HONEYCRISP EMULATOR

TECHNICAL DOCUMENTATION

Document Version 1.4
Revision F5

ACKNOWLEDGEMENTS AND LICENSE NOTICE

The APPLE-1 microcomputer was designed by Steve Wozniak and introduced in 1976. The APPLE-1 System Monitor, Integer BASIC, and the Apple Cassette Interface ROM were written by Steve Wozniak. The MOS Technology 6502 microprocessor was designed by Chuck Peddle and the MOS Technology engineering team.

The SIGNETICS 2513 character font bitmap referenced in the HoneyCrisp program code and in this documentation is the property of its respective rights holders and is used here for purposes of historical accuracy and compatibility.

NOTE: The resident system monitor is commonly referred to as WOZMON (WOZniak MONitor) throughout this documentation. Either name may be used interchangeably.

Special thanks to: @tangelic of cyberspace.online for the interface design; Bobby Nijssen of apple1software.com and 8bitflux.com; Jeff Jetton of applefritter.com for programs and debugging tips; @UncleBernie of applefritter.com for the TurboType™ protocol, the Extended Format ACI firmware, the GEN2 HGR Graphics Card, and accuracy suggestions; San Bergmans of sbprojects.net for his documentation efforts; Ken Wessen of thewessens.net for the KRUSADER PROM file; SmartyKit, Inc. of smartykit.io for the SMARTYKIT1 PROM file; Rich Dreher of dreher.net (R&D Automation) for the original CFFA1 PROM file; Egon Rath (@egrath on github.com) for the 8KB Microsoft BASIC ROM; and the team at apple1registry.com for their archival and documentation efforts. Document edited by M. Jason Smith.

The MOSe 6502 emulation core was written in JavaScript by Landon J. Smith. The HoneyCrisp Emulator, this documentation, and all original software components are copyrighted.

(See MIT license listed below.)

THE HONEYCRISP EMULATOR, MOSe EMULATION CORE (revision r230626.247)

SPECIFICATIONS

MICROPROCESSOR:

Emulation Core:	MOSe 6502, (revision r230626.247)
Core Architecture:	MOS Technology 6502
Instruction Set:	Full MOS 6502 Set (151 official opcodes)
Addressing Modes:	13 (all MOS 6502 modes)
Nominal Clock:	1.023 MHz (emulated; live rate shown in the status bar)
CPU Speed Adjustment:	0.25x to 10.00x multiplier
BCD Arithmetic:	Supported (*NOT USED BY THE APPLE-1 MICROCOMPUTER*)
Interrupts:	IRQ, NMI, RESET
Self Test:	Optional 8-stage POST on page load

VIDEO DISPLAY:

Format:	40 characters/line, 24 lines; with automatic scrolling
Character Font:	Converted SIGNETICS 2513 bitmap (embedded as a WOFF2 file)
Display Modes:	LCD or CRT
Color Themes:	7 selectable: Mint, Sherbert, Vanilla, Grape, Lemon, Blueberry, Strawberry
Graphics Cards:	P-LAB TMS9918A ($CC00/$CC01) or GEN2 HGR (16KB+ RAM); one at a time
Fullscreen:	Supported

RAM MEMORY:

Selectable Capacity:	4KB, 8KB, 16KB, 32KB, or 48KB
Default Capacity:	32KB
User RAM Base:	$0000

ROM MEMORY:

WOZMON Monitor:	$FF00-$FFFF (256 bytes, always resident)
KRUSADER:	$F000-$FBFF (may be disabled to free RAM)
Integer BASIC:	$E000-$EFFF (may be disabled)
Microsoft BASIC:	$E000-$EFFF (optional; conflicts with Integer BASIC)
AppleSoft Lite:	$6000-$6FFF (optional; requires 32KB+)
Enhanced BASIC:	$5800-$7FFF (optional; requires 8KB+)
AppleSoft TMS9918:	$4000-$7FFF (optional; requires 32KB+ and TMS9918A)
ACI ROM:	$C100-$C1FF (256 bytes, always resident)
Extended ACI:	$C500-$C5FF (Uncle Bernie's Extended Format firmware)
CFFA1 Firmware:	$8700-$AFFF (entry point $9000)
JMON:	Loaded to $0280 (requires 16KB+ RAM)

PERIPHERAL INTERFACE ADAPTER:

Device:	MCS 6820 (emulated)
KBD Data Register:	$D010
KBD Control Reg.:	$D011
DSP Data Register:	$D012
DSP Control Reg.:	$D013

APPLE CASSETTE INTERFACE:

Address Range:	$C100-$C1FF (standard); $C500-$C5FF (extended format)
Compatible Files:	WAV, AIFF, MP3 audio
Encoding:	Apple-1 FSK (frequency-shift keying)
Decode Threshold:	1%-99%, adjustable (default 30%)
Speaker Output:	Live audio on $C000 access; mute and volume controls

MASS STORAGE:

Device:	CFFA1 CompactFlash Interface (emulated)
Image Format:	.po (ProDOS order) only; one image mounted at a time

PLATFORM:

Requirements:	Any web browser with HTML 5 & ES6 support
Save-State Format:	.hcstate (save/load sessions)
Program Formats:	.hc (machine code), .bas or .txt (Integer BASIC code), .mon or .apl (monitor keystroke dump), .tur (TurboType™), .bin (raw binary), .WAV, .AIFF or .MP3 (Audio Tape Files)

INTRODUCTION

The HoneyCrisp Emulator is a complete APPLE-1 microcomputer emulation system, consisting of the MOSe 6502 emulation core, a fully emulated MOS 6820 Peripheral Interface Adapter, an Apple Cassette Interface, a CFFA1 CompactFlash disk interface, optional graphics cards, and resident system software including the original system monitor PROM program and Integer BASIC interpreter, among other user applications. It is delivered as a single self-contained HTML document and requires no installation, running directly in any web browser with ES6 support. It contains resident system monitor software, enabling the user, via the keyboard and display, to write, examine, debug, and run programs efficiently; thus, being an educational tool for the learning of microprocessor programming, and an aid in the development of software.

The integral video display section renders a 40-character, 24-line terminal with automatic scrolling. The display section contains its own rendering logic, leaving all the RAM for user programs. Output is restricted to the upper 128 ASCII characters, in keeping with the behavior of the original APPLE-1. Seven selectable color themes are provided, in either an LCD or a simulated CRT presentation, offering a range of visual environments suited to the user's preference.

The emulated RAM is fully configurable from 4 kilobytes to 48 kilobytes, selectable through the Settings panel. The system uses the same memory organization as the APPLE-1 microcomputer: user RAM occupies the address space beginning at $0000, the Integer BASIC interpreter occupies $E000 through $EFFF, the Apple Cassette Interface ROM occupies $C100 through $C1FF, and the WOZMON monitor occupies $FF00 through $FFFF. The PIA is mapped at $D010 through $D013. Several optional ROMs and expansion cards occupy additional address space when enabled; a complete map appears at the end of Section IV. The emulated system is fully compatible with original APPLE-1 machine language software.

The MOSe emulation core implements the complete MOS Technology 6502 instruction set, all thirteen addressing modes, accurate processor status flag behavior, hardware interrupt handling, and Binary Coded Decimal arithmetic. A full description of the MOSe core appears in Section IV of this manual.

