Exceptions

From OSDev Wiki

Exceptions as described in this article are generated by the CPU when an 'error' occurs. Some exceptions are not really errors in most cases, such as page faults. Exceptions are a type of interrupt.

Exceptions are classified as:

- **Faults**: These can be corrected and the program may continue as if nothing happened.
- **Traps**: Traps are reported immediately after the execution of the trapping instruction.
- **Aborts**: Some severe unrecoverable error.

Some exceptions will push a 32-bit "error code" on to the top of the stack, which provides additional information about the error. This value must be pulled from the stack before returning control back to the currently running program. (i.e. before calling IRET)

Contents

Name	Vector nr.	Type	Mnemonic	Error code?
Divide-by-zero Error	0 (0x0)	Fault	#DE	No
Debug	1 (0x1)	Fault/Trap	#DB	No
Non-maskable Interrupt	2 (0x2)	Interrupt	-	No
Breakpoint	3 (0x3)	Trap	#BP	No
Overflow	4 (0x4)	Trap	#OF	No
Bound Range Exceeded	5 (0x5)	Fault	#BR	No
Invalid Opcode	6 (0x6)	Fault	#UD	No

Device Not Available	7 (0x7)	Fault	#NM	No
Double Fault	8 (0x8)	Abort	#DF	Yes
~~**Coprocessor Segment Overrun**~~	9 (0x9)	Fault	-	No
Invalid TSS	10 (0xA)	Fault	#TS	Yes
Segment Not Present	11 (0xB)	Fault	#NP	Yes
Stack-Segment Fault	12 (0xC)	Fault	#SS	Yes
General Protection Fault	13 (0xD)	Fault	#GP	Yes
Page Fault	14 (0xE)	Fault	#PF	Yes
Reserved	15 (0xF)	-	-	No
x87 Floating-Point Exception	16 (0x10)	Fault	#MF	No
Alignment Check	17 (0x11)	Fault	#AC	Yes
Machine Check	18 (0x12)	Abort	#MC	No
SIMD Floating-Point Exception	19 (0x13)	Fault	#XM/#XF	No
Virtualization Exception	20 (0x14)	Fault	#VE	No
Reserved	21-29 (0x15-0x1D)	-	-	No
Security Exception	30 (0x1E)	-	#SX	Yes
Reserved	31 (0x1F)	-	-	No
Triple Fault	-	-	-	No
~~**FPU Error Interrupt**~~	IRQ 13	Interrupt	#FERR	No

Exceptions

Faults

Divide-by-zero Error

The **Divide-by-zero Error** occurs when dividing any number by 0 using the DIV or IDIV instruction. Many OS developers use this exception to test whether their exception handling code works. This exception may also occur when the result is too large to be represented in the destination.

The saved instruction pointer points to the DIV or IDIV instruction which caused the exception.

Bound Range Exceeded

This exception can occur when the BOUND instruction is executed. The BOUND instruction compares an array index with the lower and upper bounds of an array. When the index is out of bounds, the Bound Range Exceeded exception occurs.

The saved instruction pointer points to the BOUND instruction which caused the exception.

Invalid Opcode

The Invalid Opcode exception occurs when the processor tries to execute an invalid or undefined opcode, or an instruction with invalid prefixes. It also occurs when an instruction exceeds 15 bytes, but this only occurs with redundant prefixes.

The saved instruction pointer points to the instruction which caused the exception.

Device Not Available

The Device Not Available exception occurs when an FPU instruction is attempted but there is no FPU. This is not likely, as modern processors have built-in FPUs. However, there are flags in the CR0 register that disable the FPU/MMX/SSE instructions, causing this exception when they are attempted. This feature is useful because the operating system can detect when a user program uses the FPU or XMM registers and then save/restore them appropriately when multitasking.

The saved instruction pointer points to the instruction that caused the exception.

