Python Emulation of the NES's MOS6502 Processor

David Kersh
20th July 2023

PyData

Toumetis

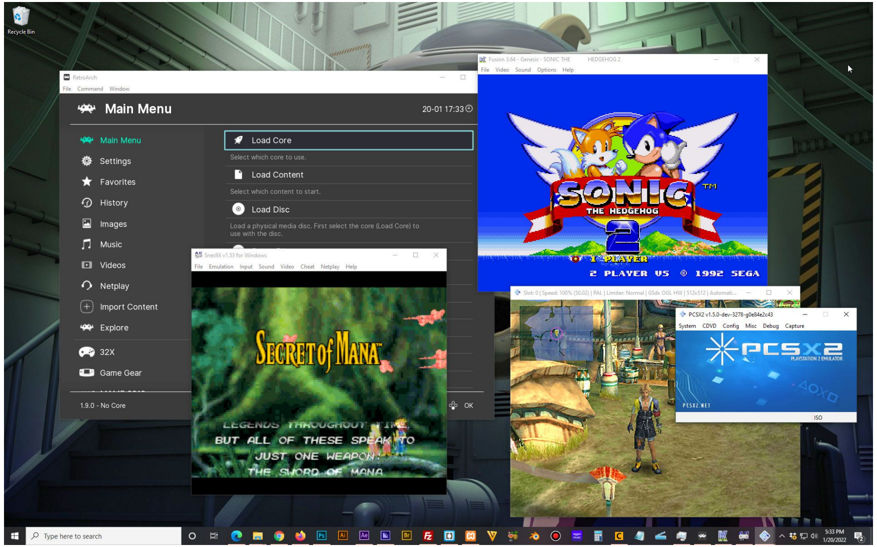
Contents

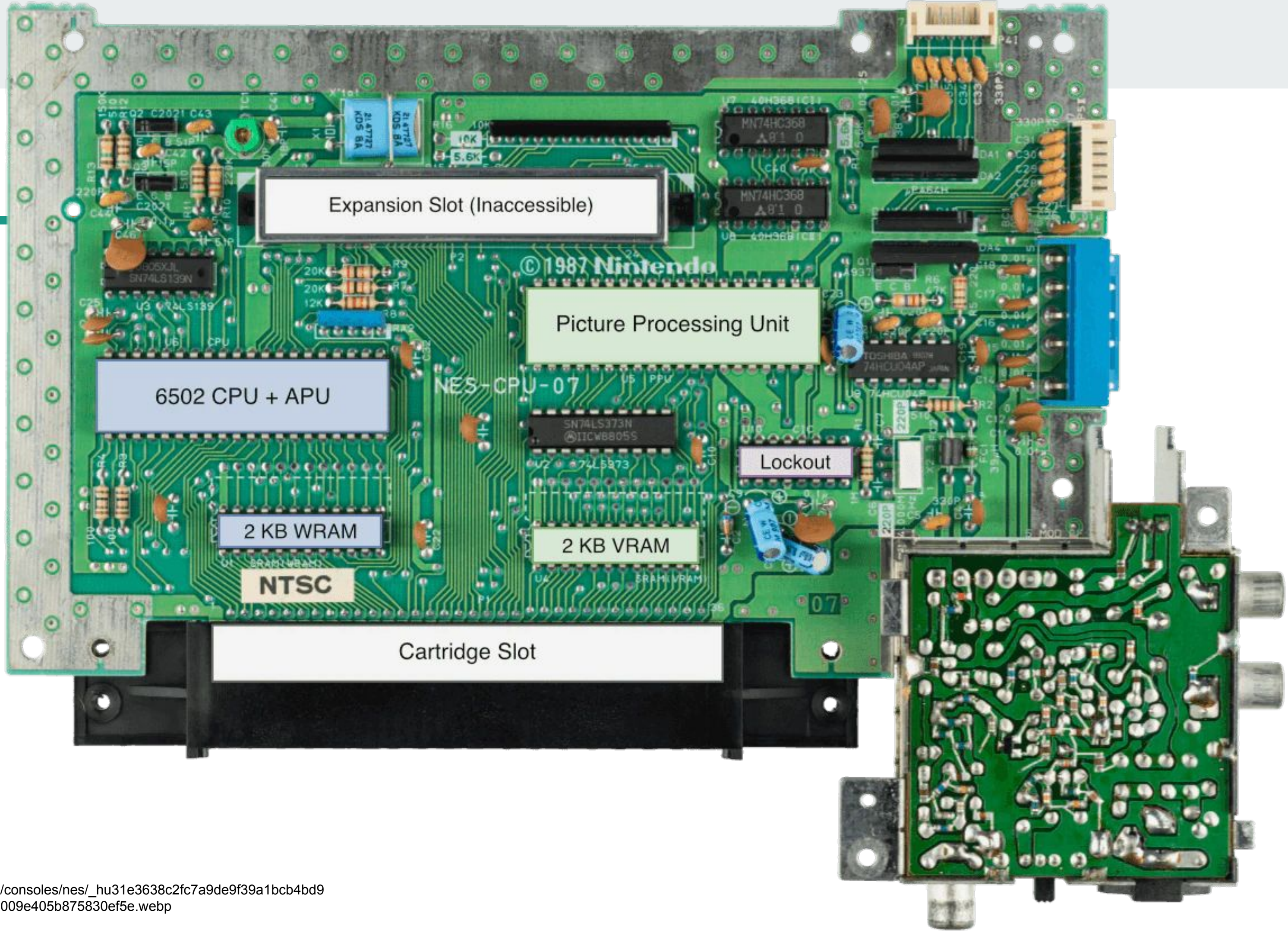
Why?

