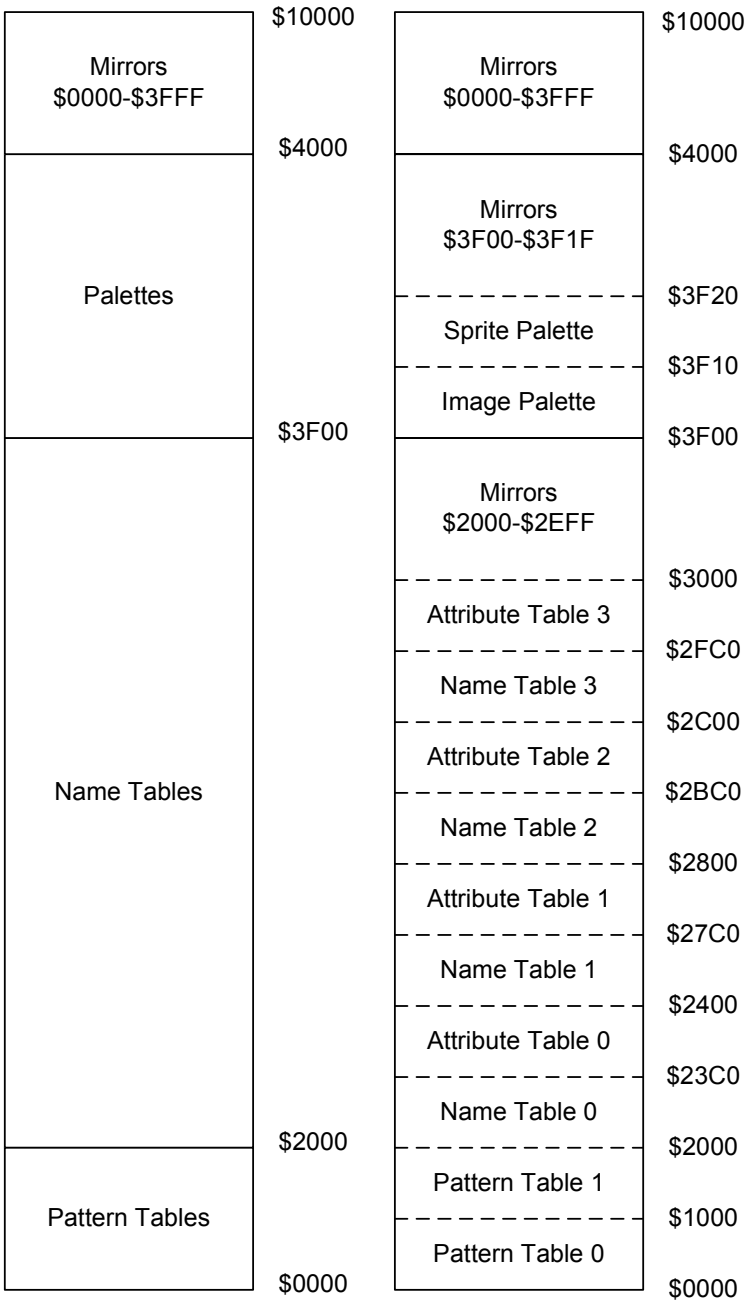

Figure 3-1. PPU memory map.

3.3 PPU Registers

Communication between the CPU and other devices takes place via memory mapped I/O registers. The registers used by the PPU are located in main memory at $2000-$2007 with an additional register used for Direct Memory Access at $4014. Remember that locations $2000-$2007 are mirrored every 8 bytes in the region $2008-$3FFF. A summary of all I/O registers can be found in Appendix B.

The actions of the PPU can be controlled by the CPU by writing to $2000 and $2001, known as PPU Control Register 1 and PPU Control Register 2 respectively. Both registers should

only be written to. Bit 7 of £2000 can be used to disable NMIs. Remember that this type of interrupt is generated whenever a V-Blank occurs and is unaffected by the interrupt disable flag of the status register. Clearing this bit will prevent an NMI from occurring on V-Blank. Since the NES supports both 8x8 and 8x16 sprites, setting bit 5 of $2000 will switch to 8x16 sprites. The next address in PPU memory to read or write from will be incremented after each I/O occurs. The value to increment by is adjusted by setting the value of bit 2 of $2000. If this is clear, the address is incremented by 1 (horizontal), otherwise the increment is 32 (vertical). Using $2001, the background can be hidden by clearing bit 3 and, similarly, the sprites can be hidden by clearing bit 4.