This manual is divided into four sections:

Section I - GETTING THE EMULATOR RUNNING
Section II - USING THE SYSTEM MONITOR
Section III - SYSTEM EXPANSION
Section IV - THE MOSe EMULATION CORE

Please read Section I thoroughly before attempting to operate the emulator, and study Section III carefully before expanding or configuring the system. Section IV provides complete technical documentation of the MOSe core for users engaged in software development or low-level system work.

SECTION I

GETTING THE EMULATOR RUNNING

The HoneyCrisp Emulator is fully assembled and ready to run. No external peripherals, power supplies, or installation procedures are required. The emulator runs entirely within a standard web browser from a single HTML file. To begin operation, open the file in any modern browser. The emulator will initialize the MOSe core, optionally run the Power-On Self Test, load the selected startup ROM into its respective ROM area, and present the terminal display in an uninitialized state.

Keyboard:

Any standard ASCII-compatible keyboard attached to the host system interfaces directly with the emulator. The emulator accepts all 64 printable ASCII characters and converts them to the uppercase format expected by WOZMON and other user programs. The WOZMON PROM program accepts only uppercase alpha (A-F, R) for command characters; the emulator performs this conversion automatically. The RETURN key on the host keyboard sends a carriage return ($8D) to the emulated microcomputer.

Certain control key combinations are intercepted by the emulator and mapped to hardware control functions, as described in the keyboard shortcut reference at the end of this section. All other character input is passed directly to the emulated keyboard buffer.

> NOTE: The ESCAPE key, when pressed while text has been entered on the current line, cancels the line and echoes a backslash followed by a carriage return.

Initial Startup:

Upon opening the emulator, the terminal display will be filled with characters. This indicates that the system is uninitialized. To begin operating, press the CLEAR SCREEN button (or Ctrl+E) to clear the display, then press the SOFT RESET button (or Ctrl+S) to initialize the WOZMON monitor. A backslash character (\) will appear on the display, and the cursor will drop to the next line. The system is now ready to accept monitor commands.

If the CLEAR SCREEN step is omitted, the display may contain residual content from the uninitialized startup state. If session auto-restore is enabled, the screen may contain characters from a left-over session if you do not trigger a CLEAR SCREEN operation. Pressing SOFT RESET at any time returns control to the resident monitor at $FF00 without disturbing RAM content.

If a startup ROM other than WOZMON has been selected in the Settings panel, the emulator boots into that ROM instead. See Section III, Startup Options.

Power-On Self Test:

The Settings panel provides a "Run POST (Power-On Self Test) on page load" option. When enabled, the emulator runs a short diagnostic sequence before booting the selected ROM and prints a pass/fail line for each of the following checks:

> CPU REGISTERS · CPU FLAGS · ZERO PAGE RAM · STACK PAGE · RAM MARCH $AA/$55 · RAM SIZE · RESET VECTORS · IRQ/BRK VECTORS

The POST is informational and non-destructive to the configuration; the system boots normally when it completes. The test is skipped when a session is auto-restored.

Control Buttons:

The interface is arranged in groups below the terminal display: SYSTEM CONTROLS, STATE & SETTINGS, and PROGRAM MANAGEMENT. The system control buttons map directly to emulated hardware functions as follows:

HARD RESET (Ctrl+H):

Clears all emulated RAM to zero, reinitializes the PIA and ACI, reloads the selected startup ROM into its proper ROM location, and starts execution from the RESET vector at $FFFC-$FFFD. All user programs and data in RAM are lost. This is equivalent to cycling power on the original APPLE-1 microcomputer. A mounted CFFA1 disk image remains mounted.

SOFT RESET (Ctrl+S):

Returns the processor to the WOZMON entry point without disturbing RAM contents. This is equivalent to triggering the reset line on an APPLE-1 microcomputer. A backslash and carriage return will appear on the display.

CLEAR SCREEN (Ctrl+E):

Clears the terminal display without affecting RAM or the state of the processor. The cursor is moved to the home position at the upper left of the display.

BREAK (Ctrl+C):

Sends a software interrupt to the running BASIC interpreter, stopping execution of the current BASIC program and returning control to the BASIC prompt. This is the equivalent of the ESCAPE key trigger inside BASIC, and works with both Integer BASIC and Microsoft BASIC.

FULLSCREEN (Ctrl+F):

Expands the terminal display to fill the browser window, scaling the character display proportionally for maximum readability. Pressing ESCAPE returns to the standard windowed view.

TUTORIAL:

Opens a guided introduction to the APPLE-1 and to monitor operation. Recommended for users new to hexadecimal notation or to the WOZMON command set.

CPU Speed Control:

A slider control labeled "CPU SPEED" is provided below the main control buttons. This slider adjusts the effective execution speed of the emulated 6502 processor from 0.25x (one quarter of nominal APPLE-1 clock speed) to 10.00x (ten times nominal speed). The current multiplier is displayed to the right of the slider, and the resulting clock frequency is reported in the status bar above the terminal. The nominal APPLE-1 clock frequency of 1.023 MHz corresponds to a multiplier of 1.00x.

NOTE: Unlike earlier versions, tape operations no longer require a 1.00x setting. Tape position is derived from the emulated CPU's cycle count, so cassette data decodes byte-for-byte identically at any speed and simply finishes proportionally sooner. Tape audio still plays at its native rate, so at high speeds the sound lags behind the load; this is cosmetic only. Some timing-sensitive demonstration programs are still best viewed at 1.00x.

Test Program:

After completing the initial startup sequence, the following test may be performed to verify that the system and all attachments are functioning correctly. This test enters a short machine language program into RAM, lists it, and runs it. It does not test all areas of the emulator, but will confirm correct operation of the keyboard, display, and WOZMON.

FIRST:

Press CLEAR SCREEN and then the SOFT RESET button. A backslash should be displayed, and the cursor should drop to the next line. WOZMON has been activated.

SECOND:

Type: 0:E8 8A 20 EF FF (RET) (0 is a zero, NOT the alpha-character "O"; spaces separate the bytes; (RET) means press RETURN.)

THIRD:

Type: 0.A (RET) (This should print out the program you have just entered.)

FOURTH:

Type: 0R (RET) (R means execute the program at the specified address.)

The program should then print a continuous sequence of incrementing ASCII characters. To stop the program and return to the system monitor, press the SOFT RESET button. To run again, type: 0R (RET).

Keyboard Shortcut Reference:

Ctrl+H	HARD RESET - power cycle; clears RAM, restarts from RESET vector
Ctrl+S	SOFT RESET - returns to WOZMON without clearing RAM
Ctrl+E	CLEAR SCREEN - clears terminal display only
Ctrl+C	BREAK - halts running BASIC program
Ctrl+F	FULLSCREEN - maximizes terminal display
Ctrl+B	PROGRAM CATALOG - opens the built-in program catalog
Ctrl+L	CFFA1 DISK LOADER - opens the CompactFlash image loader

SECTION II

USING THE SYSTEM MONITOR

The System Monitor is a PROM program in locations $FF00 to $FFFF which uses the keyboard and display to perform the functions of examining memory, depositing data into memory, and running programs. The monitor program is entered by pressing the SOFT RESET button (or Ctrl+S), which displays backslash-return. A backslash alone, with the cursor remaining on the same line, indicates bad page 0 RAM.