- Because *you can.*
  - Large number of resources available
    - Documentation, literature, forums, etc
- Powerful learning exercise
  - How processors work.
  - Assembly Language.
  - Coding Exercise.
- Appreciation for old technology.
- Projects are important.

https://fantasyanime.com/emuhelp/images0428/intro-video-game-emulators-2022.jpg

Background - What is emulation?

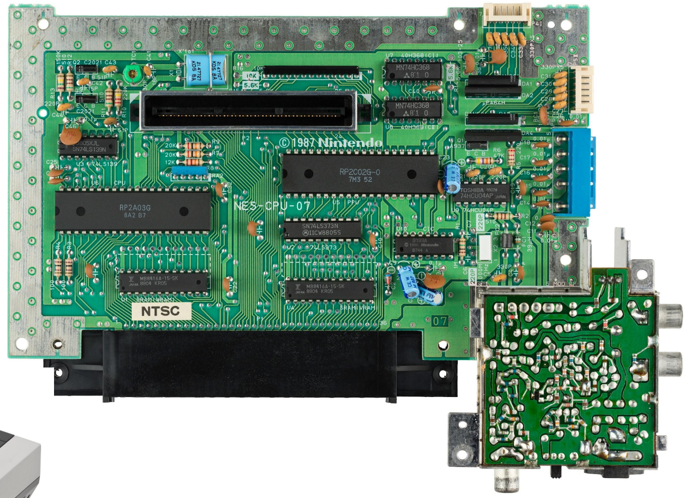




https://www.copetti.org/images/consoles/nes/_hu31e3638c2fc7a9de9f39a1bcb4bd9501_354137_7dbeaac9fb0451009e405b875830ef5e.webp https://en.wikipedia.org/wiki/Nintendo_Entertainment_System?useskin=vector#/media/File:Nintendo-NES-Mk1-Motherboard-Top.jpg

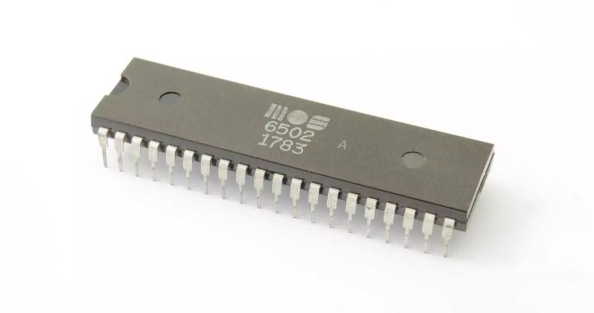
Background - The Nintendo Entertainment System

- CPU: Ricoh 2A03 / 2A07
    - MOS6502
        - Binary-coded decimal mode disabled.
            - Avoid patent issues (apparently)
    - 2x *Audio Processing Units* (APUs)
- WRAM: 2 KB
- PPU: Ricoh 2C02
- VRAM: 2 KB
- NTSC and PAL variants.



https://www.embeddedrelated.com/showarticle/1453.php

"To understand how big of an impact this chip had, all you have to do is look at [the 6502's] presence in many of the 8-bit systems of the era, sold by the millions."