The PPU Status Register is located at $2002 and is read only. The register is used by the PPU to report its status to the CPU. The programs will frequently cause the CPU to read from this location in order to ascertain the PPU's status. Bit 7 is set by the PPU to indicate that V-Blank is occurring. Bit 6 and bit 7 relate to sprites and are described later. Bit 4 indicates whether the PPU is willing to accept writes to VRAM, if it clear then writes are ignored. When a read from $2002 occurs, bit 7 is reset to 0 as are $2005 and $2006.

3.3.1 Direct Memory Access

When transferring a large amount of data between devices it is inefficient to transfer this through the processor. To transfer data from CPU memory to sprite memory, for example, takes the following steps:

1. Load required SPR-RAM address into CPU.
2. Write required SPR-RAM address to $2003.
3. Load byte into CPU.
4. Write byte to $2004.

When filling the contents of sprite memory, this technique would have to be repeated 256 times. Direct Memory Access (DMA) is a technique which allows more efficient copying of data from CPU memory to sprite memory. Using DMA, the whole of sprite memory can be filled by using a single instruction, a write to $4014. The starting address in CPU memory is specified by the operand for the write multiplied by $100. The 256 bytes starting at this address are copied directly into sprite memory without the further involvement of the CPU.

When the DMA is occurring, the memory bus is in use, preventing the CPU from accessing memory and, therefore, preventing it from accessing any more instructions. This is referred to as cycle stealing and the CPU has to wait until the DMA transfer is complete. On the NES, the DMA takes the equivalent of 512 cycles (about 4.5 scanlines worth) after which the CPU can resume. This is considerably less than would be required to copy manually through the CPU.

3.4 Colour Palette

The NES has a colour palette containing 52 colours although there is actually room for 64. However, not all of these can be displayed at a given time. The NES uses two palettes, each with 16 entries, the image palette ($3F00-$3F0F) and the sprite palette ($3F10-$3F1F). The image palette shows the colours currently available for background tiles. The sprite palette shows the colours currently available for sprites. These palettes do not store the actual colour values but rather the index of the colour in the system palette. Since only 64 unique values are needed, bits 6 and 7 can be ignored.

The palette entry at $3F00 is the background colour and is used for transparency. Mirroring is used so that every four bytes in the palettes is a copy of $3F00. Therefore $3F04, $3F08, $3F0C, $3F10, $3F14, $3F18 and $3F1C are just copies of $3F00 and the total number of

colours in each palette is 13, not 16 [5]. The total number of colours onscreen at any time is therefore 25 out of 52. Both palettes are also mirrored to $3F20-$3FFF. The colour palette is shown in Appendix F.

3.5 Pattern Tables

The NES has two pattern tables at $0000 and $1000. The pattern tables store the 8x8 pixel tiles which can be drawn on the screen. Many games store the pattern tables in CHR-ROM on the cartridge, however, games without CHR-ROM will use RAM for the pattern tables and fill them during execution. The pattern tables store the least significant two bits of the 4-bit number needed to identify the image or sprite palette entry used by that pixel such that 00b is palette entry 0, 01b is 1, 10b is 2 and 11b is 3.