Invalid TSS

An Invalid TSS exception occurs when an invalid segment selector is referenced as part of a task which, or as a result of a control transfer through a gate descriptor, which results in an invalid stack-segment reference using an SS selector in the TSS.

When the exception occurred before loading the segment selectors from the TSS, the saved instruction pointer points to the instruction which caused the exception. Otherwise, and this is more common, it points to the first instruction in the new task.

Error code: The Invalid TSS exception sets an error code, which is a selector index.

Segment Not Present

The Segment Not Present exception occurs when trying to load a segment or gate which has it's Present-bit set to 0. However when loading a stack-segment selector which references a descriptor which is not present, a Stack-Segment Fault occurs.

The saved instruction pointer points to the instruction which caused the exception.

Error code: The Segment Not Present exception sets an error code, which is the segment selector index of the segment descriptor which caused the exception.

Stack-Segment Fault

The Stack-Segment Fault occurs when:

- Loading a stack-segment referencing a segment descriptor which is not present.
- Any PUSH or POP instruction or any instruction using ESP or EBP as a base register is executed, while the stack address is not in canonical form.
- When the stack-limit check fails.

The saved instruction pointer points to the instruction which caused the exception.

Error code: The Stack-Segment Fault sets an error code, which is the stack segment selector index when a non-present segment descriptor was referenced. Otherwise, 0.

General Protection Fault

A General Protection Fault may occur for various reasons. The most common are:

- Segment error (privilege, type, limit, read/write rights).
- Executing a privileged instruction while CPL != 0.
- Writing a 1 in a reserved register field.
- Referencing or accessing a null-descriptor.

The saved instruction pointer points to the instruction which caused the exception.

Error code: The General Protection Fault sets an error code, which is the segment selector index when the exception is segment related. Otherwise, 0.

Page Fault

> *Main article:* Page fault

A Page Fault occurs when:

- A page directory or table entry is not present in physical memory.
- Attempting to load the instruction TLB with a translation for a non-executable page.
- A protection check (privileges, read/write) failed.
- A reserved bit in the page directory or table entries is set to 1.

The saved instruction pointer points to the instruction which caused the exception.

Error code

The Page Fault sets an error code:

	Length	Name	Description
P	1 bit	Present	When set, the page fault was caused by a page-protection violation. When not set, it was caused by a non-present page.
W	1 bit	Write	When set, the page fault was caused by a page write. When not set, it was caused by a page read.
U	1 bit	User	When set, the page fault was caused while CPL = 3. This does not necessarily mean that the page fault was a privilege violation.
R	1 bit	Reserved write	When set, the page fault was caused by reading a 1 in a reserved field.
I	1 bit	Instruction Fetch	When set, the page fault was caused by an instruction fetch.

In addition, it sets the value of the CR2 register to the virtual address which caused the Page Fault.

x87 Floating-Point Exception

The x87 Floating-Point Exception occurs when the FWAIT or WAIT instruction, or any waiting floating-point instruction is executed, and the following conditions are true:

- CR0.NE is 1;
- an unmasked x87 floating point exception is pending (i.e. the exception bit in the x87 floating point status-word register is set to 1).

The saved instruction pointer points to the instruction which is about to be executed when the exception occurred. The x87 instruction pointer register contains the address of the last instruction which caused the exception.

Error Code: The exception does not push an error code. However, exception information is available in the x87 status word register.

Alignment Check

An Alignment Check exception occurs when alignment checking is enabled and an unaligned memory data reference is performed. Alignment checking is only performed in CPL 3.

Alignment checking is disabled by default. To enable it, set the CR0.AM and RFLAGS.AC bits both to 1.

The saved instruction pointer points to the instruction which caused the exception.

SIMD Floating-Point Exception

The SIMD Floating-Point Exception occurs when an unmasked 128-bit media floating-point exception occurs and the CR4.OSXMMEXCPT bit is set to 1. If the OSXMMEXCPT flag is not set, then SIMD floating-point exceptions will cause an Undefined Opcode exception instead of this.