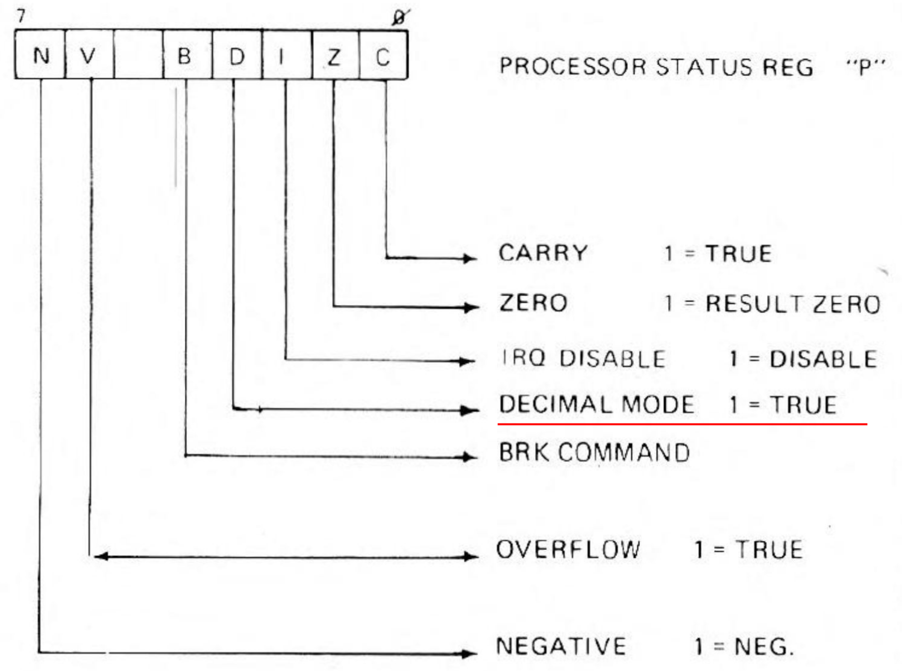
https://www.flickr.com/photos/hernan43/39614486174

MOS6502 (Ricoh 2A03 / 2A07)

- Six registers:
  - Program Counter (16 b)
  - Stack Pointer (8 b)
  - Accumulator (8 b)
  - Index X (8 b)
  - Index Y (8 b)
  - Status (8 b)
- 13 Addressing Modes
- Clock Speed = 1.79 (NTSC), 1.66 (PAL) MHz
- Each cycle either READ or WRITE
- 56 Instructions (*Op Codes*)
  - A bunch of unofficial ones too.

MOS6502 Registers

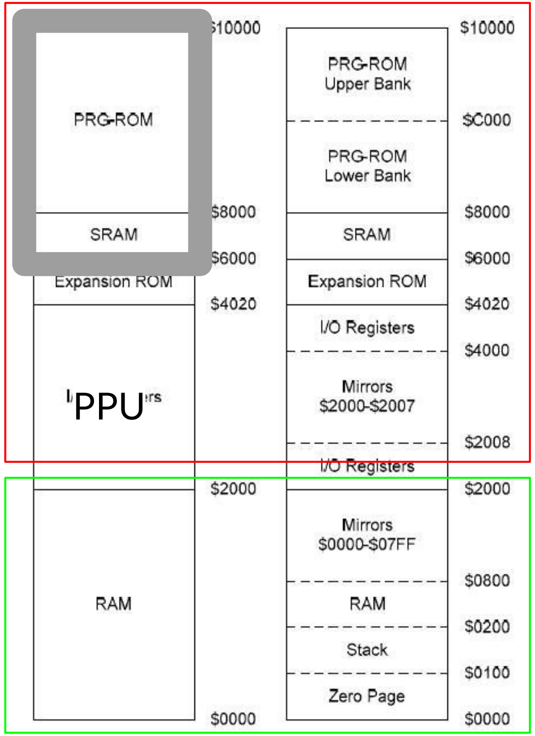
- Program Counter (16 b)
  - Points to the next instruction in memory.
  - Modified automatically as instructions are executed.
- Stack Pointer (8 b)
  - 256 B located between $0100 and $01FF
  - Decremented/Incremented with push/pop
- Accumulator, Index X, Index Y (8 b)
- Status (8 b)
  - Flags are set / cleared as instructions are executed.
  - Bitwise operators used to extract flags.



https://en.wikibooks.org/wiki/6502_Assembly
https://www.nesdev.org/wiki/CPU_memory_map