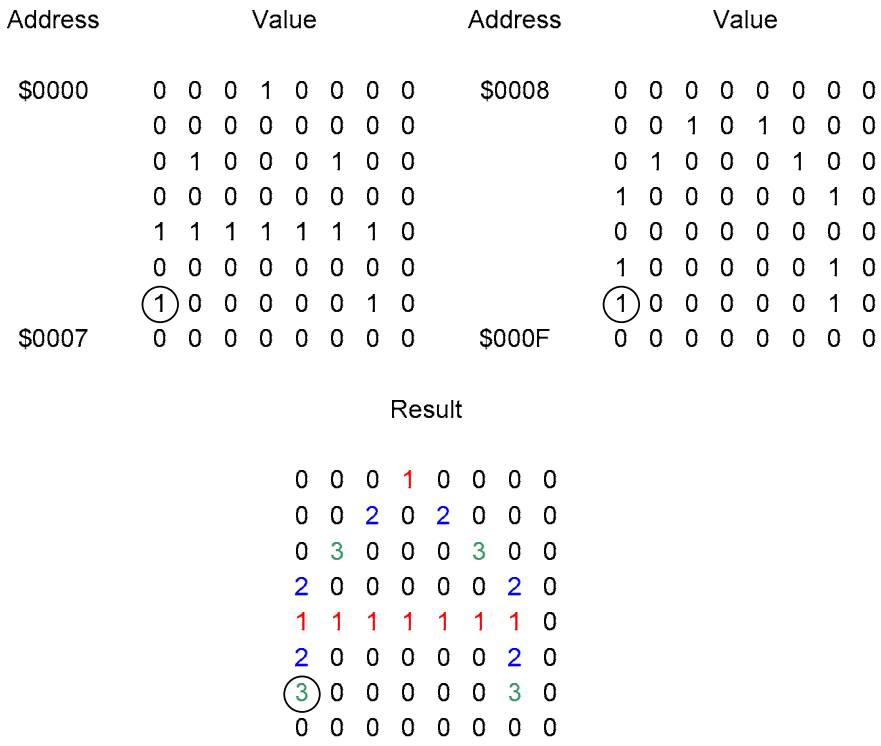

Figure 3-2. Pattern tables. Adapted from [7].

Figure 3-2 shows how the pattern tables work. The character ‘A’ is the final result, shown at the bottom. The character is constructed pixel by pixel by taking one bit from the top left and one from the top right to make a 2-bit colour. The other two bits of the colour are taken from the attribute tables. The colours shown are not genuine NES colour palette values.

3.6 Name Tables / Attribute Tables

Name tables are essentially a matrix of tile numbers, pointing to the tiles stored in the pattern tables. The name tables are 32x30 tiles and since each tile is 8x8 pixels, the entire name table is 256x240 pixels.

Each name table has an associated attribute table. Attribute tables hold the upper two bits of the colours for the tiles. Each byte in the attribute table represents a 4x4 group of tiles, so an attribute table is an 8x8 table of these groups. Each 4x4 group is further divided into four 2x2 squares as shown in figure 3-3 [9]. The 8x8 tiles are numbered $0-$F. The layout of the byte

is 33221100 where every two bits specifies the most significant two colour bits for the specified square.

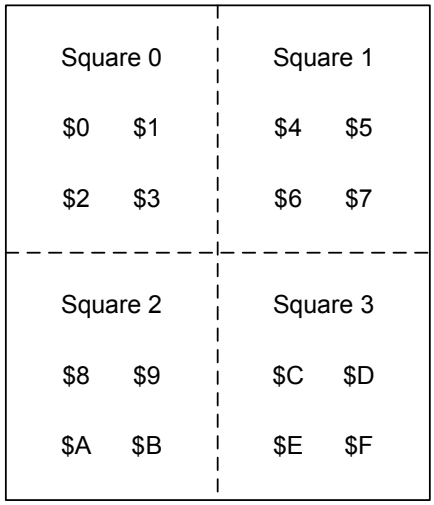

Figure 3-3. 4x4 tile group layout. Adapted from [20].

The NES only has 2 KB to store name tables and attribute tables, allowing it to store two of each. However it can address up to four of each. Mirroring is used to allow it to do this. There are four types of mirroring which are described below, using abbreviations for logical name tables (those that can be addressed), L1 at $2000, L2 at $2400, L3 at $2800 and L4 at $2C00:

- Horizontal mirroring maps L1 and L2 to the first physical name table and L3 and L4 to the second as shown in figure 3-4.

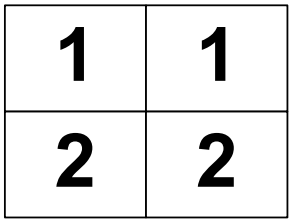

Figure 3-4. Horizontal mirroring.

- Vertical mirroring maps L1 and L3 to the first physical name table and L2 and L4 to the second as shown in figure 3-5.

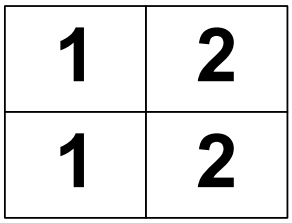

Figure 3-5. Vertical mirroring.

- Single-screen mirroring points all four logical name tables to the same physical name table as shown in figure 3-6.

