in this area.

Further analysis of the] [emulation disk's boot sequence needs to be done since I'm
unknowledgable about this area. Also, I've heard that the] [emulation accesses an
I/O location which disables some /// features.

14.0   WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I highly recommend that the /// emulator be written in a high level language such as
Pascal or C. This should make the emulator more compatible with different target
computers and make development and maintenance of the emulator much easier. I
recommend avoiding low-level languages such as assembly.

15.0   WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines that
are commonly used today by most computer users. This means either the IBM PC or the
Apple Macintosh machine family. For the PC world I recommend the /// emulator run
under Windows 95 and Windows NT. For the Macintosh world I recommend the emulator
run on most Macintosh models which means support the Macintosh 512 and above. Color
display should also be supported by the /// emulator (for the Macintosh this means
use Color QuickDraw if the machine supports CQD and if CQD is not supported by a
Macintosh model use the Classic B/W QD and maybe use patterns as "colors").

Any of these machines should be fast enough to emulate the /// and most likely will be too fast in many areas. I recommend some type of speed control be built into the emulator so that users can control how fast the emulator works. For many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will be immaterial since these programs typically wait for the user to enter data and then do their thing. But for programs such as games the user will want to control the emulator speed otherwise the game's actions will be super fast and unplayable.

Some people say that the older machines such as pre-68040 Macintoshs will be too slow for a reasonable /// emulator. I would like to see this /// emulator run on a Mac 512 machine an onwards. Running on a Mac 128 machine seems a problem due to this machine's small memory size and should not be supported (if a virtual memory scheme was used by the emulator the Mac 128 could be supported but I think having this extra level of support in the emulator would not be worth it). I disagree and am willing to wager a small sum that I'm right.

16.0   EMULATOR DEBUGGING FACILITIES

The emulator should support a comprehensive built-in debugger. This debugger's purpose should be to let the sophisticated emulator user access any part of the emulator's /// address space. This should include all of the memory that is allocated to the /// as its memory. This memory would encompass the 256K (or 512K) of /// RAM, the /// ROM (4K), the character set RAM (1K), the 6502 registers, and the special /// registers (e.g. bank register).

This debugger will prove invaluable in diagnosing emulator bugs. Not only will the user be able to type commands for the debugger but the emulator will be able to send messages to the debugger.

Logging of all debugger sessions should be stored to a text file for possible analysis. This text file would be created when the emulator starts. The log file should be appended to by the emulator. Only the user can delete the file.

The debugger should exist as a separate window that does not in any way affect the emulator's main window. This window should display only commands that the user enters or replies returned by the debugger. There should not exist a separate window area showing things such as the 6502 registers since all such information should appear in the debugger log file. The window should support at least 80 columns of text and 24 rows.

The emulator user interface should be based on a simple command line control scheme. All commands and command outputs should be text-based. This scheme could be based on the ///'s Monitor's commands or on a little more readable command scheme such as in Apple's MacsBug debugger. There should be full on-line help that discusses the debugger commands in general and each command should also have on-line help available. The debugger should show at the beginning of each line a prompt character to indicate when it is waiting for a command. I recommend the prompt be the ">" character. The debugger should also show a cursor which I recommend to be a black square.

The debugger should support the standard debugging commands such as displaying/setting memory, displaying/setting registers, and disassembling 6502 instructions. This disassembly should support the special SOS BRK call by listing the word "BRK/SOS" instead of just "BRK" and following this with the SOS command number/name and the parameter list address:

    SOS C0/CREATE 345A

The user should be able to enable or disable this feature.

Note: It may be good to also support the Apple][ProDOS command calling scheme in case this emulator ever becomes an Apple][emulator.

The debugger should support break points, single stepping, and timing buckets. The

timing buckets would be used in conjunction with break points to record how long a
sequence of 6502 instructions took to execute. This can be very useful in locating
emulator bottlenecks. The debugger supports many break point commands since I have a
feeling that this facility will be very powerful and useful during the emulator's
development.

The debugger should support the collection of statistics about the emulator. I
recommend tracking how many times specific 6502 opcodes are executed (obviously, the
debugger would need commands to display and clear this information). I would also
track memory accesses on at least a page (256 bytes) basis.

The debugger should be accessible at any time that the emulator is running. I
recommend some type of key press combination that the emulator would detect and
display the debugger window. Once the debugger window is active it should remain on
the screen until the user closes the window.

The emulator should also support a special key press combination at emulator startup
time that activates the debugger just before the /// ROM is run. This can give the
emulator developer a good way of tracing ROM execution.

The emulator should activate the debugger if any fatal emulation errors are detected
and the debugger should show a message detailing the reason for the activation. All
of these errors display a dump of the 6502 and SOS control registers. Reasons for
debugger activation from the emulator are:

1. A program writes to write-protected memory (e.g. SOS's address space). The
displayed message is "EMULATOR EXCEPTION: WRITING TO WRITE-PROTECTED MEMORY".

2. A program executes an undefined 6502 instruction (e.g. 6502 opcode $02). The
displayed message is "EMULATOR EXCEPTION: UNDEFINED 6502 OPCODE".

When the debugger is initialized (which should be when the emulator starts) the
debugger should check if a text file named "DDT.TXT" exists. If so, the debugger
should read each line from this file and execute it. Obviously, this file should
contain debugger instructions. This can be very useful for setting up commonly used
break points which if you use many would be tedious to type everytime you wanted to
use the emulator.

A memory snapshot facility should also exist. When activated by a debugger command
this facility would write to the host computer's disk a binary file containing a copy
of all the /// memory areas. This snapshot should also be readable by the debugger
so that the user could restart a specific emulation session from the snapshot.

I recommend the following emulator debugger commands which are based on the ///
Monitor commands so that these debugger commands will be familiar to Monitor users.
These commands for the most part have the general syntax of address-command. See my
document "Inside the Apple /// Computer ROM" for a list of the /// Monitor commands.
For information about the Apple] [Monitor commands, which the /// Monitor commands
are based upon, see "Apple] [Reference Manual" (Chapter 3: The System Monitor, dated
1981).

Addresses appearing in debugger commands may be prefaced by "N/" where N is a bank
number. For example, to reference address 2000 of bank 4 use 4/2000. If no bank
number precedes an address the current bank is used. To reference a ROM address use a
bank "number" of "R", for example "R/F000". To reference a character set address use
a bank "number" of "C", for example "C/0000". To reference the SOS system bank use
"S", e.g. "S/1400".

Commands should be case-insensitive (none of the UNIX case-sensitivity gobbly-gook).

Commands that display more than a screen full of information should either
automatically pause when the screen is full, or the user can use the SPACE key.

Note: Commands using ":" may also use ";" which is easier to type since this

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 9 / 23

character does not need the user of the shift key. Same for "<" and "/".

Most debugger command numeric arguments must be specified in hexadecimal. The exception is the X command which supports hexadecimal, decimal, and binary.

The debugger command parser should be very liberal. This means that users should be able to include extra spaces (or no spaces) and the command should be parsable. For example, if a command needs a list of bytes the user should be able to enter any of the following: "AABBCC", "AA BB CC", " A ABBC C " and the debugger will see these as "AABBCC".

The debugger should also support a command macro facility. This facility allows you to define a macro consisting of other debugger commands. Typing the name of the macro will then type the commands as if you entered them manually.

===

HELP (or ?) *cmd name*
  2 sps

Display debugger on-line help for all commands. Help info should be stored in an external text file for easier modification. I recommend that this section of this document be the help file.

cmd name = 1st part or all cmd name

Example: HELP *— shows all help info*

Note: ? also same *HELP S -- show all cmds starting w/ S*

BYE *HELP SS -- shows SS cmd*

Return to the emulator.

Example: BYE

CARRIAGE RETURN keypress
  2 sps
Repeat last command.

Example: If the last command was HELP and you press the CARRIAGE RETURN key then HELP will be displayed and executed again.

SPACE keypress
  2 sps
Pause current command's output. Press again to continue.

Example: If a command is executing and you press the SPACE key the comand's output will be paused, pressing SPACE again resumes the command's output. Pausing/Resuming are done on an output line basis only.

DELETE keypress
  2 sp
Stop current command's output.

Example: If a command is executing and you press this key then the command will stop executing and you will be returned to the debugger's prompt.

RD

Display 6502 registers and /// system control registers.

Show table explaining bits in P and E

Example: RD

bit names   *bit names*
A=04   X=01   Y=D8   P=30/00000011   S=F8   PC=034A : E=77/01110111   Z=1A   B=03
  1 sp   NV-BDIZC   STVRWSRR
                   unused — shows 0 or 1

N negative
V over flow
B break command
D decimal mode
I interrupt disable
Z zero
C carry

S - System Clock rate
I - I/o space
C - Screen (video) state

R - Reset enable   *R - ROM*
W - Write protect   *R - ROM*
S - Stack in use

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 10 / 23

```
----------------------------------------------------------------
```

byte:SA

Set 6502 A register to byte.

Example: 45:SA

```
----------------------------------------------------------------
```

byte:SX

Set 6502 X register to byte.

Example: 7B:SX

```
----------------------------------------------------------------
```

byte:SY

Set 6502 Y register to byte.

Example: FF:SY

```
----------------------------------------------------------------
```

byte:SP

Set 6502 P register to byte.

Example: 56:SP

```
----------------------------------------------------------------
```

byte:SS

Set 6502 S register to byte.

Example: AA:SS

```
----------------------------------------------------------------
```

word:SPC

Set 6502 PC register to word.

Example: 2000:SPC

```
----------------------------------------------------------------
```

byte:SE

Set /// E system control register to byte.

Example: 34:SE

```
----------------------------------------------------------------
```

byte:SZ

Set /// Z system control register to byte.

Example: 19:SZ

```
----------------------------------------------------------------
```

byte:SB

Set /// B system control register to byte.

Example: 06:SB

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 11 / 23

```
-----------------------------------------------------------------
addr1.addr2

Dump memory data to screen from address 1 to address 2 and display ASCII character at
the right of the screen.

Example (assumes current bank is bank 4): 300.30F
         0  1  2  3 . .
4/0300- B900 080A 0A0A 9900  08C8 D0F4 A62B A909 [F..d.uy%^&90@..G]

-----------------------------------------------------------------
ZPAGE

Dump the contents of the current interpreter's Zero Page (256 bytes).  Also supported
are commands for the Stack Page and the Extend Page:

    SPAGE    -   stack page
    EPAGE    -   extend page

To dump the pages for SOS (and drivers) use the following commands:

    SZPAGE   -   zero page
    SSPAGE   -   stack page
    SEPAGE   -   extend page

Example:  ZPAGE

Zero Page (interpreter)
      00 1 2 3 4 5 6 7 ...
1400- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF
1420- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF
...
14E0- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF

-----------------------------------------------------------------
addr:bytes

Store starting at the address the bytes.

Example: 2000:AA BB CC DD EE FF
         2000:AABBCCDDEEFF

-----------------------------------------------------------------
addr:'text'

Store text starting at address (high bit clear).

Example: 2000:'Hello World'
         2000:'David''s Dog'        -- (this stores) David's Dog

-----------------------------------------------------------------
addr:"text"

Store text starting at address (high bit set).

Example: 2000:"Hello World"
         2000:"David's Dog"         -- (this stores) David's Dog
         2000: "I said ""Hi""                        I said "Hi"
-----------------------------------------------------------------
addr3<addr1.addr2M

Move data in address range to address 3.

Example: 2000<3000.3100M
```

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 12 / 23

```
--------------------------------------------------------------
```
addr3<addr1.addr2V

Verify data in address range equals data starting at address 3.

Example: 2000<3000.3100V

Displays either "OK" if the verification suceeds, or "MISMATCH" if the verification
fails.

```
--------------------------------------------------------------
```
bytes<addr1.addr2S

Search memory in address range for the bytes.

Example: AA<3000.3100S   -- searches for byte AA
         AABBCC<3000.3100S   -- searches for bytes AA BB CC

If a search finds a match then the starting address of the match is displayed,
otherwise "PATTERN NOT FOUND" is displayed.

PATTERN FOUND AT addr

```
--------------------------------------------------------------
```
'text'<addr1.addr2S

Search memory in address range for text (high bit clear).

Example: 'D'<3000.3100S
         'David'<3000.3100S

```
--------------------------------------------------------------
```
"text"<addr1.addr2S

Search memory in address range for text (high bit set).

