NMI

From Nesdev wiki

The 2A03 and most other 6502 family CPUs are capable of processing a **non-maskable interrupt** (**NMI**). This input is edge-sensitive, meaning that if other circuitry on the board pulls the /NMI pin from high to low voltage, this sets a flip-flop in the CPU. When the CPU checks for interrupts and find that the flip-flop is set, it pushes the processor status register and return address on the stack, reads the NMI handler's address from $FFFA-$FFFB, clears the flip-flop, and jumps to this address.

"Non-maskable" means that no state inside the CPU can prevent the NMI from being processed as an interrupt. However, most boards that use a 6502 CPU's /NMI line allow the CPU to disable the generation of /NMI signals by writing to a memory-mapped I/O device. In the case of the NES, the /NMI line is connected to the NES PPU and is used to detect vertical blanking.

Contents

Operation

Two 1-bit registers inside the PPU control the generation of NMI signals. Frame timing and accesses to the PPU's PPUCTRL and PPUSTATUS registers change these registers as follows, regardless of whether rendering is enabled:

1. Start of vertical blanking: Set NMI_occurred in PPU to true.
2. End of vertical blanking, sometime in pre-render scanline: Set NMI_occurred to false.
3. Read PPUSTATUS: Return old status of NMI_occurred in bit 7, then set NMI_occurred to false.
4. Write to PPUCTRL: Set NMI_output to bit 7.

The PPU pulls /NMI low if and only if both NMI_occurred and NMI_output are true. By toggling NMI_output (PPUCTRL.7) during vertical blank without reading PPUSTATUS, a program can cause /NMI to be pulled low multiple times, causing multiple NMIs to be generated.

Caveats

Old emulators

Some platforms, such as the Game Boy, keep a flag turned on through the whole vertical blanking interval. Some early emulators such as NESticle were developed under the assumption that PPUSTATUS.7 worked the same way and thus do *not* turn off NMI_occurred in line 3. Thus, some defective homebrew programs developed in this era will wait for PPUSTATUS.7 to become false and expect this to happen at the end of vblank. (The right way to wait

for the end of vblank involves triggering a sprite 0 hit and waiting for *that* flag to become 0.) Some newer homebrew programs have been known to display a diagnostic message if an emulator incorrectly returns true on two consecutive reads of PPUSTATUS.7.

Race condition

If 1 and 3 happen simultaneously, PPUSTATUS.7 is read as false, and NMI_occurred is set to false anyway. This means that the following code that waits for vertical blank by spinning on PPUSTATUS.7 is likely to miss an occasional frame:

```
wait_status7:
  bit PPUSTATUS
  bpl wait_status7
  rts
```

Code like `wait_status7` is fine while your program is waiting for the PPU to warm up. But once the game is running, the most reliable way to wait for a vertical blank is to turn on NMI_output and then wait for the NMI handler to set a variable:

```
wait_nmi:
  lda retraces
@notYet:
  cmp retraces
  beq @notYet
  rts

nmi_handler:
  inc retraces
  rti
```

This handler works for simple cases, and in many cases, it is the simplest thing that could possibly work (http://c2.com/xp/DoTheSimplestThingThatCouldPossiblyWork.html). But if your game code takes significantly longer than 24,000 cycles, such as if you have too many critters moving on the screen, it may take longer than one frame. Waiting for NMI in this way would miss an NMI that happens while other code is running. In some cases, this could cause a sprite 0-triggered scroll split to flicker (or worse). The next step up involves doing VRAM updates and sprite 0 waiting in a separate NMI thread that is guaranteed to run every frame.

Because Gradius puts its status bar at the bottom, it can't just spin on sprite 0 all the time. Instead, it counts the approximate time that each object handler takes and deliberately delays the remaining calculations to the next frame when it might otherwise come close to missing a sprite 0 hit. Games developed by Micronics are likely to reduce a game's overall frame rate far below 60 frames per second to match the worst case of lag.