```
Table Of Contents

1:                       Introduction
1a:                      Thank you's
1b:                      Acronyms

2:                       The 2A03 (CPU + Sound)
2a:                      Memory Map
2b:                      General Information
2c:                      Interrupts
2d:                      Registers
2e:                      Notes
2f:                      Summary of CPU and APU Registers

3:                       2C02 (Picture Processing Unit)
3a:                      Memory Map
3b:                      Tileset
3c:                      Name Tables
3d:                      Name Tables Mirroring
3e:                      Palette
3f:                      Sprite Ram
3g:                      Scrolling
3h:                      Rendering
3i:                      Data Transfer
3j:                      Summary of PPU Registers

4:                       External Device
4a:                      General Information
4b:                      Joypad
4c:                      4 Player Adapter
4d:                      Light Zapper
4e:                      Paddles
4f:                      Power Pad

5:                       NES Formats
5a:                      iNes Header
5b:                      UNIF header
5c:                      Boards

+------------+
|Introduction|
+------------+

1a Thank you's
--------------

        Brad Taylor:        For writing great technical documents, reverse engineering
many pieces
                            of the NES to get the information.

        Blargg:             For finding a lot of information about the NES sound.

        Blof:               Tested most of the ROMs Quietust's & I have written.

        Chris Covell:       Documenting the colour emphasis bits, creating a general FAQ,
making a great
                            palette and making many great NES demo's. Also for having a
cool NES page.

        Jamethiel:          Helping me out with some information I just didn't get and
many Q`s.

        Kevin Horton:       Reverse engineering *many* mappers and documenting them. Also
for making CopyNes!

        Loopy:              For documenting the scrolling during rendering.

        Mark Knibbs:        Reverse engineering mappers, assisting with sound and other
```

```
document.
                            Creating the nesdev mailing list.

        Matrixz:            For testing some rom's i written and accidently giving me
idea to test.

        Memblers:           Creating a great NES development site, organizing and
collecting information and
                            providing a great message board along with answering some Q's
I had.


        Quietust:           Giving me loads of information I could not find anywhere
else. Writing some NES
                            images I tested with my emulator, advice, non stop Q`s and
being a pain in my ass.

        ReaperSMS:          Thanks for knocking some information into me such as the
input for the pad
                            and some other misc info I just didn't get along with many
Q`s.

        Tennessee           For his unif format and his many post in the nesdev mailling
list.
        Carmel-Veilleux:

        Myself:             Gathering information, testing some on hardware, writing my
emulator (Reminesce)
                            to make sure I understood and tested information. And for
writing this doc. ^_^

        And anyone else who written a doc, reverse engineered carts or has found
information about the NES.


1b Acronyms
-----------

        You may not find these words in the doc but you'll find find them around the
nesdev community.

        APU:                Audio Processing Unit, what most people refer to the PSG in
the 2A03.

        Chrrom/chrram:      Character rom/ram. This is holds the tile set rom/ram which
is located on cart. All carts
                            must have at least 8k of this (for both set 0 & 1) or writing
to mirror through out the 8k.

        CPU:                The 2A03 which holds the 6502, most people do not refer the
PSG when they mention this.

        DMA:                Direct Memory Access.

        HBlank:             Horizontal Blank, a time during rendering where the the
scanline being draw is finished
                            and video ram is not being used.

        Mappers/Memory Maps: A usually small chip that redirect wires based on infomation
fed to it. This can be used
                            to switch prgrom wram, chrrom and other things located on
cart.

        NSF:                NES Sound Format. Holds various music and sound effects.

        PPU:                Picture Processing Unit, the 2C02.
        prgrom:             Program rom. Which is located at $8000-FFFF, and could be
switched in & out
```

```
        PSG:                Programmable Sound Generator. Its built into the A203 chip
along with the modified 6502.

        Sprram              Sprite Ram, located inside of the 2C02.

        W/S Ram:            Work / Save ram. Extra ram located at $6000-7FFF for general
use. If it is batter packed it
                            is refered as save ram. May be switchable depending on the
cart.

        VBlank:             Vertical blank. This happens after the video is finished
being drawn and video
                            memory is free to access.

        vram:               Video ram, holds memory it that is used to draw on screen.

+-------+
|  CPU  |
+-------+

2a Memory Map
-------------

                +--------------+---------------+--------------+----------------------+
                |   Address    |   End         |    Size      | Description          |
                +--------------+---------------+--------------+----------------------+
                |    $0000     |   $1FFF       |    $0800     | Ram                  |
                |    $2000     |   $3FFF       |    $0008     | PPU Registers        |
                |    $4000     |   $4017       |    $0018     | APU & Input Registers |
                |    $4018     |   $5FFF       |    $1FE8     | Written to cart *1   |
                |    $6000     |   $7FFF       |    $2000     | s/w ram on cart *2   |
                |    $8000     |   $FFFF       |    $8000     | Program Rom *2       |
                +--------------+---------------+--------------+----------------------+
                        *1: Carts may use this offset range & written values for a number
of things
                        *2: May be switch able depending on cart Along with *1

2b General Information
----------------------
        The NTSC NES uses a chip called 2A03 which holds a modified NMOS 6502 CPU with a
PSG built into the die.
        The PAL NES holds a 2A07 which has some minor changes.

        The modified 6502 lack the standard decimal mode. For information about 6502
instructions,
        addressing and opcodes or cycle by cycle steps, Look for 6502.txt, 6502_cpu.txt
at nesdev
        (http://nesdev.parodius.com/ or http://nesdev.com/) and drop by
http://www.6502.org/

        For technical information about the 2A03 look for Brad Taylor's excellent
document called "2A03
        technical reference" which covers absolutely everything about the chip. The speed
of the CPU on NTSC
        is 1.7897725 MHz and 1.6626070 MHz for PAL. The CPU run exactly 341/3(NTSC) and
341/3.2 (PAL) cycles
        per scanline. Below is some general information to help you understand the 6502
better.

2c Interrupts
-------------

        The 6502 has 3 in Interrupts each of them jump to an address based on their
vectors.