Commands are typed on a line-at-a-time basis with editing. Each line may consist of any number of commands (up to 128 characters). None are executed until (RETURN) is typed. The (SHIFT-0) backspace key backspaces and echoes an underline. The (ESC) key cancels a line and echoes backslash-return.

One or more hexadecimal digits (0-9, A-F) are used for address and data values. Addresses use the four least significant digits of a group, and data values use the two least significant digits. The following examples illustrate the variety of acceptable commands:

1. Opening a location (examining the contents of a single address).

   USER TYPES: FF00 (RET)
   MONITOR TYPES: $FF00: D8

2. Examining a block; from the last examined location, to a specified one.

   USER TYPES: .FF0F (RET)
   MONITOR TYPES: $FF00: D8 58 A0 7F
   $FF04: 8C 12 D0 A9
   $FF08: A7 8D 11 D0
   $FF0C: 8D 13 D0 C9
   Note: FF00 is still the most recently opened location.

3. Combining examples 1 and 2 to print a block in a single command.

   USER TYPES: FF00.FF0F (RET)
   MONITOR TYPES: (same block output as example 2)
   Note: Only the first location ($FF00) is considered "opened".

4. Examining several individual locations at once.

   USER TYPES: 300 302 304 (RET)
   MONITOR TYPES: $0300: FF
   $0302: 00
   $0304: A9

5. Depositing data in a single location.

   USER TYPES: 300: A9 (RET)
   MONITOR TYPES: $0300: FF (prior contents)
   Note: Location 300 is now opened and contains A9.

6. Depositing data in successive locations.

   USER TYPES: 300: A9 42 8D 12 D0 60 (RET)
   MONITOR TYPES: $0300: FF (prior contents)
   Deposits A9 in $0300, 42 in $0301, 8D in $0302, 12 in $0303, D0 in $0304, 60 in $0305.

7. Depositing in successive locations using a colon to continue.

    USER TYPES: 300: A9 42 (RET)

    MONITOR TYPES: $0300: FF

    USER TYPES: : 8D 12 D0 60 (RET)

    A colon means "start depositing from the most recently deposited or opened location."

8. Running a program at a specified address.

    USER TYPES: 300R (RET)

    MONITOR TYPES: $0300: A9

    Note: The cursor is left immediately to the right of the contents byte.

9. Running a program at the most recently examined location.

    USER TYPES: R (RET)

10. Entering a program and running it in one line.

    USER TYPES: 300: A9 42 8D 12 D0 60 R (RET)

    MONITOR TYPES: $0300: FF (prior contents)

11. Examining a block, then depositing into it.

    USER TYPES: 300.305 (RET)

    MONITOR TYPES: $0300: A9 42 8D 12

    USER TYPES: : B0 B1 B2 B3 B4 B5 (RET)

    New data deposited beginning at most recently opened location ($0300).

Useful Monitor Routines:

The following routines within WOZMON may be called by user programs directly. All addresses are in the ROM area at $FF00-$FFFF.

GETLINE: location $FF1F -

Monitor entry point. Jumping to $FF1F will enter the monitor and echo a carriage return. You may then examine memory locations with the monitor.

ECHO: location $FFEF -

Prints one byte (ASCII). Data from A (accumulator); contents of A not disturbed. Example: A9 42 20 EF FF (load $42 into A, JSR ECHO).

PRBYTE: location $FFDC -

Prints one byte in hexadecimal (two hex digits). Data from A; contents of A disturbed.

PRHEX: location $FFE5 -

Prints one hexadecimal digit (the four least significant bits of A). Contents of A disturbed.

Alternate Monitors and Assemblers:

Three additional resident programs may be selected as the startup ROM in the Settings panel, or launched from the WOZMON prompt:

KRUSADER ($F000-$FBFF):

Ken Wessen's 6502 assembler, disassembler, and monitor. Type F000R at the WOZMON prompt to launch it. From within KRUSADER, type $ to return to WOZMON. The ROM may be disabled in Settings to return $F000-$FBFF to user-accessible space.

SMARTYKIT:

The SmartyKit, Inc. PROM set. From SmartyKit, type F000R for the "Apple 30th Anniversary" demo, F800R for the counter, FC00R for pixel art, or FD00R for Hello World.

JMON:

An extended monitor loaded at $0280. Requires 16KB or more of RAM.

Integer BASIC:

Integer BASIC is resident in the ROM area at $E000 through $EFFF and is available whenever the emulator is running with 8KB of RAM or more. To enter Integer BASIC, type E000R at the WOZMON prompt and press RETURN. The BASIC prompt (›) will appear, and the interpreter is ready to accept BASIC program lines.

BASIC programs may also be loaded from .bas or .txt files using the LOAD PROGRAM FILE control. When a BASIC file is loaded, the emulator automatically enters the appropriate interpreter and the program is ready to RUN. Type RUN and press RETURN to execute the loaded program. To break from a running BASIC program, press the BREAK button or Ctrl+C.

Four additional interpreters may be enabled in the Settings panel and are described in Section III under BASIC ROM Options.

NOTE: RAM locations $0024 to $002B are used as index pointers by the monitor and are invalid for user use when the monitor is active. Locations $0200 to $027F are used as input buffer storage and are invalid for user modification.

SECTION III

SYSTEM EXPANSION

The HoneyCrisp Emulator may be expanded to include the operation of additional programs, memory configurations, alternate startup ROMs, additional BASIC interpreters, graphics cards, cassette tape support, CompactFlash mass storage, and a variety of display themes. All expansion and configuration functions are accessible from the controls panel located below the terminal display. This section describes each expansion feature in detail.

Program Catalog:

The Program Catalog (formerly the Program Library) is a built-in collection of APPLE-1 compatible programs spanning machine language code, Integer BASIC code, and cassette tape images. To open the catalog, click the PROGRAM CATALOG button or press Ctrl+B. A modal window will appear showing the available programs with their names, types, and descriptions. Programs may be searched by name or description, and filtered by category (Games, Demos, Development, Utilities) and by type (BASIC, Tape, Machine Code). Favorite programs may be starred for quick retrieval.

Programs listed as "machine code" (.hc format) are assembled 6502 machine language programs that load directly into emulated RAM and execute immediately upon loading. Programs listed as "BASIC" (.bas format) load into the selected BASIC interpreter and are auto-executed. Programs listed as "Tape" are cassette audio images loaded through the ACI.

Adding Your Own Programs:

The ADD YOUR OWN PROGRAM control within the catalog accepts a program either by URL (a raw .bas, .hc, or .wav file link) or by direct upload from the host computer. The entry form records a name, type, category, run address, tape load ranges, the BASIC interpreter to use, a minimum RAM requirement, a description, and instructions. Custom entries are stored in the browser only and appear alongside the built-in catalog. Keep uploaded files small, as browser storage is limited.

Loading Program Files:

Programs may be loaded from external files on the host system using the LOAD PROGRAM FILE control. Clicking the file browse button opens the host system's file selector. Any of the accepted file types may also be dragged and dropped directly onto the terminal window. The following file types are accepted:

.hc files:

> Machine code program files. These files contain a load address on the first line (e.g., 0300) followed by lines of hexadecimal data in the WOZMON deposit format (e.g., : A9 42 8D 12 D0). The data is loaded into emulated RAM beginning at the specified address.

.bas / .txt files:

> BASIC source files, one statement per line, in standard Integer BASIC VERSION C/D syntax or in the syntax of the selected interpreter. Loading a .bas file automatically enters BASIC and loads the program for the user. At least 8KB of RAM must be configured for BASIC to operate.

