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

CPU signals and 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 (39375000/11 Hz)	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 use UMC UA6540+6541, which also 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

- 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

Memory map

Address range	Size	Device
$0000-$07FF	$0800	2KB 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	$BFE0	Cartridge space: PRG ROM, PRG RAM, and mapper registers (See Note)

Some parts of the 2 KiB of internal RAM at $0000-$07FF have predefined purposes dictated by the 6502 architecture. The zero page is $0000-$00FF, and the stack always uses some part of the $0100-$01FF page. Games may divide up the rest however the programmer deems useful. See Sample RAM map for an example allocation strategy for this RAM.

Note: Most common boards and iNES mappers address ROM and Save/Work RAM in this format:

- $6000-$7FFF = Battery Backed Save or Work RAM
- $8000-$FFFF = Usual ROM, commonly with Mapper Registers (see MMC1 and UxROM for example)

If using DMC audio:

- $C000-$FFF1 = DPCM samples

The CPU expects interrupt vectors in a fixed place at the end of the cartridge space:

- $FFFA-$FFFB = NMI vector
- $FFFC-$FFFD = Reset vector
- $FFFE-$FFFF = IRQ/BRK vector

If a mapper doesn't fix $FFFA-$FFFF to some known bank (typically, along with the rest of the bank containing them, e.g. $C000-$FFFF for a 16KiB banking mapper) or use some sort of reset detection, the vectors need to be stored in all banks.

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

The following results are from a US (NTSC) NES, original front-loading design, RP2A03G CPU chip, NES-CPU-07 main board revision, manufactured in 1988. The memory values are probably slightly different for each individual NES console. Please note that you should NOT rely on the state of any registers after Power-UP and especially not the stack register and RAM ($0000-$07FF).

At power-up

P = $34[3] (IRQ disabled)[4]
A, X, Y = 0
S = $FD[5]
$4017 = $00 (frame irq enabled)
$4015 = $00 (all channels disabled)
$4000-$400F = $00
$4010-$4013 = $00 [6]
All 15 bits of noise channel LFSR = $0000[7]. The first time the LFSR is clocked from the all-0s state, it will shift in a 1.

2A03G: APU Frame Counter reset. (but 2A03letterless: APU frame counter powers up at a value equivalent to 15)

Internal memory ($0000-$07FF) has unreliable startup state. Some machines may have consistent RAM contents at power-on, but others do not.

- Emulators often implement a consistent RAM startup state (e.g. all $00 or $FF, or a particular pattern), and flash carts like the PowerPak may partially or fully initialize RAM before starting a program, so an NES programmer must be careful not to rely on the startup contents of RAM.

After reset

A, X, Y were not affected
S was decremented by 3 (but nothing was written to the stack)[5]
The I (IRQ disable) flag was set to true (status ORed with $04)
The internal memory was unchanged
APU mode in $4017 was unchanged
APU was silenced ($4015 = 0)
APU triangle phase is reset to 0 (i.e. outputs a value of 15, the first step of its waveform)
APU DPCM output ANDed with 1 (upper 6 bits cleared)
2A03G: APU Frame Counter reset. (but 2A03letterless: APU frame counter retains old value) [8]

See also

- PPU power up state

Notes

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.

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

```
7  bit  0
---- ----
NVss DIZC
|||| ||||
|||| |||+- Carry
|||| ||+-- Zero
|||| |+--- Interrupt Disable
|||| +---- Decimal
||++------ No CPU effect, see: the B flag
|+-------- 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, including the NMI and also the pseudo-interrupt BRK instruction, implicitly push the status register to the stack.
- Interrupts returning with RTI will implicitly pull the saved status register from the stack.

C: Carry

- After ADC, this is the carry result of the addition.
- After SBC or CMP, 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, CLC.

Z: Zero

- After most instructions that have a value result, if that value is zero, this flag will be set.

I: Interrupt Disable

- When set, all interrupts except the NMI are inhibited.
- Can be set or cleared directly with SEI, CLI.
- Automatically set by the CPU when an IRQ is triggered, and restored to its previous state by RTI.
- If the /IRQ line is low (IRQ pending) when this flag is cleared, an interrupt will immediately be triggered.

D: Decimal

- On the NES, this flag has no effect.
- 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, CLD.

V: Overflow

- ADC and SBC will set this flag if the signed result would be invalid[9], necessary for making signed comparisons[10].
- 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.

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, when transferred to the stack, there are two additional bits. These do not represent a register that can hold a value but can be used to distinguish how the flags were pushed.

Some 6502 references call this the "B flag", though it does not represent an actual CPU register.

Two interrupts (/IRQ and /NMI) and two instructions (PHP and BRK) push the flags to the stack. In the byte pushed, bit 5 is always set to 1, and bit 4 is 1 if from an instruction (PHP or BRK) or 0 if from an interrupt line being pulled low (/IRQ or /NMI). This is the only time and place where the B flag actually exists: not in the status register itself, but

in bit 4 of the copy that is written to the stack.

Instruction	Bits 5 and 4	Side effects after pushing
PHP	11	None
BRK	11	I is set to 1
/IRQ	10	I is set to 1
/NMI	10	I is set to 1

Two instructions (PLP and RTI) pull a byte from the stack and set all the flags. They ignore bits 5 and 4.

The only way for an IRQ handler to distinguish /IRQ from BRK is to read the flags byte from the stack and test bit 4. The slowness of this 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 PROM, UVEPROM, flash, etc. would be forced to 0, and the IRQ handler would pick something to do instead based on the program counter.

Unlike bits 5 and 4, bit 3 actually exists in P, even though it doesn't affect the ALU operation on the 2A03 or 2A07 CPU the way it does in MOS Technology's own chips.

External links

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

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)
3. The golden log of nestest differs from this in the irrelevant bits 5 and 4 of P
4. IRQ was first asserted about 1/60 second after power-up, by APU.
5. RESET uses the logic shared with NMI, IRQ, and BRK that would push PC and P. However, like some but not all 6502s (http://visual6502.org/wiki/index.php?title=6502_BRK_and_B_bit#masking_of_the_stack_writes_during_RESET), 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
6. Eliminator Boat Duel (https://forums.nesdev.org/viewtopic.php?t=18278)
7. Noise channel init log (https://forums.nesdev.org/viewtopic.php?p=172797#p172797)
8. 2A03letterless is missing transistor to set frame counter LFSR on reset (https://forums.nesdev.org/viewtopic.php?p=214939#p214939)
9. Article: The Overflow (V) Flag Explained (http://www.6502.org/tutorials/vflag.html)
10. Article: Beyond 8-bit Unsigned Comparisons (http://www.6502.org/tutorials/compare_beyond.html#5), Signed Comparisons

References