        The address is 2 bytes long with the low byte first.

        $FFFA: NMI
        $FFFC: RESET
        $FFFE: IRQ/BRK
```

```
        Non-Maskable Interruption (NMI):
              Pushes Flags, set I flag then jump, 7 cycles long.
              It is triggered when reg $2000.7 is set when the PPU hits vblank.
              If you clear $2000.7 then set it while $2002.7 is still set, (look at PPU
regs) you can trigger
              another NMI. NMI is used to jump to code that updates the screen or time
somthing.

        Reset:
              Jumps to Address. This happens when you hit the reset button on the NES
hardware, or when you
              turn on your NES. This can also happen if you run a bad opcode, not all-bad
opcodes will cause a
              reset but they may cause one. You should wait 2 VBlanks so everything can
get in sync.

        Interrupt ReQuest (IRQ):
              if the I flag is cleared, IRQ pushes the Flags, set the I flag and spends 7
cycles doing it.
              If the I flag is already set nothing will happen. Based on what is causing
the IRQ it may be
              fired regularly until it is acknowledged (usually in the IRQ code).

              It is usually timed for mid frame tricks or to regulate something such as
sound.

        Break (BRK):
              Pushes Flags with the B flag set. Then sets the I flag, it is 7 cycles
long.
              This is caused when the 6502 runs the instruction BRK which is 2 bytes
long.
              The 2nd byte of BRK is not used in any way, usually set a specific value
for debugging
              purposes. This Instruction shares vectors with IRQ and is mostly used for
debugging.

              NOTE: The B flag does not exist in the 6502 Status flags. The bit is only
pushed, when read in
              It is cleared.

2d Registers
------------

DMA (Direct Memory Access, $4014):
        The DMA is activated when a value is written to $4014. It reads a value at AABB
then
        writes it to $2004. AA is the value written to $4014 and BB is the current write
number from 0 to FF.
        Reading and writing a byte takes one CPU cycle each. It does this 256 times thus
being 512
        CPU cycles long. Most if not all games write $00 to $2003 before writing to the
DMA. The DMA purpose
        is to update the PPU's sprram quickly during the precious vblank time.


Input Ports ($4016 / $4017):
        $4016 (port 1)
        $4017 (port 2)

        The NES has 2 ports to plug in your peripheral device (pad, light zapper, etc).
        You can read the both ports through the $4016 and $4017. Both ports has the
highest 2 bits opened,
        bit 5 is 0 and rest (bits 4-0) are read in from the port. Writing to the $4016
sends the
        written value to both ports. Meanwhile writes to the $4017 will go to the APU.
        Read the input section below to learn more about available device.

2e Notes
```

```
--------
        - The chip does *not* support decimal mode.
        - May or may not reset on bad opcodes. Unexpected behaviours may happen
        - Read Modify Write instructions (inc, asl etc) will read, write back the same
value then
          modify and write the right value to the memory address. Beware of the dummy
write.
        - Any kind of Index will have a dummy read before fixing high address byte
        - BRK is 2 bytes
        - Wraps around in the zero page (zp index X, Y, Indirect X, and Indirect Y)
        - Wraps around on indirect instructions
                (JMP)  $89FF                      LDA ($FF), Y (Y=3)
                $8900: $A2                        $0000: $9A
                $89FF: $3C                        $00FF: $24
                $8A00: $78                        $0100: $72
                PC = $A23C                        A = $9A24 + Y ($9A27)


2f Summary of CPU And APU Registers
-----------------------------------
        +-------+-----------------+------------------------------------------------------+
        | $4000 | Rectangle 1    W|                                                      |
        | $4001 | Rectangle 1    W|                                                      |
        | $4002 | Rectangle 1    W|                                                      |
        | $4003 | Rectangle 1    W|   Too be written. in the mean time check out         |
        +-------+-----------------+   Brad Taylor's Document                             |
        | $4004 | Rectangle 2    W|           "2A03 technical reference"                 |
        | $4005 | Rectangle 2    W|                                                      |
        | $4006 | Rectangle 2    W|   Also check out                                     |
        | $4007 | Rectangle 2    W|          "NES APU Sound Hardware Reference".         |
        +-------+-----------------+    By Blargg                                         |
        | $4008 | Triangle       W|                                                      |
        | $4009 | Triangle       W|                                                      |
        | $400A | Triangle       W|                                                      |
        | $400B | Triangle       W|                                                      |
        +-------+-----------------+                                                      |
        | $400C | Noise          W|                                                      |
        | $400D | Noise          W|                                                      |
        | $400E | Noise          W|                                                      |
        | $400F | Noise          W|                                                      |
        +-------+-----------------+                                                      |
        | $4010 | DMC            W|                                                      |
        | $4011 | DMC            W|                                                      |
        | $4012 | DMC            W|                                                      |
        | $4013 | DMC            W|                                                      |
        +-------+-----------------+------------------------------------------------------+
        | $4014 | Sprite DMA     W|    Refer Above                                       |
        +-------+-----------------+------------------------------------------------------+
        | $4015 | Sound Status  RW|    Refer to registers $4000 through $4013            |
        +-------+-----------------+------------------------------------------------------+
        | $4016 | Input 1       RW|    Read Section 4: External Device for more details |
        | $4017 | Input 2       RW|                                                      |
        +-------+-----------------+------------------------------------------------------+

+-------+
|  PPU  |
+-------+

3a Memory Map
-------------

        +---------------+---------------+---------------+-----------------------+
        |    Address    |   End         |    Size       | Description           |
        +---------------+---------------+---------------+-----------------------+
        |    $0000      |    $0FFF      |    $1000      | Tile Set #0           |
        |    $1000      |    $1FFF      |    $1000      | Tile Set #1           |
        +---------------+---------------+---------------+-----------------------+
        |    $2000      |    $23FF      |    $0400      | Name Table #0         |
        |    $2400      |    $27FF      |    $0400      | Name Table #1         |
```

