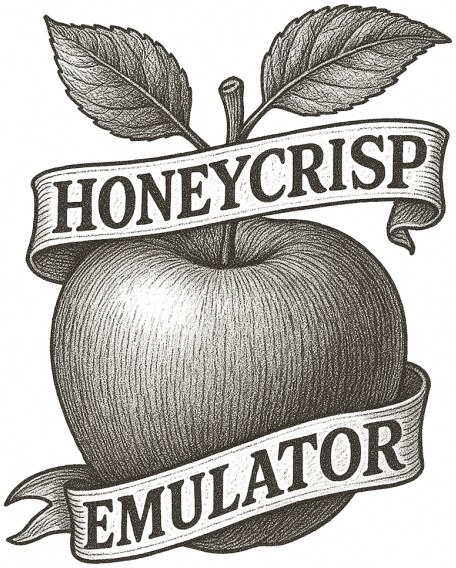

HONEYCRISP EMULATOR

Technical Documentation

HoneyCrisp Emulator | Technical Documentation

TABLE OF CONTENTS

HoneyCrisp Emulator: Technical Documentation

Version 1.1 | Author: Landon Smith | Date: October 23rd, 2025

Welcome to the official technical documentation for The HoneyCrisp Emulator. This document is your guide to understanding the inner workings of this APPLE-1 emulator. It covers a variety of topics concerning the technicalities of the emulation system and using the system for the first time.

1. Introduction

The HoneyCrisp Emulator is a browser-based recreation of the original APPLE-1 microcomputer designed by Steve Wozniak in 1976. This manual provides comprehensive documentation for developers, historians, and hobbyists who desire to understand this emulator within a greater scope.

1.1 Purpose

This emulator serves a primary purpose of being an educational tool for the people who wish to experiment with and learn about the first Apple computer, while also providing a test bed environment for hobbyist developers to easily build and run programs.

1.2 Design Philosophy

The HoneyCrisp Emulator prioritizes authentic hardware accuracy over emulation performance shortcuts. It implements cycle-accurate CPU instruction execution, original video display characteristics, proper timing mechanisms for I/O operations, and more.

1.3 Emulation Requirements

To efficiently experience and operate an APPLE-1 using HoneyCrisp, it is recommended that your system meets the following requirements:

- A modern web-browser with HTML 5.0 and JavaScript ES6 support.
- A keyboard for input operations.
- A stable connection to the internet

If you have a slower internet connection, there's no need to worry. The HoneyCrisp Emulator is fully contained within one file of 50 kilobytes in size, making for a fast and easy experience on most connection types.

Section 2 of this documentation covers the system architecture of The HoneyCrisp Emulator. It includes detailed information about the specifications that are emulated and breaks down the specifications individual steps/components.

2. HoneyCrisp System Architecture

2.1 MOS Technology 6502 CPU Emulation

At the core of The HoneyCrisp Emulator lies a complete MOS Technology 6502 CPU emulator implemented within the JavaScript source code as the **CPU6502** class. This class that contains the CPU emulator includes the following:

2.1.1 Register Implementations

All MOS 6502 registers are implemented as JavaScript properties.

- **A…….Accumulator (8-bit)**
- **X,Y……Index Registers (8-bit each)**
- **PC……..Program Counter (16-bit)**
- **S……Stack Pointer (8-bit)**
- **C,Z,I,D,B,V,N……..Status Flags (1-bit each)**

2.1.2 Instruction Execution

The step() JavaScript function (within the source-code) executes one instruction per call by performing the following operations:

1. **Fetch opcode from memory at PC**
2. **Increment the PC**
3. **Decode opcode using an instruction map**
4. **Execute the corresponding operation**
5. **Update the cycle count**
6. **Check for and apply page crossing penalties**

2.2 Memory Map

Below is a table that contains information about the memory mapping of The HoneyCrisp Emulator. Address ranges shown here are identical to the original hardware.

