CPU ALL

From NESdev Wiki

Contents

CPU

The NES CPU core is based on the 6502 processor and runs at approximately 1.79 MHz (1.66 MHz in a PAL NES). It is made by Ricoh (http://en.wikipedia.org/wiki/Ricoh) and lacks the MOS6502's decimal mode. In the NTSC NES, the RP2A03 (http://en.wikipedia.org/wiki/Ricoh_2A03) chip contains the CPU and APU; in the PAL NES, the CPU and APU are contained within the RP2A07 (http://en.wikipedia.org/wiki/Ricoh_2A03) chip.

Sections

- CPU instructions
- CPU addressing modes
- CPU memory map
- CPU power-up state
- CPU registers
- CPU status flag behavior
- CPU interrupts
- Unofficial opcodes
- CPU pin-out and signals, and other hardware pin-outs

Frequencies

The CPU generates its clock signal by dividing the master clock signal.

Rate	NTSC NES/ Famicom	PAL NES	Dendy
Color subcarrier frequency f_{sc} (exact)	$3579545.\overline{45}$ Hz (315/88 MHz)	4433618.75 Hz	4433618.75 Hz
Color subcarrier frequency f_{sc} (approx.)	3.579545 MHz	4.433619 MHz	4.433619 MHz
Master clock frequency $6f_{sc}$	21.477272 MHz	26.601712 MHz	26.601712 MHz
Clock divisor d	12	16	15
CPU clock frequency $6f_{sc}/d$	1.789773 MHz (~559 ns per cycle)	1.662607 MHz (~601 ns per cycle)	1.773448 MHz (~564 ns per cycle)

* The vast majority of PAL famiclones use a chipset or NOAC with this timing. A small number have UMC UA6540+6541, which also uses PAL NES timing.[1]

Notes

- All illegal 6502 opcodes execute identically on the 2A03/2A07.
- Every cycle on 6502 is either a read or a write cycle.
- A printer friendly version covering all section is available here.
- Emulator authors may wish to emulate the NTSC NES/Famicom CPU at 21441960 Hz ((341×262−0.5)×4×60) to ensure a synchronised/stable 60 frames per second.[2]

See also

- Cycle reference chart

- 2A03 technical reference (http://nesdev.org/2A03%20technical%20reference.txt) by Brad Taylor. (Pretty old at this point; information on the wiki might be more up-to-date.)

References

1. nesdev forum: Eugene.S provides a list of famiclones (https://forums.nesdev.org/viewtopic.php?f=3&t=17213#p216082)
2. nesdev forum: Mesen - NES Emulator (http://forums.nesdev.org/viewtopic.php?p=223679#p223679)

Memory map

Address range	Size	Device
$0000–$07FF	$0800	2 KB internal RAM
$0800–$0FFF	$0800	Mirrors of $0000–$07FF
$1000–$17FF	$0800	
$1800–$1FFF	$0800	
$2000–$2007	$0008	NES PPU registers
$2008–$3FFF	$1FF8	Mirrors of $2000–$2007 (repeats every 8 bytes)
$4000–$4017	$0018	NES APU and I/O registers
$4018–$401F	$0008	APU and I/O functionality that is normally disabled. See CPU Test Mode.
$4020–$FFFF • *$6000–$7FFF* • *$8000–$FFFF*	$BFE0 *$2000* *$8000*	Unmapped. Available for cartridge use. *Usually cartridge RAM, when present.* *Usually cartridge ROM and mapper registers.*

- Some parts of the 2 KiB of internal RAM at $0000–$07FF have predefined purposes dictated by the 6502 architecture:
  - $0000-$00FF: The zero page, which can be accessed with fewer bytes and cycles than other addresses
  - $0100–$01FF: The page containing the stack, which can be located anywhere here, but typically starts at $01FF and grows downward

  Games may divide up the rest however the programmer deems useful. See Sample RAM map for an example allocation strategy for this RAM. Most commonly, $0200-$02FF is used for the OAM buffer to be copied to PPU OAM during vblank.

