Computer Organization and Networks

(INB.06000UF, INB.07001UF)

Chapter 13b: Interrupts

Winter 2019/2020

Stefan Mangard, www.iaik.tugraz.at

So Far Everything Was Deterministic

- CPU: Fetch an instruction from memory, decode and execute it.
- Program counter points to next instruction.
- Instruction set defines all possible “actions” of a CPU.
- Some instructions modify the value in the program counter: Branch, CALL, RET,…
- Memory-mapped I/O: some addresses are mapped to “strange places” (a.k.a. “devices”)

Do we always want this continuous execution of instructions?

Remember Chapter 7: Communication via a Slow Communication Interface

- Polling is highly inefficient: the CPU is stuck in a loop until
  - an I/O peripheral sets a ready signal
  - the timer has reached a certain value
  - the user has pressed a key
  - ….
- Alternative
  - CPU keeps executing some useful code in the first place
  - We use concept of interrupts to react to “unexpected” events
  - Basic idea: Instead of waiting for an event, we execute useful code and then let an event trigger a redirection of the instruction stream

How to handle unexpected external events?

- We add an input signal to the CPU called “interrupt”.
- An external source can activate this input signal “interrupt”.
- After executing an instruction, the CPU checks for the value of this input signal “interrupt” before it fetches the next instruction.
- If the signal “interrupt” is active, the next instruction to be executed is the first instruction of the “interrupt-service routine”.
- After “handling” the interrupt by executing the interrupt-service routine, the CPU returns to the interrupted program.

Interrupts in RISC-V

- **Hardware Aspects**
  - External interrupt is an input signal to the processor core
  - Control & Status registers (CSRs) for interrupt configuration (e.g. mie, mtvec, mip, …)
  - Additional instructions for interrupt handling (mret)
  - Dedicated interrupt controllers on bigger processors

- **Software Aspects**
  - When an interrupt occurs, the program execution is interrupted
  - Special functions have to be provided to handle interrupts → Interrupt Service Routines (ISR)
  - Software needs to configure and enable interrupts manually
  - Software has to bridge the gap between the calling convention and preserving the interrupted context
    → Interrupt entry points are typically written in assembler

The Interrupt Service Routine (ISR)

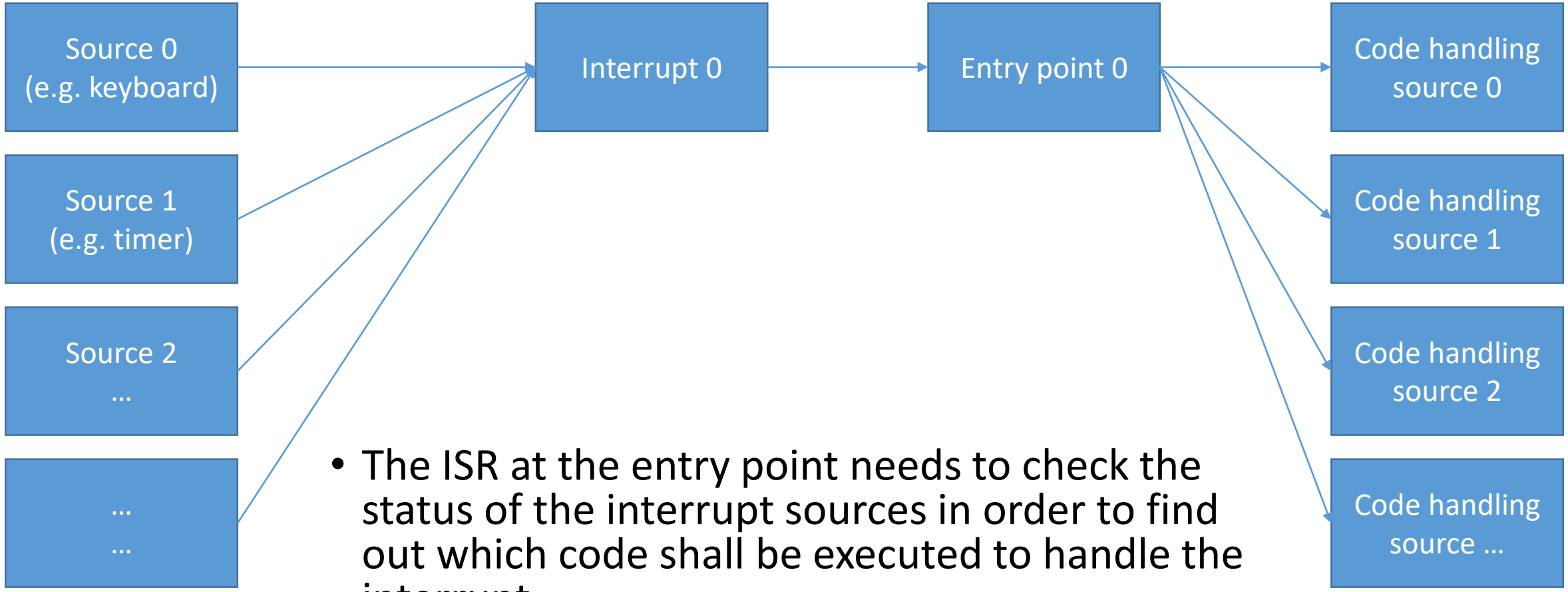
- Entering the ISR
  - Upon an interrupt, the processor
    - jumps to a location in memory specified by the **mtvec** CSR.
    - automatically stores the previous location into **mepc** CSR.
- Executing the ISR
  - The ISR can execute arbitrary code; However, the processor context (program counter, register) needs to have exactly the same values when returning to the interrupted code → “From the view of the interrupted program, the execution after the interrupt continues as if nothing had happened”
- Leaving the ISR
  - Upon the execution of the **mret** instruction, the processor
    - returns to the original location stored in the mepc CSR