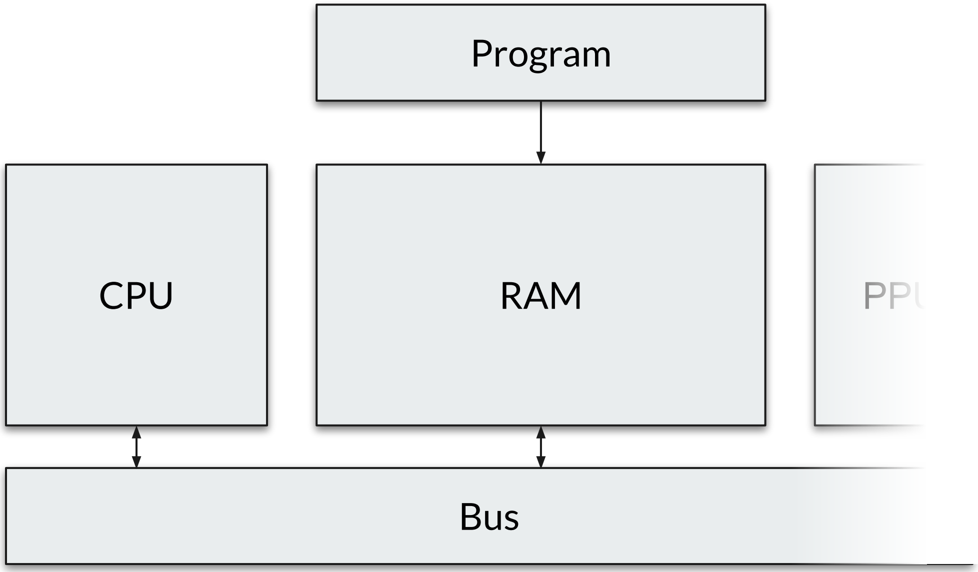
NES RAM and Memory Map

- 2 kB RAM
- [$0000 - $00FF] - Zero Page
  - Faster since only 1 B required to address.
- [$0100 - $01FF] - Stack
  - *Last-in First-out* data structure.
  - Grows backwards .
  - Tracks defined subroutines.
  - Efficient
- [$0200 - $0800] - General Purpose
- Since program counter is 16 bit, mirroring is used.
  - [$0000 - $0800] is mirrored three times:
  - [$0800 - $1000], [$1000-$1800], [$1800 - $2000]
    - Eg. Addressing $13FF == $03FF



System Architecture

- CPU
  - Read/Write
  - Ignore Clock
  - Ignore cycles
- BUS
- RAM
  - Program Loaded directly.
- Everything else ignored.
  - PPU
  - Cartridge
  - Expansion Slot
  - Mapper
  - etc



https://www.masswerk.at/6502/6502_instruction_set.html#layout
https://archive.org/details/mos_6500_mpu_preliminary_may_1976/page/n5/mode/2up?view=theater

MOS6502 Op Codes

HI	*LO-NIBBLE*															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	BRK impl	ORA X,ind				ORA zpg	ASL zpg		PHP impl	ORA #	ASL A			ORA abs	ASL abs	
1-	BPL rel	ORA ind,Y				ORA zpg,X	ASL zpg,X		CLC impl	ORA abs,Y				ORA abs,X	ASL abs,X	
2-	JSR abs	AND X,ind			BIT zpg	AND zpg	ROL zpg		PLP impl	AND #	ROL A		BIT abs	AND abs	ROL abs	
3-	BMI rel	AND ind,Y				AND zpg,X	ROL zpg,X		SEC impl	AND abs,Y				AND abs,X	ROL abs,X	
4-	RTI impl	EOR X,ind				EOR zpg	LSR zpg		PHA impl	EOR #	LSR A		JMP abs	EOR abs	LSR abs	
5-	BVC rel	EOR ind,Y				EOR zpg,X	LSR zpg,X		CLI impl	EOR abs,Y				EOR abs,X	LSR abs,X	
6-	RTS impl	ADC X,ind				ADC zpg	ROR zpg		PLA impl	ADC #	ROR A		JMP ind	ADC abs	ROR abs	
7-	BVS rel	ADC ind,Y				ADC zpg,X	ROR zpg,X		SEI impl	ADC abs,Y				ADC abs,X	ROR abs,X	
8-		STA X,ind			STY zpg	STA zpg	STX zpg		DEY impl		TXA impl		STY abs	STA abs	STX abs	
9-	BCC rel	STA ind,Y			STY zpg,X	STA zpg,X	STX zpg,Y		TYA impl	STA abs,Y	TXS impl			STA abs,X		
A-	LDY #	LDA X,ind	LDX #		LDY zpg	LDA zpg	LDX zpg		TAY impl	LDA #	TAX impl		LDY abs	LDA abs	LDX abs	
B-	BCS rel	LDA ind,Y			LDY zpg,X	LDA zpg,X	LDX zpg,Y		CLV impl	LDA abs,Y	TSX impl		LDY abs,X	LDA abs,X	LDX abs,Y	
C-	CPY #	CMP X,ind			CPY zpg	CMP zpg	DEC zpg		INY impl	CMP #	DEX impl		CPY abs	CMP abs	DEC abs	
D-	BNE rel	CMP ind,Y				CMP zpg,X	DEC zpg,X		CLD impl	CMP abs,Y				CMP abs,X	DEC abs,X	
E-	CPX #	SBC X,ind			CPX zpg	SBC zpg	INC zpg		INX impl	SBC #	NOP impl		CPX abs	SBC abs	INC abs	
F-	BEQ rel	SBC ind,Y				SBC zpg,X	INC zpg,X		SED impl	SBC abs,Y				SBC abs,X	INC abs,X	

