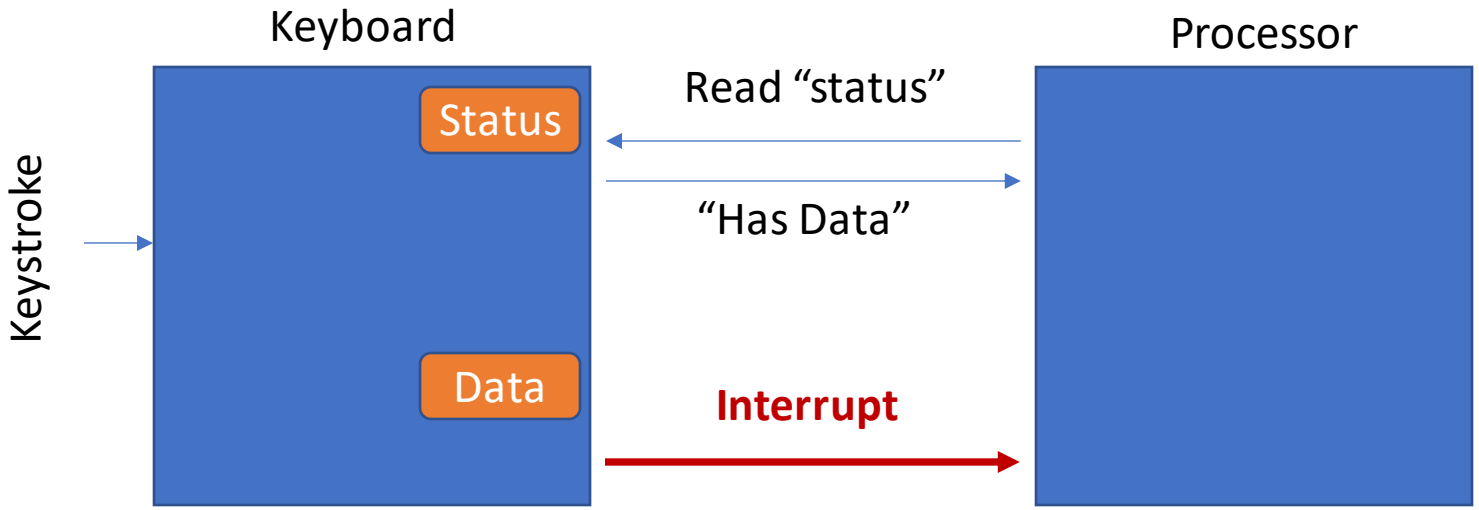
Interrupts

EECS388 Fall 2022

Context

- Recommended reading:
- Chapter 9 of “Introduction to Computing,” Patt, Patel
- Si-five Programmer’s Manual



Polling

- CPU periodically checks the I/O device

```
while (true)
{
  if(IO_REG == 0)
    do "something"
}
```

Interrupt

- The I/O device notifies CPU when it needs attention

```
main( )
{
  Do your common task
}

Int_service_routine() {
    whenever IO_REG is 0 then
    call this function and do
    "something"
}
```

Polling vs. Interrupt

- Polling
  - Waste CPU cycles
  - It is fast
- Interrupt issues
  - Save CPU cycles
  - High interrupt processing overhead
- Hybrid mode?

Interrupt == Notifications for the processor

- Hardware interrupts
  - Devices (timer, disk, keyboard, …) to CPU
- Software interrupts
  - Used for inter-process communication in a multi-core micro-controller
- Exceptions
  - Divide by zero, segmentation fault, …

Interrupt vs. TRAP?

- Interrupts causes the CPU to run an “unexpected” code
  - E.g., interrupting the execution of a word processing application to show a warning
- TRAP requests a service from OS to run an “expected” code
  - E.g., read from keyboard while running a word processing application

- But, the way that CPU handles interrupts has similarities with handling of TRAP

There are two parts to interrupt-driven I/O

1. Causing the Interrupt to occur

2. Handling the interrupt request

Part 1: Causing the Interrupt to Occur

Conditions for Having an Interrupt

1. I/O device requires attention
2. I/O device has the right to request the service
3. Servicing the I/O device is more urgent to the current operation of the processor

I/O device needs to be serviced

Examples:

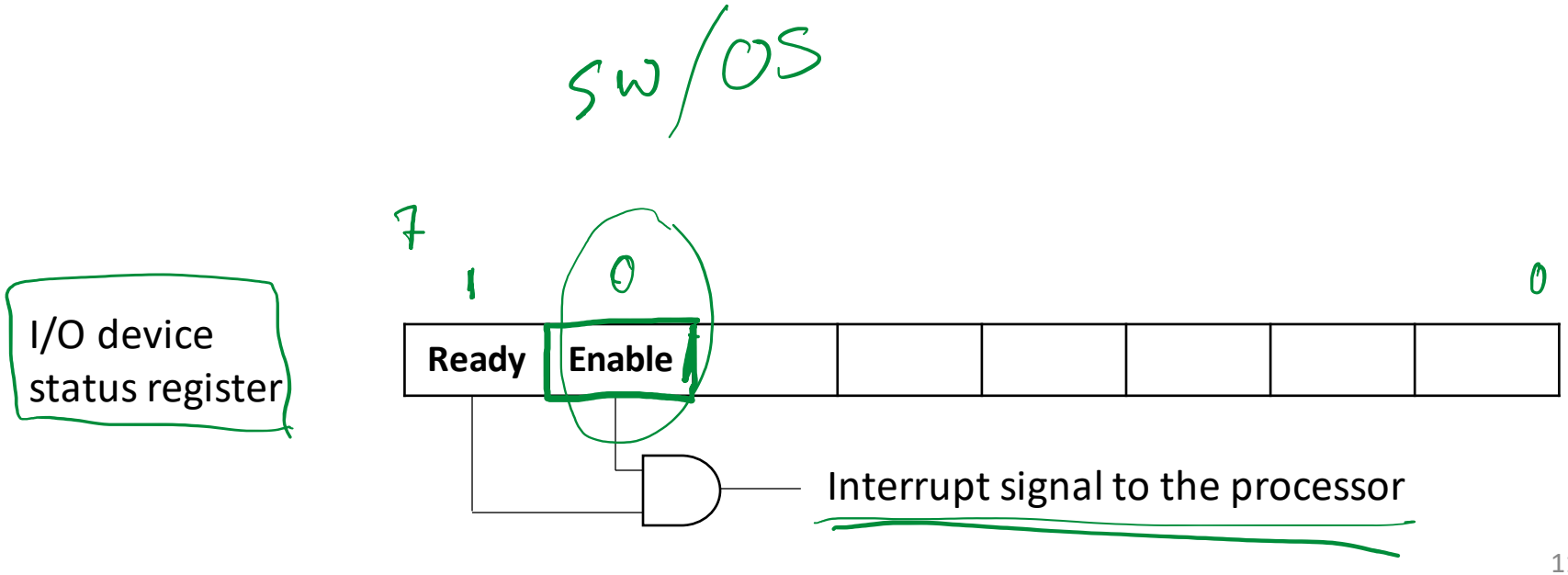
- An ADC finished analog to digital conversion
- A keystroke in a keyboard

A bit is set in a device register

I/O device status register

I/O device has the right to request the service

There is an interrupt enable bit in each I/O device that can be set/reset by the processor

