A Behavioral Approach to 6502 Emulation *

M. J. Coppola, N. S. Shelby, L. E. Carleton
Computer Science 330
Siena College
Loudonville, NY, 12211

Abstract

An abstract goes here. We made a 6502 emulator that was partially implemented and easy to extend. There were some results and we're really proud. Or something along those lines.

1 Overview

The RP2A03 is the CPU found in the Nintendo Entertainment System (NES), launched October 18th, 1985 [3, 6]. This processor was based off the MOS Technology 6502, differences being a lack of a decimal mode and the inclusion of the Audio Processing Unit (APU) [3]. Our goal is to create a partial implementation of an emulator for this CPU designed to be easily extended to a simple NES emulator.

The emulator was designed with the NES in mind. That being said, we only emulate the behavior of the 6502, not precisely the actual hardware. This serves us a huge simplification in implementation. Official operations of the 6502 only need to be calculated, and not emulated. We opted in for calculating the output of the operations immediately, and then waiting the number of clock cycles that operation took to execute.

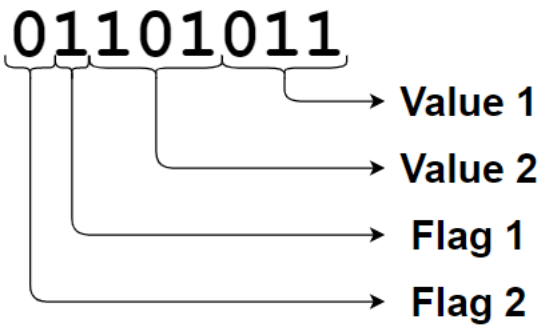
Overview to be continued...

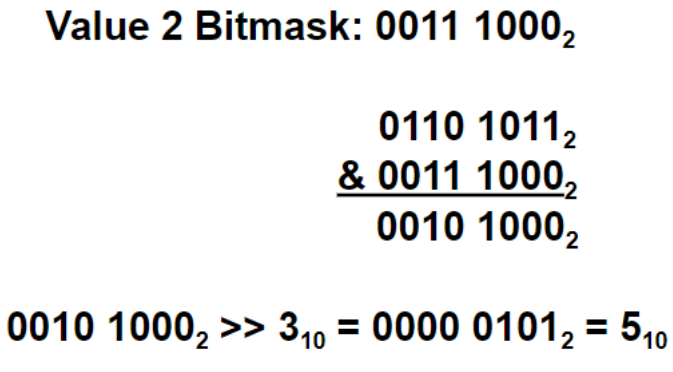
*This work was completed in partial fulfillment of the final project requirement for Computer Science 330 at Siena College, Fall 2020.

2 Bitwise Operations

Bitwise operations are major aspect of making an efficient emulator. Suppose we have an 8-bit value. We typically express this value as a number, or possibly two hexidecimal digits. One way the 6502 uses 8-bit numbers is by defining parts of the binary expression as flags or smaller width values, shown in figure 1a. In emulation, it's important that we can extract information from this 8-bit number, as well as convert information we already have into information we can store into this number. To achieve this, we use bitwise operations.

To extract information from a binary number, we would first need to isolate that information using a bitmask. We can create a bitmask by creating a new 8-bit number that has 1's placed over the important digits, and 0's on digits that are not. With this mask, we can use a bitwise AND operation (&) and have a result that only includes the digits defined by the bitmask shown in





(a) A single binary value representing multiple values. (b) Using a bitmask to extract values from a binary number.

figure 1b [7].

In order to make this value useful to us, we may have to then use a bitwise shift-left ($>>$) to push the important digits to the least significant order of the digit. By doing this, the entirety of the resulting 8-bit value numerically represents the value stored in the section of the original 8-bit value [7].

A similar process is used for setting bits as well. We create a bitmask for the bits we want to set and or it with our value. To unset, we can & with the inverse (˜) of our mask. We can toggle/flip bits with a mask that is XOR'd (ˆ) with our value.