.mon / .apl files:

> WOZMON keystroke dumps, in the form address: bytes ... followed by ADDRR. The contents are injected straight into memory as though typed at the monitor, and run automatically.

.tur files:

Uncle Bernie's TurboType™ format. An auto-typed loader stub is entered, followed by a CRC-verified fast transfer. The program runs automatically on success; on a checksum mismatch the emulator falls back to the monitor with an EE error.

.bin files:

Raw binary dumps. Because .bin files carry no address information, the emulator prompts for one or more load address ranges and a run address, all in hexadecimal. File bytes are written in order, filling each range before moving to the next; the total range size must be at least as large as the file.

.hcstate files:

Saved emulator state. Restores the complete session exactly where it was left off.

NOTE: If the program's load address exceeds the currently configured RAM size, the emulator will display an error and will not load the file. In this case, increase the RAM configuration to at least the size required by the program.

Apple Cassette Interface:

The Apple Cassette Interface (ACI) is an expansion card for the APPLE-1 microcomputer that enables the saving and loading of programs to and from standard audio cassette tapes using frequency-shift keying (FSK) encoding. The ACI is mapped to address $C100 through $C1FF. HoneyCrisp emulates the ACI, allowing real APPLE-1 cassette audio files in WAV, AIFF, or MP3 format to be decoded directly into emulated RAM, and allowing emulated RAM to be encoded into WAV audio files for archival or transfer. The real ACI code reads the signal at $C081 bit-by-bit, exactly as on hardware.

NOTE: MP3 files are decoded through the browser's audio engine, so decode quality can vary slightly. WAV remains the most reliable format for critical reads.

Reading a Cassette Tape:

To load a program from an ACI-compatible audio file, proceed as follows:

1. In the Tape File Loader section, click the BROWSE button and select the cassette audio file from the host file system.

2. Type C100R at the monitor prompt and press RETURN. The monitor will respond with "C100: A9*" and await the address range command.

3. Enter the memory address range or ranges - e.g. 0300.03FFR for a single range, or E000.EFFFR 004A.00FFR for multiple ranges (up to eight).

4. Click the PLAY TAPE button. The cassette audio will be decoded and data written into emulated RAM; a progress bar shows the decode progress in real time. PAUSE and REWIND controls are provided alongside. If the ACI begins reading before PLAY is pressed, the tape auto-starts so that the leader tone cannot be missed.

5. When the progress bar completes, the program has been fully loaded and the monitor returns with a backslash. Verify the data by examining the loaded range, then run the program using its entry point address followed by R.

Writing a Cassette Tape:

To save emulated RAM contents to a WAV audio file in Apple-1 cassette format, type C100R to reach the C100: A9* prompt, then enter ranges using W in place of R - e.g. E000.EFFFW to save $E000-$EFFF. Multiple ranges may be chained: 0300.03FFW E000.EFFFW. The browser downloads a WAV file named after the address range (e.g. E000-EFFF.wav). The resulting file uses the same Apple-1 FSK encoding as standard APPLE-1 cassettes and can be reloaded with the Tape File Loader, or used with other emulation applications such as MAME.

Threshold Control:

The THRESHOLD control sets the comparator sensitivity of the FSK audio decoder when reading cassette files. It is expressed as a percentage from 1% to 99%, with a default value of 30%. Lower values make the decoder more sensitive to weak or quiet signals; higher values require a stronger signal before a transition is recognized. If a tape decode fails or produces garbage data in RAM, try reducing the threshold to 15-25%. If the decoder is producing too many false reads from a noisy tape, raise the threshold slightly above the default. For most standard Apple-1 cassette recordings, the default value of 30% will work correctly without adjustment.

ACI Speaker Output:

On real hardware, any access to $C000 toggles the cassette-out line; through a microphone or instrument amplifier this is an audible square wave, and it is how tone-generator BASIC programs (such as Corey Cohen's "Twinkle Twinkle") produce sound. HoneyCrisp reproduces this behavior: every $C000 access plays live through the host speakers. A Mute checkbox and a volume slider are provided above the Tape File Loader, along with a separate tape volume control.

Tape Timing and CPU Speed:

Tape position is derived from the emulated CPU's own cycle count rather than from wall-clock time. Raising the CPU Speed slider therefore does not break tape reads; it makes them finish proportionally sooner. A 24-second tape loads in about 2.4 seconds at 10x and still decodes byte-for-byte identically. Tape audio, however, always plays at its native rate in real time, so at higher speeds the sound lags behind the load and continues playing after it finishes. This is cosmetic only and does not affect the data.

Uncle Bernie's Extended Format ACI ($C500-$C5FF):

The Extended Format ACI is a published community firmware for the Apple-1 Cassette Interface, used on Uncle Bernie's GEN1 and GEN2 improved ACIs, which adds checksummed cassette I/O at much higher data rates. It is a supplementary page at $C500-$C5FF that coexists with the standard ACI at $C100 and shares its underlying tape routines. To use it:

1. Mount the cassette with BROWSE as usual.
2. Type C500R and press RETURN to reach its * prompt.
3. Use RX / WX in place of R / W, then click PLAY TAPE.

Thanks to Uncle Bernie for the firmware and for permission to include it in HoneyCrisp.

CFFA1 CompactFlash Disk Interface ($8700-$AFFF):

The CFFA1 emulates Rich Dreher's CompactFlash for Apple card, giving the emulator a virtual hard disk for ProDOS. The firmware occupies $8700 through $AFFF and provides an interactive menu-driven interface for loading and saving programs, cataloging files, and managing storage volumes. The firmware entry point is $9000.

Mounting a disk:

1. Open the CFFA1 Disk Loader with Ctrl+L, or drop a .po file directly onto the terminal.
2. Click the drop area to browse, or drag a ProDOS disk image onto it.

3. Type 9000R at the WOZMON prompt to boot into the CFFA1 menu and access the mounted disk. The system outputs "CFFA1›" to indicate the firmware has initialized.

CFFA1 menu commands:

C	Catalog - list files on the mounted disk
P	Prefix - change the current ProDOS directory
L	Load - load a file from disk into memory
N	New Directory - create a subdirectory
S	Save (BASIC) - save a BASIC program to disk
W	Write File - write a raw memory range to disk
R	Rename - rename a file or directory
D	Delete - remove a file or directory
!	Format - format the mounted disk image
B	Read Block - read a raw ProDOS block
M	Memory Display - view a range of memory
Q	Quit - return to WOZMON

Programs are loaded from the CF card using the L (Load) command, which accepts any ProDOS-compatible binary file and deposits it into emulated RAM at the address specified in the file's ProDOS auxiliary type field. Programs are saved using the W (Write File) or S (Save BASIC) commands, as appropriate for the file type. Only the first 32MB partition on the CF card is accessible.

NOTE: The CFFA1 emulation requires a disk image file in .po (ProDOS Order) format to be mounted before any read or write operation is attempted. No other image format is supported. Attempting Catalog, Load, Save, Write, or Format without a .po image inserted will cause the firmware to return I/O error $27 and abort the operation. Only one image may be mounted at a time; use EJECT in the CFFA1 Disk Loader before mounting a different image. Mounted disks persist across resets.