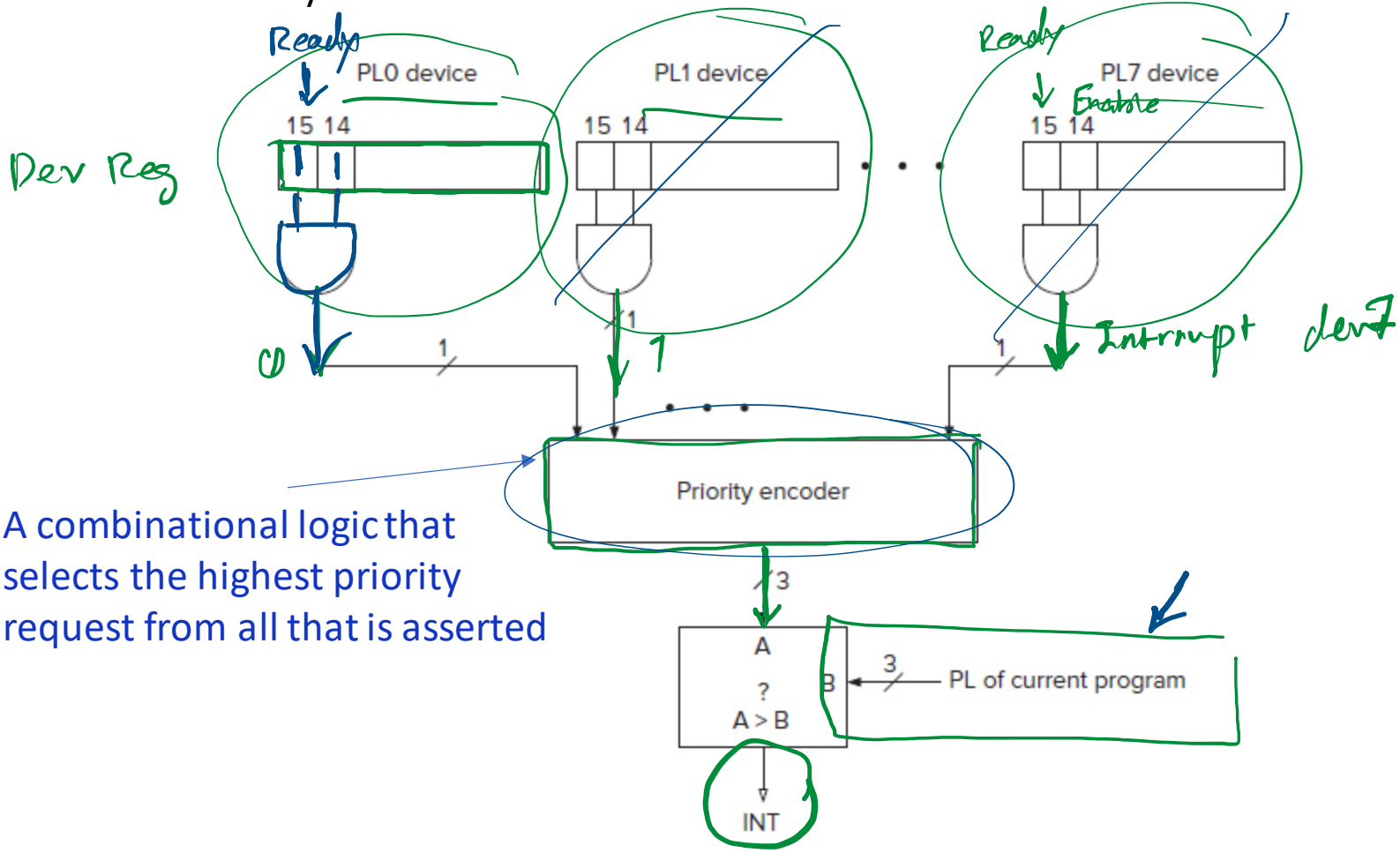


The Urgency of the Interrupt

- Each program runs at a specific priority level
- For an interrupt to go through, the interrupt should have a priority level higher than the program running on the processor
  - We do not want a radio volume control to interrupt an image processing application in an autonomous vehicle!

Generation of an Interrupt Signal

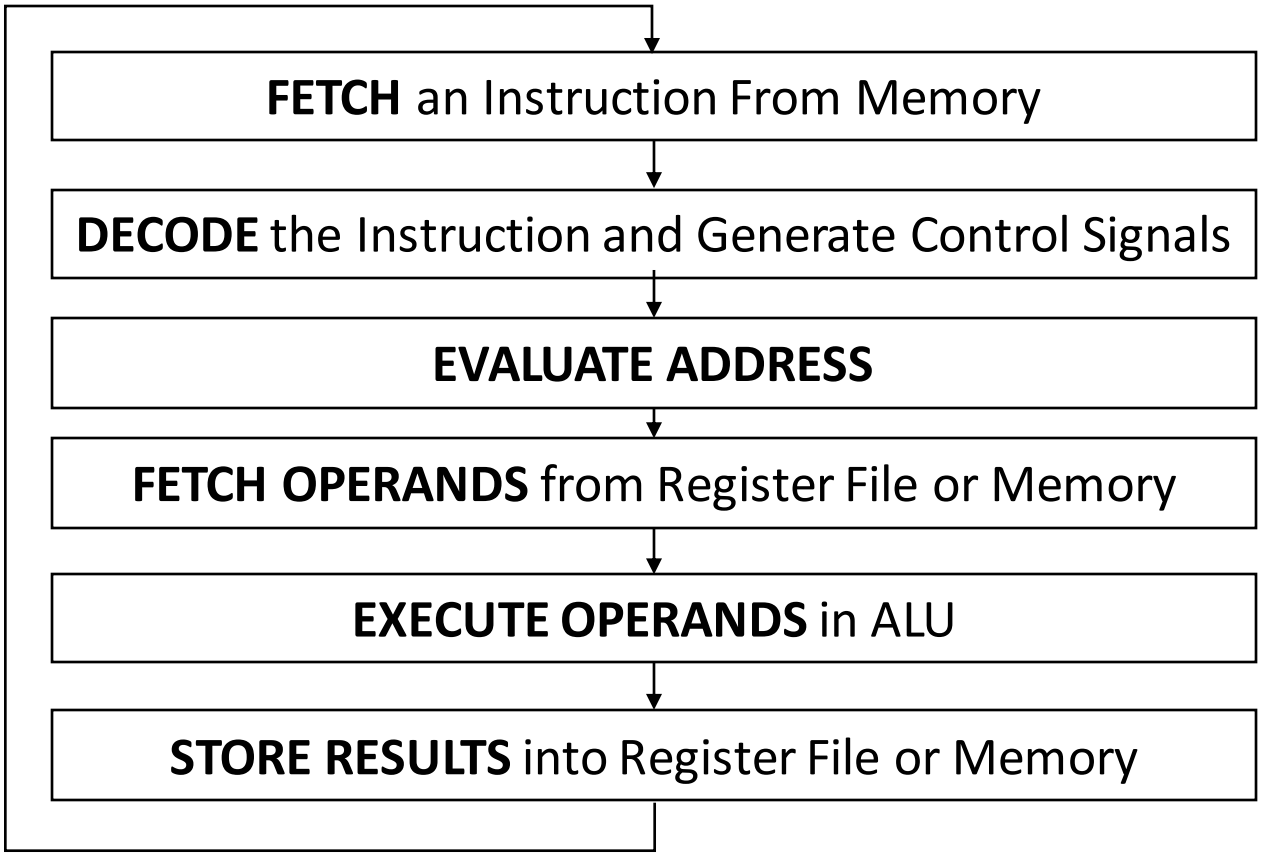
PL = Priority level



A combinational logic that selects the highest priority request from all that is asserted

source: Patel, S., and Yale Patt. *Introduction to Computing Systems: from bits & gates to C & beyond*