The saved instruction pointer points to the instruction which caused the exception.

Error Code: The exception does not push an error code. However, exception information is available in the MXCSR register.

Traps

Debug

The Debug exception occurs on the following conditions:

- Instruction fetch breakpoint (Fault)
- General detect condition (Fault)
- Data read or write breakpoint (Trap)
- I/O read or write breakpoint (Trap)
- Single-step (Trap)
- Task-switch (Trap)

When the exception is a fault, the saved instruction pointer points to the instruction which caused the exception. When the exception is a trap, the saved instruction pointer points to the instruction after the instruction which caused the exception.

Error code: The Debug exception does not set an error code. However, exception information is provided in the debug registers.

Breakpoint

A Breakpoint exception occurs at the execution of the INT3 instruction. Some debug software replace an instruction by the INT3 instruction. When the breakpoint is trapped, it replaces the INT3 instruction with the original instruction, and decrements the instruction pointer by one.

The saved instruction pointer points to the byte after the INT3 instruction.

Overflow

An Overflow exception is raised when the INTO instruction is executed while the overflow bit in RFLAGS is set to 1.

The saved instruction pointer points to the instruction after the INTO instruction.

Aborts

Double Fault

A Double Fault occurs when an exception is unhandled or when an exception occurs while the CPU is trying to call an exception handler. Normally, two exception at the same time are handled one after another, but in some cases that is not possible. For example, if a page fault occurs, but the exception handler is located in a not-present page, two page faults would occur and neither can be handled. A double fault would occur.

The saved instruction pointer is undefined. A double fault cannot be recovered. The faulting process must be terminated.

In several starting hobby OSes, a double fault is also quite often a misdiagnosed IRQ0 in the cases where the PIC hasn't been reprogrammed yet.

Machine Check

The Machine Check exception is model specific and processor implementations are not required to support it. It uses model-specific registers to provide error information. It is disabled by default. To enable it, set the CR4.MCE bit to 1.

Machine check exceptions occur when the processor detects internal errors, such as bad memory, bus errors, cache errors, etc.

The value of the saved instruction pointer depends on the implementation and the exception.

Triple Fault

Main article: Triple Fault

The Triple Fault is not really an exception, because it does not have an associated vector number. Nonetheless, a triple fault occurs when an exception is generated when attempt to call the double fault exception handler. It results in the processor resetting. See the main article for more information about possible causes and how to avoid them.

Selector Error Code

	Length	Name	Description
E	1 bit	External	When set, the exception originated externally to the processor.
Tbl	2 bits	IDT/GDT/LDT table	This is one of the following values: **0b00** The Selector Index references a descriptor in the GDT. **0b01** The Selector Index references a descriptor in the IDT. **0b10** The Selector Index references a descriptor in the LDT. **0b11** The Selector Index references a descriptor in the IDT.
Index	13 bits	Selector Index	The index in the GDT, IDT or LDT.

Legacy

The following exceptions happen on outdated technology, but are no longer used or should be avoided. They apply mostly to the intel 386 and earlier, and might include CPUs from other manufacturers around the same time.

FPU Error Interrupt

In the old days, the floating point unit was a dedicated chip that could be attached to the processor. It lacked direct wiring of FPU errors to the processor, so instead it used IRQ 13, allowing the CPU to deal with errors at it's own leasure. When the 486 was developed and multiprocessor support was added, the FPU was embedded on die and a global interrupt for FPUs became undesirable, instead getting an option for direct error handling. By default, this method is not enabled at boot for backwards compatibility, but an OS should update the settings accordingly.

Coprocessor Segment Overrun

When the FPU was still external to the processor, it had separate segment checking in protected mode. Since the 486 this is handled by a GPF instead like it already did with non-FPU memory accesses.

See Also

External Links

- Intel® 64 and IA-32 Architectures Software Developer's Manual (http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf) , Volume 3 (System Programming Guide), Chapter 6 (Interrupt and exception handling)