ADDRESS RANGE	SIZE	DESCRIPTION
$0000-$0FFF	Default of 4K-bytes (Expandable to 8K-bytes)	Emulated memory's ZERO-PAGE
$C100-C1FF	256 bytes	Apple Cassette Interface ROM
$E000-$EFFF	4K-bytes	Integer BASIC ROM
$FF00-$FFFF	256 bytes	Relative System Monitor

2.2.1 I/O Registers

- $D010 - Keyboard data input (bit 7 = ready flag)
- $D011 Keyboard status (bit 7 = data available)
- $D012 Display Output Register
- $D013 Display Control

2.2.2 Memory Access Methods

The read(addr) and write (addr, val) operations handles all memory access, implementing proper I/O mapping, RAM read/write functions, and ROM write protection.

2.3 Video Display System

The video subsystem emulates the APPLE-1's 40-column by 24-row character display using a virtual frame buffer.

2.3.1 Display Architecture

- **Screen Buffer:** 2D Array (40 columns by 24 rows)
- **Cursor (@) Tracking:** X and Y coordinates
- **Scrolling:** Automatic when last value of Y is reached. (as per the original hardware)
- **Rendering:** TTF character-based rendering. No bitmaps were used to save space.

2.3.2 Character Set

HoneyCrisp uses a custom font, reproducing the Signetics 2513 character ROM, limiting the font to a set of 64 characters, as per the original hardware.

- **Uppercase Letters: A-Z**
- **Digits: 0-9**
- **Special Characters: @ [\ ] ^ _ ! # $ % & ‘ ( ) * + , - . / : ; < = > ?**
- **Blank Space (BLSP)**

2.3.3 Cursor Blink Interval

The relative system monitor (WOZMON) prompt cursor (@) blinks in 530ms intervals, matching the original hardware behavior. This is implemented via setInterval() calling the renderScreen() function.

3. Primary Technical Specifications

3.1. Instruction Set Implementation

HoneyCrisp implements all 151 documented 6502 opcodes plus common undocumented opcodes. Not all operation codes used are listed within this manual for brevity. The table on **page six** lists some of these opcodes.

3.1.1 Documented Instructions

Category	Opcodes
Load/Store	LDA, LDX, LDY, STA, STX, STY
Transfer	TAX, TXA, TAY, TSX, TXS
Stack	PHA, PLA, PHP, PLP
Arithmetic	ADC, SBC, INC, INX, INY, DEC, DEX, DEY
Logic	AND, ORA, EOR
Shift	ASL, LSR, ROL, ROR
Compare	CMP, CPX, CPY, BIT
Brach	BCC, BCS, BEQ, BNE, BMI, BPL, BVC, BVS
Jump	JMP, JSR, RTS, RTI
Flag	CLC, SEC, CLI, SEI, CLV, CLD, SED
System	BRK, NOP

3.1.2 Undocumented Instructions

Category	Opcodes
Load/Store Combos	LAX, SAX
Decrement/Increment Combos	DCP, ISP
Shift/Rotate Combos	SLO, RLA, SRE, RRA
Accumulator Ops	ANC, ALR, ARR, XAA, AXS
Store High Byte	SHY, SHX

3.1.3 Addressing Modes

All 13 standard 6502 addressing modes are implemented including:
Immediate, Zero Page, Absolute, Indirect, Relative, and all indexed variations.

3.2 Cycle Timing

HoneyCrisp implements cycle-accurate timing for all instructions executed.
The cycleTable array stores the base cycle count for each opcode.

3.2.1 Timing Accuracy

- **Base cycles are pulled from a 256-entry lookup table.**
- **Page crossing penalties are added automatically.**
- **Branch taken penalties are calculated (+1 cycle, +2 if page is crossed).**
- **A total cycle counter tracks the emulation's progress.**

3.2.2 Execution Throttling

The tick(timestamp) function runs continuously via requestAnimationFrame(), executing 16,667 (or more, depending on CPU Speed turboMultiplier value) per frame to approximate the original 1MHz operation at 60 fps.

3.3 I/O Handling

3.3.1 Keyboard Input (Breakdown)

Keyboard input is buffered in the _kbdBuf array.

- **All typed characters are converted to uppercase for font compatibility.**
- **ASCII codes received are OR'd with $80 (bit 7 is set)**
- **Reading $D010 returns a character and clears the buffer array.**
- **Reading $D011 returns $80 is a character is available.**

3.3.2 Display (Terminal) Output

Character output is handled through the _videoHook

- **Writing to $D012 triggers output**
- **The character is masked to 7-bit ASCII**
- **Output is queued in videoOutputQueue and rate-limited to a value set by default or the turboMultiplier.**
- **A carriage return, $0D triggers a newline.**

4. ROM Components

4.1 System Monitor (WOZMON)

The system monitor, WOZMON (or Wozniak Monitor) provides an environment for programming the APPLE-1.