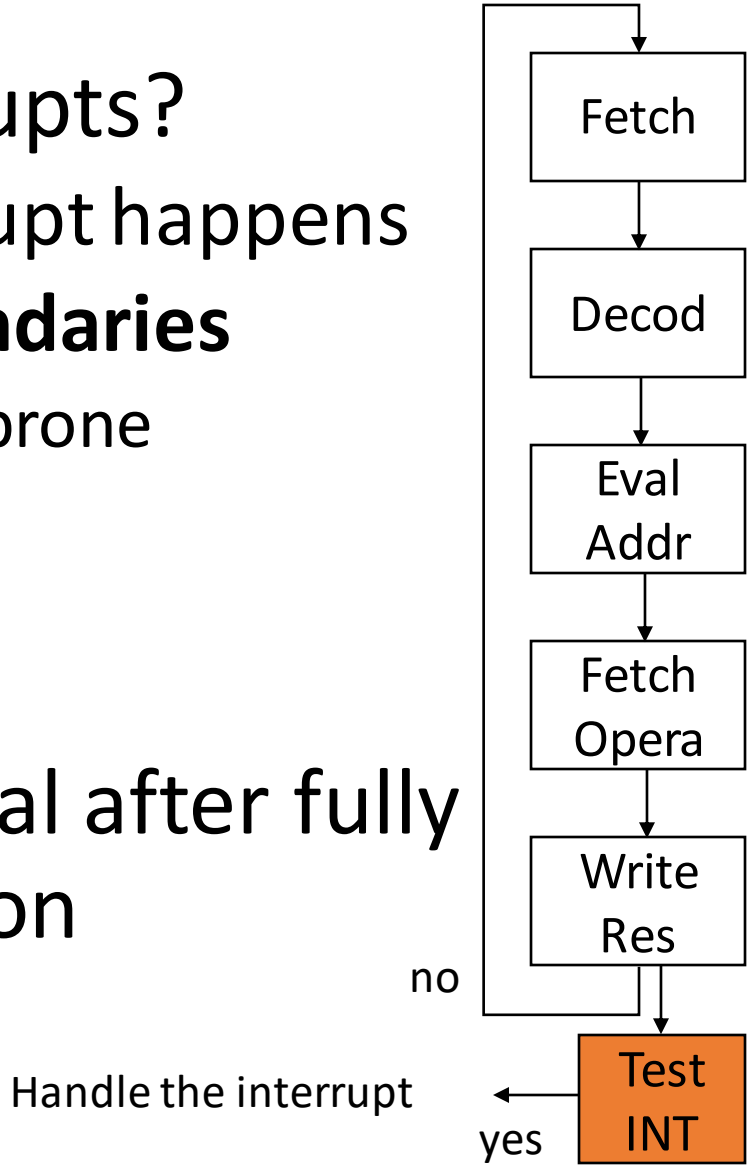
Interrupts can happen at anytime!



Instruction Processing Phases

- When to handle interrupts?
  - Anytime that an interrupt happens
  - **In the instruction boundaries**
    - Simpler and less error prone

Check the Interrupt signal after fully executing each instruction



Part 2: Handling the Interrupt Request

Handing an interrupt goes through 3 stages:

1. Initiate the interrupt
2. Service the interrupt
3. Return from the interrupt

```
  Program A is executing instruction n
  Program A is executing instruction n+1
  Program A is executing instruction n+2
1: Interrupt signal is detected
1: Program A is put into suspended animation
1: PC is loaded with the starting address of Program B
2: Program B starts satisying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B finishes satisfying I/O device's needs
3: Program A is brought back to life

  Program A is executing instruction n+3
  Program A is executing instruction n+4
                      .
                      .
                      .
```

source: Patel, S., and Yale Patt. *Introduction to Computing Systems: from bits & gates to C & beyond*

1- Initiate the Interrupt

The processor does two things:

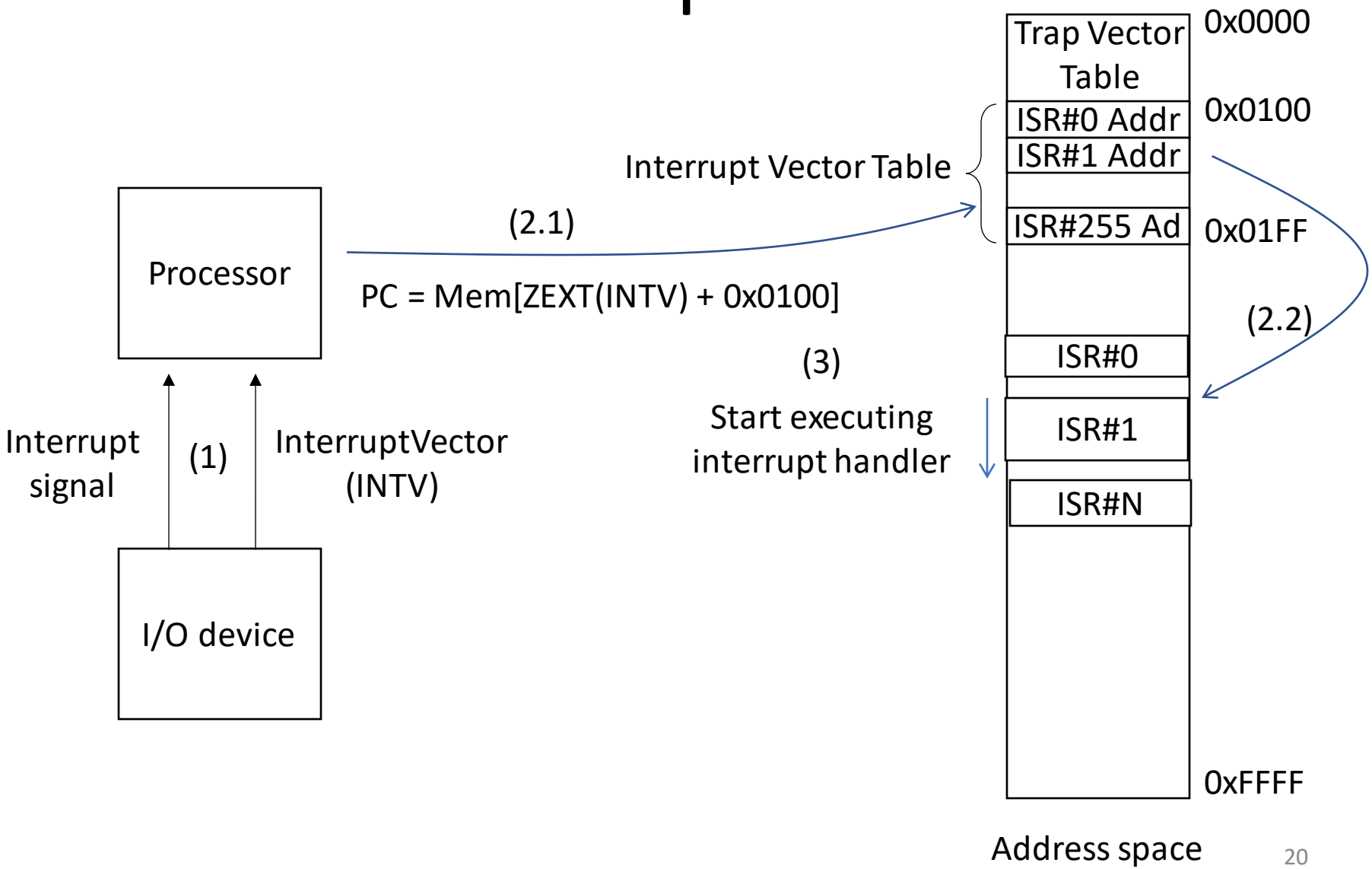
- Save the state of the interrupted (current) program
  - So that we can restart the program once we handled the interrupt
    - Save PC and PSR in supervisor stack (like TRAP)
  - Do we need to save the content of the register file?
- Load the state of the interrupting program

Vectored Interrupts

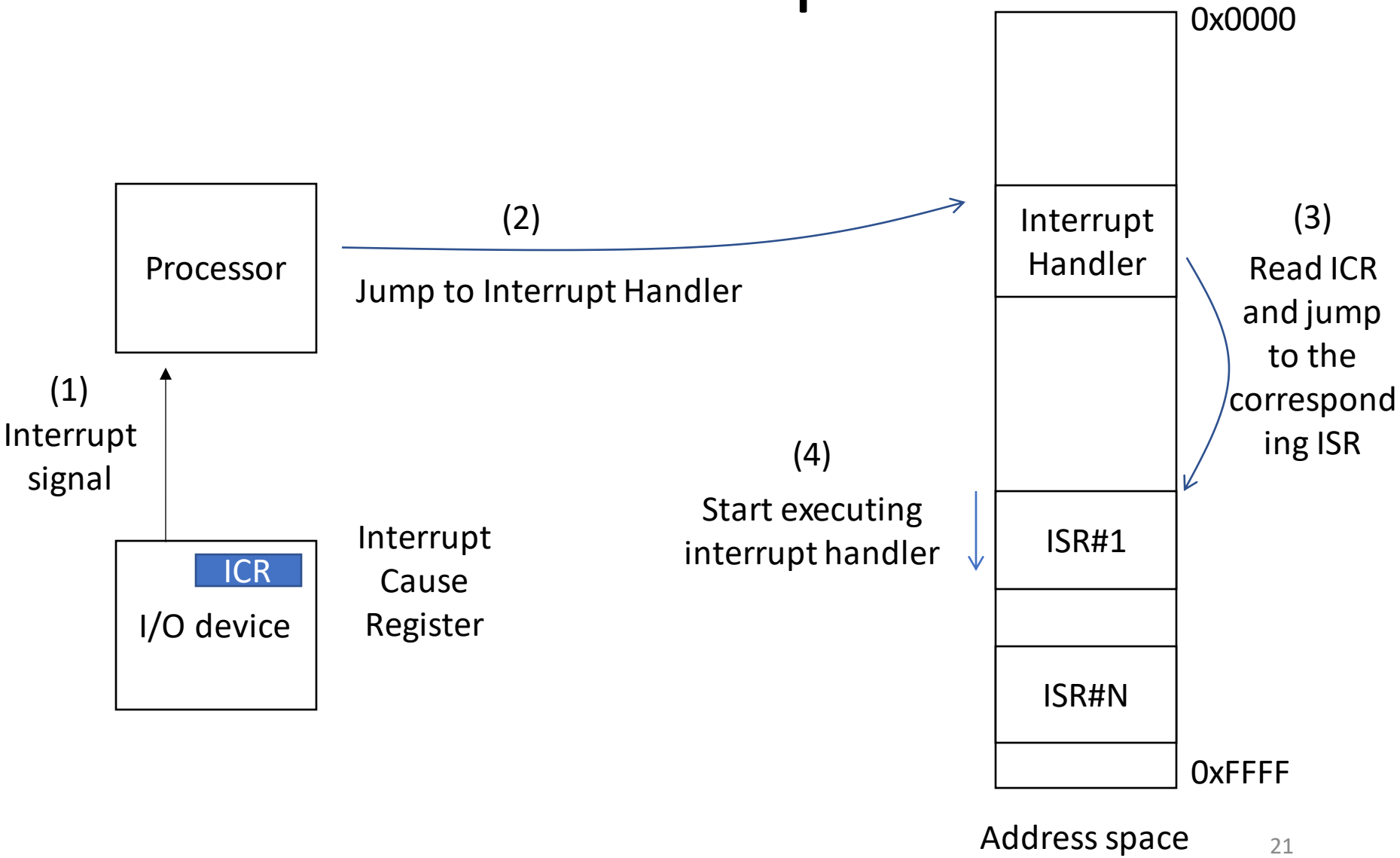
- An ***interrupt vector*** is provided to the processor by the interrupting device
- The *interrupt vector* is use by the processor to reference an entry in the ***Interrupt Vector Table***
- *Interrupt Vector Table* consists of memory locations each containing the starting address of an ***Interrupt Service Routine***
- *Interrupt Service Routines* are program fragments stored in memory that service interrupt requests

