Interrupt

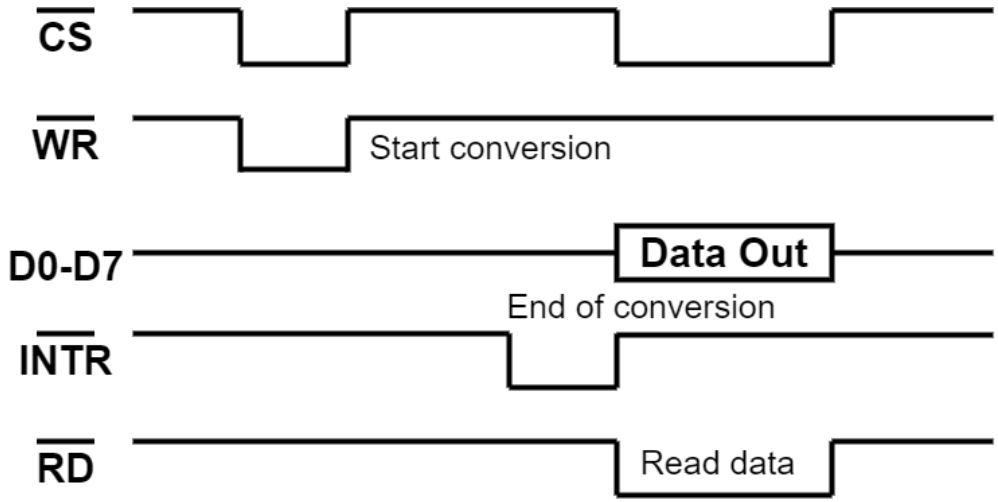
EECS 388 – Spring 2023

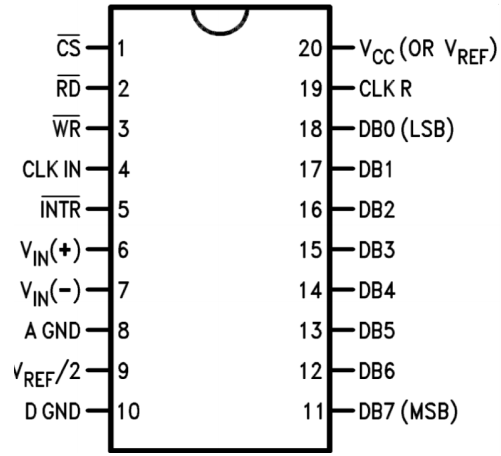
© Prof. Tamzidul Hoque

Lecture notes based in part on slides created by Mohammad Alian and Heechul Yun

Context from last lecture

- We saw that Interrupt was sent by ADC to CPU when data conversion is done and digital output is read.
- But interrupt is not the only way an external device can communicate with CPU