Finding the Interrupt Service Routine

- Two approaches are common:
  - Single entrypoint for all interrupts.
    - the ISR has to determine what caused the interrupt and then handles the corresponding interrupt
  - Multiple entrypoints for different interrupts organized in a table (vectored interrupts)
    - A table defines the entry point for different causes of interrupts
- RISC-V permits both approaches
  - Ibex only supports vectored interrupts
  - Each interrupt vector table entry has 4 bytes
    - Interrupt cause 0 leads to a jump to mtvec
    - Interrupt cause 1 leads to a jump to mtvec+4
    - Interrupt cause 2 leads to a jump to mtvec+8
    - …

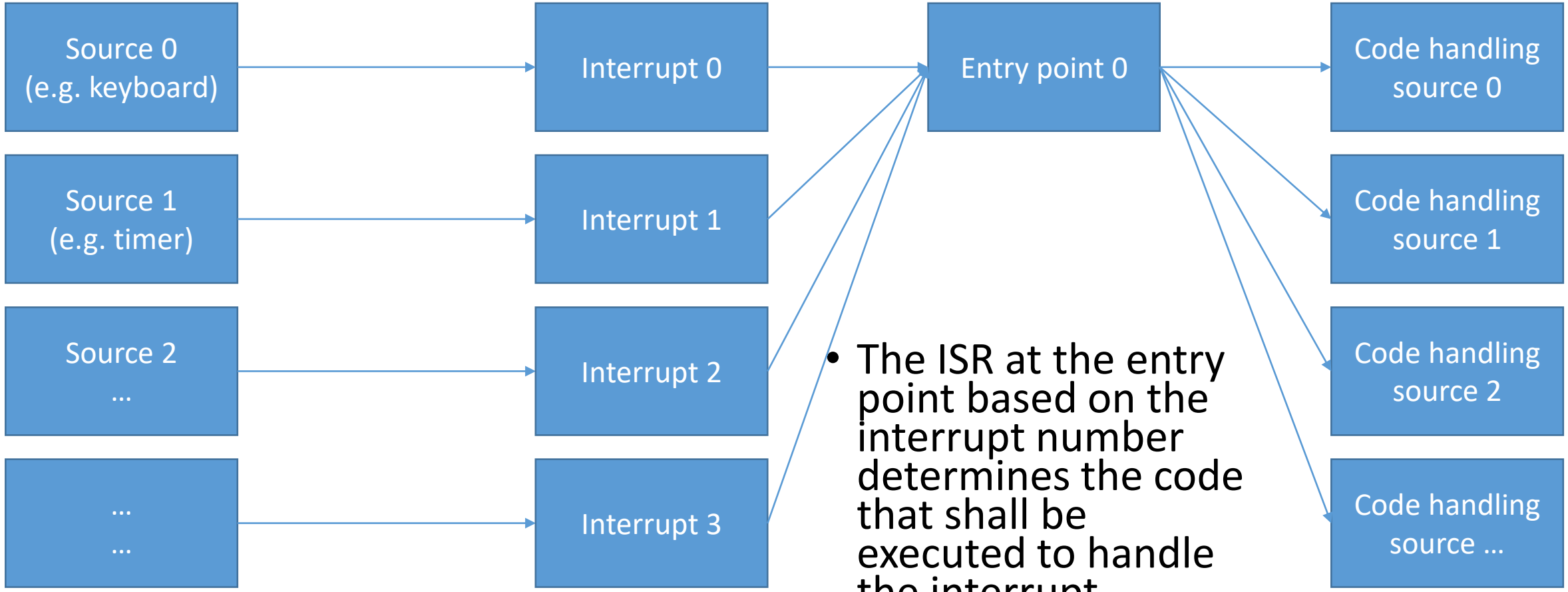
    →just enough space to place a single **jal** instruction to the actual ISR handler code at each entry location