Example: "D"<3000.3100S
         "David"<3000.3100S

```
--------------------------------------------------------------
```
disk.block<addr1.addr2W

Write address range to disk # disk starting at disk block. If disk # is not present
then uses disk .D1. Disk should equal 1, 2, 3, or 4. The address range always ends
on a block boundary no matter what you type.

Example: 1.117<2000.21FFW   -- write 512 bytes to disk 1 block $117

Note: Disk /// disks contain 280 blocks ($118) sot he block range is 0-117
(hexadecimal).

```
--------------------------------------------------------------
```
disk.block<addr1.addr2R

Read from disk # disk starting at block to the address range. If disk # is not
present then uses disk .D1. See the W command for more info.

Example: 1.117<2000.21FFR   -- read 512 bytes from disk 1 block $117

```
--------------------------------------------------------------
```
disk.block-block:DISK

Read block range from disk # disk to a special debugger 4K buffer which is not used
by the emulator. If the typed block range is greater than 4K then only the first 4K
will be read. You can then examine this buffer's contents either with a hex/ascii

dump or with a disassembly (command L). This command is useful when you want to
examine a disk's contents. For disassembly purposes, you can specify the logical
starting address for the buffer. See the DISKBUFFER command.

To disassemble the special disk buffer (see the L command) use bank X (stands for
"extra") as part of the disassembly address parameter (e.g. "X/100"). Same for
dumping memory or whatever commands you want to use with this special buffer.

Example: 1.0-7:DISK   -- read 8 blocks (0 to 7) from disk 1

----
addr:DISKBUFFER

Set disk buffer starting logical address. Default address is 2000. See the DISK
command.

Example: A000:DISKBUFFER   -- 〜

(Range is

0000-FFFF

----
addr1.addr2L

Disassemble instructions in address range. If only addr1 appears then disassemble 20
instructions. Disassembly includes the opcode cycle count.

Example: 300L   -- assumes bank 4 is current

```
4/0300-   A9 C1      'X.'    (2)   LDA #$C1        ;
4/0302-   20 ED FD   '...'   (5)   JSR $FDED       ;
4/0305-   18         '.'     (2)   CLC             ;
4/0306-   69 0A      'T.'    (4)   ADC #$01        ;
4/0308-   C9 DB      '..'    (3)   CMP #$DB        ;
4/030A-   D0 F6      '..'    (3)   BNE $0302       ;
4/030C-   60         'U'     (4)   RTS             ;

1         2          3       4     5               6   (see Note)
```

Note: Column 1 = bank register/address
         Column 2 = memory bytes
         Column 3 = ASCII for the memory bytes
         Column 4 = opcode cycle count
         Column 5 = disassembled instructions
         Column 6 = remark character ";" (optional, see DISASMREM)

L by itself disassembles the next 20 instructions.

DISASMREM

Display ";" after each disassembly line that is produced by the L command. Default
is to not display the remark. Useful if you plan to add comments to a disassembly.
See also DISASMREMOFF.

Example: DISASMREM

DISASMREMOFF

Turn off DISASMREM. See also DISASMREM.

Example: DISASMREMOFF

----
addrG

Call subroutine at the address.

Example: A000G

----
addrJ

Jump to the address.

Example: A000J

----
wordX

Convert word (or up to 4 hex digits) to hexadecimal, decimal, and binary (X stands
for "translate"). Prefix character for byte determines its base: no prefix = hex, .
= dec, t = binary.   *Put in a table for*
                                                                       easier viewing

Example: AX   ->   A(16)   10(10)   0000 0000 0000 1010(2)
         .10X   ->   A(16)   10(10)   0000 0000 0000 1010(2)
         t1010   ->   A(16)   10(10)   0000 0000 0000 1010(2)
         FFFFX   ->   FFFF(16)   65535(10)   1111 1111 1111 1111(2)

----
addr1.addr2:CS

Calculate and display a checksum for address range. Checksum is a 4 byte quantity
which is calculated the same as the disk image file checksums.

Example: 300.500:CS
         CHECKSUM=AF897CEE

----
addrT

Trace instructions starting at the address. Each traced instruction displays
register contents. Press the SPACE to pause the trace, press DELETE to stop the
trace. The displayed registers contain values _after_ the previously listed command
executes.

Example: A000T   -- assuming bank 4 is current

4/A000-   A9 C1   'X.'   (2)   LDA #$C1
A=C1 X=01 Y=D8 P=30/00000011 S=F8 PC=A002 : E=77/01110111 Z=1A B=04
4/A002-   20 ED FD   '...'   (5)   JSR $FDED
A=C1 X=01 Y=D8 P=30/00000011 S=F6 PC=FDED : E=77/01110111 Z=1A B=04
        Lisp   *bit names*   *bit names*
Note: Press the DELETE key to stop the trace, SPACE to pause/resume.

----
addrSS

Single step trace starting at the address. After each step pause and wait for user
to press SPACE to continue or DELETE to stop the single step.

Example: A000*(SS)*-- assuming bank 4 is current

4/A000-   A9 C1   'X.'   (2)   LDA #$C1
A=C1 X=01 Y=D8 P=30/00000011 S=F8 PC=A002 : E=77/01110111 Z=1A B=04
        Lisp   *names*   *names*
Note: Press SS by itself to single step the next instruction, or press CARRIAGE
RETURN to repeat the SS.

----

addr:BP

Set a break point at address. When address is accessed the debugger is entered and displays the registers. Up to 100 break points should be supported.

Example: A000:BP

----
addr:BPC

Clear break point at address.

Example: A000:BPC

SOS:BP

Set a break point when a SOS call is made. This means when the BRK opcode is executed. Same as M00:BP.

Example: SOS:BP

Mopcode:BP

Set a break point when opcode is executed.

Example: M60:BP -- set break point when the RTS instruction (60) is executed.

ROM:BP

Set a break point when a call is made to the ROM.

Example: ROM:BP

----
addr1.addr2:BPW

Set a break point when any address within address range is written to. BPW = Break Point Write.

Example: 300.123AR:BPW

----
addr1.addr2:BPR

Set a break point when any address within address range is read from. BPR = Break Point Read.

Example: 300.123A:BPR

make just 1 BPE command

----
addr.byte:BPE

addr1.addr2.byte1 byte2... : BPE   (a2 options)

Set a break point when the address contents equal the byte value. BPE = Break Point Equals.

Example: 300.AA:BPE

addr1.addr2.byte1-byte2:BPE   (a2 options)

----
addr.byte1-byte2:BPE

Set a break point when the address contents equal a byte value in the byte range. BPE

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 16 / 23

"_129.PICT" 405 KB 2001-08-13 dpi: 600h x 600v pix: 4379h x 6107v

new cmd
BPNE BP not equals
Same syntex as BPE

= Break Point Equals.

Example: 300.AA-BB:BPE

----
addr.byte1 byte2 ... :BPEA *ℓ*

Set a break point when the address contents equal byte 1 value, or equals byte 2
value, etc. Supports up to 16 byte values. BPEA = Break Point Equals Any.

Example: 300.AABBCCDD:BPEA
         300.AA BB CC DD:BPEA

----
addr1.addr2.byte1 byte2 ... :BPEA *ℓ*

Set a break point when the address range contains any bytes equalling the byte
values. BPEA = Break Point Equals Any.

Example: 300.400.AABBCCDD:BPEA

----
addr1.addr2.byte1-byte2:BPEA *ℓ*

Set a break point when the address range contents equal the byte range. BPEA = Break
Point Equals Any.

Example: 300.400.AA-BB:BPEA

BPD

Display break point table.

Example: BPD

#	Address Range	BP	Setting
1	4/2000-4/21FF	BPEA	AA-BB

BPC

Clear break point table.

Example: BPC

----
addr1.addr2:TB

Set timing bucket for address range. When address 1 is accessed timing starts. When
address 2 is accessed timing stops. Up to 100 timing buckets should be supported.

Example: A000.A1FF:TB

TBD

Display timing bucket table. Shows all set timing buckets and the time in 1/60th of
a second and in seconds spent in each bucket.

Example: TBD

#	Address Range	Time (1/60s)	Time (secs)

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 17 / 23

```
---   -------------     -------------     -----------
  1   4/A000-4/A1FF           34              0.567
  2   4/A300-4/A310            5              0.083
---   -------------     -------------     -----------
                             39              0.650
```

```
-------------------------------------------------------------
```
addr:TBC

Clear timing bucket starting at address.

Example: A000:TBC

```
-------------------------------------------------------------
```
TBC

Clear timing bucket table.

Example: TBC

```
-------------------------------------------------------------
```
error:SOSE

List SOS general error message for the error number. If no error number is present
then list all general errors. Error info should be stored in an external text file
for easier modification. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSE

BADSCNUM - Invalid SOS call number

```
-------------------------------------------------------------
```
error:SOSFE

Display SOS fatal error message for the error number. If no error number is present
then list all fatal errors. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSFE

BADBRK - Invalid BRK

```
-------------------------------------------------------------
```
command:SOS

Display SOS command name and SOS command area (e.g. file system) for the command
number. If no command number present then list all SOS command numbers and their
names. Command info should be stored in an external text file for easier
modification. See the SOS Reference Manual for a list of these commands.

Example: C0:SOS

CREATE (File System)

```
-------------------------------------------------------------
```
SOSON

Turn on disassembly of SOS calls which displays SOS followed by the command number
and parameter address. The emulator defaults to this.

Example: SOSON

```
-------------------------------------------------------------
```
SOSOFF

Turns off SOSON.

Example: SOSOFF

----
disk:CAT

Display catalog of SOS disk stored in disk # disk. Includes recursive list of all
subdirectories. Should show same file info as Apple's System Utilities program.

Note: Other commands that may be supported include CATPASCAL for Apple][Pascal
disks and CATDOS for Apple][DOS disks. This may come in handy if you want to see
what these disks contain if you have them as disk image files.

Example: 1:CAT

----
disk.file_name:INFO

Displays information about the specified file in the disk. Information includes
standard SOS file information but also block list of all index blocks (if any)
associated with the file and block list of all data blocks for the file.

Example: 1.APPLE3.TEXT:INFO

----
disk.block:DUMP

Display contents of specified disk block in the standard hex/ascii dump format.

Example: 1.0:DUMP

----
disk:DRIVERS

Display list of contents of the SOS.DRIVER file stored on the disk. List includes
driver names, driver information, and other items that are in the driver file (e.g.
character sets).

Example: 1:DRIVERS

----
disk:CHECKIMAGE

Check validity of disk image in disk # disk. Computes header and data part checksums
and compares against the image file's listed checksums.

Example: 1:CHECKIMAGE

DIT

Display Driver Information Table (DIT), a data structure maintained by this debugger.
Contains list of all loaded drivers, their names, sizes, and entry point addresses.

Example: DIT

MIT

Display Memory Information Table (MIT), a data structure maintained by this debugger.
See section EMULATOR MEMORY STRUCTURE for what this structure contains.

Example: MIT

OPCODES

Display a histogram of opcode execution counts. Includes the actual number of the
counts. Sorted by frequency. Opcodes not executed are listed below the histogram.

Example: OPCODES

```
LDA   2,188,973  **********************************************
STA      12,123  *****************************
CMP         467  **************
      -----------
        2,201,563
```

Unexecuted opcodes: TXS NOP

OPCODESCLR

Reset opcode histogram table.

Example: OPCODESCLR

----
page1.page2:MEMORYR

Display memory write access table. This table lists on a 256 byte page basis counts
for each time the page was read. If page1.page2 specified then lists only those
pages. If a single page is specified then display only that page's access count.

Example: 0.5:MEMORYR

----
page1.page2:MEMORYW

Display memory read access table. This table lists on a 256 byte page basis counts
for each time the page was written. See MEMORYW for page options.

Example: 0.5:MEMORYW

MEMORYCLR

Reset both memory access tables.

Example: MEMORYCLR

----
value:SCROLL   *if scroll > 0 then cmds showing more than screen of info do not pause for user when screen full*

Set debugger display scrolling rate interline delay. Value is in 1/10th of a second.
Default is no delay (value = 0). Useful if you want to for example dump lots of
memory and don't want to mess with the SPACE key to read what is displayed. Set the
scrolling delay to a comfortable value, sit back, and enjoy the show.