3 NES Hardware

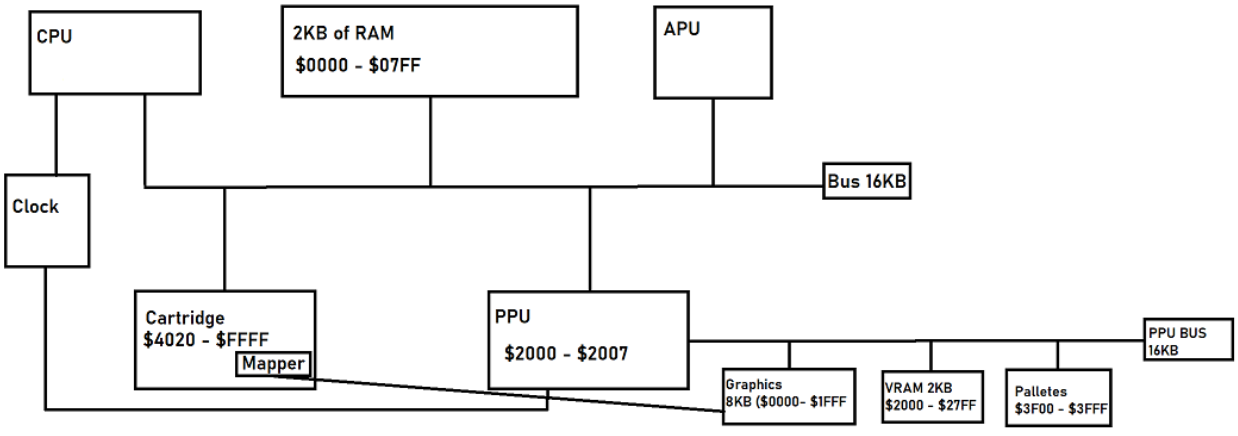


Figure 2: NES hardware bus diagram

The RP20A3 (6502) in the NES communicated to other devices on the machine using a 16-bit

addressable hardware bus. The CPU can read and write information to any device connected to the bus [4]. Addresses $0000 to $00FF are mapped to the NES's 2KiB of internal memory. Parts of this memory has predefined purposes dictated by the CPU architecture. The zero page is mapped to $0000-$00FF and the stack is mapped to address $0100 and onwards up to $01FF [4]. In our implementation, we map the entirtey of the bus to read/write memory. A full NES implementation would have other devices on the bus.

The 6502 features 6 registers. The first major register is the 1-byte wide Accumulator (A) register. This register is used with the ALU, and supports using the status register to describe its behavior, such as denoting if its value is overflowing, negative, zero, etc. The X and Y registers are used for different addressing modes. These registers were useful for looping when used with the increment and decrement instructions along side addressing with the value found in them. The program counter (PC) is a 2-byte wide register for tracking the current instruction in memory. The stack pointer register (S) is a 1-byte wide register that holds the manipulatable value that points to the stack portion of memory. The status (P) register hold 8 bytes (6 used) that store the current status flags of the CPU [1].

The 6502 has 3 index addressing modes. Addressing modes define where the CPU operations source operands from. Zero page indexing uses a 1-byte address to find a value found in the hard-coded zero page. We can offset the address to be read using the values in the X and Y register. Absolute addressing uses a 2-byte wide address to find memory outside of the zero page. Similarly, the X and Y registers can be used to offset this address. The 6502 also features indirect addressing, which is it's solution for simulating the function of pointers. The mode reads the value at the address found at the supplied address. Unlike the previous addressing modes, the X and Y offsets server different purposes from each other. The X register is used to offset the supplied address for the mode, while the Y register is used to offset the value found at the supplied address [2].

Other addressing modes are also used. Many instructions operate directly on the accumulator. Others address have immediate values from the program memory. Relative addressing is used by branching instructions to move to a new instruction in program memory from -128 to 128 bytes away from the branch's address.

As instructions execute, the CPU updates the status register to reflect 6 different boolean flags. The carry (C) flag turns on when an addition or subtraction carries or borrows a bit. This flag

is also set if a logic shift pushes out a 1. The zero (Z) flag is turned on if an instruction results in a zero. The interrupt disable flag is used to enable and disable maskable CPU interrupts. The overflow flag (V) is set when overflow is detected during an addition or subtraction, and in a few other niche cases. The negative flag (N) is set when an instruction results in a negative value [**?**].

The first byte of an instruction is the opcode. Although only 56 opcodes are documented for the 6502, there are 256 addressable opcodes in the CPU. The undocumented opcodes, known as illegal opcodes, usually contain microcode the assist in the execution of the outward facing legal opcodes. An accurate emulator will simulate the microcode, therefore the legal opcodes utilizing the microcode, as well as executing them on the rare chance a developer uses one. Although a small subset of NES titles utilize these unofficial opcodes, we will not implement them in the interest of time at the cost of 8 titles [5].

References

[1] Cpu registers. `http://wiki.nesdev.com/w/index.php/CPU_registers`, 2015. Accessed: 2020-11-20.

[2] Cpu addressing modes. `http://wiki.nesdev.com/w/index.php/CPU_addressing_modes`, 2018. Accessed: 2020-11-20.

[3] Cpu. `wiki.nesdev.com/w/index.php/CPU`, 2019. Accessed: 2020-11-17.

[4] Cpu memory map. `http://wiki.nesdev.com/w/index.php/CPU_memory_map`, 2019. Accessed: 2020-11-20.

[5] Cpu unofficial opcodes. `https://wiki.nesdev.com/w/index.php/CPU_unofficial_opcodes`, 2020. Accessed: 2020-11-20.

[6] Nintendo entertainment system. `https://en.wikipedia.org/wiki/Nintendo_Entertainment_System`, 2020. Accessed: 2020-11-17.

[7] O. Lawlor. Bits and bitwise operators. `https://www.cs.uaf.edu/courses/cs301/2014-fall/notes/bits-bitwise/`, 2014. Accessed: 2020-11-17.