Connecting Interrupt Sources to Interrupt Service Routines



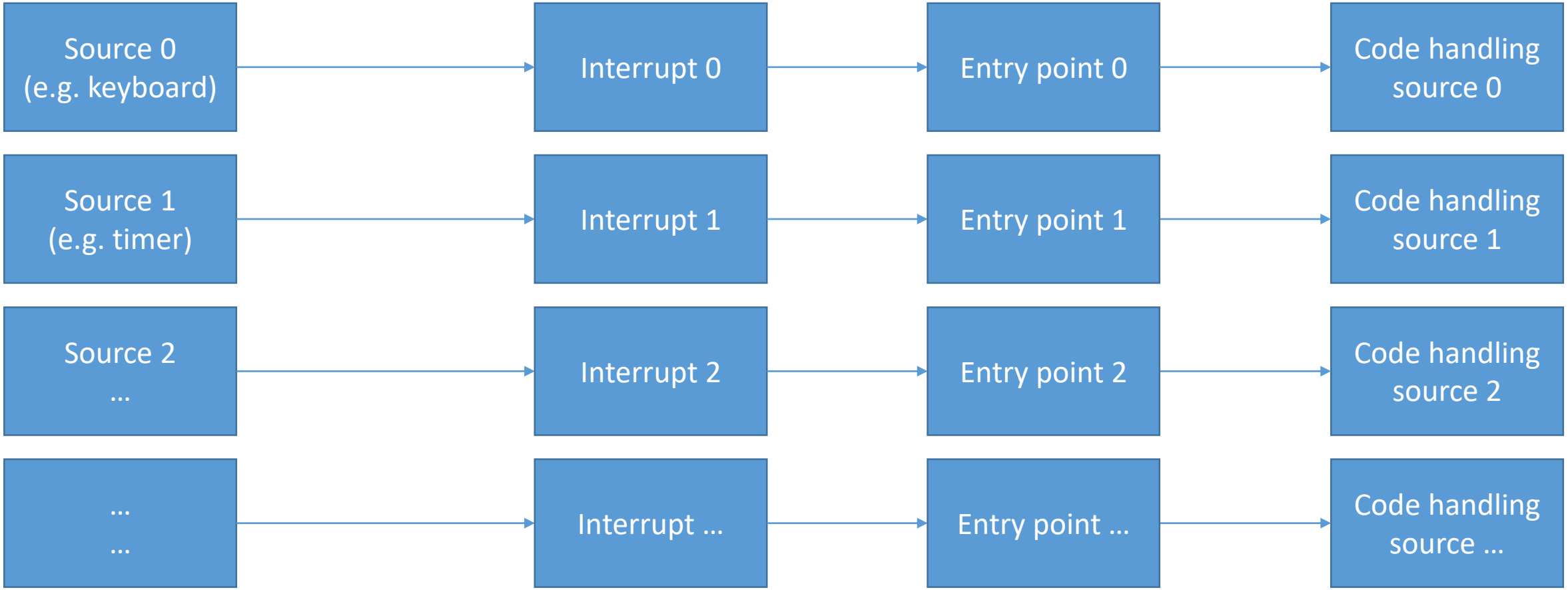
- The ISR at the entry point needs to check the status of the interrupt sources in order to find out which code shall be executed to handle the interrupt

Connecting Interrupt Sources to Interrupt Service Routines



- The ISR at the entry point based on the interrupt number determines the code that shall be executed to handle the interrupt

Connecting Interrupt Sources to Interrupt Service Routines



- Vectored handling with different entry points for different interrupts

Connecting Interrupt Sources to Interrupt Service Routines

- In practice all kinds of combinations are possible for interrupt handling
- There is also the option for having interrupts with different priorities
- Dedicated interrupt controllers are available on larger systems to handle priorities, entry points, nested interrupts, …