Example: 10:SCROLL   -- sets scrolling delay to 1 second

----
filename:LOG

Close log file, create a new one with filename, and output all debugger displays to
this new file. Useful if you're running the emulator from a write-protected disk and
you want to re-direct the output to a writable disk file.

```
Example: MyDiary:LOG


------------------------------------------------------------
SNAPSHOTW

Write the contents of all of the emulator's memory to binary file on the host
computer's hard disk.  This snapshot could prove useful in diagnosing an emulator
problem.  The binary file should be named "Snapshot_YYYYMMDD_HHMMSS.BIN".

Example: SNAPSHOTW


------------------------------------------------------------
SNAPSHOTRfile-name

Read a snapshot file into the emulator's memory.

Example: SNAPSHOTR Snapshot_19971225_123456.BIN


------------------------------------------------------------
MACRO name commands

Define a macro name and commands for this macro.  You can use any name containing
alphnumeric characters or periods with a maximum length of 31 characters.  Up to 25
macros may be defined.  All commands are verified and if any syntax errors occur you
will be told and the macro will not be defined.  Macro commands cannot include other
macro commands.

Example: MACRO my.dump 300.400 A000.A1FF A000L


------------------------------------------------------------
MACROL

List all defined macros.

Example: MACROL

  #  Name / Contents
  -- ------------------------------
  1  my.dump
     300.400 A000.A1FF A000L


------------------------------------------------------------
!macro-name

Execute a macro with the name "macro-name".  Each command within the macro is
displayed followed by the commands' display.

Example: !my.dump

300.400
...
A000.A1FF
...
A000L
...
```

FONT display current font bitmap   FONTROM ROM font bitmap (handwritten)

```
------------------------------------------------------------
VERSION

Display debugger version information.  Includes version number and creation
date/time.
============================================================
```

17.0   EMULATOR MEMORY STRUCTURE

I recommend that the emulator's internal memory structure for the Apple /// memory resources be structured as follows:

o Memory block containing the size of memory and references to each /// memory bank (the references can be whatever is appropriate -- on the Mac these could be Mac memory pointers or handles):

```
- number of switchable banks (1..15)

- reference to bank  S                        (32K: 0000-1FFF, A000-FFFF) *

- reference to bank  0/$0 - switchable        (32k: 2000-9FFF)
- reference to bank  1/$1 - switchable        (32k: 2000-9FFF)
- ...
- reference to bank 14/$E - switchable        (32k: 2000-9FFF)

- reference to Boot ROM ROM address space     (4k: F000-FFFF)
- reference to Boot ROM RAM address space     (4k: F000-FFFF)

- reference to I/O RAM address space          (4k: C000-CFFF)
```

* The system (S) bank is always on-line and is never bank switched. SOS and part of the interpreter reside here.

o Memory block containing the 6502 registers:

```
- Accumulator         (A)        8 bits
- X index             (X)        8 bits
- Y index             (Y)        8 bits
- Status Register     (P)        8 bits
- Stack Pointer       (S)        8 bits
- Program Counter     (PC)      16 bits
```

o Memory block containing the special /// System Control Registers:

```
- E: Environment Register (FFDF)      8 bits
- Z: Zero Page Register   (FFD0)      8 bits
- B: Bank Register        (FFEF)      8 bits
```

18.0   WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should try to obtain as much /// technical information as possible. The author has lots of info which he can copy at minimal charge (10 cents per page plus postage). These persons should also have access to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does not address. This includes support for other input devices such as the mouse which does have a 3rd party driver available.

19.0   REFERENCES

Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Plus Owner's Guide, Apple Computer, 1982

Apple /// System Data Sheet, Apple Computer, July 1983

Apple /// Plus System Data Sheet, Apple Computer, October 1983

Apple /// Standard Device Drivers Manual, Apple Computer, 1981

Apple /// SOS Reference Manual, Apple Computer, 1982

Apple /// SOS Device Driver Writer's Guide, Apple Computer, 1982

Apple /// Service Reference Manual (Level 2), Apple Computer, 1983

/// Bits: John Jeppson's Guided Tour of Highway ///, Softalk magazine, May 1983

Bank Switch Razzle-Dazzle, Softalk magazine, August 1982

The Apple Nobody Knows, Apple Orchard magazine, Fall 1981

Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981

Inside the Apple /// Computer ROM, David Craig, November 1997

###

Some Ideas about an Apple /// Computer Emulator -- Version 4
David T Craig -- 12 Dec 1997 -- 23 / 23

Apple III Computer Information
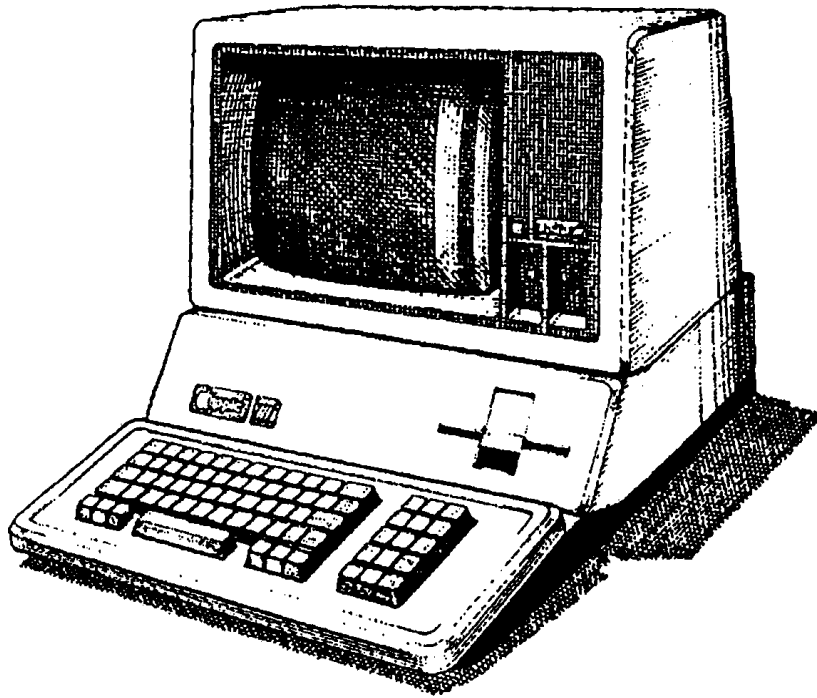
Apple III Emulator Ideas

Version 3 • 09 Dec 1997

SOME IDEAS ABOUT AN APPLE /// COMPUTER EMULATOR

David T. Craig -- 09 December 1997

941 Calle Mejia #1006, Santa Fe, NM 87501 USA
e-mail: 71533.606@compuserve.com

TABLE OF CONTENTS

MODIFICATION HISTORY

28 Nov 1997 -- Version 1
Created by David T. Craig.

04 Dec 1997 -- Version 2
New sections: MONITOR SUPPORT, EMULATOR DEBUGGING FACILITIES.
Updated sections: DISK IMAGES, MEMORY BANK SWITCHING EMULATOR, SOS SYSTEM CALL
EMULATION, REFERENCES.
Added several good comments by Chris Smolinski.

09 Dec 1997 -- Version 3
DISK IMAGES: Updated info about DTCMake3///DiskImage Mac application, made disk
image file an all-text file.
SOS SYSTEM CALL EMULATION: typo Silentypr --> Silentype.
WHAT TARGET MACHINES SHOULD BE SUPPORTED: More pre-68040 Mac comments.
EMULATOR DEBUGGING FACILITIES: typo affects --> affect, added info about
enabling/disabling SOS BRK disassembly, same for ProDOS, added list of emulator
debugging commands.
EMULATOR MEMORY STRUCTURE: New section.

"_138.PICT" 161 KB 2001-08-13 dpi: 300h x 300v pix: 2166h x 3036v

1.0   PURPOSE

This document describes some ideas about implementing a software emulator for the Apple /// computer. These ideas are based on my experiences with the Apple /// computer and its software programming. No specific target machine is mentioned in this document since these ideas should be non-target machine specific. These ideas are submitted to stimulate thought about such an emulator and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS), and /// programs (e.g. AppleWriter ///) are based on my extensive collection of /// technical manuals, specification sheets, and many /// technical articles (Dr. John Jeppson's articles are very exhaustive and full of lots of neat /// techoid stuff). I have around 15 Apple manuals, the majority of which were published by Apple, which include user manuals and the technical programming manuals.

For those people seriously interested in implementing an Apple /// emulator program I highly recommend that they have at least the Apple /// Service Reference Manual. This manual, which is almost 500 pages long, is the definitive reference for how the Apple /// computer works. Most of its contents describe theory of operation even though its title suggests service-type information only. The important features of this manual for a /// emulator writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I programmed the /// many moons ago and have worked professionally as an Apple Macintosh computer programmer since 1984.

Note: All comments are welcome. If you have anything to add or correct please let me know and I will update the master copy of this document.

2.0   EMULATOR GOALS

The /// emulator should provide a complete emulation environment for the faithful execution of Apple /// and /// Plus programs. As far as the emulator user is concerned when they run the emulator program their computer should work just like an Apple /// computer and all // visual fidelity should be maintained. Emulation of the Apple /// Plus computer may also be supported (this means the /// Plus' interlaced screen). If the /// Plus is supported by the emulator you may want to let the user specify if they want to run a /// or a /// Plus.

I think it would be beyond neat if the emulator could run Apple's running horses demo and the other /// demos.

The /// emulator should support an Apple /// computer with at least 256K of memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of memory may also exist since the ///'s operating system (SOS) supports up to 512K of memory. Memory size, if variable, should always be a multiple of 32K. I believe the lowest memory size supported by the /// (ROM?) is 96K. Support for a ProFile disk may also exist (for this disk there would need to be a disk image with a size of 5M). The first floppy disk (.D1) would correspond to the floppy disk drive that is built into the Apple ///. The other disks correspond to external disks and should exist as image files with specific file names (e.g. "Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file should also have a specific file name (e.g. "Apple 3 ProFile").

Image file names should have an extension (e.g. ".D3I") since this is needed by PCs.

3.0   EMULATOR   USER   INTERFACE

When the user runs the Apple /// emulator program the user should see on their computer screen a screen (or a window representing the screen on GUI systems) corresponding to the ///'s screen which the user would see if they were in front of a real Apple /// computer. All /// text and graphic modes should be supported by the /// emulator (this includes the special modes supported by the /// Plus and its interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes the current /// screen to either a text file (for text modes) or to a graphic file (for graphic modes) or always just creates a graphic file. The screen dump graphic file should be a standard graphic file for whatever target machine your support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple Macintosh produce PICT files). Since the /// supports custom character sets dumping the screen to a PICT file (or to the target computer's clipboard) may be the best solution.

The emulator screen if implemented in a GUI window may also display a status area at the bottom of the window. This status area would display at least two lines of text and would keep the user informed of what the emulator was doing internally.

4.0   DISK   IMAGES

The /// emulator should read disk image files which correspond directly to real /// 140K disks. When the /// emulator starts it should look in its folder and if there exists a /// disk image file the emulator should boot this image. If there are multiple disk image files then the emulator may want to display a list of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of these disks there should exist an utility program that runs on the /// computer and which outputs disk block data to the /// serial port (I plan to make this utility and call it DTCDumpIt). This utility's output should be a hex/ascii dump that specifies block numbers and has a checksum for each line of data. This utility should ask the user if it should dump a file or a disk.

On the target machine there should exist a similar utility that inputs the disk block data and creates a disk image file. I recommend that the transmitted disk block data consist of a hex dump with block number and checksum information in a human readable fashion. The receiving program (on the target computer) would read this human readable information, verify that the data was sent correctly, and produce binary disk image file images (I plan to create this utility for the Apple Macintosh and call it DTCMake///DiskImage).

There should also exist a disk image file for the ///'s Boot ROM (recommended file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image. This ROM should be the Revision 1 ROM (not Revision 0) since this was the last ROM produced and SOS 1.3 (the last SOS) requires this ROM.

Users should also be able to format a disk image by specifying the disk drive

device name (e.g. .D2). Users should then be able to name the disk image so that
they can use it later. Users should be able to assign specific disk images to
specific disk drives.

I recommend that all disk image files have a very specific internal format. This
format should support the verification of disk image files so that if a disk
image file becomes corrupted in some fashion the /// emulator can detect this
corruption, not use the image, and alert the user.