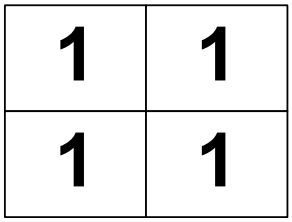

Figure 3-6. Single-screen mirroring.

- Four-screen mirroring uses an additional 2 KB of RAM in the cartridge itself to allow logical name tables to each map to separate physical name tables as shown in figure 3-7.

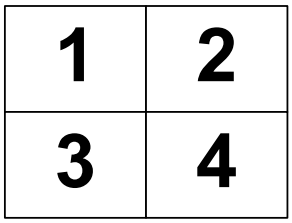

Figure 3-7. Four-screen mirroring.

3.7 Sprites

Sprites are the characters to draw on the screen. Sprites can be either 8x8 pixels or 8x16 pixels. Most characters are composed of multiple sprites. The sprite data is stored in the pattern tables while the sprite attributes are stored in SPR-RAM. There are a maximum of 64 sprites, each of which uses four bytes in SPR-RAM. The bytes work as follows:

- Byte 0 - Stores the y-coordinate of the top left of the sprite minus 1.
- Byte 1 - Index number of the sprite in the pattern tables.
- Byte 2 - Stores the attributes of the sprite.
  - Bits 0-1 - Most significant two bits of the colour.
  - Bit 5 - Indicates whether this sprite has priority over the background.
  - Bit 6 - Indicates whether to flip the sprite horizontally.
  - Bit 7 - Indicates whether to flip the sprite vertically.

8x16 sprites use different pattern tables based on their index number. If the index number is even the sprite data is in the first pattern table at $0000, otherwise it is in the second pattern table at $1000.

Sprites can be read or written one at a time by first writing the required address to $2003 and then reading or writing $2004. Alternatively the whole of SPR-RAM can be written in one DMA operation by writing to $4014.

Sprites are given priority based on their position in SPR-RAM. The first sprite is known as sprite 0 and has higher priority. On each line the system calculates which sprites are on that line and draws them, lowest priority first, to ensure high priority sprites are drawn on top. Only eight sprites are allowed per scanline, and the system indicates when this number has been reached by setting bit 5 of I/O register $2002.

A common technique used for scrolling involves determining whether sprite 0 is overlapping a non-transparent background pixel. If the system is drawing sprite 0, and any non-transparent pixel in it is in the same position as a non-transparent background pixel, the system sets the sprite 0 hit flag in bit 6 of $2002. Therefore if the background tile contains only transparent pixels the sprite 0 hit flag will not be set. Figure 19 shows sprite 0 detection. The left image shows the background, the centre image shows the sprite and the right image shows the composite of the two. Colour 0 represents transparency and the circled pixel indicates where the sprite 0 hit flag is set. Figure 3-8 is adapted from [20], however the original incorrectly indicated where the hit flag was set.

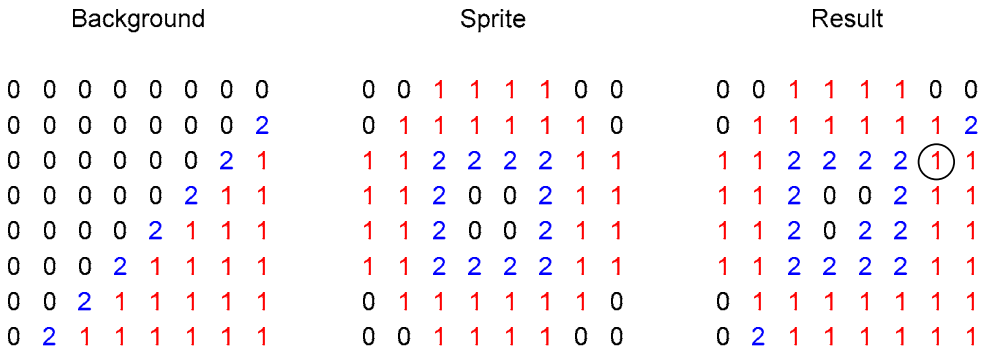

Figure 3-8. Sprite 0 detection. Adapted from [7].

Characters are generally larger than a single sprite and so are constructed by combining multiple sprites. For example figure 3-9 shows how the Mario character is constructed of eight separate 8x8 sprites.