- The unmapped space at $4020-$FFFF can be used by cartridges for any purpose, such as ROM, RAM, and registers. Many common mappers place ROM and save/work RAM in these locations:
  - $6000–$7FFF: Battery-backed save or work RAM (usually referred to as WRAM or

PRG-RAM)
    - $8000–$FFFF: ROM and mapper registers (see MMC1 and UxROM for examples)

  The cartridge is able to passively observe reads from and writes to any address in the CPU address space, even outside this unmapped space, except for reads from $4015, the only readable register that is internal to the CPU. The cartridge can map writable registers anywhere, but its readable memory can only be placed where it does not interfere with other readable hardware, which would produce a bus conflict. While cartridges can map readable memory at $4000-$4014 and $4018-$401F, a quirk in the 2A03's register decoding can cause DMA to misbehave if the CPU is halted while reading from $4000-$401F, so it is recommended that cartridges only map readable memory from $4020-$FFFF.

- If using DPCM playback, samples are limited to the following practical range:
    - $C000–$FFF1: DPCM sample data

  Sample playback wraps around from $FFFF to $8000. The highest sample starting address is $FFC0 and longest sample is $FF1 bytes, so the full DPCM range is $C000-$FFFF and $8000-$8FB0, but making use of the wraparound is challenging because of banking and the presence of the CPU vectors.

- The CPU expects interrupt vectors in a fixed place at the end of the unmapped space:
    - $FFFA–$FFFB: NMI vector, which points at an NMI handler
    - $FFFC–$FFFD: Reset vector, which points at code to initialize the NES chipset
    - $FFFE–$FFFF: IRQ/BRK vector, which may point at a mapper's interrupt handler (or, less often, a handler for APU interrupts)

  These vectors are supplied by the cartridge. Unless a mapper fixes $FFFA–$FFFF to some known bank (normally by fixing an entire bank-sized region at the top of the address space, such as $C000-$FFFF, to a specific bank) or uses some sort of reset detection, the vectors (and a suitable reset code stub) must be present in all banks.

- Reading from memory that is not mapped to anything normally returns open bus. The cartridge hardware may affect open bus behavior across the entire CPU address space, such as by pulling bits high or low.

Pin out and signal description

Pin out

```
        .--\/--.
 AD1 <- |01  40| -- +5V
 AD2 <- |02  39| -> OUT0
/RST -> |03  38| -> OUT1
 A00 <- |04  37| -> OUT2
 A01 <- |05  36| -> /OE1
 A02 <- |06  35| -> /OE2
 A03 <- |07  34| -> R/W
 A04 <- |08  33| <- /NMI
 A05 <- |09  32| <- /IRQ
```

```
   A06 <- |10  31| -> M2
   A07 <- |11  30| <- TST (usually GND)
   A08 <- |12  29| <- CLK
   A09 <- |13  28| <> D0
   A10 <- |14  27| <> D1
   A11 <- |15  26| <> D2
   A12 <- |16  25| <> D3
   A13 <- |17  24| <> D4
   A14 <- |18  23| <> D5
   A15 <- |19  22| <> D6
   GND -- |20  21| <> D7
          `------'
```

Signal description

Active-Low signals are indicated by a "/". Every cycle is either a read or a write cycle.