Note: Support for existing Apple][disk image files may be feasible but I
recommend against this since the format of these images could change.

The proposed image format:

The disk image file contains two parts, a header part and a data part. The
header part appears first followed by the data part. The header part contains
identification and verification information. The data part contains the actual
disk blocks for the /// disk. This file contains only text, no binary data
appears here in any fashion. The only non-text information that can appear in
these files is the Carriage Return (CR) and the Line Feed (LF) characters. The
emulator should ignore LFs if appropriate. All information appears in lines with
a maximum length of 255 characters. Character case is immaterial. Blank lines
are ignored. The reason for this format is so these image files can be
transferred over the internet without the need for any binary-to-text conversion.
Also, text-only files can easily be viewed by people using a word processor.

The header part contains:

Line	Comments
Signature	"APPLE /// DISK IMAGE"
Version	"VERSION" version number (e.g. "1")
Image Name	"IMAGE NAME" name of image, anything the user wants, most likely the name of the interpreter on the disk, e.g. "Apple Writer ///"
Creation Date	"CREATED" date image file created, "YYYY-MM-DD"
Created by Name	"CREATED BY" name of person or company who created this image
Comment	"COMMENT" comment for anything user wants
Data Size	"DATA SIZE" size of data part (decimal, e.g. "143360")
Data Checksum	"DATA CHECKSUM" hexadecimal checksum (e.g. "FA7C3188")
Reserved 1	"RESERVED"
Reserved 2	"RESERVED"
Reserved 3	"RESERVED"
Reserved 4	"RESERVED"
Tech Comment	"TECH COMMENT" name of program that this is for
Header Checksum	"HEADER CHECKSUM" hexadecimal checksum (e.g. "B97C31D5")

Notes:

The checksum should be calculated as the exclusive-OR of each byte followed by a
left rotation of 1 bit. Checksum starts with zero. Checksums should always be 4
bytes in size and be stored in the header as an 8 character string.

The Tech Comment's purpose is to allow people who obtain an image file to be able
to contact someone about the file's purpose.

"_141.PICT" 237 KB 2001-08-13 dpi: 300h x 300v pix: 2196h x 3030v

The data part contains lines representing 16 bytes from the original disk. Each
line has a specific format which begins with the starting disk address for the
line, 16 bytes, the ASCII equivalent of the 16 bytes, and a checksum for the
bytes of the line with the format:

[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [12345678901234561] 12345678

The last line of the file must be the word "FINIS".

Sample disk image file:

APPLE /// DISK IMAGE
VERSION 1
IMAGE NAME Apple Writer ///
CREATED 1997-10-11
CREATED BY David T. Craig
COMMENT Thanks to Paul Lutus
DATA SIZE 16
DATA CHECKSUM FA7C3188
RESERVED
RESERVED
RESERVED
RESERVED
TECH COMMENT Created for David Craig's /// Emulator - 71533.606@compuserve.com
HEADER CHECKSUM B97C31D5

[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [Apple.///.Emul..] FA7C3188

FINIS

5.0   6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The heart
may be referred to as the "6502 engine." The emulator should support all of the
6502 instructions, the 6502 registers, and the special Apple /// registers (e.g.
the bank switch register, the environment register, and the zero-page register).
Special register descriptions and usage can be found in the Apple /// SOS
Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and
environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the ///
emulator should work much more easily.

Support for special /// features may also exist at this level of the ///
emulator. For example, the /// emulator may not want to emulate all of the ////'s
memory-mapped I/O features, but instead intercept access to special areas or
routines and call the target machine's operating system to handle these features.
See sections ROM EMULATION and MEMORY-MAPPED I/O EMULATION for more details.

6.0   ROM EMULATION

The /// emulator should also support as much as possible the ////'s Boot ROM.
This means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation discussion.

For the Boot ROM's floppy disk I/O support I recommend that all the gory details here not be supported directly at the memory-mapped I/O level but instead the /// emulator should emulate this I/O. Specifically, the /// emulator should intercept any access to the Boot ROM routines which read or write disk blocks and use the appropriate target machine operating system routines to accomplish this feature.

The /// emulator should also initialize the ROM's character set which the ROM normally loads into a special RAM chip that is not accessible to the ///'s 6502 processor. See section MEMORY BANK SWITCHING EMULATION for more details.

7.0   MEMORY-MAPPED   I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world need to be handled by the /// emulator. These areas include such addresses as the speaker addresses. The Apple /// Service Reference Manual provides detailed information about these addresses.

All accesses to memory by the /// emulator must respect the bank switch and environment register settings so that the emulator does not try to access a memory-mapped address when that address is not mapped into the 6502 address space.

Programs which access low-level I/O locations such as the disk I/O addresses should not be supported. I assume most /// programs will access hardware components using SOS or device drivers.

8.0   MEMORY BANK SWITCHING EMULATION

The /// emulator must also fully support the ////'s bank switched and enhanced indirect addressing memory architecture. Detailed descriptions and usage of /// memory handling can be found in the Apple /// SOS Reference Manual.

The /// emulator should also support the ////'s character set RAM chip. This holds the bitmap descriptions of each of the 128 characters in the /// character. This RAM area, which is not accessible to the ////'s 6502 CPU, holds 1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the storage of the character set in the SOS.DRIVER file. I believe the ROM character set has bits that are reversed compared to the SOS.DRIVER character set.

The storage of text and graphics in memory should be supported also. This should happen automatically when a /// program writes to the text/graphic memory buffers. The emulator needs to detect such writes and update its screen as appropriate.

9.0   SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be intercepted by the /// emulator. But certain calls may need to be intercepted unless a lower level of the /// emulator intercepts these feature already.

"_143.PICT" 252 KB 2001-08-13 dpi: 300h x 300v pix: 2196h x 3036v

System calls to SOS or drivers that may need intercepting by the /// emulator
could be:

o Disk I/O   (.D[1-4] and .PROFILE drivers)
o Keyboard I/O   (.CONSOLE driver)
o Screen I/O   (.CONSOLE and .GRAPHIC drivers)
o Sound generation   (.AUDIO driver)
o Serial port I/O   (.RS232 driver)
o Silentype Printer (.SILENTYPE) [I'm not sure about support for this]
o Clock I/O   (Y2K dates may be a problem)

I recommend that the /// emulator intercept all activity dealing with the above
and have the target machine perform the equivalent features. For example, to
read or write a disk block the /// emulator should have a routine that accesses
the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so
that the user can specify specific properties of some of the above drivers. For
example, if the target machine supports several output ports the emulator may let
the user specify which port to use (e.g. for the .PRINTER driver the user could
assign it to a specific serial or parallel port on the target machine).

Note: The ///'s clock does not support the year 2000 or greater. I think the
emulator should support Y2K dates but I'm not sure if SOS's file system date
stamps will support this easily.

10.0   DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM
CALL EMULATION. I suspect the programming within the /// emulator for this area
could be the most work since there are lots of device drivers that make up a
simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is
interrupt handling. Since the emulator does not have physical devices connected
to it in any direct fashion I don't think interrupts exist as far as the emulator
is concerned. Interrupts dealing with disks or the keyboard can be handled at a
lower level by having the /// emulator call the appropriate system call in the
target machine. These low-level I/O handlers should set up the appropriate
driver data areas so that the rest of the ///'s software (SOS and the
interpreter) will work correctly. For example, keyboard I/O should be setup in
the /// emulator so that when the keyboard input memory-mapped I/O location is
accessed the target machine OS really reads the keyboard and sets up the memory-
mapped location as appropriate.

11.0   KEYBOARD SUPPORT

11.1   User interface support

The /// computer's keyboard layout is basically compatible with modern keyboards.
The /// keyboard does have two extra keys, Open Apple and Closed Apple which are
positioned to the left of the Apple /// keyboard. Also present on the keyboard
are four arrow keys. The emulator should support these keys either directly
(i.e., the target machine has similar keys) or associate other keys with the
///'s special keys (e.g., the Macintosh computer's two Option keys could be used
to simulate the special Open and Closed Apple keys). The emulator's associated

keys need not physically be in the same location as the ///'s special keys but having them in the general area will be beneficial.

Note: The /// Plus keyboard contains an extra key, Delete, compared to the /// keyboard.

11.2   Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped I/O locations as detailed in section DEVICE DRIVER EMULATION.

12.0   MONITOR SUPPORT

The emulator should support the Apple's built-in ROM Monitor. Entry to the Monitor should be similar to how this is done on a real /// (at startup if Open Apple and Control keys are pressed). The code in the ROM which tests for Monitor entry should work.

13.0   APPLE][EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple][Emulation Disk. I'm not sure of what would be involved here but suspect that if the ///'s 6502 CPU and the memory-mapped I/O locations are robustly supported that the][emulation should work also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do not exist in the Apple][world.][emulation details can be found in the Apple /// Owner's Guide and the Apple /// Service Reference Manual.

Note: I have a disassembled listing of the Apple][Emulation Disk ROM source listing which could prove useful in this area.

Further analysis of the][emulation disk's boot sequence needs to be done since I'm unknowledgable about this area. Also, I've heard that the][emulation accesses an I/O location which disables some /// features.

14.0   WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I highly recommend that the /// emulator be written in a high level language such as Pascal or C. This should make the emulator more compatible with different target computers and make development and maintenance of the emulator much easier. I recommend avoiding low-level languages such as assembly.

15.0   WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines that are commonly used today by most computer users. This means either the IBM PC or the Apple Macintosh machine family. For the PC world I recommend the /// emulator run under Windows 95 and Windows NT. For the Macintosh world I recommend the emulator run on most Macintosh models which means support the Macintosh 512 and above. Color display should also be supported by the /// emulator (for the Macintosh this means use Color QuickDraw if the machine supports CQD and if CQD is not supported by a Macintosh model use the Classic B/W QD and maybe use patterns as "colors").

Any of these machines should be fast enough to emulate the /// and most likely

will be too fast in many areas. I recommend some type of speed control be built
into the emulator so that users can control how fast the emulator works. For
many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will be
immaterial since these programs typically wait for the user to enter data and
then do their thing. But for programs such as games the user will want to
control the emulator speed otherwise the game's actions will be super fast and
unplayable.

Some people say that the older machines such as pre-68040 Macintoshs will be too
slow for a reasonable /// emulator. I would like to see this /// emulator run on
a Mac 512 machine an onwards. Running on a Mac 128 machine seems a problem due
to this machine's small memory size and should not be supported (if a virtual
memory scheme was used by the emulator the Mac 128 could be supported but I think
having this extra level of support in the emulator would not be worth it). I
disagree and am willing to wager a small sum that I'm right.

16.0   EMULATOR   DEBUGGING   FACILITIES

The emulator should support a comprehensive built-in debugger. This debugger's
purpose should be to let the sophisticated emulator user access any part of the
emulator's /// address space. This should include all of the memory that is
allocated to the /// as its memory. This memory would encompass the 256K (or
512K) of /// RAM, the /// ROM (4K), the character set RAM (1K), the 6502
registers, and the special /// registers (e.g. bank register).

This debugger will prove invaluable in diagnosing emulator bugs.

Logging of all debugger sessions should be stored to a text file for possible
analysis. This text file would be created when the emulator starts. The log
file should be appended to by the emulator. Only the user can delete the file.

The debugger should exist as a separate window that does not in any way affect
the emulator's main window. This window should display only commands that the
user enters or replies returned by the debugger. There should not exist a
separate window area showing things such as the 6502 registers since all such
information should appear in the debugger log file. The window should support at
least 80 columns of text and 24 rows.

The emulator user interface should be based on a simple command line control
scheme. All commands and command outputs should be text-based. This scheme
could be based on the ///'s Monitor's commands or on a little more readable
command scheme such as in Apple's MacsBug debugger. There should be full on-line
help that discusses the debugger commands in general and each command should also
have on-line help available.

The debugger should support the standard debugging commands such as
displaying/setting memory, displaying/setting registers, and disassembling 6502
instructions. This disassembly should support the special SOS BRK call by
listing the word "BRK/SOS" instead of just "BRK" and following this with the SOS
command number/name and the parameter list address:

   SOS   C0/CREATE   345A

The user should be able to enable or disable this feature.

Note: It may be good to also support the Apple // ProDOS command calling scheme

in case this emulator ever becomes an Apple // emulator.

