Exceptions

Isaac Pérez Andrade

ITESM Guadalajara
Department of Engineering & Architecture
Department of Computer Science

February - June 2020

Exceptions & Interrupts

Exceptions & Interrupts

- Exceptions and interrupts are unexpected events that disrupt the normal execution of a program.
- They are not `branch` or `jump` instructions executed in a program.
- They transfer the control of the Central Processing Unit (CPU) to a special program, known as **handler**.
- Exceptions and interruptions are mistakenly used as interchangeable terms. However, they differ from each other.
- The difference between an exception and an interrupt is the source that triggered the change in the flow control.

Exceptions & Interrupts

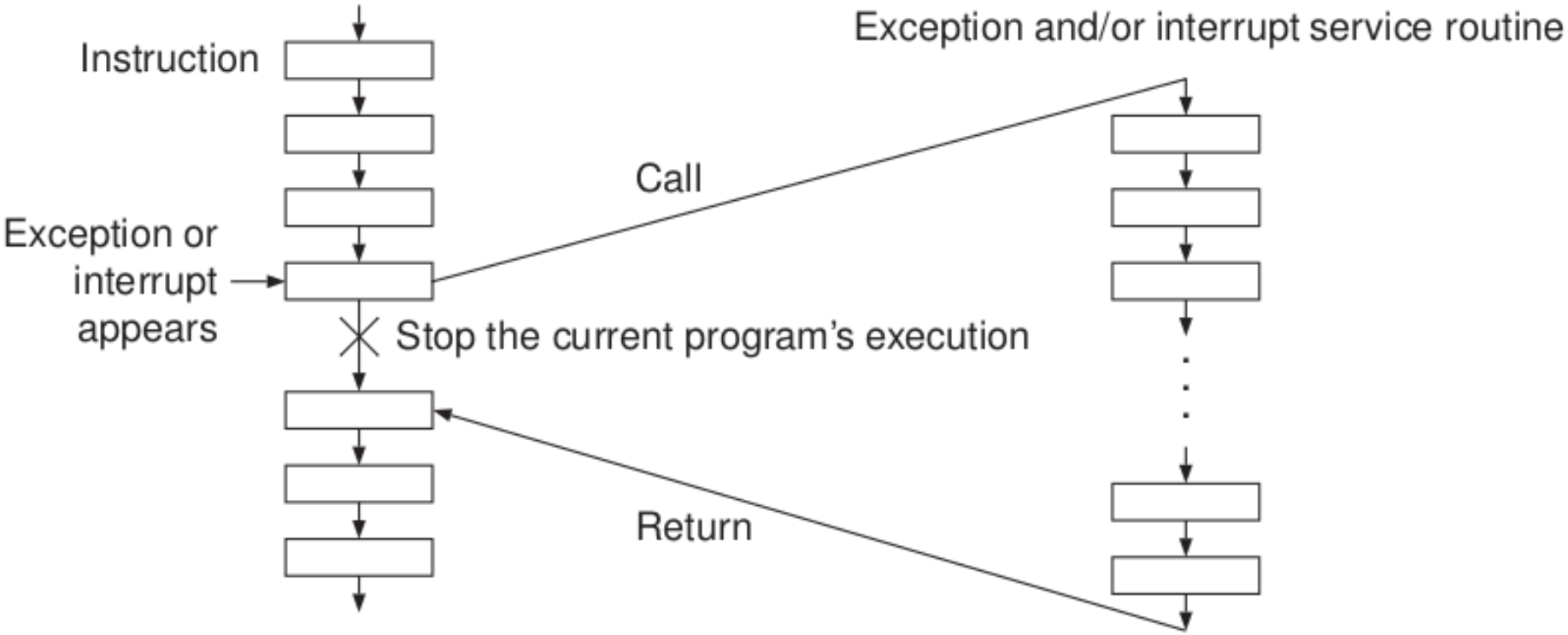


Figure 1: Exceptions and interruptions flow control.

Exceptions & Interrupts

- **Exceptions** are **synchronous** unscheduled events that occur **internally** to the processor.

Exceptions & Interrupts

- **Exceptions** are **synchronous** unscheduled events that occur **internally** to the processor.
  - Arithmetic overflow.
  - Unsupported instruction.

Exceptions & Interrupts

- **Exceptions** are **synchronous** unscheduled events that occur **internally** to the processor.
  - Arithmetic overflow.
  - Unsupported instruction.
- **Interrupts** are **asynchronous** unscheduled events that occur **externally** to the processor.

Exceptions & Interrupts

- **Exceptions** are **synchronous** unscheduled events that occur **internally** to the processor.
  - Arithmetic overflow.
  - Unsupported instruction.
- **Interrupts** are **asynchronous** unscheduled events that occur **externally** to the processor.
  - IO devices such as keyboard or mouse.
  - Timers.