$2800	$2BFF	$0400	Name Table #2
$2C00	$2FFF	$0400	Name Table #3
$3000	$3EFF	$3EFF	Name Table Mirror *1
$3F00	$3FFF	$0020	Palette *2
$4000	$FFFF	$C000	Mirrors of Above *3

Notes:
*1: use (Address & 1FFF) + $2000 for Mirrored Address
*2: Mirrors 3F00 every $20 bytes, check the palette section for more details
*3: high 2 bits are ignored, simply and the address with $3FFF

3b Tile set

Tile sets hold 8x8 pixel tiles used to draw pictures on screen.
It is located inside of the cart and may be switched at any given time depending on the cartridge.
The tiles hold 2 bits per pixel which give you 16 byte per tile.
Each tile set holds 256 tiles which will result in 4k ($1000 or 4096) per set. There are two tile
sets, normally one set is for the sprites while the other is for the background (nametable).
Sprite can be 8x16 pixels in which case the NES will allow sprites to access both tile sets.

```
Cleared  bits:                          '.'
Set bits (low  at First  8 bytes):      '*'
Set bits (high at Second 8 bytes):      '-'
Combine bits (value 3):                 '#'

byte 0: *.....*.        *.....*.        clear bits '.' are considered transparent
byte 1: **....*.        #*....#.        so when this happens, you draw the pixel
byte 2: *.*...*.        #-*...#.        behind It.
byte 3: *..*..*.        #.-*..#.        High priority sprites -> Nametable ->
byte 4: *...*.*.        #..-*.#.        Low priority sprites -> Colour #0
byte 5: *....**.        #...-*#.        The left side is drawn over the right side. But special cases
byte 6: *.....*.        #....-#.        will disable pixels or change priority.
byte 7: ........        -.....-.

byte 8: ........
byte 9: -.....-.
byte A: --....-.
byte B: -.-...-.
byte C: -..-..-.
byte D: -...-.-.
byte E: -....--.
byte F: -.....-.
```

The tiles above will have a second pair of bits to select the type of palette.
If the tile is for a name table the bits will be found in the 'attribute' part of the tables.
If the tile is used for a sprite the high 2 bits will be found inside the sprite ram.