The debugger should **support** break points, single stepping, and timing buckets. The timing buckets would be used in conjunction with break points to record how long a sequence of 6502 instructions took to execute. This can be very useful in locating emulator bottlenecks. The debugger supports many break point commands since I have a feeling that this facility will be very powerful and useful during the emulator's development.

The debugger should support the collection of statistics about the emulator. I recommend tracking how many times specific 6502 opcodes are executed (obviously, the debugger would need commands to display and clear this information). I would also track memory accesses on at least a page (256 bytes) basis.

The debugger should be accessible at any time that the emulator is running. I recommend some type of key press combination that the emulator would detect and display the debugger window. Once the debugger window is active it should remain on the screen until the user closes the window.

The emulator should also support a special key press combination at emulator startup time that activates the debugger just before the /// ROM is run. This can give the emulator developer a good way of tracing ROM execution.

The emulator should activate the debugger if any fatal emulation errors are detected. For example, if a /// program attempts to write to write-protected memory (e.g. SOS's address space) then the debugger window should appear stating the reason for the error.

I recommend the following emulator debugger commands which are based on the /// Monitor commands so that these debugger commands will be familiar to Monitor users. These commands for the most part have the general syntax of address-command. See my document "Inside the Apple /// Computer ROM" for a list of the /// Monitor commands. For information about the Apple][Monitor commands, which the /// Monitor commands are based upon, see "Apple][Reference Manual" (Chapter 3: The System Monitor, dated 1981).

Addresses appearing in debugger commands may be prefaced by "N/" where N is a bank number. For example, to reference address 2000 of bank 4 use 4/2000. If no bank number precedes an address the current bank is used.

To reference a ROM address use a bank "number" of "R", for example "R/F000".

To reference a character set address use a bank "number" of "C", for example "C/0000".

Commands should be case-insensitive.

Commands that display more than a screen full of information should either automatically pause when the screen is full, or the user can use the SPACE key.

Note: Commands using ":" may also use ";" which is easier to type since this character does not need the user of the shift key. Same for "<" and "/".

HELP (or ?)   Display debugger on-line help for all commands. Help info should be
                         stored in an external text file for easier modification.

Some Ideas about an Apple /// Computer Emulator -- Version 3
David T Craig -- 09 Dec 1997 -- 10 / 16

```
BYE                       Return to the emulator.

CARRIAGE  RETURN          Repeat last command.

SPACE                     Pause current command's output.  Press again to continue.

DELETE                    Stop current command's output.

RD                        Display 6502 registers and /// system control registers.

byte:SA                   Set 6502 A register  to byte.
byte:SX                   Set 6502 X register  to byte.
byte:SY                   Set 6502 Y register  to byte.
byte:SP                   Set 6502 P register  to byte.
byte:SS                   Set 6502 S register  to byte.
word:SPC                  Set 6502 PC register  to word.

byte:SE                   Set /// E system control register  to byte.
byte:SZ                   Set /// Z system control register  to byte.
byte:SB                   Set /// B system control register  to byte.

addr1.addr2               Dump memory data to screen  from address 1 to address 2 and display
                          ASCII character  at the right  of the screen. E.g., 300.310  displays  the
                          following (this example  assumes  the current  bank  is bank 4):

                          4/0300-  B900 080A 0A0A 9900   08C8 D0F4 A62B A909 [F..d.uy%^&909..G]

addr:bytes                Store  starting  at the address  the bytes.

addr:'text'               Store  text  starting  at address  (high bit clear).

addr:"text"               Store  text  starting  at address  (high bit set).

addr3<addr1.addr2M        Move  data  in address  range  to address  3.

addr3<addr1.addr2V        Verify  data  in address  range  equals  data  starting  at address  3.

bytes<addr1.addr2S        Search  memory  in address  range  for the bytes.

'text'<addr1.addr2S       Search  memory  in address  range  for text  (high bit clear).

"text"<addr1.addr2S       Search  memory  in address  range  for text  (high bit set).

disk.block<addr1.addr2W   Write  address  range  to disk # disk  starting  at disk block.  If disk #
                          is not present  then uses disk .D1.

disk.block<addr1.addr2R   Read  from disk # disk  starting  at block  to the address  range.  If disk
                          # is not present  then uses disk .D1.

disk.block-block:DISK     Read  block  range  from disk # disk  to a special  debugger  4K buffer
                          which  is not used  by the emulator.  You can then examine  this buffer's
                          contents  either  with a hex/ascii  dump  or with a disassembly  (command
                          L).  This command  is useful  when you want to examine  a disk's
                          contents.  For disassembly  purposes,  you can specify  the logical
                          starting  address  for the buffer.  See the DISKBUFFER  command.
```

To disassemble the special disk buffer (see the L command) use bank S as part of the disassembly address parameter. Same for dumping memory or whatever commands you want to use with this special buffer.

addr:DISKBUFFER   Set disk buffer starting logical address. Default address is 2000. See the DISK command.

addr1.addr2L   Disassemble instructions in address range. If only addr1 appears then disassemble 20 instructions. Disassembly includes the opcode cycle count. A sample disassembly follows:

```
4/0300-    A9 C1       'X.'    (2)    LDA #$C1        ;
4/0302-    20 ED FD    '...'   (5)    JSR $FDED       ;
4/0305-    18          '.'     (2)    CLC             ;
4/0306-    69 0A       'T.'    (4)    ADC #$01        ;
4/0308-    C9 DB       '..'    (3)    CMP #$DB        ;
4/030A-    D0 F6       '..'    (3)    BNE $0302       ;
4/030C-    60          'U'     (4)    RTS             ;

1          2           3       4      5               6  (see Note)
```

Note: Column 1 = bank register/address
         Column 2 = memory bytes
         Column 3 = ASCII for the memory bytes
         Column 4 = opcode cycle count
         Column 5 = disassembled instructions
         Column 6 = remark character ";" (optional, see DISASMREM)

L by itself disassembles the next 20 instructions.

DISASMREM   Display ";" after each disassembly line that is produced by the L command. Default is to not display the remark. Useful if you plan to add comments to a disassembly.

DISASMREMOFF   Turn off DISASMREM.

addrG   Call subroutine at the address.

addrJ   Jump to the address.

wordX   Convert word (or up to 4 hex digits) to hexadecimal, decimal, and binary. Prefix character for byte determines its base: no prefix = hex, . = dec, t = binary. E.g.,

```
AX          ->      A(16)        10(10)    0000 0000 0000 1010(2)
.10X        ->      A(16)        10(10)    0000 0000 0000 1010(2)
t1010       ->      A(16)        10(10)    0000 0000 0000 1010(2)
FFFFX       ->      FFFF(16)     65535(10) 1111 1111 1111 1111(2)
```

addr1.addr2:CS   Calculate and display a checksum for address range. Checksum is a 4 byte quantity which is calculated the same as the disk image file checksums.

addrT   Trace instructions starting at the address. Each traced instruction displays register contents. Press the SPACE to pause the trace, press DELETE to stop the trace.

Some Ideas about an Apple /// Computer Emulator -- Version 3
David T Craig -- 09 Dec 1997 -- 12 / 16

addrSS	Single step trace starting at the address. After each step pause and wait for user to press SPACE to continue or DELETE to stop the single step.
addr:BP	Set a break point at address. When address is accessed the debugger is entered and displays the registers. Up to 100 break points should be supported.
addr:BPC	Clear break point at address.
SOS:BP	Set a break point when a SOS call is made. This means when the BRK opcode is executed. Same as M00:BP.
Mopcode:BP	Set a break point when opcode is executed. E.g., to break when RTS ($60) is executed, M60:BP.
ROM:BP	Set a break point when a call is made to the ROM.
addr1.addr2R:BP	Set a break point when any address within address range is written to.
addr1.addr2W:BP	Set a break point when any address within address range is read from.
addr.byte:BPE	Set a break point when the address contents equal the byte value.
addr.byte1-byte2:BPE	Set a break point when the address contents equal a byte value in the byte range.
addr/b1.b2...:BPEA	Set a break point when the address contents equal byte 1 value, or equals byte 2 value, etc. Supports up to 16 byte values.
addr/byte1-byte2:BPEA	Set a break point when the address contents equal the byte range.
addr1.addr2/b1...:BPEA	Set a break point when the address range contains any bytes equalling the byte values.
addr1.addr2/b1-b2:BPEA	Set a break point when the address range contents equal the byte range.
BPD	Display break point table.
BPC	Clear break point table.
addr1.addr2:TB	Set timing bucket for address range. When address 1 is accessed timing starts. When address 2 is accessed timing stops. Up to 100 timing buckets should be supported.
TBD	Display timing bucket table. Shows all set timing buckets and the time in 1/60th of a second spent in each bucket.
addr:TBC	Clear timing bucket starting at address.
TBC	Clear timing bucket table.
error:SOSE	List SOS general error message for error number. If no error number present then list all general errors. E.g., 01:SOSE -> "BADSCNUM -

	Invalid SOS call number". Error info should be stored in an external text file for easier modification.
error:SOSFE	Display SOS fatal error message for error number. If no error number present then list all fatal errors. E.g., 01:SOSFE -> "BADBRK - Invalid BRK".
command:SOS	Display SOS command name for the command number. If no command number present then list all SOS command numbers and their names. E.g., C0:SOS -> CREATE. Command info should be stored in an external text file for easier modification.
SOSON	Turn on disassembly of SOS calls which displays SOS followed by the command number and parameter address. The emulator defaults to this.
SOSOFF	Turns off SOSON.
disk:CAT	Display catalog of SOS disk stored in disk # disk. Includes recursive list of all subdirectories.

Note: Other commands that may be supported include CATPASCAL for Apple][Pascal disks and CATDOS for Apple][DOS disks. This may come in handy if you want to see what these disks contain if you have them as disk image files. |
disk.file_name:INFO	Displays information about the specified file in the disk. Information includes standard SOS file information but also block list of all index blocks (if any) associated with the file and block list of all data blocks for the file.
disk.block:DUMP	Display contents of specified disk block in the standard hex/ascii dump format.
diskCHKIMAGE	Check validity of disk image in disk # disk. Computes header and data part checksums and compares against the image file's listed checksums.
DIT	Display Driver Information Table (DIT), a data structure maintained by this debugger. Contains list of all loaded drivers, their names, sizes, and entry point addresses.
MIT	Display Memory Information Table (MIT), a data structure maintained by this debugger. See section EMULATOR MEMORY STRUCTURE for what this structure contains.
OPCODES	Display a histogram of opcode execution counts. Includes the actual number of the counts. Sorted by frequency. Opcodes not executed at all appear below the histogram. E.g.,

LDA   2,188,973   ***
STA   12,123   *****************************
CMP   467   **************

Unexecuted opcodes: TXS NOP |
| OPCODESCLR | Reset opcode histogram table. |

page1.page2:MEMORYR	Display memory write access table. This table lists on a 256 byte page basis counts for each time the page was read. If page1.page2 specified then lists only those pages. If a single page is specified then display only that page's access count.
page1.page2:MEMORYW	Display memory read access table. This table lists on a 256 byte page basis counts for each time the page was written. See MEMORYW for page options.
MEMORYCLR	Reset both memory access tables.
value:SCROLL	Set debugger display scrolling rate interline delay. Value is in 1/10th of a second. Default is no delay. Useful if you want to for example dump lots of memory and don't want to mess with the SPACE key to read what is displayed. Set the scrolling delay to a comfortable value, sit back, and enjoy the show.
filename:LOG	Close log file, create a new one with filename, and output all debugger displays to this new file. Useful if you're running the emulator from a write-protected disk and you want to re-direct the output to a writable disk file.
VERSION	Display debugger version information. Includes version number and creation date/time.