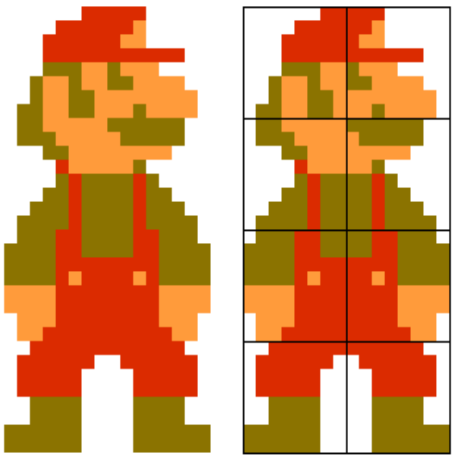

Figure 3-9. Character construction. Adapted from [33].

3.8 Scrolling

The background can be scrolled horizontally or vertically. Scrolling makes use of the separate name tables. At any given time background on the screen is either taken straight from one of the name tables or will be a combination of two name tables. This is shown in figures 3-10 and 3-11. Figure 3-10 shows the contents of two of the name tables (the other

two are, of course, mirrors) and figure 3-11 shows the composite image displayed on the screen, including sprites.

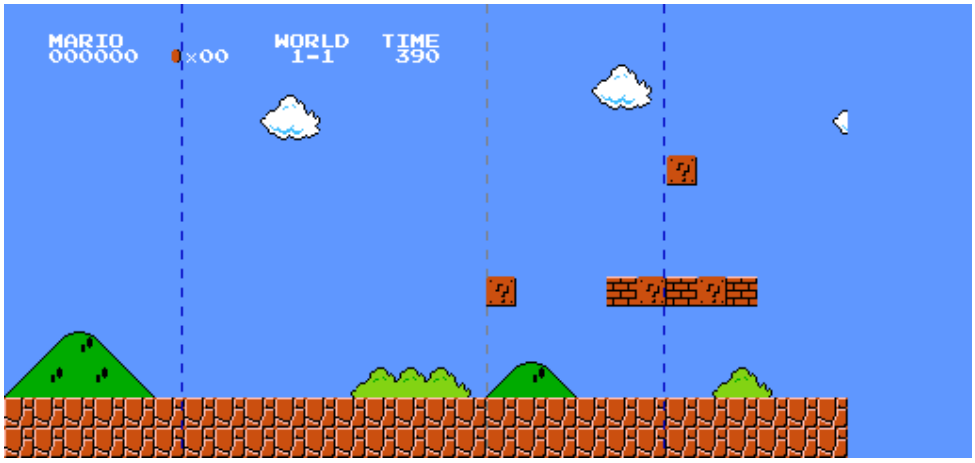

Figure 3-10. Horizontal scrolling in Super Mario Bros.

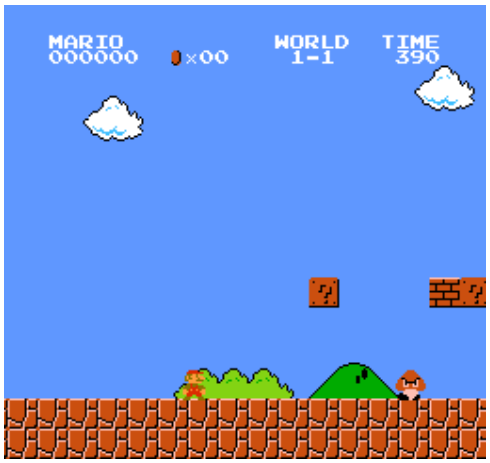

Figure 3-11. Composite image.

The final image starts on the first name table and stretches across to the second. The division between the two name tables is shown on figure 3-10 by the grey line. The two blue lines indicate the area which is shown on the screen. To the left of the on-screen portion is the section which has already been displayed, and which has now scrolled off the screen. To the right of the on-screen portion is where the system is currently filling the name table with what lies ahead and will be displayed on the screen as Mario continues to move along. As demonstrated by the cloud which is cut in half, not all of this area has yet been filled by the system. Some games only allow movement in one direction while others allow scrolling in both directions. This is described by Nintendo as follows [33]:

> "The PPU may display only 960 characters at a time, but it actually stores twice that amount. In a one way scroll, new characters constantly replace old characters behind the scroll. This is why in games like Super Mario Bros. the screen can scroll only one way. In Metroid, however, scrolling occurs in two directions and new characters are continually added in the direction of the scroll."