Location: $FF00-$FFFF

Commands: - Examine Memory Address: (AAAA)
- Modify Memory Address Contents: (AAAA: BB CC…)
- Run Programs at Address: (AAAA R)

4.2 Integer BASIC

Apple Integer BASIC provides a high-level programming environment.

Location: $E000-$EFFF
Features: Integer-only arithmetic, 26 available variables (A-Z), arrays (DIM), loops (FOR/NEXT), subroutines (GOSUB/RETURN) and memory access (PEEK/POKE).
Activation: To enter Integer BASIC, type **E000R** from the system monitor.

5. User Interface

5.1 Hardware Control Buttons

HARD RESET: Clears RAM, resets CPU registers, and un-initializes VRAM
SOFT RESET: Resets the system to WOZMON, preserving memory contents.
CLEAR SCREEN: Erases characters on the display without changing the system state.
BREAK: Interrupts an Integer BASIC program. (Only applicable to $E000-$EFFF)
FULLSCREEN: Enlarges the terminal display for easier viewing.

Hardware operation shortcuts are implemented as key-binds for fullscreen mode.

SOFT-RESET: Control+S, **BREAK**: Control+C, **PASTE:** Control/CMD+V
HARD-RESET: Control+H, **CLEAR-SCREEN:** Control+W,
FULLSCREEN TOGGLE: Control+F

5.2 Memory Configuration

Radio buttons allow the user to configure the memory size between 4K-bytes ($0000-$0FFF) and 8K-bytes ($0000-$1FFF). If memory configuration is changed, the system will automatically hard-reset.

5.3 MOS 6502 CPU Speed

Under the memory configuration are two radio buttons. These radio buttons allow a user to configure the execution speed of the CPU with **Normal** or **Fast**.
Normal Execution Speed = 16,667 cycles
Fast Execution Speed = 16,667 cycles by 10 times

6. Program Loading

6.1 File Formats

HoneyCrisp v1.1 has support for three program file types:

- **.hc (Standard HoneyCrisp format)**
- **.bas (BASIC file format. Usually contains BASIC program source code)**
- **.txt (Standard text file format. Can also be used for BASIC program loading.)**

Examples of each format mentioned are listed below.

Example of .hc:

0300 ***(starting address)***
: A9 4C 20 EF FF A9 0D 20 EF FF
: A9 41 20 EF FF A9 0D 20 EF FF ***(Contents for the address and subsequents)***
: A9 4E 20 EF FF A9 44 20 EF FF

Example of .bas/.txt — Simply just raw source-code.

0 DIM A$(5) : PRINT : PRINT
10 INPUT “WHAT’S YOUR NAME”,A$
20 PRINT “HELLO, “;A$
30 END

6.2 Loading Procedure

The user clicks the browse button labeled "Load a Program" and selects a supported file. A parser reads the file, and depending on the file type, either:

A, .hc format: Sets the memory pointer based on the address lines, and writes the data bytes into the specified memory address.

B, .bas/.txt: These file types are automatically assumed to contain BASIC program source-code. As such, when they are loaded, the emulator automatically injects E 0 0 R, waits approximately two seconds, and proceeds to inject the code from the file into the interpreter.
After injection, a user can type RUN to start the loaded program.

7. Source Code

The HoneyCrisp emulator source code is available for download on GitHub.
The repository includes versions 1.0 and 1.1 of the emulator.

Use a web-browser to download the source code at the following address.
https://github.com/landonjsmith/honeycrisp

8. License and Attribution

8.1. MIT License

8.2. Acknowledgments/Thanks

- Steve Wozniak - Original Apple-1 hardware/software design
- 6502 CPU designers Chuck Peddle, Rod Orgill, and Wil Mathys at MOS Technology
- San Bergmans at www.sbprojects.net for his extensive Apple-1 documentation
- Will Scullin's **APPLE-1JS** as inspiration for the development of this emulator.
- Jeff Jetton for pointing out significant inaccuracies of v1.0

8.3. Contact Information

For questions, bug reports, or suggestions, please contact landon@producerjason.com.