17.0   EMULATOR MEMORY STRUCTURE

I recommend that the emulator's internal memory structure for the Apple /// memory resources be structured as follows:

o Memory block containing the size of memory and references to each /// memory bank (the references can be whatever is appropriate -- on the Mac these could be Mac memory pointers or handles):

```
    - number of switchable banks (1..15)

    - reference to bank   S                    (32K: 0000-1FFF, A000-FFFF) *

    - reference to bank   0/$0 - switchable    (32K: 2000-9FFF)
    - reference to bank   1/$1 - switchable    (32K: 2000-9FFF)
    - ...
    - reference to bank 14/$E - switchable     (32K: 2000-9FFF)

    - reference to Boot ROM ROM address space  (4k: F000-FFFF)
    - reference to Boot ROM RAM address space  (4k: F000-FFFF)

    - reference to I/O RAM address space       (4k: C000-CFFF)

    * The system (S) bank is always on-line and is never bank switched.
      SOS and part of the interpreter reside here.
```

o Memory block containing the 6502 registers:

```
    - Accumulator        (A)      8 bits
    - X index            (X)      8 bits
    - Y index            (Y)      8 bits
```

```
        - Status Register      (P)        8 bits
        - Stack Pointer        (S)        8 bits
        - Program Counter      (PC)      16 bits

  o  Memory block containing the special /// System Control Registers:

        - E: Environment Register   (FFDF)    8 bits
        - Z: Zero Page Register     (FFD0)    8 bits
        - B: Bank Register          (FFEF)    8 bits
```

18.0   WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should try
to obtain as much /// technical information as possible. The author has lots of
info which he can copy at minimal charge. These persons should also have access
to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does
not address. This includes support for other input devices such as the mouse
which does have a 3rd party driver available.

19.0   REFERENCES

Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Plus Owner's Guide, Apple Computer, 1982

Apple /// System Data Sheet, Apple Computer, July 1983

Apple /// Plus System Data Sheet, Apple Computer, October 1983

Apple /// Standard Device Drivers Manual, Apple Computer, 1981

Apple /// SOS Reference Manual, Apple Computer, 1982

Apple /// SOS Device Driver Writer's Guide, Apple Computer, 1982

Apple /// Service Reference Manual (Level 2), Apple Computer, 1983

/// Bits: John Jeppson's Guided Tour of Highway ///, Softalk magazine, May 1983

Bank Switch Razzle-Dazzle, Softalk magazine, August 1982

The Apple Nobody Knows, Apple Orchard magazine, Fall 1981

Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981

Inside the Apple /// Computer ROM, David Craig, November 1997

###

Apple III Computer Information

Apple III Emulator Ideas

Version 2   •   04 Dec 1997

SOME IDEAS ABOUT AN APPLE /// COMPUTER EMULATOR

David T. Craig -- 04 December 1997

941 Calle Mejia #1006, Santa Fe, NM 87501 USA
e-mail: 71533.606@compuserve.com

TABLE OF CONTENTS

MODIFICATION HISTORY

28 Nov 1997 -- Version 1

Created by David T. Craig

04 Dec 1997 -- Version 2

Created several new sections:
12.0 MONITOR SUPPORT, 16.0 EMULATOR DEBUGGING FACILITIES
Updated several sections:
DISK IMAGES, MEMORY BANK SWITCHING EMULATOR, SOS SYSTEM CALL EMULATION,
REFERENCES
Added several good comments by Chris Smolinski

changes : p.2, p.8, p.9, p.6, p.3

Some Ideas about an Apple /// Computer Emulator -- Version 2
David T Craig -- 04 Dec 1997 -- 1 / 10

1.0 PURPOSE

This document describes some ideas about implementing a software emulator for the Apple /// computer. These ideas are based on my experiences with the Apple /// computer and its software programming. No specific target machine is mentioned in this document since these ideas should be non-target machine specific. These ideas are submitted to stimulate thought about such an emulator and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS), and /// programs (e.g. AppleWriter ///) are based on my extensive collection of /// technical manuals, specification sheets, and many /// technical articles (Dr. John Jeppson's articles are very exhaustive and full of lots of neat /// techoid stuff). I have around 15 Apple manuals, the majority of which were published by Apple, which include user manuals and the technical programming manuals.

For those people seriously interested in implementing an Apple /// emulator program I highly recommend that they have at least the Apple /// Service Reference Manual. This manual, which is almost 500 pages long, is the definitive reference for how the Apple /// computer works. Most of its contents describe theory of operation even though its title suggests service-type information only. The important features of this manual for a /// emulator writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I programmed the /// many moons ago and have worked professionally as an Apple Macintosh computer programmer since 1984.

NOTE: All comments are welcome. If you have anything to add or correct please let me know and I will update the master copy of this document.

2.0 EMULATOR GOALS

The /// emulator should provide a complete emulation environment for the faithful execution of Apple /// and /// Plus programs. As far as the emulator user is concerned when they run the emulator program their computer should work just like an Apple /// computer and all // visual fidelity should be maintained. Emulation of the Apple /// Plus computer may also be supported (this means the /// Plus' interlaced screen). If the /// Plus is supported by the emulator you may want to let the user specify if they want to run a /// or a /// Plus.

I think it would be beyond neat if the emulator could run Apple's running horses demo and the other /// demos.

The /// emulator should support an Apple /// computer with at least 256K of memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of memory may also exist since the ///'s operating system (SOS) supports up to 512K of memory. Memory size, if variable, should always be a multiple of 32K. I believe the lowest memory size supported by the /// (ROM?) is 96K. Support for a ProFile disk may also exist (for this disk there would need to be a disk image with a size of 5M). The first floppy disk (.D1) would correspond to the floppy disk drive that is built into the Apple ///. The other disks correspond to external disks and should exist as image files with specific file names (e.g.

"Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file should also have
a specific file name (e.g. "Apple 3 ProFile").

Image file names should have an extension (e.g. ".D3I") since this is needed by
PCs.

3.0   EMULATOR   USER   INTERFACE

When the user runs the Apple /// emulator program the user should see on their
computer screen a screen (or a window representing the screen on GUI systems)
corresponding to the ///'s screen which the user would see if they were in front
of a real Apple /// computer. All /// text and graphic modes should be
supported by the /// emulator (this includes the special modes supported by the
/// Plus and its interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes
the current /// screen to either a text file (for text modes) or to a graphic
file (for graphic modes) or always just creates a graphic file. The screen dump
graphic file should be a standard graphic file for whatever target machine your
support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple
Macintosh produce PICT files). Since the /// supports custom character sets
dumping the screen to a PICT file (or to the target computer's clipboard) may be
the best solution.

The emulator screen if implemented in a GUI window may also display a status
area at the bottom of the window. This status area would display at least two
lines of text and would keep the user informed of what the emulator was doing
internally.

4.0   DISK   IMAGES

The /// emulator should read disk image files which correspond directly to real
/// 140K disks. When the /// emulator starts it should look in its folder and
if there exists a /// disk image file the emulator should boot this image. If
there are multiple disk image files then the emulator may want to display a list
of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of
these disks there should exist an utility program that runs on the /// computer
and which outputs disk block data to the /// serial port (I plan to make this
utility and call it DTCDumpIt). This utility's output should be a hex/ascii
dump that specifies block numbers and has a checksum for each line of data.
This utility should ask the user if it should dump a file or a disk.

On the target machine there should exist a similar utility that inputs the disk
block data and creates a disk image file. I recommend that the transmitted disk
block data consist of a hex dump with block number and checksum information in a
human readable fashion. The receiving program (on the target computer) would
read this human readable information, verify that the data was sent correctly,
and produce binary disk image file images (I plan to create this utility ~~for my~~
~~Apple /// and~~ for ~~my~~ Apple Macintosh and call it DTCMake3DiskImage).
               the   &///

There should also exist a disk image file for the ///'s Boot ROM (recommended
file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image.
This ROM should be the Revision 1 ROM (not Revision 0) since this was the last

ROM produced and SOS 1.3 (the last SOS) requires this ROM.

Users should also be able to format a disk image by specifying the disk drive
device name (e.g. .D2). Users should then be able to name the disk image so
that they can use it later. Users should be able to assing specific disk images
to specific disk drives.

assign

I recommend that all disk image files have a very specific internal format.
This format should support the verification of disk image files so that if a
disk image file becomes corrupted in some fashion the /// emulator can detect
this corruption, not use the image, and alert the user. The proposed image
format is as follows:

header appears 1st, then data

The disk image file contains two parts, a header part and a data part. The
header part contains identification and verification information. The data part
contains the actual disk blocks for the /// disk. The header part is 512 bytes
in size.

The header part contains (size is in bytes);

simpler
(for internet exchanging)
change to all text, no endian
format, no length byte in strings,
sizes/checksums as 8 digit
hex #s

Field	Size	Comments
Signature *e*	16	Contains "APPLE /// IMAGE "
Endian Format	2	Contains either "LL" or "BB" [0]
Version Number	2	Contains "11" for version 1, "22" for version 2, ...
Image Name	128	Name of image, anything the user wants, most likely the name of the interpreter on the disk, e.g. "Apple Writer ///" [1]
Creation Date	10	Date image file created, format "YYYY-MM-DD"
Modification Date	10	Date image file modified, format "YYYY-MM-DD"
Created by Name	80	Name of person or company who created this image [1]
Comment	128	Comment for anything user wants [1]
Data Part Size	4	Size of data part
Data Part Checksum	4	Checksum for data part [2]
Reserved	130	Reserved for future use, fill with hex $00
Header Checksum	4	Checksum for header part [2]

Support CR-LF

file ends w/ "FINIS"

Notes: *e*

Store data as a hex dump: 00000000: XX XX·· XX [ASCII] CCS]
groups of 2 bytes, 16 bytes per line

[0] The endian format specifies the byte ordering within 2 and 4 byte integer
values. "LL" stands for "Little Endian" and "BB" stands for "Big Endian." This
is needed so disk images created on one type of machine (e.g. IBM PC - Little
Endian) can be correctly read by other machines (e.g. Apple Macintosh - Big
Endian).

[1] All strings are stored in the header as Pascal strings which begin with a
single byte specifying the length in bytes of the string's contents. Pad unused
characters with blanks. *change — string padded w/ spaces*

[2] The checksum should be calculated as the exclusive-OR of each byte followed
by a left rotation of 1 bit. Checksum starts with zero.

Support for existing Apple][disk image files may be feasible but I recommend
against this since the format of these images could change.

5.0   6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The
heart may be referred to as the "6502 engine." The emulator should support all
of the 6502 instructions, the 6502 registers, and the special Apple ///
registers (e.g. the bank switch register, the environment register, and the
zero-page register). Special register descriptions and usage can be found in
the Apple /// SOS Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and
environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the ///
emulator should work much more easily.

Support for special /// features may also exist at this level of the ///
emulator. For example, the /// emulator may not want to emulate all of the
///'s memory-mapped I/O features, but instead intercept access to special areas
or routines and call the target machine's operating system to handle these
features. See sections ROM EMULATION and MEMORY-MAPPED I/O EMULATION for more
details.

6.0   ROM EMULATION

The /// emulator should also support as much as possible the ///'s Boot ROM.
This means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation
discussion.

For the Boot ROM's floppy disk I/O support I recommend that all the gory details
here not be supported directly at the memory-mapped I/O level but instead the
/// emulator should emulate this I/O. Specifically, the /// emulator should
intercept any access to the Boot ROM routines which read or write disk blocks
and use the appropriate target machine operating system routines to accomplish
this feature.

The /// emulator should also initialize the ROM's character set which the ROM
normally loads into a special RAM chip that is not accessible to the ///'s 6502
processor. See section MEMORY BANK SWITCHING EMULATION for more details.

7.0   MEMORY-MAPPED   I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world
need to be handled by the /// emulator. These areas include such addresses as
the speaker addresses. The Apple /// Service Reference Manual provides detailed
information about these addresses.

All accesses to memory by the /// emulator must respect the bank switch and
environment register settings so that the emulator does not try to access a
memory-mapped address when that address is not mapped into the 6502 address
space.

Programs which access low-level I/O locations such as the disk I/O addresses
should not be supported. I assume most /// programs will access hardware

components using SOS or device drivers.

8.0   MEMORY BANK SWITCHING   EMULATION

The /// emulator must also fully support the ////'s bank switched and enhanced
indirect addressing memory architecture. Detailed descriptions and usage of ///
memory handling can be found in the Apple /// SOS Reference Manual.

The /// emulator should also support the ////'s character set RAM chip. This RAM
holds the bitmap descriptions of each of the 128 characters in the ///
character. This RAM area, which is not accessible to the ////'s 6502 CPU, holds
1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character
Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the
storage of the character set in the SOS.DRIVER file. I believe the ROM
character set has bits that are reversed compared to the SOS.DRIVER character
set.

The storage of text and graphics in memory should be supported also. This
should happen automatically when a /// program writes to the text/graphic memory
buffers. The emulator needs to detect such writes and update its screen as
appropriate.