It is clear that the status bar area of the screen is not scrolled in the same way as the rest and is fully resident on the first name table. This is typical of status information and is handled in Super Mario Bros. by using the sprite 0 hit flag and in Super Mario Bros. 3 by generating an IRQ.

The general picture of horizontal and vertical scrolling is shown in figure 3-12. The name table shown here as A is specified by bits 0-1 of $2000 and B is the name table after (which depends on the mirroring technique). This does not apply to games which allow simultaneous horizontal and vertical scrolling [7]. The background image will span across the name tables as shown in figure 3-13.

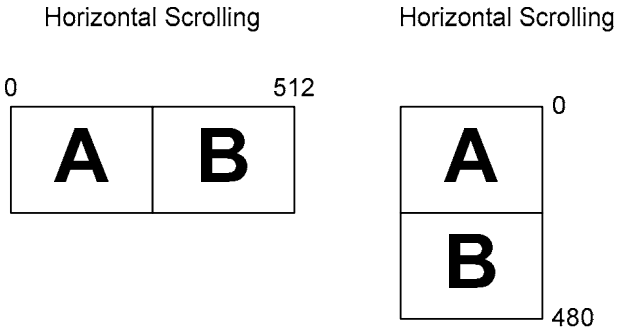

Figure 3-12. Horizontal and vertical scrolling. Adapted from [7].

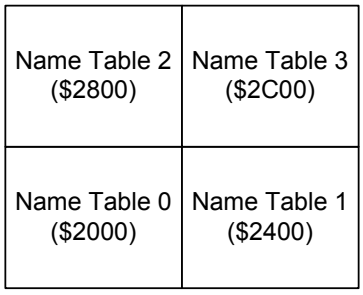

Figure 3-13. Name tables used for background. Adapted from [7].

The way scrolling works is described in [8] and is summarised here. The system maintains a 16-bit VRAM address register, the value of which is set by $2006. The layout of this register is as follows:

- Bits 0-11 - Stores the address in the name table as an offset from $2000. Bits 0-4 are the x-scroll and is incremented as the line is drawn. As this is incremented from 31, it wraps to 0 and bit 10 is switched. Bits 5-9 are the y-scroll and are incremented at the end of a line. When incremented from 29, it wraps to 0 and bit 11 is switched. If the value is set above 29 by a write to $2007, then it will wrap to 0 when it reaches 31, but bit 11 is not affected.
- Bits 12-14 are the tile y-offset.

Since the x-scroll and the y-scroll indicate tile numbers, this allows 32 tiles across the screen (256 pixels) and 30 tiles down the screen (240 pixels), for a total of 960 tiles.

There is a second, temporary VRAM address register which is also 16-bits long. Finally there is a 3-bit tile x-offset. These are updated by writes to registers and as the frame is drawn.

3.9 Television Standards

The NES connects to a television to display the game to user. As a result different versions of the system were created for the two television formats, NTSC and PAL. NTSC (National Television Standards Committee) is the standard used in North America, most of South America and parts of Asia [34]. PAL (Phase Alternating Line) is the standard used in Europe, much of Asia, Africa and Australasia [35]. Table 3-1 shows the differences between NTSC and PAL versions of the NES.

	NTSC	PAL
Frames per second	60	50
Time per frame (milliseconds)	16.67	20
Scanlines per frame (of which is V-Blank)	262 (20)	312 (70)
CPU cycles per scanline	113.33	106.56
Resolution	256 x 224	256 x 240
CPU speed	1.79 MHz	1.66 MHz

Table 3-1. Comparison of NTSC and PAL NES systems.

Images are displayed on a television screen by a stream of high speed electrons which moves across the screen, from left to right, drawing each pixel. A single line of pixels is referred to as a scanline. At the end of a scanline the electron beam must move to the next line and return to the left before it can proceed. The time it takes to do this is known as the Horizontal Blank period (H-Blank).

After drawing the screen once, the electron beam must return to the top left corner, ready to start the next frame. The time it takes to do this is known as the Vertical Blank period (V-Blank). When entering the V-Blank period, the PPU indicates this by setting bit 7 of I/O register $2002. This bit is reset when the CPU next reads from $2002.