Can you see the similarities with TRAP instruction processing?

Vectored Interrupts



Direct Mode Interrupts



Handing an interrupt goes through 3 stages:

1. Initiate the interrupt
2. **Service the interrupt**
3. Return from the interrupt

```
  Program A is executing instruction n
  Program A is executing instruction n+1
  Program A is executing instruction n+2
1: Interrupt signal is detected
1: Program A is put into suspended animation
1: PC is loaded with the starting address of Program B
2: Program B starts satisying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B finishes satisfying I/O device's needs
3: Program A is brought back to life

  Program A is executing instruction n+3
  Program A is executing instruction n+4
                    .
                    .
                    .
```

source: Patel, S., and Yale Patt. *Introduction to Computing Systems: from bits & gates to C & beyond*

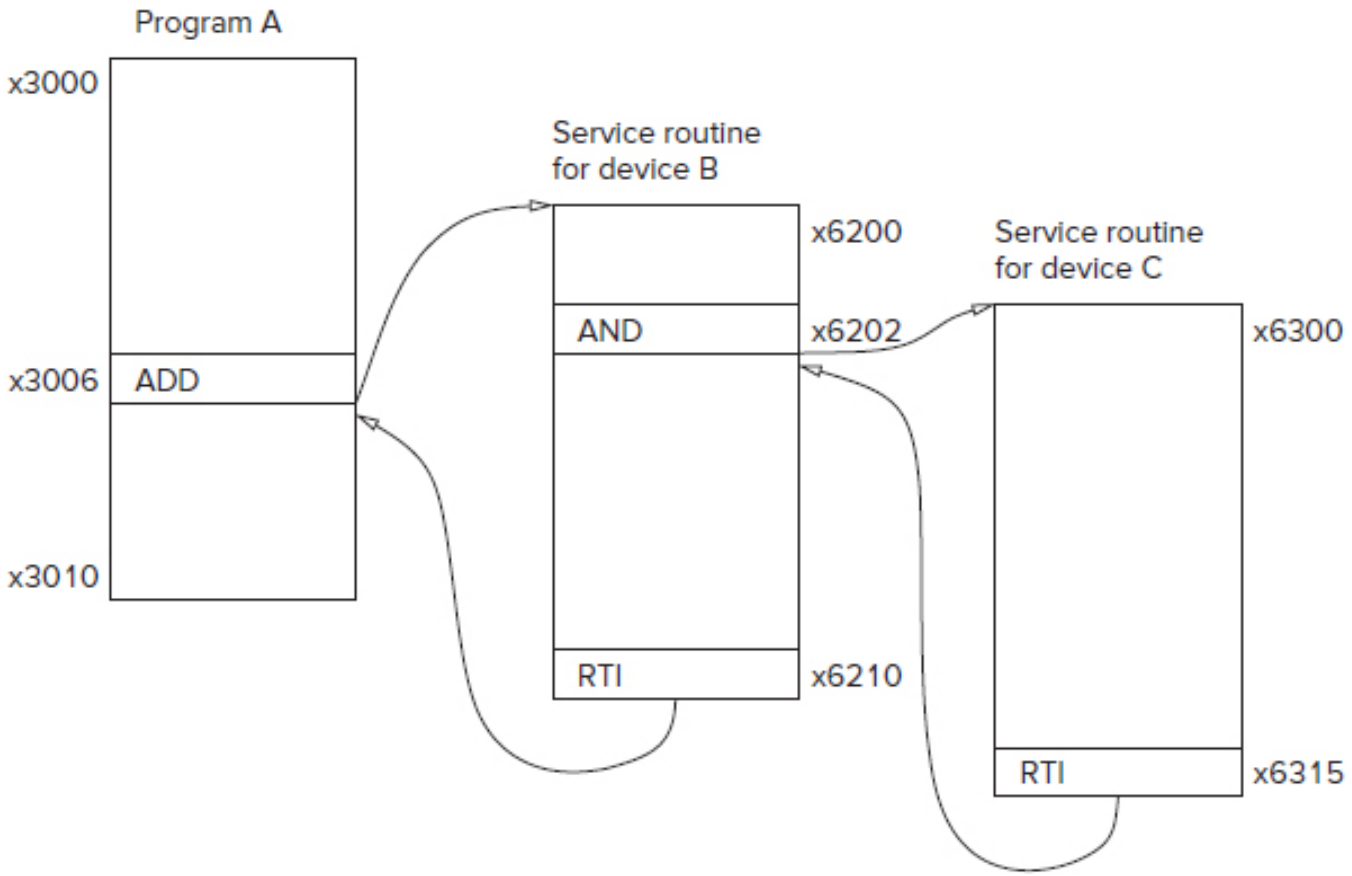
Handing an interrupt goes through 3 stages:

