Interrupt flag

The **Interrupt flag** (**IF**) is a system flag bit in the x86 architecture's FLAGS register, which determines whether or not the central processing unit (CPU) will handle maskable hardware interrupts.[1]

The bit, which is bit 9 of the FLAGS register, may be set or cleared by programs with sufficient privileges, as usually determined by the operating system. If the flag is set to `1`, maskable hardware interrupts will be handled. If cleared (set to `0`), such interrupts will be ignored. IF does not affect the handling of non-maskable interrupts (NMIs) or software interrupts generated by the INT instruction.

Contents

Setting and clearing

The flag may be set or cleared using the CLI (**Cl**ear **I**nterrupts), STI (**Se**t **I**nterrupts) and POPF (**Pop F**lags) instructions.

CLI clears IF (sets to 0), while STI sets IF to 1. POPF pops 16 bits off the stack into the FLAGS register, which means IF will be set or cleared based on the ninth bit on the top of the stack.[1]

Privilege level

In all three cases, only privileged applications (usually the OS kernel) may modify IF. Note that this only applies to protected mode code. (Real mode code may always modify IF.)

CLI and STI are privileged instructions, which trigger a general protection fault if an unprivileged application attempts to execute it, while POPF will simply not modify the IF flag if the application is unprivileged.

The privilege level required to execute a CLI or STI instruction, or set IF using POPF, is determined by the IOPL (I/O Privilege Level) in EFLAGS. If the IOPL is set to 2 for example, any program running only in ring 0 can execute a CLI. Most modern operating systems set the IOPL to be 0 so only the kernel can execute CLI/STI. The reason for this is that since clearing IF will force the processor to ignore all interrupts, the kernel may never get control back if it is not set to 1 again.

Old DOS programs

Some old DOS programs that use a protected mode DOS extender and install their own interrupt handlers (usually games) use the CLI instruction in the handlers to disable interrupts and either POPF (after a corresponding PUSHF) or IRET (which restores the flags from the stack as part of its effects) to restore it. This works if the program was started in real mode, but causes problems when such programs are run in a DPMI-based container on modern operating systems (such as NTVDM under Windows NT or later). Since CLI is a privileged instruction, it triggers a fault into the operating system when the program attempts to use it. The OS then typically stops delivering interrupts to the program until the program executes STI (which would cause another fault). However, the POPF instruction is not privileged and simply fails silently to restore the IF. The result is that the OS stops delivering interrupts to the program, which then hangs. DOS programs that do not use a protected mode extender do not suffer from this problem, as they execute in V86 mode where POPF does trigger a fault.

There are few satisfactory resolutions to this issue. It is usually not possible to modify the program as source code is typically not available and there is no room in the instruction stream to introduce a STI without massive editing at the assembly level. Removing CLI's from the program or causing the V86 host to ignore CLI completely might cause other bugs if the guest's interrupt handlers are not re-entrant safe (though when executed on a modern processor, they typically execute fast enough to avoid overlapping of interrupts).

CLI

CLI is commonly used as a synchronization mechanism in uniprocessor systems. For example, a CLI is used in operating systems to disable interrupts so kernel code (typically a driver) can avoid race conditions with an interrupt handler. Note that CLI only affects the interrupt flag for the processor on which it is executed; in multiprocessor systems, executing a CLI instruction does not disable interrupts on other processors. Thus, a driver/interrupt handler race condition can still occur because other processors may service interrupts and execute the offending interrupt handler. For these systems, other synchronization mechanisms such as locks must be used in addition to CLI/STI to prevent all race conditions.

Because the HLT instruction halts until an interrupt occurs, the combination of a CLI followed by a HLT is commonly used to intentionally hang the computer.

STI

The STI instruction enables interrupts by setting the IF.

One interesting quirk about the STI instruction is that, unlike CLI which has an immediate effect, interrupts are not actually enabled until after the instruction immediately following the STI. One side effect of this could be IF=0, then executing a CLI instruction immediately after an STI instruction means that interrupts are never recognized. The STI instruction sets the IF flag, but interrupts are not checked for until after the next instruction which in this case would be the CLI which takes effect immediately. This behavior exists so a processor that constantly takes interrupts can still make forward progress. See IA-32 manuals for details.

See also

- FLAGS register (computing)
- Intel 8259
- Advanced Programmable Interrupt Controller (APIC)
- Interrupt
- Interrupt handler
- Non-maskable interrupt (NMI)
- Programmable Interrupt Controller (PIC)
- x86

References

1. "Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual" (http://download.intel.com/design/PentiumII/manuals/24319102.PDF) (PDF). Retrieved 2007-07-13.

External links

- Intel 64 and IA-32 Architectures Software Developer Manuals (https://software.intel.com/en-us/articles/intel-sdm) - Retrieved 2017-09-14