On an NTSC version of the NES, there are 240 scanlines on the screen (although the top and bottom eight lines are cut off) and it takes an additional 3 scanlines worth of CPU cycles to enter V-Blank. The V-Blank period takes a further 20 scanlines worth before the next frame can be drawn.

The software that can be run using an emulator is usually referred to as a ROM image in reference to the original ROM chips used to store it. A simple dump of the contents of the cartridge is unlikely to be sufficient as it leaves no way to identify what each part of the file means. Two different file formats have emerged to provide this information.

The iNES file format was originally defined by Marat Fayzullin for use in his iNES emulator. The format has since been used by most emulators and is the most common format for ROM images. INES format files should have the file extension *.nes. The format provides a 16 byte header at the start of the file which contains important information. The format as described in [9] is as shown in table 4-1:

Starting Byte	Length (Bytes)	Contents
0	3	Should contain the string 'NES' to identify the file as an iNES file.
3	1	Should contain the value $1A, also used to identify file format.
4	1	Number of 16 KB PRG-ROM banks. The PRG-ROM (Program ROM) is the area of ROM used to store the program code.
5	1	Number of 8 KB CHR-ROM / VROM banks. The names CHR-ROM (Character ROM) and VROM are used synonymously to refer to the area of ROM used to store graphics information, the pattern tables.
6	1	ROM Control Byte 1: • Bit 0 - Indicates the type of mirroring used by the game where 0 indicates horizontal mirroring, 1 indicates vertical mirroring. • Bit 1 - Indicates the presence of battery-backed RAM at memory locations $6000-$7FFF. • Bit 2 - Indicates the presence of a 512-byte trainer at memory locations $7000-$71FF. • Bit 3 - If this bit is set it overrides bit 0 to indicate four-screen mirroring should be used. • Bits 4-7 - Four lower bits of the mapper number.
7	1	ROM Control Byte 2: • Bits 0-3 - Reserved for future usage and should all be 0. • Bits 4-7 - Four upper bits of the mapper number.
8	1	Number of 8 KB RAM banks. For compatibility with previous versions of the iNES format, assume 1 page of RAM when this is 0.
9	7	Reserved for future usage and should all be 0.

Table 4-1. iNES header information.

Following the header is the 512-byte trainer, if one is present, otherwise the ROM banks begin here, starting with PRG-ROM then CHR-ROM. The format allows for up to 256 different memory mappers. Each mapper is assigned a specific number and the mapper number can be obtained by shifting bits 4-7 of control byte 2 to the left by 4 bits and then adding the bits 4-7 of control byte 1. A complete list of mappers and their official iNES mapper numbers can be found in Appendix C.

5 - Input Devices

5.1 Control Pad

The 6502 used memory mapped I/O (input/output). This means that the same instructions and bus are used to communicate with I/O devices as with memory, that writing to a specific memory location writes to the appropriate device. In the NES, the I/O ports for input devices were $4016 and $4017 (see Appendix B).

The original NES used a rectangular control pad as shown in figure 5-1. The pad featured four buttons, A, B, Start and Select as well as a four-directional cross used to control movement. Although many variations were released, often with additional features such as slow motion and turbo fire, the original design was by far the most commonly used.

Figure 5-1. Original NES control pad [43].

The system reads multiple times from the I/O port to get all information about the controller. Each of the first eight reads indicates the status of one button on the standard controller in the order A, B, Select, Start, Up, Down, Left, Right. The first controller is attached to port $4016, the second to $4017. Using a four-player adapter it was possible to connect four controllers to the system, although this was rare. In this case controllers 1 and 3 were attached to $4016 and 2 and 4 to $4017. The next eight reads would get the status of the second controller on the port, otherwise they are ignored.

Reads 17-20 retrieve the signatures which identify whether a device is connected and if so, what type of device [7]. If a joypad is connected to $4016 the returned value is 01b, if one is connected to $4017 the returned value is 10b. There are four more reads which are not required before the cycle starts again.

The process of reading from an I/O device can be reset by use of a strobing method. When a reset is required, it is indicated by first writing a 1 to the port, followed by a 0.

5.2 Zapper

When the NES first launched in America, Nintendo included a light-gun known as the Zapper. Figure 5-2 shows the original version of the Zapper, although the colour was later changed to orange. By aiming using the sight, the gamer could produce quite accurate results. Several games featured Zapper support including Duck Hunt, Gumshoe and Wild Gunman [44].