3c Name Tables

The name tables are used to select tiles for the background. Each table can hold 32x30 tiles
which make a resolution of 256x240. Each byte selects a tile while the end of the table holds the
colour attributes. The high 2 bits for the tiles selected will select the palette we are using
(at $3F00-3F0F) they can be found at the end of the name table at $3C0 (32*30 = 960)
Each byte in the attribute section affects 16 tiles.

```
        00 01 | 02 03 .... 1C 1D | 1E 1F        the 2x2 tiles at each corner get their
        20 21 | 22 23 .... 3C 3D | 3E 3F        2 high bits from the appropriate bit.
        ------+------ .... ------+------              %AABB CCDD        DD | CC
        40 41 | 42 43 .... 5C 5D | 5E 5F                                ---+---
        60 61 | 62 63 .... 7C 7D | 7E 7F                                BB | AA


Affecting Registers:

        $2000 %xxxT xxNN
        $2001 %xxxx VxCx
        $2000.4:        (T)iles will be read from $Txxx
        $2000.1-0:       NN is the name-table number

        $2001.3:        (V)isibility. If cleared the PPU stops drawing data from the
name-tables.
        $2001.1:        (C)lipping. If cleared the left 8 pixels of the name-table will
be ignored.

        Note: on a NTSC screen the top 8 & bottom 8 pixels are not visible.


3d Name Tables
    Mirroring
--------------

        The NES has 2k of ram for nametables internally (which gives you 2 tables).
        Using A10 & A11 of the PPU you can control the mirroring or give it extra tables.

        A11 A10

        0   0       $2000, $2400, $2800, $2C00 are all pointed at the first k of ram.

        0   1       $2000, $2400 are pointing to the first k while $2800, $2C00 are
pointing
                    to the 2nd k. This causes horizontal mirroring

        1   0       $2000, $2800 are pointing to the first k while $2400, $2C00 are
pointing to the
                    2nd k. This causes vertical mirroring (the table above/below are the
same)

        1   1       $2000 is pointed to the first k, $2400 is pointed to the second k.
                    $2800 & $2C00 are pointing to 2k of ram (or ROM) inside of the cart.

        Check out brad taylors doc for technical infomation.

3e Palette
----------

        The NES has $20 bytes of palette data mapped to $3F00-3FFF
        You can easily get the true address by doing an AND with $1F
        Address 00-0F is for name-tables while 10-1F is for sprites.

        As mentioned before when the tile low 2 bit is 0 the pixel is transparent the PPU
only has room
        for 4 sets of 3 palettes for name-tables & sprites.
        When the colour is %xx00 the PPU either draws the pixel behind of the current
pixel of it or
        draws colour #0 when there is none. Colour #0 can be set by writing to $3Fx0.

        0123 4567 89AB CDEF        (u)nused, (a)cknowledged, (c)olour #0, (m)irror of
above
0       caaa uaaa uaaa uaaa        name-tables
1       Maaa Maaa Maaa Maaa        sprites

        These bits are converted to colour, the colours are analog and do not convert to
```

```
a consistent RGB
        palette. The PPU only has room for 6 bits in its palette section so you can only
output a range of
        64 colours plus emphasis for some output change.

Affecting Registers:
        $2001.5-7, .0        %BGRx xxxM

        Thanks to Chris Covell for this info.

        001        B: 074.3%        G: 091.5%        R: 123.9%
        010        B: 088.2%        G: 108.6%        R: 079.4%
        011        B: 065.3%        G: 098.0%        R: 101.9%
        100        B: 127.7%        G: 102.6%        R: 090.5%
        101        B: 097.9%        G: 090.8%        R: 102.3%
        110        B: 100.1%        G: 098.7%        R: 074.1%
        111        B: 075.0%        G: 075.0%        R: 075.0%

        When (M)onochrome is set bits 0-3 is ignored which only leaves you with the
highest 2 bits (5 & 6)
        This leaves you with some white & light grey colours.

While Reading:
        Reading the palette via $2007 bypasses the VRAM read buffer and immediately
returns the correct data.
        However, the VRAM read buffer will still be updated but, with the data mirrored
from $2F00-$2FFF
        rather than internal palette data. Palette RAM is only 6 bits wide; the upper 2
bits will always read back as 0.

        Also while the palette being read with M set, you will read from 3Fx0 and the low
4 bits will be
        cleared. Monochrome will not affect Writing in any way.

3f Sprite Ram
-------------
        The PPU has exactly 256 bytes for sprite ram. Each sprite needs 4 bytes of data