1. Initiate the interrupt
2. Service the interrupt
3. **Return from the interrupt**

RTI instruction

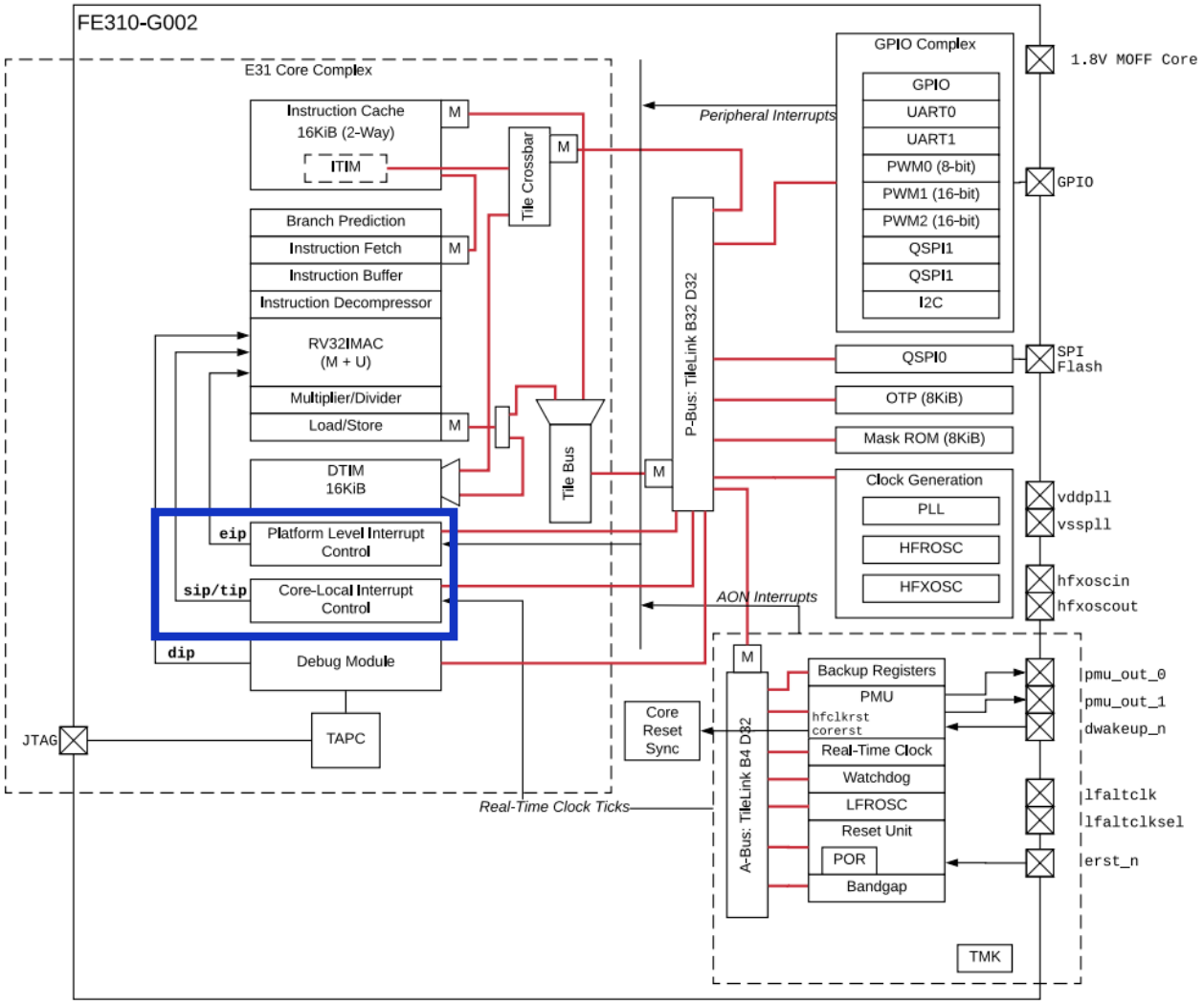
- Restoring the state of the interrupted program and bring it back to execution
  - Pop PC and PSR from the supervisor stack