Appendix B
NES I/O Registers

The following information is based on [7]:

Address	Access Level	Description
$2000	Write	PPU Control Register 1: • Bits 0-1 - Name table address, changes between the four name tables at $2000 (0), $2400 (1), $2800 (2) and $2C00 (3). • Bit 2 - Specifies amount to increment address by, either 1 if this is 0 or 32 if this is 1. • Bit 3 - Identifies which pattern table sprites are stored in, either $0000 (0) or $1000 (1). • Bit 4 - Identifies which pattern table the background is stored in, either $0000 (0) or $1000 (1). • Bit 5 - Specifies the size of sprites in pixels, 8x8 if this is 0, otherwise 8x16. • Bit 6 - Changes PPU between master and slave modes. This is not used by the NES. • Bit 7 - Indicates whether a NMI should occur upon V-Blank.
$2001	Write	PPU Control Register 2: • Bit 0 - Indicates whether the system is in colour (0) or monochrome mode (1), • Bit 1 - Specifies whether to clip the background, that is whether to hide the background in the left 8 pixels on screen (0) or to show them (1). • Bit 2 - Specifies whether to clip the sprites, that is whether to hide sprites in the left 8 pixels on screen (0) or to show them (1). • Bit 3 - If this is 0, the background should not be displayed. • Bit 4 - If this is 0, sprites should not be displayed. • Bits 5-7 - Indicates background colour in monochrome mode or colour intensity in colour mode.
$2002	Read	PPU Status Register: • Bit 4 - If set, indicates that writes to VRAM should be ignored. • Bit 5 - Scanline sprite count, if set, indicates more than 8 sprites on the current scanline. • Bit 6 - Sprite 0 hit flag, set when a non-transparent pixel of sprite 0 overlaps a non-transparent background pixel. • Bit 7 - Indicates whether V-Blank is occurring.
$2003	Write	SPR-RAM Address Register: Holds the address in SPR-RAM to access on the next write to $2004.
$2004	Write	SPR-RAM I/O Register: Writes a byte to SPR-RAM at the address indicated by $2003.
$2005	Write	VRAM Address Register 1.

$2006	Write	VRAM Address Register 2.
$2007	Read / Write	VRAM I/O Register: Reads or writes a byte from VRAM at the current address.
$4000	Write	pAPU Pulse 1 Control Register.
$4001	Write	pAPU Pulse 1 Ramp Control Register.
$4002	Write	pAPU Pulse 1 Fine Tune (FT) Register.
$4003	Write	pAPU Pulse 1 Coarse Tune (CT) Register.
$4004	Write	pAPU Pulse 2 Control Register.
$4005	Write	pAPU Pulse 2 Ramp Control Register.
$4006	Write	pAPU Pulse 2 Fine Tune Register.
$4007	Write	pAPU Pulse 2 Coarse Tune Register.
$4008	Write	pAPU Triangle Control Register 1.
$4009	Write	pAPU Triangle Control Register 2.
$400A	Write	pAPU Triangle Frequency Register 1.
$400B	Write	pAPU Triangle Frequency Register 2.
$400C	Write	pAPU Noise Control Register 1.
$400E	Write	pAPU Noise Frequency Register 1.
$400F	Write	pAPU Noise Frequency Register 2.
$4010	Write	pAPU Delta Modulation Control Register.
$4011	Write	pAPU Delta Modulation D/A Register.
$4012	Write	pAPU Delta Modulation Address Register.
$4013	Write	pAPU Delta Modulation Data Length Register.
$4014	Write	Sprite DMA Register: Writes cause a DMA transfer to occur from CPU memory at address $100 x n, where n is the value written, to SPR-RAM.
$4015	Read / Write	pAPU Sound / Vertical Clock Signal Register.
$4016	Read / Write	Joypad 1: • Bit 0 - Reads data from joypad or causes joypad strobe when writing. • Bit 3 - Indicates whether Zapper is pointing at a sprite. • Bit 4 - Cleared when Zapper trigger is released. Only bit 0 is involved in writing.
$4017	Read / Write	Joypad 2: When reading: • Bit 0 - Reads data from joypad or causes joypad strobe when writing. • Bit 3 - Indicates whether Zapper is pointing at a sprite. • Bit 4 - Cleared when Zapper trigger is released. Only bit 0 is involved in writing.