Exceptions & Interrupts

Is the following example an interrupt or an exception?

- Your current Microprocessor without Interlocked Pipeline Stage (MIPS) design.
  - R-Type and I-Type instructions are supported.

Exceptions & Interrupts

Is the following example an interrupt or an exception?

- Your current MIPS design.
  - R-Type and I-Type instructions are supported.
  - What about J-Type instructions?
  - What would happen in your design if the μP receives a `jump` instruction?

Exceptions & Interrupts

- μPs usually rely on a special register for handling exceptions.
- This register is the Exception Program Counter (EPC).
- EPC stores the address of the instruction that caused the exception before transferring flow control to the program handler.
- The handler may then provide service to the offending program with a predefined action.
- If the handler does not terminate the main program execution, it uses the EPC in order to return control to the main program and resume its execution.

Exceptions & Interrupts

- There are two ways to transfer control to the handlers.
  - Polled.
  - Vectored.

Polled exceptions/interrupts

- MIPS Microarchitecture (μA) provides a status register, called **Cause Register**, which holds a field that indicates the reason for the exception/interrupt.

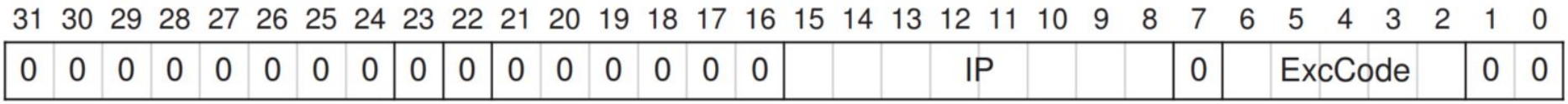


Figure 2: Cause register example.

Exceptions & Interrupts

ExcCode	Mnemonic	Description
0	Int	Interrupt (`IP[7:0]` indicates the interrupt source)
1	Mod	TLB hit on store but memory was not yet modified
2	TLBL	TLB miss on instruction fetch or load
3	TLBS	TLB miss on store
4	AdEL	Address error when fetch or load
5	AdES	Address error when store
6	IBE	Bus error on instruction fetch
7	DBE	Bus error on load or store
8	Sys	Executing system call instruction
9	Bp	Executing break instruction
10	RI	Attempt to execute reserved instruction
11	CpU	Coprocessor is not available
12	Ov	Arithmetic overflow
13	Tr	Executing trap instruction
14	–	Reserved
15	FPE	Floating-point exceptions
16–22	–	Reserved
23	WATCH	Virtual address matches value in Watch register
24	MCheck	TLB multiple match but not consistent
25–29	–	Reserved
30	CacheErr	Cache error
31	–	Reserved

Figure 3: Causes of exceptions in MIPS μA.

Exceptions & Interrupts

- In polled exceptions/interrupts, the control transfer is done in two steps.
  1. Control is transferred to a common entry address.
  2. Control is then transferred to an individual address corresponding to the ExcCode handler.
- In contrast, in **vectored exceptions/interrupts**, the address to which control is transferred is determined by the cause of the exception.
- This is done by hardware by generating a unique vector and an attached address corresponding to the address of the handler.