so that leaves you
        with a maximum of 64 sprites. Sprite ram can access by $2003, $2004 & $4014

        Byte:
        0: %YYYY YYYY Axis Y from top of the screen.
        1: %TTTT TTTT Tile Index
        2: %vhp- --cc
                v:        Vertical flip
                h:        Horizontal flip
                p:        sprite priority, 0=High priority, in front of nametables 1= Low
pri, behind hametables
                ---:      unknown/not used
                cc:       high 2 bits for tiles
        3: %XXXX XXXX Axis X from left side of the screen

Other Affecting Registers:
        $2000 %--S- T---
                Sprite (S)ize 8x16
                (T)ile table 1xxx
        $2001 %---V -C--
                Sprites are (V)isble
                (C)lip sprite pixels if fall in left 8 pixel of the screen
        $2002 %-HO- ----
                Sprite #0 pixel (H)it
                Sprite (O)verflow (9+ found on scanline)

        If $2000.5 (S) is set (8x16 sprites) the lowest bit decides the Tile Table
instead of 2000.3 (T).
        Since the low bit controls the table you can only use even numbered tiles which
is good since all odds
        would be the low 8x8 of the 8x16 tiles.
```

```
        When sprite #0 draws a pixel and the BG also drew a pixel there, sprite collision
(H)it is set ($2002.6).
        Only 8 sprite can be drawn on screen, if a 9th sprite is found on the same
scanline bit
        $2002.5 (O) is set. The first 8 sprites found will be drawn. If 2 sprites are
drawn at the same
        position the sprite lower (closer to sprite #0) will be drawn on top.

        Note on collision:
              If the pixel hits at $FF (Right most of the screen) it will not be
considered as a successful hit
              and will NOT set the collision bit ($2002.6)

        Note on priority:
              If two or more sprites occupy the same pixel position, the sprite with
lowest number is drawn.
              If a high and low priority sprite are drawn at the same position, high
priority sprite pixel will
              be considered low priority.

        Note on sprram:
              After you write data to it, you must regulary refresh it by write data over
it. If you take too long
              to refresh the data the bytes which you haven't written on will change to
an unknown value.

Registers to Access sprite-ram:
        $2003:  Write Only: Sets the offset in sprite ram.
        $2004:  Reads or writes the value in sprite ram using $2003 as an offset.
                Increase offset value by 1 on writes, does not increase on reads.

        $4014:  Sprite DMA which is a CPU register. It is write only.
                It will read XX00 + I (where XX is the value and I is 0-FF) and feed the
value to $2004.
                It takes 512 cycles and is a lot better then looping writes by code.

3g Scrolling
------------
        Thanks to Loopy for all of this info.

        Temp:           register/latch to hold scrolling info which is copied to address
on line 21.
        Address:        The 15 bits that $2007 use to sent writes to the PPU
        Fine X:         3 bits uses to control the name-tables fine x during rendering.
        Toggle:         a bit that is flipped between write 1 & 2 for the $2005 & 6.

        NN              Name-table Bits. High is the Vertical bit, low is the Horizontal
bit
        YYYYY           Tile Y
        XXXXX           Tile X
        yyy             Fine Y


        PPU Address during rendering        %-yyy NNYY   YYYX XXXX


        $2000 W %---- --NN
        temp    %---- NN--   ---- ----

        $2002 R toggle = 0

$2005   W %XXXX Xxxx (toggle is cleared)

        temp        %0--- ----   ---X XXXX
        Fine X      %xxx
        toggle = 1

$2005   W %YYYY Yyyy (toggle is set)
        temp        %0yyy --YY   YYY- ----
```

```
        toggle = 0

Total bits affected with $2005:
        temp        %0yyy --YY   YYYX XXXX



$2006   W %--yy NNYY (toggle is cleared)
        temp        %00yy NNYY   ---- ----

$2006   W %YYYX XXXX (toggle is set)
        temp        %0--- ----   YYYX XXXX
        address = temp

Total bits affected with $2006:
        temp        %00yy NNYY   YYYX XXXX
        address = temp

        As you can see above, $2006 can change every bit in the temp, but unfortunately
it
        will clear the high fine Y bit. You can set the Fine X,Y & Tile X, Y with $2005
but you
        need a write to $2000 (or write $2006 then $2005) to change the name-table bits

3h Rendering
------------
