Reference

CPU

NES Specs
Ricoh 2A03 (modified MOS 6502)
CPU Clock: 1.7897725 MHz (NTSC)
PPU Clock: 5.3693175MHz (NTSC)
Internal Memory: 2K WRAM, 2K VRAM, 256 Bytes SPR-RAM

Timing
The PPU clock runs 3x faster than the CPU. So during 1 CPU cycle, 3 PPU cycles will pass.
CPU vs. PPU Timings:

CPU cycles per scanline:	113.6666
CPU cycles per frame (1st frame):	29780.66
CPU cycles per frame (2nd frame):	29780.33
CPU cycles per two frames:	59561.00

VBlank lasts 262-243 = 19 scanlines = 6,479 PPU cycles = 2,159.66 CPU cycles
HBlank lasts 341-256-1 = 84 PPU cycles = 28 CPU cycles

- A “dot” (pixel) takes 1/3 CPU cycles to draw.
- 341 dots are “drawn” in every scanline (HBlank starts after 256 dots).
- 262 scanlines are drawn in every frame.
  - Scanline 0 (first after VBlank is over) is a dummy scanline.
  - Scanlines 1 to 241 render the 240 lines of the display.
  - Scanline 242 does nothing.
  - Scanlines 243 - 262 are the VBlank period
- Since a fractional number of CPU cycles run during a frame, NTSC outputs a short scanline (scanline 0) with only 340 dots every 2nd frame to make cycle count an integer.

Reference:
CPU & PPU Clock Timings
PPU Timing Detail (Everynes)
PPU Timing Detail (Nesdev)
NTSC PPU Timing Diagram

CPU Status Registers

```
bit ->   7                           0
       +---+---+---+---+---+---+---+---+
       | N | V |   | B | D | I | Z | C |  <-- flag, 0/1 = reset/set
       +---+---+---+---+---+---+---+---+


N  =  NEGATIVE. Set if bit 7 of the accumulator is set.

V  =  OVERFLOW. Set if the addition of two like-signed numbers or the
      subtraction of two unlike-signed numbers produces a result
      greater than +127 or less than -128.

B  =  BRK COMMAND. Set if an interrupt caused by a BRK, reset if
      caused by an external interrupt.

D  =  DECIMAL MODE. Set if decimal mode active.

I  =  IRQ DISABLE.  Set if maskable interrupts are disabled.

Z  =  ZERO.  Set if the result of the last operation (load/inc/dec/
      add/sub) was zero.

C  =  CARRY. Set if the add produced a carry, or if the subtraction
      produced a borrow.  Also holds bits after a logical shift.
```

Ref: http://nesdev.com/6502guid.txt

The Zero Flag:
The Z flag of the status register is set to 1 in the following cases:

- INC, INX, INY if result (new value of M/X/Y) is 0.
- LDA, LDX, LDY if A/X/Y was set to 0.

The Carry Flag:
A carry is produced any time a 1 bit is added to a 1 bit (1+1 = 0 with C=1 set).
A borrow is produced any time a 1 bit is subtracted from a 0 bit (0-1 = 1 with C=0 set).
If there is no carry with an add, C=0 is set. If there is no borrow with a subtract, C=1 is set.

The Negative Flag:
The N flag is cleared (N=0) if the last instruction resulted in a value < $80 (Bit7=0).
The N flag is set (N=1) if the last instruction resulted in a value >= $80 (Bit7=1).

Programming

Opcodes

Logic:
ORA: OR
EOR: Exclusive OR
AND: AND

Comparison:
CMP: Compare Memory and Accumulator
CPX: Compare Memory and X
CPY: Compare Memory and Y

Condition	N	Z	C
Register < Memory	1	0	0
Register = Memory	0	1	1
Register > Memory	0	0	1

Branching:

Opcode	Name	Condition
BCC	Branch on Carry Clear	C = 0
BCS	Branch on Carry Set	C = 1
BNE	Branch on Result Not Zero	Z = 0
BEQ	Branch on Result Zero	Z = 1
BPL	Branch on Result Plus	N = 0
BMI	Branch on Result Minus	N = 1
BVC	Branch on Overflow Clear	V = 0
BVS	Branch on Overflow Set	V = 1

Note: Branching uses relative addressing, which can jump to an instruction -128 to +127 bytes from the current position of the PC. For longer jumps, use JMP.