Exceptions & Interrupts

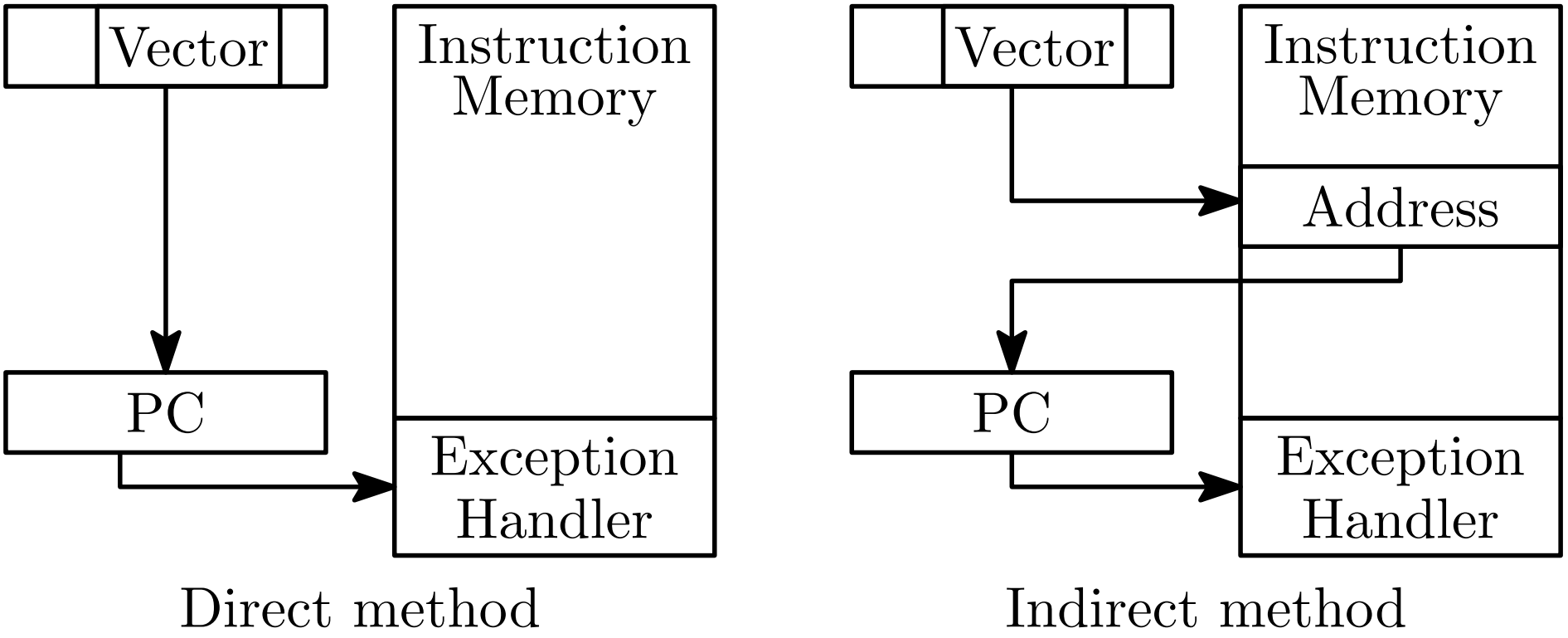


Figure 4: Mechanisms for vectored interrupts.

Exceptions & Interrupts

- How does a handler return the control to the main program?

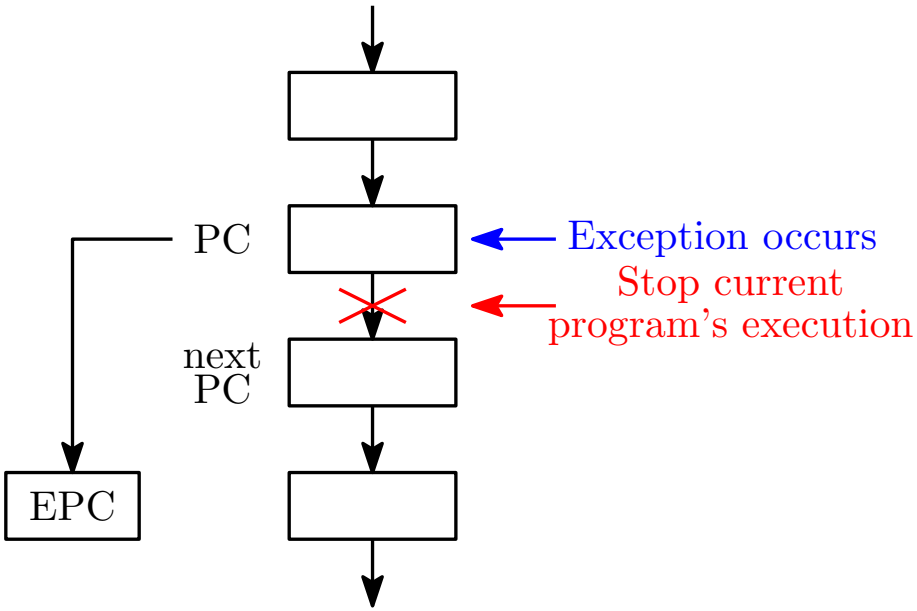
Exceptions & Interrupts

- How does a handler return the control to the main program?
- The return address must be saved prior entering the handler's routine.
- Based on the ISAs, the return address may be saved to
  - A general-purpose register.
  - A special register.
  - Stack memory.