- **CLK** : 21.47727 MHz (NTSC) or 26.6017 MHz (PAL) clock input. Internally, this clock is divided by 12 (NTSC 2A03) or 16 (PAL 2A07) to feed the 6502's clock input φ0, which is in turn inverted to form φ1, which is then inverted to form φ2. φ1 is high during the first phase (half-cycle) of each CPU cycle, while φ2 is high during the second phase.
- **AD1** : Audio out pin (both pulse waves).
- **AD2** : Audio out pin (triangle, noise, and DPCM).
- **Axx** and **Dx** : Address and data bus, respectively. **Axx** holds the target address during the entire read/write cycle. For reads, the value is read from **Dx** during φ2. For writes, the value appears on **Dx** during φ2 (and no sooner).
- **OUT0..OUT2** : Output pins used by the controllers ($4016 output latch bits 0-2). These 3 pins are connected to either the NES or Famicom's expansion port, and **OUT0** is additionally used as the "strobe" signal (OUT) on both controller ports.
- **/OE1** and **/OE2** : Controller ports (for controller #1 and #2 respectively). Each enable the output of their respective controller, if present.
- **R/W** : Read/write signal, which is used to indicate operations of the same names. Low is write. **R/W** stays high/low during the entire read/write cycle.
- **/NMI** : Non-maskable interrupt pin. See the 6502 manual and CPU interrupts for more details.
- **/IRQ** : Interrupt pin. See the 6502 manual and CPU interrupts for more details.
- **M2** : Can be considered as a "signals ready" pin. It is a modified version the 6502's φ2 (which roughly corresponds to the CPU input clock φ0) that allows for slower ROMs. CPU cycles begin at the point where **M2** goes low.
  - In the NTSC 2A03E, G, and H, **M2** has a duty cycle of 15/24 (5/8), or 350ns/559ns. Equivalently, a CPU read (which happens during the second, high phase of **M2**) takes 1 and 7/8th PPU cycles. The internal φ2 duty cycle is exactly 1/2 (one half).
  - In the PAL 2A07, **M2** has a duty cycle of 19/32, or 357ns/601ns, or 1.9 out of 3.2 pixels.
  - In the original NTSC 2A03 (no letter), M2 has a duty cycle of 17/24, or 396ns/559ns, or 2 and 1/8th pixels.
- **TST** : (tentative name) Pin 30 is special: normally it is grounded in the NES, Famicom, PC10/VS. NES and other Nintendo Arcade Boards (Punch-Out!! and Donkey Kong 3). But if it is pulled high on the RP2A03G, extra diagnostic registers to test the sound hardware are enabled from $4018 through $401A, and the joystick ports $4016 and $4017 become open bus. On the RP2A07 (and presumably also the RP2A03E), pulling

pin 30 high instead causes the CPU to stop execution by means of activating the embedded 6502's RDY input.

- **/RST** : When low, holds CPU in reset state, during which all CPU pins (except pin 2) are in high impedance state. When released, CPU starts executing code (read $FFFC, read $FFFD, ...) after 6 M2 clocks.

Power up state

Initial tests on the power-up/reset state of the CPU/APU and RAM contents were done using an NTSC front-loading NES from 1988 with a RP2A03G CPU on the NES-CPU-07 board revision.

Countless bugs in commercial and homebrew games exist because of a reliance on the initial system state. An NES programmer should not rely on the state of CPU/APU registers and RAM contents not guaranteed at power-up/reset.

CPU

Initial CPU Register Values

Register	At Power	After Reset
A, X, Y	0	unchanged
PC	($FFFC)	($FFFC)
S[1]	$00 - 3 = $FD	S -= 3
C	0	unchanged
Z	0	unchanged
I	1	1
D	0	unchanged
V	0	unchanged
N	0	unchanged

APU

Initial APU Register Values

Register	At Power	After Reset
Pulses ($4000-$4007)	0	unchanged?
Triangle ($4008-$400B)	0	unchanged?
Triangle phase	?	0 (output = 15)
Noise ($400C-$400F)	0	unchanged?
Noise 15-bit LFSR	$0000 (all 0s, first clock shifts in a 1)[2]	unchanged?
DMC flags and rate ($4010)[3]	0	unchanged
DMC direct load ($4011)[3]	0	[$4011] &= 1
DMC sample address ($4012)[3]	0	unchanged
DMC sample length ($4013)[3]	0	unchanged
DMC LFSR	0? (revision-dependent?)	? (revision-dependent?)
Status ($4015)	0 (all channels disabled)	0 (all channels disabled)
Frame Counter ($4017)	0 (**frame IRQ enabled**)	unchanged
Frame Counter LFSR[4]	$7FFF (all 1s)	revision-dependent