Complete documentation for the CFFA1 card, including hardware installation, dip switch configuration, firmware entry points, the API, and the ProDOS file system interface, is available in the CFFA1 manual at: dreher.net/projects/CFforApple1/downloads/CFFA1_Manual_1.1.pdf

Settings Panel:

The Settings panel is opened by clicking the SETTINGS button. Changes are applied with the SAVE CHANGES button; an unsaved-changes indicator appears when edits are pending. It provides the following configuration options:

Startup Options:

Selects which ROM program the emulator boots into on page load: WOZMON (default), KRUSADER, SMARTYKIT, or JMON. JMON requires 16KB or more of RAM. The panel reports the current selection. A checkbox enables the Power-On Self Test described in Section I.

Memory Configuration:

The RAM amount may be selected from five options: 4KB, 8KB, 16KB, 32KB (default), or 48KB. The selected amount of RAM is installed beginning at $0000 and extends upward. BASIC requires at least 8KB of RAM to operate. Some programs in the catalog require specific minimum RAM amounts; the catalog description for each program will indicate any such requirements. Changes to the RAM configuration take effect after a HARD RESET.

Display Settings:

Seven terminal color themes are available via the Terminal Theme selector, each setting the foreground (character) and background colors of the display: Mint (green on black), Sherbert (amber on black), Vanilla (white on black), Grape (purple on black), Lemon (yellow on black), Blueberry (cyan on black), and Strawberry (red on black). A DISPLAY MODE selector chooses between LCD DISPLAY and CRT DISPLAY presentation. The selected theme and mode are applied immediately and saved for future sessions.

Graphics Cards:

Two optional video cards may be shown to the right of the main terminal. The TMS9918A Graphics Card exposes a P-LAB TMS9918 VDP at $CC00/$CC01. Uncle Bernie's GEN2 HGR Graphics Card maps its display directly into RAM and requires 16KB or more. Both are off by default and cannot be enabled at the same time; turning one on switches the other off. Both cards are described in full under GRAPHICS CARD EXPANSION, following this section.

Session Management:

The "Auto-restore last session on startup" checkbox, when enabled, causes the emulator to automatically restore the last saved session state each time the page is loaded. The session includes full RAM contents, processor register values, terminal display state, and all settings.

Save and Load State:

Session state may be manually saved and restored using the SAVE STATE and LOAD STATE buttons. Clicking SAVE STATE causes the browser to download a binary file in .hcstate format containing the complete system state - all RAM contents, processor registers, terminal display contents, and current settings. This file may be stored externally and reloaded at any future time by clicking LOAD STATE and selecting the .hcstate file.

Saving RAM Contents:

The SAVE RAM button downloads the current RAM contents as a .hc format file. Three save options are provided: a modified-pages option that saves only pages of RAM written since the last reset, a full RAM dump that saves all configured RAM, and a custom range option accepting comma-separated hex ranges (e.g. 0000-0FFF, 2000-2FFF). An option to skip empty (all-zero) memory is provided, and a filename may be specified; the .hc extension is added automatically. The resulting file may be reloaded using the LOAD PROGRAM FILE control.

ROM Options:

The following ROM regions may be enabled or disabled. Because several of them share address space, the emulator will refuse conflicting combinations.

Disable KRUSADER ROM ($F000-$FBFF):

Relocates $F000-$FBFF into user-accessible space. The KRUSADER assembler will not be available when this option is checked.

Enable TMS9918 Graphics Demo ROM ($4000-$7FFF):

Loads the TMS9918 Graphics Demo Suite into $4000-$7FFF at startup, so that it survives a HARD RESET like a real ROM. Requires 32KB or more of RAM and the TMS9918A Graphics Card. Type 4000R to launch. Cannot be enabled alongside AppleSoft Lite or Enhanced BASIC.

BASIC ROM Options:

Integer BASIC ($E000-$EFFF):

Enabled by default. When unchecked, $E000-$EFFF becomes user-accessible RAM and Integer BASIC is unavailable. Launch with E000R.

AppleSoft Lite ($6000-$6FFF):

Requires 32KB or more of RAM. Type 6000R to launch. Cannot be enabled alongside Enhanced BASIC.

Enhanced BASIC / EhBASIC ($5800-$7FFF):

Requires 8KB or more of RAM. Type 5800R to launch. Cannot be enabled alongside AppleSoft Lite.

Microsoft BASIC ($E000-$EFFF):

The 8KB Microsoft BASIC ROM contributed by Egon Rath. Requires 8KB or more of RAM. Type E000R to launch. Cannot be enabled alongside Integer BASIC.

AppleSoft TMS9918 ($4000-$7FFF):

Loads AppleSoft Lite (same command set and token layout as AppleSoft BASIC Lite) into $4000-$7FFF at startup, but HGR, HCOLOR=, HPLOT and related commands drive the P-LAB TMS9918 VDP at $CC00/$CC01 instead of a memory-mapped framebuffer. Requires 32KB or more of RAM and the TMS9918A Graphics Card. Type 4000R to launch. Cannot be enabled alongside AppleSoft Lite, Enhanced BASIC, or the TMS9918 Graphics Demo ROM.

When a BASIC program is loaded from a file or from the catalog and the target interpreter is ambiguous, the emulator presents a SELECT BASIC INTERPRETER dialog offering Integer BASIC (E000R), AppleSoft Lite (6000R), AppleSoft TMS9918 / Graphics Card (4000R), and Enhanced BASIC / EhBASIC (5800R).

GRAPHICS CARD EXPANSION

The HoneyCrisp Emulator supports two optional video expansion cards. Neither card is part of the original APPLE-1 design; both are period-appropriate expansions that add bitmapped color graphics to a machine that otherwise has only a 40-column character terminal. When a card is enabled, its output appears in a second display panel to the right of the main terminal. The terminal itself is unaffected and continues to operate normally, so a program may print to the display and draw to the graphics card at the same time.

> IMPORTANT: Only one graphics card may be active at a time. There is not enough screen space to show both panels alongside the terminal, so enabling one card in the Settings panel automatically switches the other off. Both are off by default.

Each card is enabled from the DISPLAY SETTINGS group of the Settings panel. In addition, the emulator watches for accesses to each card's address space: if a running program touches the TMS9918A VDP ports or the GEN2 soft switches while the corresponding card is off, the emulator will enable that card's display automatically and print a notice. This means most graphics programs from the Program Catalog will work without visiting the Settings panel first, provided enough RAM is configured.

The P-LAB TMS9918A Video Card:

The TMS9918A is the Texas Instruments Video Display Processor used in the TI-99/4A, the ColecoVision, the MSX line, and a number of later Apple-1 expansion projects. It is a self-contained video chip with its own private 16KB of video RAM, which the host processor never addresses directly; instead, all access is performed through two I/O ports. Because the display memory does not come out of the emulated APPLE-1 RAM, the TMS9918A card imposes no RAM requirement of its own.

Register Interface:

$CC00	VDP data port. Reading or writing this address transfers a byte to or from video RAM at the current auto-incrementing VRAM address pointer.
$CC01	VDP address / register port. Writing two bytes in sequence sets the 14-bit VRAM address pointer, or loads one of the eight internal control registers, depending on the two high bits of the second byte. Reading this port returns the status register and clears the interrupt flag.