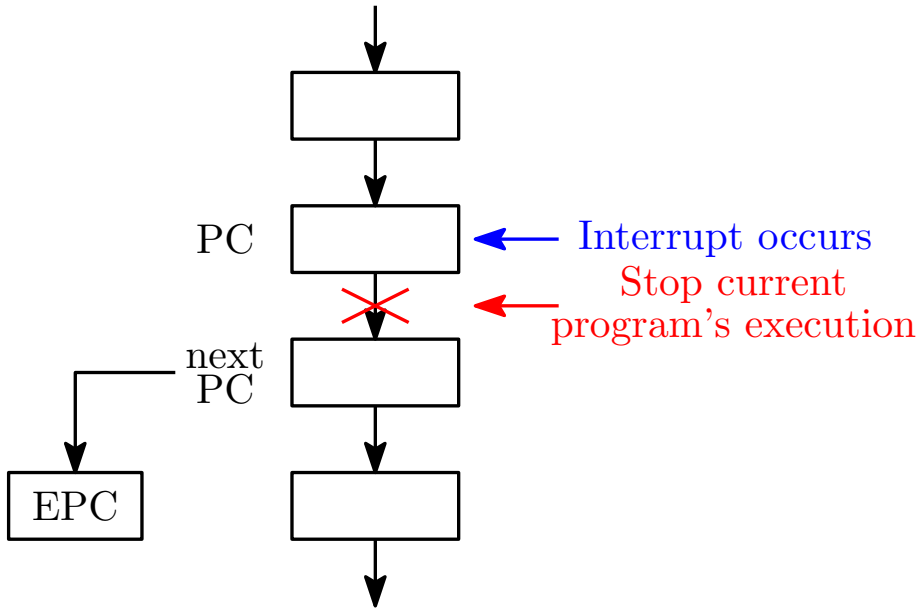
Exceptions & Interrupts

- Exceptions store the current Program Counter (PC) value.
  - This is done in case the offending instructions needs to be executed again.
- Interrupts, on the other hand, store the **next** PC value.
  - Current instruction is completed before transferring control to handler.

Exceptions & Interrupts



Return address for exceptions



Return address for interrupts

Figure 5: Return address in exceptions and interruptions.

Exceptions & Interrupts

- MIPS provides the special instruction `eret` for returning from exception handler.
- This instruction writes into PC the contents of EPC.
- If the offending instruction does not need to be executed again, the return address is incremented before being loaded into PC.

Exceptions & Interrupts

- Interrupts may occur any time, even when the handler is executing an interrupt subroutine.
- These further interrupt request may be enabled or disabled with a special control register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				IM					0	0	0	0	0	0	0	IE

Figure 6: Interrupt masking in Status register.

- The i^{th} bit in `IM` field represents the i^{th} interruption request.
- Bit `IE` is the global interrupt enable.
- When an interrupt is attended, μP may or may not disable (mask) all other `IM` bits, based on whether the μP supports nested interrupts.

Exceptions & Interrupts

- If nested interrupts are supported, some states, including return addresses must be saved to the stack memory.
- Moreover, some interrupts may have priority over others. For example, timers have priority over keyboard inputs.

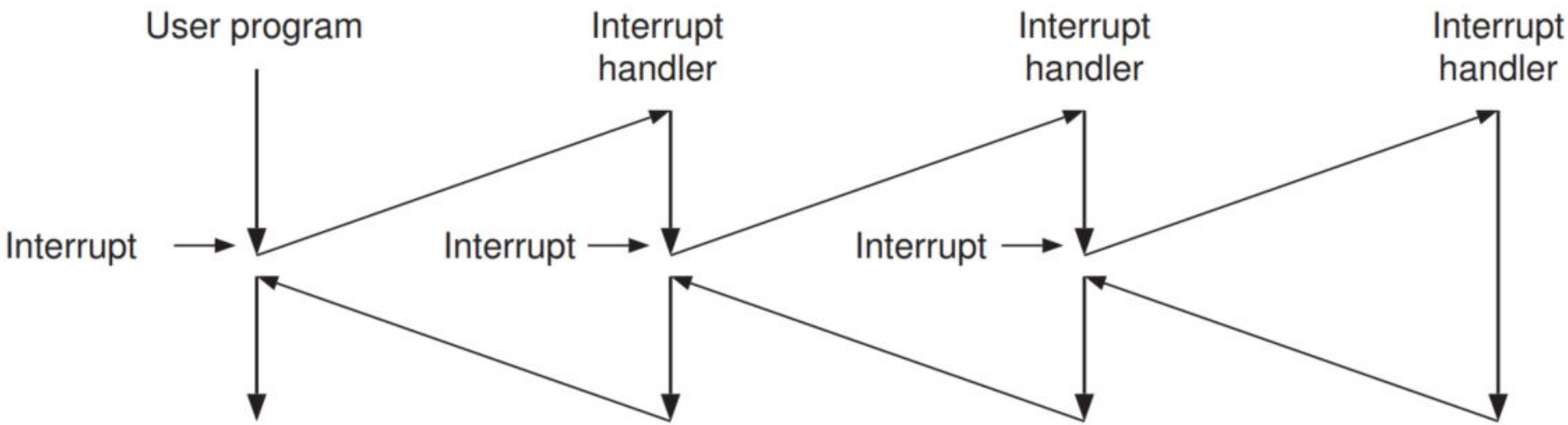


Figure 7: Interrupt nesting.