        Check out brad Taylor's doc for full information about rendering.
        Here is my simpler explanation of it.

        Fine X controls what pixel to start rendering from (0-7). After drawing one bit
the NES increase fine
        X, when it wraps from 7 to 0 Tile X is then increase to get to the next tile.
When tile X wraps from
        31 to 0, the Tile Y is not increase, but the low bit of the 2 nametable bits is
flipped. This effect
        allows the PPU to skip the Tiles in the next row and jump to the next name-table.

        Fine Y is increase at the end of every line. After Fine Y wraps from 7 to 0 the
Tile Y is increased.
        Unlike tile X, tile Y does not wrap from 31 to 0, it wraps from 29 to 0 so it
will not read from its
        attribute bits. (remember screen is 240 pixels high. 30 tile * 8 pixel per tile =
240)

        When Tile Y wrap from 29 to 0 it flips the high name-table bit. If Tile Y is set
to 30 or 31 from
        $2006, it will wrap from 31 to 0 but the high name-table bit will not flip

        Line 0: Junk/Refresh
            This line clears bits $2002.5-7. Also throws away all the sprite data.
            The PPU Address copies the PPU's temp at the beginning of the line if sprite
or name-tables
            are visible.

            It is unknown if this line affects the PPU like 1-241, according to my
emulator, decreasing the MMC 3
            scanline counter causes problems. Maybe the point of it is to destory all
sprite objects.

        Line 1-241:
            Renders the screen for 240 lines, at the end of each line it increases Fine Y
and gets the low NT
            bit, fine X & Tile X position from PPU's temp.

        Line 242: Dead/Junk
            Does nothing but run.

        Line 243 - 262(NTSC) - 312(PAL): VBlank
            At the beginning of line 0 bit $2002.7 is set. If $2000.7 is also set a
```

```
hardware NMI occurs.
            you can access the PPU any time without corrupting data.

        Note: If $2001.3 & 4 is cleared (sprite & name-tables not visible) you can access the PPU without
              corrupting data, as if it were in VBlank.

              A few games changes the palette and jump to the next nametable. This is done by turning off the ppu
              usually at the end of the line (HBlank) writing to the palette, then setting the address to the
              location it want to start from. Turning off the ppu will destory the sprites for the next scanline.

              You have exactly (341-256(pixels)) / 3(NTSC) or 3.2(PAL) cpu cycle for HBlank.


3i Data Transfer
---------------

        Register $2007 is the only way for the NES to read & write data in the PPU's Memory map. The PPU's
Address controls where the value is written too. To set the address you must write to $2006 while the toggle
is set (2nd $2005/6 write) which copies the temp to the address. When reading or writing to $2007 the
address is increased by 1 or 32 based on $2000.2.

        When reading Address $0000 - $2FFF from $2007 the value return is the *last* value read from the PPU.
So if you were to set the Address to $2000 and start reading the nametable bytes, the first byte would be
whatever you last *read* and the next read will be the byte at $2000. however, if the Read if $3F00 - $3FFF
(palette), you will get the palette byte right away, the value at 2xxx will replace the pipe value which
you will receive the next time you read from 0-3EFF.

3j Summary of PPU Registers
---------------------------

        +-------+---------------+-------------------------------------------------------+
        | $2000 | PPU Control 1 |        %7654 3210                                     |
        |       |               |                                                       |
        |       |               | 7 Execute NMI on VBlank   Disabled / Enabled          |
        |       |      Write    | 6 unknown/unused                                      |
        |       |               | 5 Sprite Size                8x8   /  8x16            |
        |       |               | 4 Background Tile Table    $0000   / $1000            |
        |       |               | 3 Sprite Tile Table        $0000   / $1000            |
        |       |               | 2 PPU Address Increment      1     / 32               |
        |       |               | 1-0 Name-table bit        Read Scrolling Section      |
        +-------+---------------+-------------------------------------------------------+
        | $2001 | PPU Control 2 |        %7654 3210                                     |