Example: execution flow for Interrupt-driven I/O



Interrupts in Real World Processors (Not LC-3)

Case Study: Si-Five Interrupts



Source: SiFive FE310-G002 Manual

Si-Five Interrupt Architecture

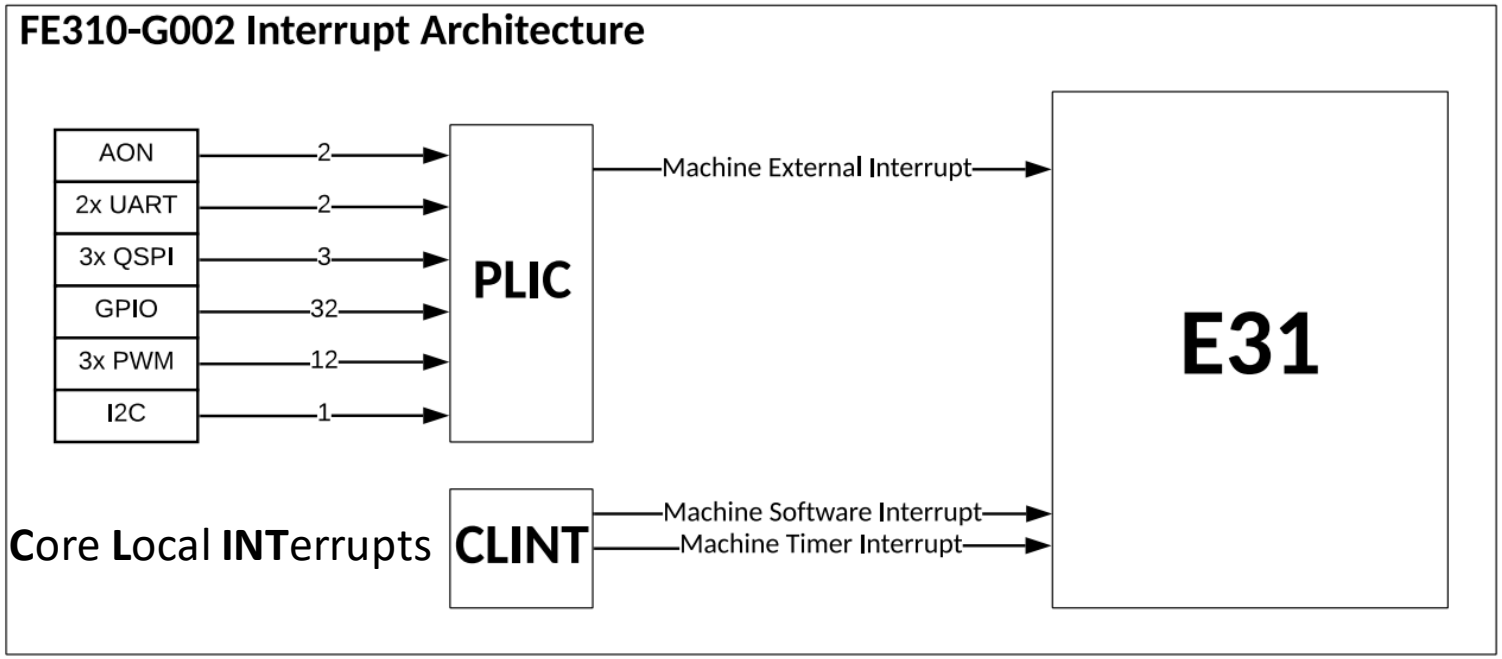


Figure 4: FE310-G002 Interrupt Architecture Block Diagram.

Source: Si-five Programmer's Manual

Control and Status Registers (CSRs)

- Privilege registers for software/hardware communication
- Use special instructions to read/write

hart = HARdware **T**hread

Interrupt related CSRs:

`mstatus`: control and track hart's current status (global interrupt enable/disable)

`mtvec`: base address of the interrupt vector

`mie`: enable/disable individual interrupts

`mip`: which interrupts are currently pending

`mcause`: identify the cause of the interrupt

Memory-Map Registers for CLINT

- MSIP is used for inter-process communication
- Trigger timer interrupt when `mtime` > `mtimecmp`

Address	Width	Attr.	Description	Notes
0x2000000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x2004008			Reserved	
…				
0x200bff7				
0x2004000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x2004008			Reserved	
…				
0x200bff7				
0x200bff8	8B	RW	mtime	Timer Register
0x200c000			Reserved	

Table 24: CLINT Register Map