9.0   SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be
intercepted by the /// emulator. But certain calls may need to be intercepted
unless a lower level of the /// emulator intercepts these feature already.
System calls to SOS or drivers that may need intercepting by the /// emulator
could be:

o Disk I/O   (.D[1-4] and .PROFILE drivers)
o Keyboard I/O   (.CONSOLE driver)
o Screen I/O   (.CONSOLE and .GRAPHIC drivers)
o Sound generation   (.AUDIO driver)
o Serial port I/O   (.RS232 driver)
o Silentype Printer (.SILENTYPE) [I'm not sure about support for this]
o Clock I/O   (Y2K dates may be a problem)

I recommend that the /// emulator intercept all activity dealing with the above
and have the target machine perform the equivalent features. For example, to
read or write a disk block the /// emulator should have a routine that accesses
the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so
that the user can specify specific properties of some of the above drivers. For
example, if the target machine supports several output ports the emulator may
let the user specify which port to use (e.g. for the .PRINTER driver the user
could assign it to a specific serial or parallel port on the target machine).

Note: The ////'s clock does not support the year 2000 or greater. I think the
emulator should support Y2K dates but I'm not sure if SOS's file system date
stamps will support this easily.

10.0   DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM
CALL EMULATION. I suspect the programming within the /// emulator for this area
could be the most work since there are lots of device drivers that make up a
simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is
interrupt handling. Since the emulator does not have physical devices connected
to it in any direct fashion I don't think interrupts exist as far as the
emulator is concerned. Interrupts dealing with disks or the keyboard can be
handled at a lower level by having the /// emulator call the appropriate system
call in the target machine. These low-level I/O handlers should set up the
appropriate driver data areas so that the rest of the ///'s software (SOS and
the interpreter) will work correctly. For example, Keyboard I/O should be setup
in the /// emulator so that when the keyboard input memory-mapped I/O location
is accessed the target machine OS really reads the keyboard and sets up the
memory-mapped location as appropriate.

11.0   KEYBOARD SUPPORT

11.1   User interface support

The /// computer's keyboard layout is basically compatible with modern
keyboards. The /// keyboard does have two extra keys, Open Apple and Closed
Apple which are positioned to the left of the Apple /// keyboard. Also present
on the keyboard are four arrow keys. The emulator should support these keys
either directly (i.e., the target machine has similar keys) or associate other
keys with the ///'s special keys (e.g., the Macintosh computer's two Option keys
could be used to simulate the special Open and Closed Apple keys). The
emulator's associated keys need not physically be in the same location as the
///'s special keys but having them in the general area will be beneficial.

NOTE: The /// Plus keyboard contains an extra key, Delete, compared to the ///
keyboard.

11.2   Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped
I/O locations as detailed in section DEVICE DRIVER EMULATION.

12.0   MONITOR SUPPORT

The emulator should support the Apple's built-in ROM Monitor. Entry to the
Monitor should be similar to how this is done on a real /// (at startup if Open
Apple and Control keys are pressed). The code in the ROM which tests for
Monitor entry should work.

13.0   APPLE][EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple][Emulation Disk. I'm
not sure of what would be involved here but suspect that if the ///'s 6502 CPU
and the memory-mapped I/O locations are robustly supported that the][emulation
should work also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do not exist in the Apple][world.][emulation details can be found in the Apple /// Owner's Guide and the Apple /// Service Reference Manual.

Note: I have a disassembled listing of the Apple][Emulation Disk ROM source listing which could prove useful in this area.

Further analysis of the][emulation disk's boot sequence needs to be done since I'm unknowledgable about this area. Also, I've heard that the][emulation accesses an I/O location which disables some /// features.

14.0   WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I highly recommend that the /// emulator be written in a high level language such as Pascal or C. This should make the emulator more compatible with different target computers and make development and maintenance of the emulator much easier. I recommend avoiding low-level languages such as assembly.

15.0   WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines that are commonly used today by most computer users. This means either the IBM PC or the Apple Macintosh machine family. For the PC world I recommend the /// emulator run under Windows 95 and Windows NT. For the Macintosh world I recommend the emulator run on most Macintosh models which means support the Macintosh 512 and above. Color display should also be supported by the /// emulator (for the Macintosh this means use Color QuickDraw if the machine supports CQD and if CQD is not supported by a Macintosh model use the Classic B/W QD and maybe use patterns as "colors").

Any of these machines should be fast enough to emulate the /// and most likely will be too fast in many areas. I recommend some type of speed control be built into the emulator so that users can control how fast the emulator works. For many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will be immaterial since these programs typically wait for the user to enter data and then do their thing. But for programs such as games the user will want to control the emulator speed otherwise the game's actions will be super fast and unplayable.

Some people say that the older machines such as the pre-68040 Macintoshs will be too slow for a reasonable /// emulator. I disagree and am willing to wager a small sum that I'm right. *I would like to see emul run on a Mac 512 machine.*

16.0   EMULATOR DEBUGGING FACILITIES   *add list of commands — base on MacsBug*

The emulator should support a comprehensive built-in debugger. This debugger's purpose should be to let the sophisticated emulator user access any part of the emulator's /// address space. This should include all of the memory that is allocated to the /// as its memory. This memory would encompass the 256K (or 512K) of /// RAM, the /// ROM (4K), the character set RAM (1K), the 6502 registers, and the special /// registers (e.g. bank register).

This debugger could prove invaluable in diagnosing emulator bugs.

The emulator should exist as a seperate window that does not in any way affects

the emulator's main window.

The emulator user interface should be based on ~~a simple~~ command line ~~control scheme.~~ All commands and command outputs should be text-based. This scheme could be based on the ///'s Monitor's commands or on a little more readable command scheme such as in Apple's MacsBug debugger. There should be full on-line help that discusses the debugger commands in general and each command should also have on-line help available.

Logging of all debugger sessions should be stored to a text file for possible *future* analysis. This text file would be created when the emulator started. The log file should be appended to by the emulator and only the user can delete the file.

The debugger should support the standard debugging commands such as displaying/setting memory, displaying/setting registers, and disassembling 6502 instructions (this disassembly should support the special SOS BRK call). *ProDOS?.   command to turn this ON/OFF*
The debugger should support break points, single stepping, and timing buckets. The timing buckets would be used in conjunction with break points to record how long a sequence of 6502 instructions took to execute. This can be very useful in locating emulator bootlenecks.

The debugger should support the collection of statistics about the emulator. I recommend tracking how many times specific 6502 opcodes are executed (obviously, the debugger would need commands to display and clear this information). I would also track memory accesses on at least a page (256 bytes) basis.

The debugger should be accessible at any time that the emulator is running. I recommend some type of key press combination that the emulator would detect and display the debugger window. Once the debugger window is active it should remain on the screen until the user closes the window.

The emulator should also support a special key press combination at emulator startup time that activates the debugger just before the /// ROM is run. This can give the emulator developer a good way of tracing ROM execution.

The emulator should activate the debugger if any fatal emulation errors are detected. For example, if a /// program attempts to write to write-protected memory (e.g. SOS's address space) then the debugger window should appear stating the reason for the error.

17.0   WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should try to obtain as much /// technical information as possible. The author has lots of info which he can copy at minimal charge. These persons should also have access to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does not address. This includes support for other input devices such as the mouse which does have a 3rd party driver available.
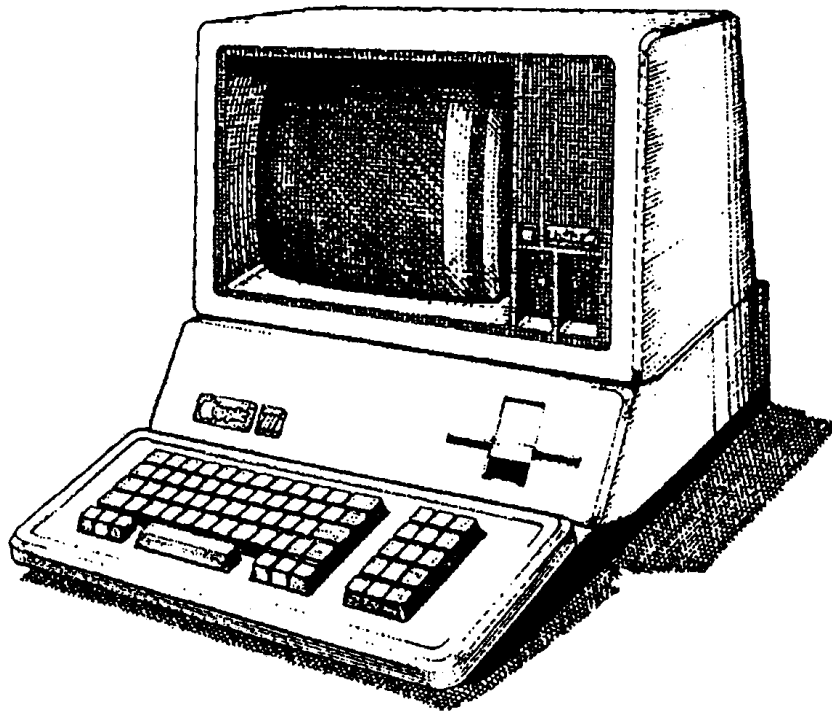
18.0   REFERENCES

Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Plus Owner's Guide, Apple Computer, 1982

Apple /// System Data Sheet, Apple Computer, July 1983

Apple /// Plus System Data Sheet, Apple Computer, October 1983

Apple /// Standard Device Drivers Manual, Apple Computer, 1981

Apple /// SOS Reference Manual, Apple Computer, 1982

Apple /// SOS Device Driver Writer's Guide, Apple Computer, 1982

Apple /// Service Reference Manual (Level 2), Apple Computer, 1983

/// Bits: John Jeppson's Guided Tour of Highway ///, Softalk magazine, May 1983

Bank Switch Razzle-Dazzle, Softalk magazine, August 1982

The Apple Nobody Knows, Apple Orchard magazine, Fall 1981

Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981

Inside the Apple /// Computer ROM, David Craig, November 1997

###

Apple III Computer Information

Apple III Emulator Ideas

Version 1 • 28 Nov 1997

Some Ideas about an Apple /// Computer Emulator

David T. Craig -- 28 November 1997

941 Calle Mejia #1006, Santa Fe, NM 87501 USA
e-mail: 71533.6062compuserve.com

TABLE OF CONTENTS

See Chris G's 30 Nov email comments

13.0 MONITOR SUPPORT

16.0 EMULATOR DEBUGGING FACILITIES

ERRORS: 4.0, 8.0, 9.0, 16.0

MOD/ADD:
- /// clock does not handle years 2000+
/// FS date stamp may also not support 2000+

-2.0: .IMG extension
-2.0: Mem range, multiple of 32k
- 16.0: new section #18.0
-4.0: DTCDump.It explain (///)
DTC Make 3D VolImage (Mac)
checksums - keep at 4 bytes
Pad PSTRINGS w/ blanks
-9.0 Silentype driver
-12.0 refs for AII emul

MODIFICATION HISTORY
----

28 Nov 1997 -- Version 1 -- Created by David T. Craig

1.0   PURPOSE

This document describes some ideas about implementing a software emulator for
the Apple /// computer. These ideas are based on my experiences with the Apple
/// computer and its software programming. No specific target machine is
mentioned in this document since these ideas should be non-target machine
specific. These ideas are submitted to stimulate thought about such an emulator
and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS),
and /// programs (e.g. AppleWriter ///) are based on my extensive collection of
/// technical manuals, specification sheets, and many /// technical articles
(Dr. John Jeppson's articles are very exhaustive and full of lots of neat ///
techoid stuff). I have around 15 Apple manuals, the majority of which were
published by Apple, which include user manuals and the technical programming
manuals.

For those people seriously interested in implementing an Apple /// emulator
program I highly recommend that they have at least the Apple /// Service
Reference Manual. This manual, which is almost 500 pages long, is the
definitive reference for how the Apple /// computer works. Most of its contents
describe theory of operation even though its title suggests service-type
information only. The important features of this manual for a /// emulator
writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I
programmed the /// many moons ago and have worked professionally as an Apple
Macintosh computer programmer since 1984.

NOTE
All comments are welcome. If you have anything to add or correct please let me
know and I will update the master copy of this document.

2.0   EMULATOR   GOALS

image file extensions (eg ".IMG")?