Revision-dependent Register Values

2A03 letterless

Register	At Power	After Reset
DMC LFSR	0?	?
Frame Counter LFSR[4]	$7FFF (all 1s)	unchanged

2A03E, 2A03G, 2A07, various clones

Register	At Power	After Reset
DMC LFSR	0?	?
Frame Counter LFSR[4]	$7FFF (all 1s)	$7FFF (all 1s)

RAM contents

Internal RAM ($0000-$07FF) and cartridge RAM (usually $6000–$7FFF, depends on mapper) have an unreliable state on power-up and is unchanged after a reset. Some machines may have consistent RAM contents at power-up, but others may not. Emulators often implement a consistent RAM startup state (e.g. all $00 or $FF, or a particular pattern), and flashcarts may partially or fully initialize RAM before starting a program.

Battery-backed save RAM and other types of SRAM/NVRAM have an unreliable state on the first power-up and is generally unchanged after subsequent resets and power-ups. However, there is an added chance of data corruption due to loss of power or other external factors (bugs, cheats, etc). Emulators and flashcarts may initialize save files with a consistent state (much like other sections of RAM) and persist this data without corruption after closing or reloading a game.

Because of these factors, an NES programmer must be careful not to blindly trust the initial contents of RAM.

Best practices

- Configure the emulator so it provides a random system state and random RAM contents on power-up.
  - Mesen (https://www.mesen.ca/) provides a set of such emulation options recommended for developers, along with a debugger setting to break execution on all reads from uninitialized RAM.
- Refer to the init code article when setting up the reset handler. The sample implementation is a good point to start from.
  - If you are using an audio driver, make sure to call its initialization routine in the reset handler before playing any sound.
- If some RAM state is intended to persist across resets, ensure that the checks used to do so are robust against random initial RAM contents. (e.g. unique multi-byte signatures, checksum calculations, etc)
- Validate any data read from potentially unreliable sources before using it. For example, the stats of an RPG character could be checked against valid ranges when loading them from a save.

See also

- PPU power up state

References

1. RESET uses the logic shared with NMI, IRQ, and BRK that would push PC and P. However, like some but not all 6502s, the 2A03 prohibits writes during reset. This test (https://forums.nesdev.org/viewtopic.php?p=184247#p184247) relies on open bus being precharged by these reads. See 27c3: Reverse Engineering the MOS 6502 CPU (en) (https://www.youtube.com/watch?v=fWqBmmPQP40&t=41m45s) from 41:45 onward for details
2. Noise channel init log (https://forums.nesdev.org/viewtopic.php?p=172797#p172797)
3. DMC power-up state manifests as buzzing in Eliminator Boat Duel (https://forums.nesdev.org/viewtopic.php?p=231773#p231773)
4. 2A03letterless is missing transistor to set frame counter LFSR on reset (https://forums.nesdev.org/viewtopic.php?p=214939#p214939)

Status flag behavior

The **flags** register, also called **processor status** or just **P**, is one of the six architectural registers on the 6502 family CPU. It is composed of six one-bit registers. Instructions modify one or more bits and leave others unchanged.

Flags

Instructions that save or restore the flags map them to bits in the architectural 'P' register as follows:

```
7  bit  0
---- ----
NV1B DIZC
|||| ||||
|||| |||+- Carry
|||| ||+-- Zero
|||| |+--- Interrupt Disable
|||| +---- Decimal
|||+------ (No CPU effect; see: the B flag)
||+------- (No CPU effect; always pushed as 1)
|+-------- Overflow
+--------- Negative
```

- The PHP (Push Processor Status) and PLP (Pull Processor Status) instructions can be used to retrieve or set this register directly via the stack.
- Interrupts (NMI and IRQ/BRK) implicitly push the status register to the stack.
- Interrupts returning with RTI will implicitly pull the saved status register from the stack.
- The two bits with no CPU effect are ignored when pulling flags from the stack; there are no corresponding registers for them in the CPU.
- When P is displayed as a single 8-bit register by debuggers, there is no convention for what values to use for bits 5 and 4 and their values should not be considered meaningful.