        |       |               |                                                       |
        |       |               | 7-5 Colour Emphasis        Read Palette Section       |
        |       |        Write  | 4 Sprite Visibility         False  /  True            |
        |       |               | 3 Background Visibility     False  /  True            |
        |       |               | 2 Sprite Clipping           True   /  False           |
        |       |               | 1 Background Clipping       True   /  False           |
        |       |               | 0 Monochrome Colours        False  /  True            |
        +-------+---------------+-------------------------------------------------------+
        | $2002 | PPU Status    |        %VHO- ----                                     |
        |       |               |                                                       |
        |       |      Read     | 7 VBlank                                              |
        |       |               | 6 Sprite #0 Collision Hit    False  / True            |
        |       |               | 5 Sprite Overflow            False  / True            |
        |       |               |                                                       |
        |       |               | Note: All bits are cleared on line 0.                 |
```

```
        |       |               |     Bit 7 is cleared on read.                         |
        |       |               |     The PPU Toggle Bit is also cleared                |
        +-------+---------------+--------------------------------------------------------+
        | $2003 | Sprite Index  | Write only, Used with $2004.                           |
        +-------+---------------+--------------------------------------------------------+
        | $2004 | Sprite Data   | Uses the Index in $2003 to read or write from Sprit   |
        |       | Read & Write  | Ram. Increase Sprite Index by 1 on reads or writes.   |
        +-------+---------------+--------------------------------------------------------+
        | $2005 | Write Only    | Read Scrolling for more information.                   |
        | $2006 |               |                                                        |
        +-------+---------------+--------------------------------------------------------+
        | $2007 | Read & Write  | Read PPU Transfer                                      |
        +-------+---------------+--------------------------------------------------------+

+-----------------+
| External Device |
+-----------------+

4a General Information
----------------------
        The NES has many types of input, Ranging from the power pad to the light zapper.
        Here are all the known inputs and all the info I can get from them.
        $4016 & $4017

        Reads
        %oo0i ieei
                oo  open bus, will be the last bit read in (high byte of PC)
                iii input device, if no input is available it gets the expansion port
bits
                ee  expansion port

        writes
        %ss-- -eea on write
                ss if writing to $4017 you can set 2 sound bits mention in sound docs.
                ee expansion port
                (a)ll ports (expansion, input 1 and 2) get this bit.
                - unused (0's)

Note:       NTSC and PAL NES do not have any games or peripherals that uses the
                expansion port (or its bits).

4b Joypad
----------
        The NES standard pad. While plugged in $4016 & $4017 you should reset the shift
register in both pads
        before reading. To do this you write 1 to the lowest bit in $4016, which will set
the latch inside of
        the pad connect to any port. When the latch is set the shift register will be
brought down to the first
        avaible bit and this will constally done until you clear the latch. (write 0 to
low bit in $4016).

        After reading $4016 or $4017 the shift register will move to the next button. The
lowest bit will hold the
        button state. If the bit is set the button is currently being held. The order of
the buttons are

        A B Select Start Up Down Left Right

        A would be the first bit you read and Right would be the 8th. After these 8 reads
the rest of the bit will
        be set (on normal pads). To get player 2 input you would use the lowest bit from
$4017.

4c 4 Player Adapter
---------------
        This is used the same way as the joy pad except after the 8 reads from $4016 you
will get the 8 bits
        from player 3 which is in the same order. Player 4 input will be the next 8 from
```