The two-byte latch on the address port follows the standard TMS9918A protocol. The first byte written is held; the action taken is determined by the top two bits of the second byte: 00 selects a VRAM address for reading, 01 selects a VRAM address for writing, and 10 writes the first byte into the control register named by the low three bits of the second byte. The VRAM pointer wraps within the 16KB space ($0000-$3FFF) after each data-port access, so consecutive bytes may be transferred without resetting the address.

Display Characteristics:

Resolution:	256 x 192 pixels
Video RAM:	16KB, private to the card (not part of the emulated 64K address space)
Palette:	15 colors plus transparent
Modes:	Graphics I, Graphics II, Multicolor, and Text
Sprites:	Supported, with the standard per-scanline limit
Registers:	Eight internal control registers, plus a status register

Graphics II is the mode most commonly used by the demonstration programs in the catalog, as it allows an independent color pair for every eight-pixel group and therefore produces the most detailed images. Multicolor mode trades resolution for free color placement, and Text mode provides a 40-column monochrome character display independent of the main terminal.

Enabling the card:

> Open the Settings panel and check "Enable TMS9918A Graphics Card" under DISPLAY SETTINGS, then click SAVE CHANGES. The video panel appears immediately to the right of the terminal. If the GEN2 HGR card was enabled, it is switched off at the same time.

TMS9918A Software Support:

Two optional ROMs in the Settings panel are written specifically for this card. Both occupy $4000-$7FFF and cannot be enabled together, nor alongside AppleSoft Lite or Enhanced BASIC, which claim overlapping space.

TMS9918 Graphics Demo ROM ($4000-$7FFF):

> Loads the TMS9918 Graphics Demo Suite at startup, so that it survives a HARD RESET the way a real ROM would. Requires 32KB or more of RAM and the TMS9918A card. Type 4000R at the monitor prompt to launch it.

AppleSoft TMS9918 ROM ($4000-$7FFF):

> A build of AppleSoft Lite with the same command set and token layout as the standard AppleSoft Lite ROM, but in which HGR, HCOLOR=, HPLOT and the related graphics statements drive the TMS9918 VDP through $CC00/$CC01 rather than writing to a memory-mapped framebuffer. This is the most convenient way to write original graphics programs for the card, since no knowledge of the VDP register protocol is required. Requires 32KB or more of RAM and the TMS9918A card. Type 4000R to launch.

When adding a program to the catalog by hand, the BASIC Interpreter field offers "AppleSoft TMS9918 (force switch)" so that a graphics program always loads into the correct interpreter regardless of what is currently running.

Several ready-made TMS9918A programs ship in the Program Catalog, including Mandelbrot, Plasma, Conway's Game of Life, Galaga, Apple LOGO, and a photographic display. Their catalog entries note the RAM each one requires.

Uncle Bernie's GEN2 HGR Graphics Card:

The GEN2 HGR card takes the opposite approach to the TMS9918A. Rather than providing private video memory behind an I/O port, it maps its display directly into the emulated APPLE-1 RAM and scans it out in the manner of the Apple II video circuit. Programs draw by writing bytes into ordinary memory, which makes plotting extremely fast and makes the card straightforward to drive from machine language, but it also means the display competes with user programs for RAM.

> NOTE: The GEN2 HGR card requires 16KB or more of configured RAM, because its display pages occupy $2000-$3FFF and $4000-$5FFF. If a program accesses the card while less RAM is installed, the emulator declines to enable the display and reports the RAM requirement rather than showing a corrupt image.

Display Characteristics:

Resolution:	280 x 192 pixels in high-resolution mode
Display Memory:	Page 1 at $2000-$3FFF; Page 2 at $4000-$5FFF
Color:	Generated by NTSC-style artifacting from the bit pattern, with a high-order palette bit per byte, as on the Apple II
Text:	A 40-column text page and a mixed text/graphics split are provided

Paging: Two independent display pages, selectable for flicker-free animation

GEN2 Soft Switches:

The card is controlled by soft switches in the $Cxxx I/O space, in the familiar Apple II style: the switch is thrown by any access to its address, and the value read or written is irrelevant. The card decodes an address as one of its switches when bit 11 is clear, bit 9 is set, and bit 4 is set - for example the block at $C250-$C257. The low three bits select the function:

+0	Graphics mode on (clears text mode)
+1	Text mode on
+2	Full-screen mode (clears mixed mode)
+3	Mixed mode on (graphics with four text lines at the bottom)
+4	Select display Page 1
+5	Select display Page 2
+6	Low-resolution mode (clears hi-res)
+7	High-resolution mode

A read from any soft-switch address also returns the card's horizontal scan status in bit 7, which timing-sensitive programs may poll to synchronize drawing with the video scan and avoid tearing.

Programming the card:

A typical high-resolution setup selects hi-res, full-screen, graphics mode, and Page 1, then writes pixel data into $2000-$3FFF. Each byte covers seven horizontal pixels in its low seven bits; bit 7 selects the color palette for that group of pixels. Rows are not stored contiguously - the card reproduces the interleaved Apple II line ordering - so a row-address lookup table is the usual approach in user programs.

Enabling the card:

Open the Settings panel and check "Enable Uncle Bernie's GEN2 HGR Graphics Card" under DISPLAY SETTINGS, confirm that at least 16KB of RAM is configured, then click SAVE CHANGES. If the TMS9918A card was enabled, it is switched off at the same time.

Ready-made GEN2 programs in the Program Catalog include Chess, Conway's Game of Life, a low-resolution demonstration, and a multi-mode demonstration that exercises the text, mixed, and paging switches in turn.

Choosing Between the Cards:

The TMS9918A is the better choice for color-rich still images and for sprite-based work, and it costs no user RAM at all, since its frame buffer is private to the card. Its drawback is that every pixel must be moved through a two-port interface, which is comparatively slow for animation driven from 6502 code.

The GEN2 HGR card is the better choice for fast animation and for programs that manipulate the display as ordinary memory, and it supports page flipping for smooth motion. Its drawbacks are the 16KB RAM floor, the loss of $2000-$5FFF to display memory, and the artifacting color model, which constrains where colors may be placed.

Programs written for one card will not run on the other. Catalog entries indicate which card a program requires in the program title and description.

SECTION IV

THE MOSe EMULATION CORE

The MOSe emulation core, revision r230626.247, is an instruction-level software emulation of the MOS Technology 6502 microprocessor, written in JavaScript. It constitutes the central processing engine of the HoneyCrisp Emulator and is responsible for the execution of all 6502 machine language instructions, the management of the processor's internal register state, the handling of memory read and write operations, and the processing of hardware interrupts. The core is designed to provide a complete and correct execution environment for all standard APPLE-1 software, including WOZMON, Integer BASIC, and any user-written 6502 machine language programs.

The MOSe core executes instructions at the level of complete opcodes: each call to the core's execution function advances the processor through one complete 6502 instruction, including all operand fetches, address calculations, memory accesses, and register updates required by that instruction. Cycle counting is maintained to track the relative timing of instructions, allowing the emulation to present correct timing behavior to software that depends upon it. This cycle count is also the time base used by the Apple Cassette Interface, as described in Section III.

Processor Registers:

The MOSe core implements all six internal registers of the MOS Technology 6502 processor. These registers are the fundamental state elements of the CPU and are manipulated directly by 6502 instructions:

A (Accumulator):

An 8-bit general-purpose register used as the primary operand source and destination for arithmetic, logical, and data transfer instructions. The accumulator is the only register that participates directly in arithmetic and logical operations. Its contents may be loaded from memory, stored to memory, pushed to the stack, pulled from the stack, shifted, and transferred to and from the X and Y registers.