The /// emulator should provide a complete emulation environment for the
faithful execution of Apple /// and /// Plus programs. As far as the emulator
user is concerned when they run the emulator program their computer should work
just like an Apple /// computer and all // visual fidelity should be maintained.
Emulation of the Apple /// Plus computer may also be supported (this means the
/// Plus' interlaced screen). If the /// Plus is supported by the emulator you
may want to let the user specify if they want to run a /// or a /// Plus.

mem should be multiple of 32K in range 128K-512K

The /// emulator should support an Apple /// computer with at least 256K of
memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of
memory may also exist since the ///'s operating system (SOS) supports up to 512K
of memory. Support for a ProFile disk may also exist (for this disk there would
need to be a disk image with a size of 5M). The first floppy disk (.D1) would
correspond to the floppy disk drive that is built into the Apple ///. The other
disks correspond to external disks and should exist as image files with specific
file names (e.g. "Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file
should also have a specific file name (e.g. "Apple 3 ProFile").

3.0   EMULATOR   USER   INTERFACE

Some Ideas about an Apple /// Computer Emulator (version 1) - DTC - 28 Nov 1997 - 2 / 8

"_167.PICT" 288 KB 2001-08-13 dpi: 300h x 300v pix: 2243h x 2941v

When the user runs the Apple /// emulator program the user should see on their
computer screen a screen (or a window representing the screen) corresponding to
the ///'s screen which the user would see if they were in front of a real Apple
/// computer. All /// text and graphic modes should be supported by the ///
emulator (this includes the special modes supported by the /// Plus and its
interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes
the current /// screen to either a text file (for text modes) or to a graphic
file (for graphic modes) or always just creates a graphic file. The screen dump
graphic file should be a standard graphic file for whatever target machine your
support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple
Macintosh produce PICT files).

4.0   DISK IMAGES   in

The /// emulator should read disk image files which correspond directly to real
/// 140K disks. When the /// emulator starts it should look its folder and if
there exists a /// disk image file the emulator should boot this image. If
there are multiple disk image files then the emulator may want to display a list
of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of
these disks there should exist an utility program that runs on the /// computer
and which outputs disk block data to the /// serial port. On the target machine
there should exist a similar utility that inputs the disk block data and creates
a disk image file. I recommend that the transmitted disk block data consist of
a hex dump with block number and checksum information in a human readable
fashion. The receiving program (on the target computer) would read this human
readable information, verify that the data was sent correctly, and produce
binary disk image file images. Note: I plan to create such a utility for my
Apple /// and for my Apple Macintosh. /// app: DTC DumpThing.TEXT (DTC Dump It.TEXT)
                                                          DTC Make 3 Disk Image
There should also exist a disk image file for the ///'s Boot ROM (recommended
file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image.

Users should also be able to format a disk image by specifying the disk drive
device name (e.g. .D2). Users should then be able to name the disk image so
that they can use it later.

I recommend that all disk image files have a very specific internal format.
This format should support the verification of disk image files so that if a
disk image file becomes corrupted in some fashion the /// emulator can detect
this corruption, not use the image, and alert the user. The proposed image
format is as follows:

The disk image file contains two parts, a header part and a data part. The
header part contains identification and verification information. The data part
contains the actual disk blocks for the /// disk. The header part is 512 bytes
in size.

The header part contains (size is in bytes);

Field   Size Comments
------------------   ---- --
Signature   16 Contains "APPLE /// IMAGE "
                                        ← version # (2 bytes)   over

"_168.PICT" 279 KB 2001-08-13 dpi: 300h x 300v pix: 2154h x 2995v

```
Endian Format         2 Contains either "LL" or "BB" [0]        version # (2 bytes)
                                                                 $AAAA for v.1
Image Name          128 Name of image anything the user wants,   $BBBB for v.2
                        most likely the name of the interpreter on the disk,
                        e.g. "Apple Writer ///" [1]
Creation Date        10 Date image file created, format "YYYY-MM-DD"
Modification Date    10 Date image file modified, format "YYYY-MM-DD"
Created by Name      80 Name of person or company who created this image  [1]
Comment             128 Comment for anything user wants  [1]
Data Part Size        4 Size of data part
Data Part Checksum    4 Checksum for data part [2]
Reserved            132 Reserved for future use, fill with hex $00
Header Checksum       4 Checksum for header part [2]
```

Notes:

[0] The endian format specifies the byte ordering within 2 and 4 byte integer
values. "LL" stands for "Little Endian" and "BB" stands for "Big Endian." This
is needed so disk images created on one type of machine (e.g. IBM PC - Little
Endian) can be correctly read by other machines (e.g. Apple Macintosh - Big
Endian).

[1] All strings are stored in the header as Pascal strings which begin with a
single byte specifying the length in bytes of the string's contents. pad w/ blanks

[2] The checksum should be calculated as the exclusive-OR of each byte followed
by a left rotation of 1 bit. CS starts at Ø.

5.0   6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The
heart may be referred to as the "6502 engine." The emulator should support all
of the 6502 instructions, the 6502 registers, and the special Apple ///
registers (e.g. the bank switch register, the environment register, and the
zero-page register). Special register descriptions and usage can be found in
the Apple /// SOS Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and
environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the ///
emulator should work much more easily.

Support for special /// features may also exist at this level of the ///
emulator. For example, the /// emulator may not want to emulate all of the
///'s memory-mapped I/O features, but instead intercept access to special areas
or routines and call the target machine's operating system to handle these
features. See sections ROM EMULATION and MEMORY-MAPPED I/O EMULATION for more
details.

6.0   ROM EMULATION

The /// emulator should also support as much as possible the ///'s Boot ROM.
This means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation
discussion.

Some Ideas about an Apple /// Computer Emulator (version 1) - DTC - 28 Nov 1997 - 4 / 8

For the Boot ROM's floppy disk I/O support I recommend that all the gory details here not be supported directly at the memory-mapped I/O level but instead the /// emulator should emulate this I/O. Specifically, the /// emulator should intercept any access to the Boot ROM routines which read or write disk blocks and use the appropriate target machine operating system routines to accomplish this feature.

The /// emulator should also initialize the ROM's character set which the ROM normally loads into a special RAM chip that is not accessible to the ///'s 6502 processor. See section MEMORY BANK SWITCHING EMULATION for more details.

7.0   MEMORY-MAPPED   I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world need to be handled by the /// emulator. These areas include such addresses as the speaker addresses. The Apple /// Service Reference Manual provides detailed information about these addresses.

All accesses to memory by the /// emulator must respect the bank switch and environment register settings so that the emulator does not try to access a memory-mapped address when that address is not mapped into the 6502 address space.

8.0   MEMORY BANK SWITCHING   EMULATION

The /// emulator must also fully support the ///'s bank switched and enhanced indirect addressing memory architecture. The emulator's 6502 engine ?

Detailed descriptions and usage of /// memory handling can be found in the Apple /// SOS Reference Manual.

The /// emulator should also support the ///'s character set RAM chip. This holds the bitmap descriptions of each of the 128 characters in the /// character. This RAM area, which is not accessible to the ///'s 6502 CPU, holds 1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the storage of the character set in the SOS.DRIVER file. I believe the ROM character set has bits that are reversed compared to the SOS.DRIVER character set.

9.0   SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be intercepted by the /// emulator. But certain calls may need to be intercepted unless a lower level of the /// emulator intercepts these feature already. System calls to SOS or drivers that may need intercepting by the /// emulator could be:

o Disk I/O   (.D[1-4] and .PROFILE drivers)
o Keyboard I/O   (.CONSOLE driver)
o Screen I/O   (.CONSOLE and .GRAPHIC drivers)
o Sound generation (.AUDIO driver)
o Serial port I/O   (.RS232 driver)

SILENTYPE?.

Some Ideas about an Apple /// Computer Emulator (version 1) - DTC - 28 Nov 1997 - 5 / 8

|

I recommend that the /// emulator intercept all activity dealing with the above and have the target machine perform the equivalent features. For example, to read or write a disk block the /// emulator should have a routine that accesses the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so that the user can specify specific properties of some of the above drivers. For example, if the target machine supports several output ports the emulator may let the user specify which port to use (e.g. for the .PRINTER driver the user could assign it to a specific serial or parallel port on the target machine).

10.0   DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM CALL EMULATION. I suspect the programming within the /// emulator for this area could be the most work since there are lots of device drivers that make up a simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is interrupt handling. Since the emulator does not have physical devices connected to it in any direct fashion I don't think interrupts exist as far as the emulator is concerned. Interrupts dealing with disks or the keyboard can be handled at a lower level by having the /// emulator call the appropriate system call in the target machine. These low-level I/O handlers should set up the appropriate driver data areas so that the rest of the ///'s software (SOS and the interpreter) will work correctly. For example, keyboard I/O should be setup in the /// emulator so that when the keyboard input memory-mapped I/O location is accessed the target machine OS really reads the keyboard and sets up the memory-mapped location as appropriate.

11.0   KEYBOARD SUPPORT

11.1   User interface support

The /// computer's keyboard layout is basically compatible with modern keyboards. The /// keyboard does have two extra keys, Open Apple and Closed Apple which are positioned to the left of the Apple /// keyboard. Also present on the keyboard are four arrow keys. The emulator should support these keys either directly (i.e., the target machine has similar keys) or associate other keys with the ///'s special keys (e.g., the Macintosh computer's two Option keys could be used to simulate the special Open and Closed Apple keys). The emulator's associated keys need not physically be in the same location as the ///'s special keys but having them in the general area will be beneficial.

Apple /// Plus Note: The /// Plus keyboard contains an extra key, Delete, compared to the /// keyboard.

11.2   Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped I/O locations as detailed in section DEVICE DRIVER EMULATION.

12.0   APPLE][EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple][Emulation Disk. I'm

Some Ideas about an Apple /// Computer Emulator (version 1) - DTC - 28 Nov 1997 - 6 / 8

not sure of what would be involved here but suspect that if the ///'s 6502 CPU
and the memory-mapped I/O locations are robustly supported that the][emulation
should work also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do
not exist in the Apple][world.][emulation details can be found in the Apple
/// Owner's Guide (p X) and Service Manual (p X).   DTC has listing of ASC ROM amw

13.0   WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I recommend highly that the /// emulator be written in a high level language
such as Pascal or C. This should make the emulator more compatible with
different target computers and make development and maintenance of the emulator
much easier. I recommend avoiding low-level languages such as assembly.

14.0   WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines
that are commonly used today by most computer users. This means either the IBM
PC or the Apple Macintosh machine family. For the PC world I recommend the ///
emulator run under Windows 95 and Windows NT. For the Macintosh world I
recommend the emulator run on most Macintosh models which means support the
Macintosh 512 and above. Color display should also be supported by the ///
emulator (for the Macintosh this means use Color QuickDraw if the machine
supports CQD and if CQD is not supported by a Macintosh model use the Classic
B/W QD and maybe use patterns as "colors").

Any of these machines should be fast enough to emulate the /// and most likely
will be too fast in many areas. I recommend some type of speed control be built
into the emulator so that users can control how fast the emulator works. For
many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will
be immaterial since these programs typically wait for the user to enter data and
then do their thing. But for programs such as games the user will want to
control the emulator speed otherwise the game's actions will be super fast and
unplayable.

15.0   WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should
try to obtain as much /// technical information as possible. The author has
lots of info which he can copy at minimal charge. These persons should also
have access to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does
not address. This includes support for other input devices such as the mouse
which does have a 3rd party driver available.

16.0   EMULATOR DEBUGGING FACILITIES
16.0   REFERENCES
17.0

Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Plus Owner's Guide, Apple Computer, 1982

Apple /// System Data Sheet, Apple Computer, July 1983

Apple /// Plus System Data Sheet, Apple Computer, October 1983

Some Ideas about an Apple /// Computer Emulator (version 1) - DTC - 28 Nov 1997 - 7 / 8

Apple /// Standard Device Drivers Manual, Apple Computer, 1981

Apple /// SOS Reference Manual, Apple Computer, 1982

Apple /// SOS Device Drive Writer's Guide, Apple Computer, 1982

Apple /// Service Reference Manual (Level 2), Apple Computer, 1983

/// Bits: John Jeppson's Guided Tour of the Highway ///, Softalk magazine, May 1983

Bank Switch Razzle-Dazzle, Softalk magazine, August 1982

The Apple Nobody Knows, Apple Orchard magazine, Fall 1981

###