EXCEPTIONS

Control is the most challenging aspect of processor design: it is both the hardest part to get right and the hardest part to make fast. One of the hardest parts of control is implementing exceptions and interrupts—events other than branches or jumps that change the normal flow of instruction execution. An exception is an unexpected event from within the processor; arithmetic overflow is an example of an exception. An interrupt is an event that also causes an unexpected change in control flow but comes from outside of the processor. Interrupts are used by I/O devices to communicate with the processor.

Many architectures and authors do not distinguish between interrupts and exceptions, often using the older name interrupt to refer to both types of events. MIPS convention uses the term exception to refer to any unexpected change in control flow without distinguishing whether the cause is internal or external; we use the term interrupt only when the event is externally caused.

The Intel IA-32 architecture uses the word interrupt for all these events. Interrupts were initially created to handle unexpected events like arithmetic overflow and to signal requests for service from I/O devices. The same basic mechanism was extended to handle internally generated exceptions as well. Here are some examples showing whether the situation is generated internally by the processor or externally generated:

Type of event

I/O device request - External
Invoke the operating system from user program - Internal
Arithmetic overflow - Internal
Using an undefined instruction - Internal
Hardware malfunctions - Either

The operating system knows the reason for the exception by the address at which it is initiated. The addresses are separated by 32 bytes or 8 instructions, and the operating system must record the reason for the exception and may perform some limited processing in this sequence. When the exception is not vectored, a single entry point for all exceptions can be used, and the operating system decodes the status register to find the cause.

For the operating system to handle the exception, it must know the reason for the exception, in addition to the instruction that caused it. There are two main methods used to communicate the reason for an exception. The method used in the MIPS architecture is to include a status register (called the Cause register), which holds a field that indicates the reason for the exception. A second method is to use vectored interrupts. In a vectored interrupt, the address to which control is transferred is determined by the cause of the exception.

The basic action that the processor must perform when an exception occurs is to save the address of the offending instruction in the **exception program counter (EPC)** and then transfer control to the operating system at some specified address.

Interrupt & Exception

Exception Also called interrupt. An unscheduled event that disrupts program execution; used to detect overflow.

Interrupt an exception that comes from outside of the processor. (Some architectures use the term interrupt for all exceptions.)

The Intel x86 uses interrupt. We follow the MIPS convention, using the term exception. External term interrupt only when the event is externally caused. Some examples are given below in table

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

How Exceptions Are Handled in the MIPS Architecture

The basic action that the processor must perform when an exception occurs is to save the address of the offending instruction in the exception program counter (EPC) and then transfer control to the operating system at some specified address.

After performing the necessary action the operating system can either terminate the program or continue with its execution, using the EPC to determine where to restart the execution of the program.

To handle an exception, an operating system should know the reason for the exception and the instruction that caused it. Two methods are used in MIPS architecture to communicate the reason for interrupt.

1. MIPS uses a Status register or cause register which has a field that holds the reason for exception.
2. Vectored interrupts – The operating system knows the reason for the exception by the address at which it is addresses are separated by 32 bytes or eight instructions, and the initiated. The operating system must record the reason for the exception.

Exceptions in a Pipelined Implementation:

A pipelined implementation treats exceptions as a form of control hazard. For example, suppose there is an arithmetic overflow in an add instruction. We must flush the instructions that follow the add instruction from the pipeline and begin fetching instructions from the new address.

The easiest way to do this is to flush the instruction and restart it from the beginning after the exception is handled. The final step is to save the address of the offending instruction in the *exception program counter* (EPC). In reality, we save the address +4, so the exception handling the software routine must first subtract 4 from the saved value.