X (Index Register X):

An 8-bit index register used for indexed addressing and as a general-purpose counter or storage register. The X register is used in the Zero Page,X; Absolute,X; Indexed Indirect; and Stack-relative modes. It may be loaded from memory or from the accumulator, stored to memory, transferred to or from the accumulator or stack pointer, incremented, and decremented.

Y (Index Register Y):

An 8-bit index register functionally similar to X. The Y register is used in the Zero Page,Y; Absolute,Y; and Indirect Indexed addressing modes. It supports the same load, store, transfer, increment, and decrement operations as X, but cannot be transferred to or from the stack pointer.

S (Stack Pointer):

An 8-bit register that points to the next available location on the hardware stack, which occupies page 1 of RAM ($0100-$01FF). The stack pointer is decremented when data is pushed and incremented when data is pulled. The stack pointer always points to the next free location; the processor decrements S before writing on a push, and reads the location S+1 before incrementing S on a pull. The stack pointer may be transferred to and from the X register.

PC (Program Counter):

A 16-bit register that contains the address of the next instruction to be fetched and executed. The program counter is incremented automatically after each byte fetch. It is modified by branch instructions (relative offset), jump instructions (absolute or indirect address), subroutine calls (JSR), returns (RTS, RTI), and interrupt handling. The program counter cannot be accessed directly; it is manipulated only through control flow instructions and interrupt mechanisms.

P (Processor Status):

An 8-bit register whose individual bits serve as condition flags, recording the results of arithmetic and logical operations, and controlling certain processor behaviors. The P register is described in full in the following section.

Processor Status Flags:

The P register contains seven defined flag bits. An eighth bit (bit 5) is always set to 1 and has no function. Each flag is described below in order from bit 7 to bit 0:

N - Negative (bit 7):

Set to 1 if the result of the last operation had bit 7 equal to 1 (i.e., the result is negative in two's complement representation); cleared to 0 otherwise. Affected by arithmetic, logical, load, transfer, increment, decrement, and shift operations.

V - Overflow (bit 6):

Set to 1 if a signed arithmetic overflow occurred in the last ADC or SBC operation; cleared to 0 otherwise. Also manipulated directly by the CLV instruction, and loaded from bit 6 of the tested byte by the BIT instruction. The V flag is tested by the BVC and BVS branch instructions.

B - Break (bit 4):

Set to 1 when the processor executes a BRK instruction, and cleared when a hardware interrupt (IRQ or NMI) is serviced. The B flag is not a true hardware flag but rather a distinguishing marker written into the copy of P that is pushed to the stack during a BRK or interrupt. It is used by interrupt service routines to determine whether the interrupt was software (BRK) or hardware (IRQ) in origin.

D - Decimal (bit 3):

When set to 1, causes the ADC and SBC arithmetic instructions to operate in Binary Coded Decimal (BCD) mode, treating the accumulator and memory operands as two-digit packed BCD numbers. When cleared, these instructions operate in standard binary mode. The D flag is set by SED and cleared by CLD. It is not affected by RESET.

I - Interrupt Disable (bit 2):

When set to 1, maskable hardware interrupts (IRQ) are ignored. When cleared, IRQ interrupts are recognized. The I flag is set automatically when an IRQ or NMI interrupt is serviced, preventing nested interrupts. It is set by SEI, cleared by CLI, and set during RESET.

Z - Zero (bit 1):

Set to 1 if the result of the last operation was zero; cleared to 0 otherwise. Affected by arithmetic, logical, load, transfer, increment, decrement, shift, and compare operations. Tested by the BEQ and BNE branch instructions.

C - Carry (bit 0):

Serves as carry out of bit 7 for ADC, borrow indicator for SBC (inverted), and as a shift-out bit for the ASL, LSR, ROL, and ROR shift/rotate instructions. Also used as a general-purpose flag by the compare instructions (CMP, CPX, CPY), where it indicates that the accumulator or register is greater than or equal to the operand. The C flag is set by SEC and cleared by CLC.

Addressing Modes:

The MOSe core fully implements all thirteen addressing modes defined by the MOS Technology 6502 architecture. Each addressing mode specifies how the processor locates the operand for a given instruction. The thirteen modes are:

1. Implied:

   No operand is required. The instruction operates entirely on internal registers. Examples: CLC, SEC, TAX, INX.

2. Accumulator:

   The operand is the accumulator register itself. Used by shift and rotate instructions when they operate on A directly. Example: ASL A.

3. Immediate (#$nn):

   The operand is a literal 8-bit constant value contained in the byte immediately following the opcode. Example: LDA #$42.

4. Zero Page ($zz):

   The operand is located at an address in page zero ($0000-$00FF), specified by the single byte following the opcode. Zero page addressing requires only one address byte and executes faster than absolute addressing. Example: LDA $24.

5. Zero Page,X ($zz,X):

   The effective address is obtained by adding the X register to the zero page base address. The result is masked to remain within page zero (wraps at $FF). Example: LDA $24,X.

6. Zero Page,Y ($zz,Y):

   Identical to Zero Page,X but uses the Y register as the index. Used only by LDX and STX instructions. Example: LDX $24,Y.

7. Absolute ($nnnn):

   The full 16-bit effective address is given in the two bytes following the opcode, stored in little-endian byte order (low byte first). Example: LDA $D010.

8. Absolute,X ($nnnn,X):

   The effective address is formed by adding the X register to the 16-bit base address. If the addition crosses a page boundary, an extra clock cycle is consumed for read instructions; write instructions always take the extra cycle. Example: LDA $0300,X.

9. Absolute,Y ($nnnn,Y):

   Identical to Absolute,X but uses the Y register. Example: LDA $0300,Y.

10. Indirect (($nnnn)):

Used exclusively by the JMP instruction. The two bytes at the specified absolute address contain the low and high bytes of the jump target address. NOTE: the original 6502 has a hardware bug in which, if the low byte of the indirect address is $FF, the high byte is fetched from $xx00 of the same page rather than the next page. This behavior is replicated within MOSe.

11. Indexed Indirect (($zz,X)):

The zero page base address is added to X to form a pointer address (within page zero), then the two bytes at that pointer address give the effective address. Used primarily in data structure and table operations. Example: LDA ($24,X).

12. Indirect Indexed (($zz),Y):

The two bytes at the given zero page address form a 16-bit base address; the Y register is then added to this base to form the effective address. This mode is the most common indirect mode, used for table lookups and array access via a zero-page pointer. Example: LDA ($24),Y.

13. Relative ($+/-rr):

Used exclusively by the eight conditional branch instructions (BCC, BCS, BEQ, BMI, BNE, BPL, BVC, BVS). The signed 8-bit displacement byte following the opcode is added to the program counter (pointing to the byte after the branch instruction) to form the branch target. The branch range is -128 to +127 bytes from the instruction following the branch.

Instruction Set Summary:

The MOSe core executes all 151 official MOS Technology 6502 opcodes. Instructions are organized into functional groups as follows:

Load and Store:

LDA, LDX, LDY - load A, X, or Y from memory. STA, STX, STY - store A, X, or Y to memory.

Register Transfer:

TAX, TAY - transfer A to X or Y. TXA, TYA - transfer X or Y to A. TSX, TXS - transfer between stack pointer and X.

