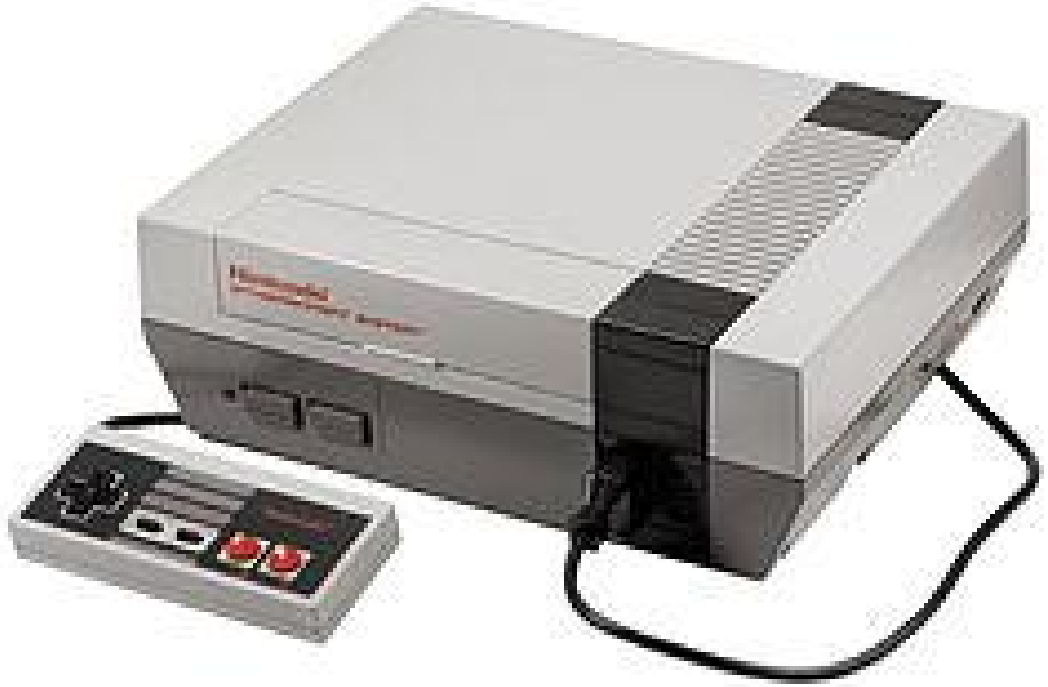
8-bit Game Programming with the NES

bit.ly/stars8bit

Barry Peddycord III
@isharacomix

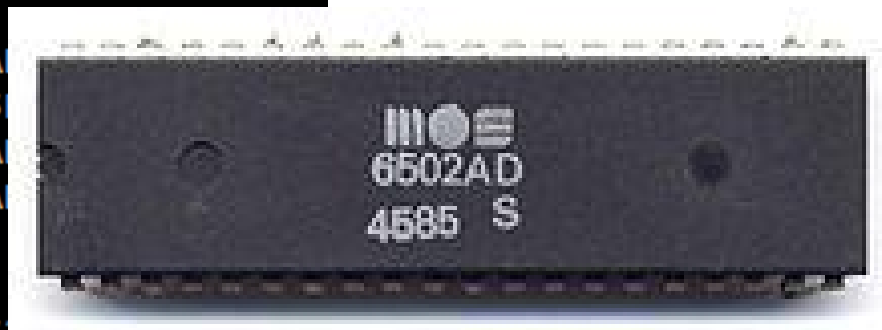
Before we get started...

Who knows what this is?