Adding/Subtracting:
ADC `M` (Add with Carry) | Result: (A + M + C) → A
SBC `M` (Subtract with Carry) | Result: (A - M - ¬C) → A

* To cleanly add two numbers, `CLC` must be run before `ADC` to clear carry.
* To cleanly subtract two numbers, `SEC` must be run before `SBC` to set carry.

Status Register Flags:

`CLC`	(Clear Carry)	Result: C = 0
`SEC`	(Set Carry)	Result: C = 1
`CLI`	(Clear Interrupt)	Result: I = 0
`SEI`	(Set Interrupt)	Result: I = 1
`CLV`	(Clear Overflow)	Result: V = 0
`CLD`	(Clear Decimal)	Result: D = 0
`SED`	(Set Decimal)	Result: D = 1

Test Bits:

`BIT` `M`

Performs M `AND` A and sets Z=1 if result is zero (i.e. %00000000), and Z=0 otherwise.

Note: *Immediate addressing is undefined for BIT.*
Use a variable to store mask instead of a constant.

Example 1: `LDA #%11111000` followed by `BIT #%00001000` will set Z=0, showing that bit 3 of `%11111000` is 1.

Example 2: `LDA #%11110000` followed by `BIT #%00001000` will set Z=1, showing that bit 3 of `%11110000` is 0.

Direct Register Addressing:

Allowed: ASL A/X/Y, LSR A/X/Y

Not Allowed: INC A/X/Y, ADC A/X/Y, SBC A/X/Y, CMP A/X/Y, CPX A/X/Y, CPY A/X/Y

Reference: 6502 Opcodes

https://www.masswerk.at/6502/6502_instruction_set.html
http://www.6502.org/tutorials/6502opcodes.html
http://www.thealmightyguru.com/Games/Hacking/Wiki/index.php/6502_Opcodes
http://obelisk.me.uk/6502/reference.html

PPU & Graphics

Specs

Screen Resolution: 256x240 pixels / 32x30 tiles

Actual Resolution: 256x224 pixels (NTSC crops the top and bottom 8 rows of pixels.)

Palettes
The NES holds color information in two 16-byte palettes: the background palette ($3F00) and the sprite palette ($3F10). Each palette holds 16 colors in four groups. Each 8x8 pixel sprite and each 16x16 pixel sub-block of the background must use one of these four 4-color groups to determine all of its colors.

Name Tables
A nametable holds one full screen of background tiles. The NES has space for 2 nametables. Nametable 0 starts at PPU address $2000 and takes up 960 bytes (1 byte per tile, 32x30 tiles).

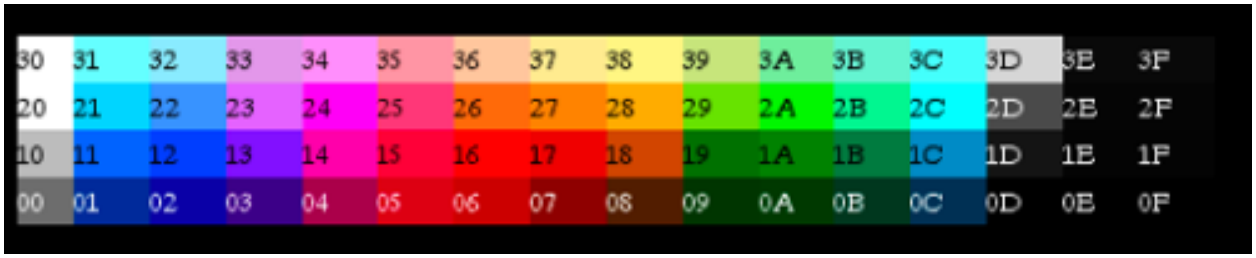
Attribute Tables
An attribute table determines which colors of the background palette a nametable can use:

1. The screen is divided into 32x32 pixel (4x4 tile) blocks. (Since the screen is only 30 tiles tall, the bottom half of the bottom row of blocks is cut off. Thus there are 8 columns and 7.5 rows of blocks rendered on-screen.)
2. Each byte in the attribute table specifies data for one of these blocks.
3. The 32x32 pixel (4x4 tile) blocks are subdivided into four 16x16 (2x2 tile) sub-blocks.
4. Each pair of bits in the attribute table byte for a given block determine the 4-color group of consecutive background palette colors that can be used in the corresponding sub-block. (Thus only 4 colors can be used in each 16x16 pixel area.)
5. Bits 0-1, 2-3, 4-5, 6-7 determine the 4-color palette group for the upper-left, upper-right, lower-left, and lower-right sub-block respectively.

Pattern Tables
The PPU has RAM area for two 4KB pattern tables called “left” ($0000) and “right” ($1000). Each table can be used for either sprites or background, which is specified using port $2000 (PPUCTRL).
Each pattern table contains 256 tiles.
Each tile consists of a 8x8 pixel bitmap with 2 bit depth (4 colors). Each tile occupies 16 bytes, the first 8 bytes contain color bit 0 for each pixel, the next 8 bytes color bit 1.
Each byte defines a row of 8 pixels (MSB left).

The 8KB Pattern Table area of PPU memory is also known as CHR-ROM.
Pattern Table Format: https://wiki.nesdev.com/w/index.php/PPU_pattern_tables

Valid Colors



Note: Color $0D produces unstable behavior.

Fonts
https://forums.nesdev.com/viewtopic.php?t=8440

Sprite 0 hit
The “Sprite 0 hit” flag ($2002.6) is set when the PPU reaches the first *non-transparent* pixel of the first sprite in OAM, sprite #0.
To hide sprite 0, its priority bit (bit 5 of the sprite attributes byte) can be set to place it behind the background. (Note, it must also be behind a non-transparent background pixel to be hidden.)

The sprite 0 flag is set when the PPU *draws* the first pixel, not when it reads it.
The PPU is always 2 pixels behind. It draws pixel 0 on cycle 2, pixel 248 on cycle 250, etc.

More conditions

Reference
https://wiki.nesdev.com/w/index.php/The_frame_and_NMIs
NN Week 9: Binary to Decimal, Displaying Numbers

Controllers

CPU RAM addresses $4016 and $4017 are used for controller 1 and 2 respectively.
The standard NES joypad only uses the 0th bit (LSB) of each data byte for all buttons.

A Joypad read is triggered by writing a 1, then a 0 to address $4016.
Then, reading $4016 (or $4017) will return one bit per read representing the state of each joypad button in the order:
A, B, Select, Start, Up, Down, Left, Right.

Using the traditional method of condensing the bitstream into a single byte variable, we have the following button values:

```
A  = %10000000 = $80
B  = %01000000 = $40
St = %00100000 = $20
Se = %00010000 = $10
U  = %00001000 = $08
D  = %00000100 = $04
L  = %00000010 = $02
R  = %00000001 = $01
```

Controller info: http://problemkaputt.de/everynes.htm#controllers

---

APU and Sound

Period and Notes
Notes are specified to the APU by an 11-bit wavelength/period. For a desired frequency F:
*P = C/(F*16) - 1*, where P=Period, C=CPU speed (in Hz), F=Frequency of the note (in Hz).