Stack Operations:

PHA, PLA - push and pull accumulator. PHP, PLP - push and pull processor status.

Arithmetic:

ADC - add with carry (binary or BCD mode). SBC - subtract with borrow (binary or BCD mode). INC, INX, INY - increment memory, X, or Y. DEC, DEX, DEY - decrement memory, X, or Y.

Logical:

AND, ORA, EOR - bitwise AND, OR, exclusive-OR with memory. BIT - test bits in memory against the accumulator.

Shift and Rotate:

ASL - arithmetic shift left. LSR - logical shift right. ROL - rotate left through carry. ROR - rotate right through carry.

Compare:

CMP, CPX, CPY - compare A, X, or Y with memory; sets N, Z, C flags.

Branch:

BCC, BCS - branch on Carry clear or set. BEQ, BNE - branch on Zero set (equal) or clear. BMI, BPL - branch on Negative set or clear. BVC, BVS - branch on Overflow clear or set.

Jump and Subroutine:

JMP - jump to address (absolute or indirect). JSR - jump to subroutine; pushes return address to stack. RTS - return from subroutine; pulls return address from stack.

Interrupt and System:

BRK - software interrupt; pushes PC+2 and P to stack; loads IRQ vector. RTI - return from interrupt; pulls P and PC from stack. NOP - no operation.

Flag Control:

CLC, SEC - clear or set Carry. CLI, SEI - clear or set Interrupt Disable. CLD, SED - clear or set Decimal mode. CLV - clear Overflow.

Interrupt Handling:

The MOSe core supports all three interrupt mechanisms of the MOS 6502. When an interrupt is recognized, the processor completes the current instruction, then pushes the high byte of PC, the low byte of PC, and the P register (with the B flag set appropriately) to the hardware stack. The interrupt disable flag (I) is then set, and the processor loads the program counter from the appropriate vector address:

IRQ ($FFFE-$FFFF):

Maskable hardware interrupt. The IRQ is recognized only when the I flag in P is clear. In the APPLE-1 and HoneyCrisp, WOZMON maps the IRQ vector to $FF00 (the WOZMON reset entry). The B flag is pushed as 0 to distinguish this from a BRK.

NMI ($FFFA-$FFFB):

Non-maskable interrupt. The NMI is always recognized regardless of the state of the I flag. It follows the same push sequence as IRQ but is not maskable. In WOZMON, the NMI vector is mapped to $FF00.

RESET ($FFFC-$FFFD):

Upon system reset, the processor does not push to the stack, but loads the program counter from the RESET vector at $FFFC-$FFFD. In HoneyCrisp and the original APPLE-1, this vector points to $FF00, the entry point of WOZMON. The I flag is set and the D flag is cleared by RESET; all other registers are indeterminate.

Binary Coded Decimal Mode:

When the D (Decimal) flag in the P register is set by the SED instruction, the ADC and SBC instructions operate in BCD mode. In this mode, the accumulator and memory operands are treated as two-digit packed BCD numbers (00 through 99). Addition and subtraction produce BCD results, and the carry flag indicates a carry out of the most significant BCD digit. All other instructions are unaffected by the D flag and continue to operate in standard binary mode. The CLD instruction clears the D flag, returning ADC and SBC to binary mode. WOZMON clears the D flag upon reset (CLD is the first instruction executed from the RESET vector).

Memory Map:

The complete emulated address space of HoneyCrisp is 65,536 bytes ($0000-$FFFF). RAM occupies the low portion of the address space up to the configured limit; locations above the RAM ceiling and below the first ROM region return $FF on read and ignore writes, consistent with the behavior of unoccupied address space on the original APPLE-1 board. Optional regions appear only when the corresponding option is enabled in Settings.

$0000 - (RAM limit)	User RAM, configurable to 4/8/16/32/48KB (JMON loads at $0280)
$4000 - $7FFF	AppleSoft TMS9918 or TMS9918 Graphics Demo ROM (optional)
$5800 - $7FFF	Enhanced BASIC / EhBASIC ROM (optional)
$6000 - $6FFF	AppleSoft Lite ROM (optional)
$8700 - $AFFF	CFFA1 CompactFlash firmware (optional; entry point $9000)
$C000	ACI cassette-out toggle (speaker output)
$C100 - $C1FF	Apple Cassette Interface ROM ($FF elsewhere in $Cxxx)
$C500 - $C5FF	Extended Format ACI firmware (optional)
$CC00 - $CC01	P-LAB TMS9918 VDP registers (optional)
$D010 - $D013	PIA (6820 keyboard and display registers)
$E000 - $EFFF	Integer BASIC ROM or Microsoft BASIC ROM (4KB)
$F000 - $FBFF	KRUSADER ROM (may be disabled to free RAM)
$FC00 - $FEFF	Unoccupied (reads as $FF)
$FF00 - $FFFF	WOZMON ROM (includes RESET, NMI, IRQ vectors)

HARDWARE NOTES

Page 0 Monitor Variables:

Symbol	Address	Description
XAML	$0024	Examine index - low byte
XAMH	$0025	Examine index - high byte
STL	$0026	Store index - low byte
STH	$0027	Store index - high byte
L	$0028	Hex input accumulator - low
H	$0029	Hex input accumulator - high
YSAV	$002A	Saved Y register (text index)
MODE	$002B	Monitor mode byte (XAM/STOR/BLKXAM)

Input Buffer:

$0200 - $027F: Monitor line input buffer. This 128-byte region stores the text typed by the user on the current input line. It is invalid for user use whenever the monitor is active. Note that JMON, when selected, is loaded at $0280, immediately above this buffer.

PIA and ACI Register Map:

Register	Address	Function
KBD	$D010	Keyboard data. High-order bit equals 1 when a key has been pressed.
KBD CR	$D011	Keyboard control. High-order bit indicates key ready; reading clears the flag. Rising edge of KBD strobe sets flag.
DSP	$D012	Display data. Lower seven bits are the ASCII character to display; high-order bit is "display busy" (1 = busy, 0 = ready).
DSP CR	$D013	Display control register.
ACI BASE	$C100	ACI ROM base. Executing C100R engages the ACI tape-read routine; the ACI load-address range is read from the tape header.
ACI OUT	$C000	Cassette-out toggle. Any access flips the output line; reproduced as live speaker audio.
ACI EXT	$C500	Extended Format ACI base. Executing C500R reaches its * prompt; use RX / WX in place of R / W.

Software Considerations:

The following sequences are the standard routines for reading the keyboard and writing to the display from user programs. These are identical to those used on the original APPLE-1 hardware.

Read Key from KBD:

LDA KBD CR ($D011) ; wait for key ready
BPL (loop back) ; high bit = 0, not ready - loop
LDA KBD DATA ($D010) ; load key ASCII value

Output to Display:

BIT DSP ($D012) ; test display ready
BMI (loop back) ; high bit = 1, busy - loop
STA DSP ($D012) ; store ASCII character to display

Tips:

- Save your system state to preserve work between sessions - use SAVE STATE and LOAD STATE.
- Enable Auto-restore last session in Settings to automatically reload your last state on startup.
- Adjust CPU Speed with the slider for faster or slower execution. Some demos are best at 1.00x.
- Program descriptions in the catalog list any special RAM requirements - check these before running.
- Any accepted file type may be dragged and dropped directly onto the terminal window.