C: Carry

- After ADC, this is the carry result of the addition.
- After SBC or CMP, both of which do subtraction, this flag will be set if no borrow was the result, or alternatively a "greater than or equal" result.
- After a shift instruction (ASL, LSR, ROL, ROR), this contains the bit that was shifted out.
- Increment and decrement instructions do not affect the carry flag.
- Can be set or cleared directly with SEC or CLC.

Z: Zero

- After most instructions that have a value result, this flag will either be set or cleared based on whether or not that value is equal to zero.

I: Interrupt Disable

- When set, IRQ interrupts are inhibited. NMI, BRK, and reset are not affected.
- Can be set or cleared directly with SEI or CLI.
- Automatically set by the CPU after pushing flags to the stack when any interrupt is triggered (NMI, IRQ/BRK, or reset). Restored to its previous state from the stack when leaving an interrupt handler with RTI.
- If an IRQ is pending when this flag is cleared (i.e. the /IRQ line is low), an interrupt will be triggered immediately. However, the effect of toggling this flag is delayed 1 instruction when caused by SEI, CLI, or PLP.

D: Decimal

- On the NES, decimal mode is disabled and so this flag has no effect. However, it still exists and can be observed and modified, as normal.
- On the original 6502, this flag causes some arithmetic instructions to use binary-coded decimal representation to make base 10 calculations easier.
- Can be set or cleared directly with SED or CLD.

V: Overflow

- ADC and SBC will set this flag if the signed result would be invalid[1], necessary for making signed comparisons[2].
- BIT will load bit 6 of the addressed value directly into the V flag.
- Can be cleared directly with CLV. There is no corresponding set instruction, and the NES CPU does not expose the 6502's Set Overflow (SO) pin.

N: Negative

- After most instructions that have a value result, this flag will contain bit 7 of that result.
- BIT will load bit 7 of the addressed value directly into the N flag.

The B flag

While there are only six flags in the processor status register within the CPU, the value pushed to the stack contains additional state in bit 4 called the B flag that can be useful to software. The value of B depends on what caused the flags to be pushed. Note that this flag does not represent a register that can hold a value, but rather a transient signal in the CPU controlling whether it was processing an interrupt when the flags were pushed. B is 0 when pushed by interrupts (NMI and IRQ) and 1 when pushed by instructions (BRK and PHP).

Cause	B flag
NMI	0
IRQ	0

BRK	1
PHP	1

Because IRQ and BRK use the same IRQ vector, testing the state of the B flag pushed by the interrupt to the stack is the only way for an IRQ handler to distinguish between them. The B flag also allows software to identify whether interrupt hijacking has occurred, where an NMI overrides the BRK instruction, but the B flag is still set by the BRK. However, testing this bit from the stack is fairly slow, which is one reason why BRK wasn't used as a syscall mechanism. Instead, it was more often used to trigger a patching mechanism that hung off the IRQ vector: a single byte in programmable ROM would be forced to 0, and the IRQ handler would pick something to do instead based on the program counter.

Some debugging tools, such as Visual6502, display the B flag as bit 4 of P as a matter of convenience. The user can see it turn off at the start of an interrupt and back on after the CPU reads the vector.

External links

- koitsu's explanation (http://forums.nesdev.org/viewtopic.php?p=64224#p64224)

References

1. Article: The Overflow (V) Flag Explained (http://www.6502.org/tutorials/vflag.html)
2. Article: Beyond 8-bit Unsigned Comparisons (http://www.6502.org/tutorials/compare_beyond.html#5), Signed Comparisons

Retrieved from "https://www.nesdev.org/w/index.php?title=CPU_ALL&oldid=22067"

This page was last edited on 22 September 2024, at 07:36.