NTSC note table: http://www.freewebs.com/the_bott/NotesTableNTSC.txt

---

Mappers and Bank Switching

MMC1 (iNES Mapper 1)
Any write to a CPU address in $8000-$FFFF with bit7=1 set resets the shift register. Five consecutive writes to a CPU address in $8000-$FFFF with bit7=0 will load the shift register with the bit0 values of each write. Bits 14 and 13 of the address of the fifth write selects the internal register destination of the shift register contents.

Register	Address Range
Control	$8000-$9FFF
CHR bank 0	$A000-$BFFF
CHR bank 1	$C000-$DFFF
PRG bank	$E000-$FFFF

Example (Strider):
We want to write the value **01110** to the **PRG bank** register ($E000-$FFFF).
Shift register (SR) will be initialized with %10000 after reset.

```
LDA #$0E      [A = %000001110]
STA $FFE1     [SR = %01000]
LSR A         [A = %00000111]
STA $FFE1     [SR = %10100]
LSR A         [A = %00000011]
STA $FFE1     [SR = %11010]
LSR A         [A = %00000001]
STA $FFE1     [SR = %11101]
LSR A         [A = %00000000]
STA $FFE1     [SR = %01110]
```

After the last byte is written and the shift register is full, the mapper chip will load the value **01110** into the PRG bank register as selected by our choice of address, $FFE1.

The MMC1 mapper chip has a maximum capacity of 256KB available for extra PRG ROM (256KB is most common, but some chips can use more or less).
In 16KB mode, this is split into up to 16 PRG ROM banks, each 16KB. In 32KB mode, it is split into 8x 32KB banks. (See chart in Appendix.)

NESASM Syntax

Directives (`.bank`)
Program ROM and character (sprite) ROM are laid out in banks. The NES has one 16KB PRG-ROM bank and one 8KB CHR-ROM bank.
PRG-ROM is stored in CPU memory. CHR-ROM is stored in PPU memory (Pattern Tables).
These are specified for the assembler by adding `.inesprg 1` and `.ineschr 1` respectively to the program header.
The **`.bank`** command switches between ROM banks in 8KB chunks. The assembler supports up to 128 banks, but the NES (without expansion) supports only 3:

- `.bank 0` = First 8KB of PRG-ROM
- `.bank 1` = Last 8KB of PRG-ROM
- `.bank 2` = All 8KB of CHR-ROM

Directives (`.org`)
“Set the location of the program counter. The thirteen lower bits of the address inform the assembler about the offset in the ROM bank and the three upper bits represent the page index.”

Directives (`.db`)
Stores one or more data bytes at the current location.
Equivalent to `.byte`.

Tutorials

Nerdy Nights Tutorial (Mirror #1)
Nerdy Nights Tutorial (Mirror #2)

NES Dev Tutorial with Environment Setup

Resource Links

NES
https://en.wikibooks.org/wiki/NES_Programming
http://problemkaputt.de/everynes.htm
https://wiki.nesdev.com/w/index.php/Programming_guide

General 6502
http://skilldrick.github.io/easy6502/#intro
http://nesdev.com/6502guid.txt

Hardware and Cartridge Making

http://www.raphnet.net/electronique/nes_cart/nes_cart_en.php

Appendix

Cartridge ROM address ranges for MMC1 mapper chip in 16KB PRG ROM bank mode

Bank number	Address range
0 [0000]	$00000 - $03FFF
1 [0001]	$04000 - $07FFF
2 [0010]	$08000 - $0BFFF
3 [0011]	$0C000 - $0FFFF
4 [0100]	$10000 - $13FFF
5 [0101]	$14000 - $17FFF
6 [0110]	$18000 - $1BFFF
7 [0111]	$1C000 - $1FFFF
8 [1000]	$20000 - $23FFF
9 [1001]	$24000 - $27FFF
10 [1010]	$28000 - $2BFFF
11 [1011]	$2C000 - $2FFFF
12 [1100]	$30000 - $33FFF
13 [1101]	$34000 - $37FFF
14 [1110]	$38000 - $3BFFF
15 [1111]	$3C000 - $3FFFF