$4017. To reset the
strobing to all 4 player you do the same writes to $4016.

4d Light Zapper

The light zapper uses 2 of the 6 bits. %TW--0
T is the Trigger, is set if it's pressed else it is cleared.
W this is White detection. All games set the target area to white and the rest to black to avoid any
colour confusion due to colours & light in the room.

4e Paddles

To be written

4f Power Pad

To be written

```
+-------------+
| NES Formats |
+-------------+
```

5a iNes Header:

Every emulator supports the iNes format and many games images use it.
There are docs that include extended infomation in the reserve section but i have not
heard or any image that uses one. This format listed is supported in every emulator i know of

```
+--------+------+------------------------------------------+
|   0    |  3   | 'NES'                                    |
|   3    |  1   | $1A                                      |
|   4    |  1   | 16K PRG-ROM page count                   |
|   5    |  1   | 8K  CHR-ROM page count                   |
|   6    |  1   | ROM Control Byte #1                      |
|        |      |   %####vTsM                              |
|        |      |     |  ||||+- 0 = Horizontal Mirroring   |
|        |      |     |  ||||   1 = Vertical Mirroring     |
|        |      |     |  |||+-- 1 = SRAM enabled           |
|        |      |     |  ||+--- 1 = Trainer *1             |
|        |      |     |  |+---- 1 = Four-screen VRAM layout |
|        |      |     |  |                                 |
|        |      |     +--+----- Mapper # (lower 4-bits)    |
|        |      |                                          |
|   7    |  1   | ROM Control Byte #2                      |
|        |      |   %####0000                              |
|        |      |     |  |                                 |
|        |      |     +--+----- Mapper # (upper 4-bits)    |
+--------+------+------------------------------------------+
|   8    |  8   | Reserved, All filled with 0's.           |
+--------+------+------------------------------------------+
```

*1 Trainers are 512 bytes and hold absolutly no useful infomation. It was supposly somthing to imitate
bankswitching in mappers but it badly executed and didnt work out, i can not find a peice of infomation
about trainers and i have no idea how they would replace it or who thought of it.

5b UNIF Header:
Most new emulator supports this format.
You can get the infomation about unif at its homepage http://www.parodius.com/~veilleux/index.html
or http://www.parodius.com/~veilleux/UNIF_current.txt

5c Boards:

```
        http://www.parodius.com/~veilleux/boardnames
        You can also find out what game uses which board (north american games)
        http://www.parodius.com/~veilleux/boardtable.txt
        RR holds the rarity value where A+ is the hardest to find.
```