https://www.nesdev.org/obelisk-6502-guide/reference.html#LDA

Example - LDA

LDA - Load Accumulator

A,Z,N = M

Loads a byte of memory into the accumulator setting the zero and negative flags as appropriate.

C	Carry Flag	Not affected
Z	Zero Flag	Set if A = 0
I	Interrupt Disable	Not affected
D	Decimal Mode Flag	Not affected
B	Break Command	Not affected
V	Overflow Flag	Not affected
N	Negative Flag	Set if bit 7 of A is set

Addressing Mode	Opcode	Bytes	Cycles
Immediate	$A9	2	2
Zero Page	$A5	2	3
Zero Page,X	$B5	2	4
Absolute	$AD	3	4
Absolute,X	$BD	3	4 (+1 if page crossed)
Absolute,Y	$B9	3	4 (+1 if page crossed)
(Indirect,X)	$A1	2	6
(Indirect),Y	$B1	2	5 (+1 if page crossed)

See also: LDX, LDY

Basic Example Program - Storing Values in Memory

Python	6502 Assembly	6502 Machine Code*

`data = [10, 3, 35]`	`LDA #0A`
	`STA $012c`
	`LDA #03`
	`STA $012d`
	`LDA #23`
	`STA $012e`
		`A9 0A 8D 2C 01 A9 03 8D 2D 01 A9 23 8D E2 01`

Implementation - RAM

Implementation - CPU - outline

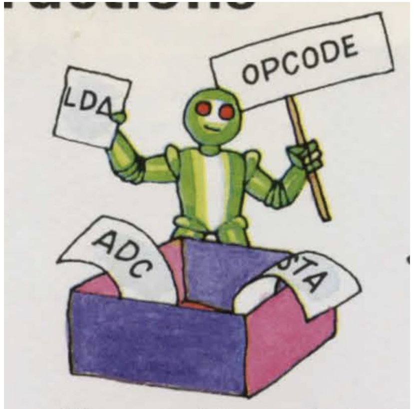
Implementation - CPU - OpCodes

Implementation - Bus

Demo

Unit Testing (if there's time)

- https://github.com/TomHarte/ProcessorTests/tree/main/nes6502
  - 10,000 tests for each opcode.
  - Assumes a 64 KB RAM
  - Performs a single operation and checks CPU state.
- `pytest-html`



Summary and Next Steps

- A MOS6502 CPU was emulated using Python.
    - Incomplete, but we're playing Snake
- Implement a clock.
    - I'd quite like to implement a random number generator too.
- Finish the last of the CPU instruction set.
- Get to work on the PPU.
- Get an actual NES game running.

Useful Resources

- Javidx9 - NES Emulation (in C++)
  https://www.youtube.com/watch?v=F8kx56OZQhg
- Writing a NES Emulator in Rust (ebook)
  https://bugzmanov.github.io/nes_ebook/chapter_1.html
- NesDev Wiki
  https://www.nesdev.org/wiki/Nesdev_Wiki
- NES Opcodes
  https://www.nesdev.org/obelisk-6502-guide/reference.html
- In-depth discussion of the NES architecture
  https://www.copetti.org/writings/consoles/nes/
- Various resources regarding the 6502 processor
  http://www.6502.org/tutorials/
- Cornucopia of retro coding books
  http://retro.hansotten.nl/6502-sbc/kim-1-manuals-and-software/books/
- https://skilldrick.github.io/easy6502/
- /r/EmuDev