Who LOVES to program in assembly?

```
        JMP checksides
hitbottom:
        LDA BA
        EOR #$
        STA BA
        INC BA
checksides:
        TYA
        CPX #$2
        BCC crossleft
        CPX #$FD
        BCS crossright
        CPX #24
        BEQ leftpaddle
        CPX #222
        BEQ rightpaddle
        JMP endcollisions
leftpaddle:
        SEC
        SBC PADDLE_1Y
        ADC #8
        CMP #40
        BCC flipflop
```



@isharacomix

Schedule

1st Half 10:00-11:00

- The Hardware and how to use it
- The NES Programming Paradigm
- Getting our hands dirty

BREAK 11:00-11:30

2nd Half 11:30-12:30

- Programming exercises
- Discussion and wrap-up

NES at a Glance

- Introduced in America in 1985
- MOS 6502 CPU
  - Same chip family as Apple][, C64, and Atari 2600
- 2 KB of RAM
- 2-bit color depth, 8 color palettes
- 64 concurrent 8x8 sprites

NES Hardware

- The CPU (central processing unit)
  - A slightly modified MOS 6502
  - Reads instructions from game cart
- The PPU (picture processing unit)
  - Handles rendering backgrounds and sprites
  - Maintains 512 8x8 pixel tiles for sprites and bkgs
- The APU (audio processing unit)
  - Does “everything else”
  - Has 5 sound channels
  - Also handles controller input and output

Memory-Mapped Registers

- Unlike modern processors, 6502 does not support interrupts
- Instead, hardware exposes pins that are directly wired to the CPU's memory lines
- This means that some memory locations, like $4017, are actually aliased to hardware registers, like player 2's controller

Takeaway #1

The hardware does the heavy lifting. As long as the programmer just puts their data in the right locations, the NES takes care of the rest.

NES programming is basically just picking up data, fiddling with it, and putting it in special memory locations.

Memory Pages

- The 6502 can access 64 KB of memory
- Addresses are 16 bits (two bytes)
- Example: $FF88
  - The high byte (FF) is called the “Page”. There are 256 addressable pages.
  - Each page then has 256 addressable bytes.

A note on syntax:

$xxxx means a hexadecimal number

%xxxx means a binary number

How do we know which locations to use?

$0000-$07FF	RAM
$0800-$1FFF	RAM Mirror
$2000-$2007	PPU Registers
$2008-$3FFF	PPU Reg Mirrors
$4000-$401F	APU Registers
$4020-$7FFF	Cart Hardware
$8000-$FFFF	Cart ROM

- The MEMORY MAP
- Explains every single memory location that the CPU access and what it does.

Checkpoint and stretch break

We've just discussed the hardware without getting too detailed. Do we have any questions before moving on to the programming flow?

What an NES Program Looks Like

Visit http://dev.8bitmooc.org/help/warmup

- First, get the starting address of the program from the Vector Table ($FFF0)
- The PPU needs to warm up. We can use this time to get the NES into a known state by zeroing out memory
- After we're done, we initialize our game and turn on the graphics.

Try it yourself!

Copy **http://dev.8bitmooc.org/help/warmup** into the sandbox.

What an NES Program Looks Like 2

- The timing of the NES is determined by the screen rate.
- Every time the screen refreshes, the NES gets a ‘non-maskable interrupt’ (NMI).
- In our warmup code, we go into an infinite loop after initializing, and then put the code we want to run each step in the NMI handler.

Two Programming Exercises

When we come back, we are going to do two programming exercises:

- Getting user input from the controller
- Drawing sprites in a Pong game

Exercise 1: User Input

Visit http://dev.8bitmooc.org/challenge/1-1

How to read the controller:

- First, write a ‘1’ to $4016
- Then write a ‘0’ to $4016
- Each time you read $4016, you will get a 1 or a 0 in the least significant bit based on which button was pressed.
- A, B, Start, Select, Up, Down, Left, Right

Exercise 2: Drawing Sprites

Visit http://dev.8bitmooc.org/challenge/1-3

Sprite information is in a 4-byte data structure

- byte 0: Y position
- byte 1: which 8x8 tile in the spritesheet
- byte 2: properties, such as color
- byte 3: X position

write the page that your sprite data is on (3, in this case) into $4014 to perform DMA

So, that’s that