CS: asserted by from uP

RD: asserted by uP

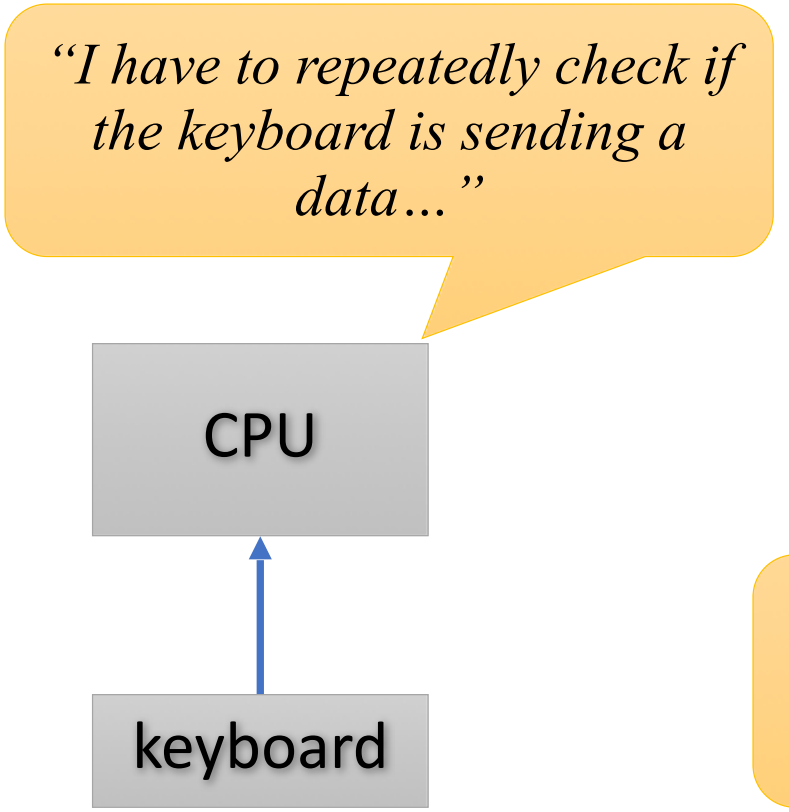
WR: asserted by uP

INTR: sent to uP

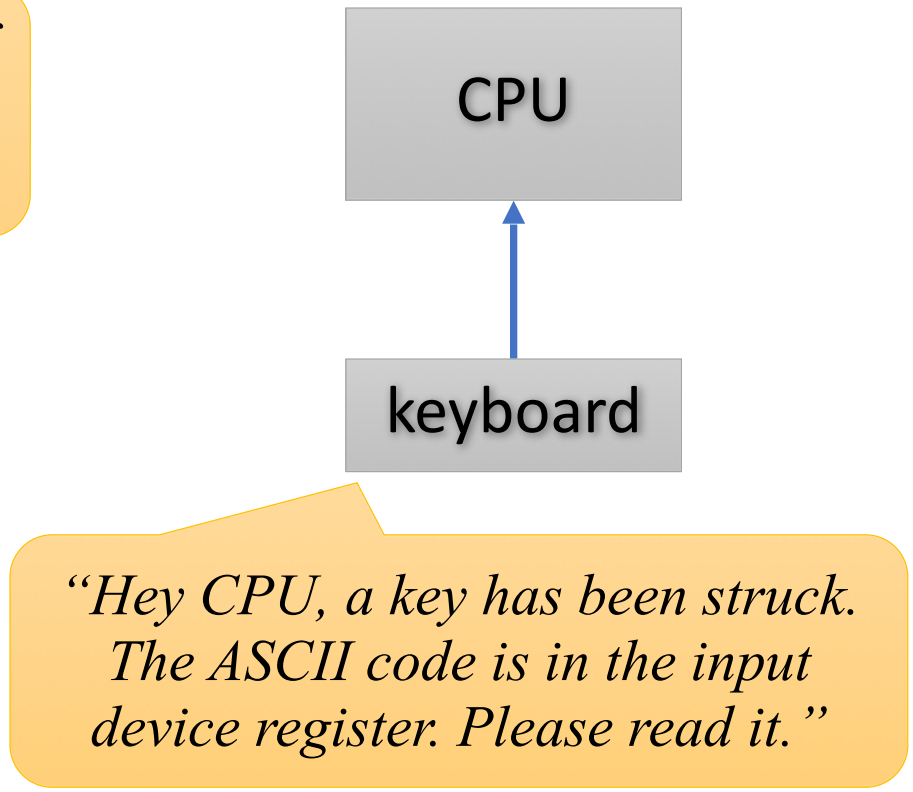
2

Interrupt-Driven vs. Polling

Polling: *The CPU checks the keyboard repeatedly if it wants to send data*



Interrupt: *The IO device interrupts when its ready to send data*



Interrupt-Driven vs. Polling

Polling: *The CPU checks the keyboard repeatedly if it wants to send data*

Example: Staring at the clock all night to so that we can have breakfast at 7:00 AM

Interrupt: *The IO device interrupts the CPU when its ready to send data*

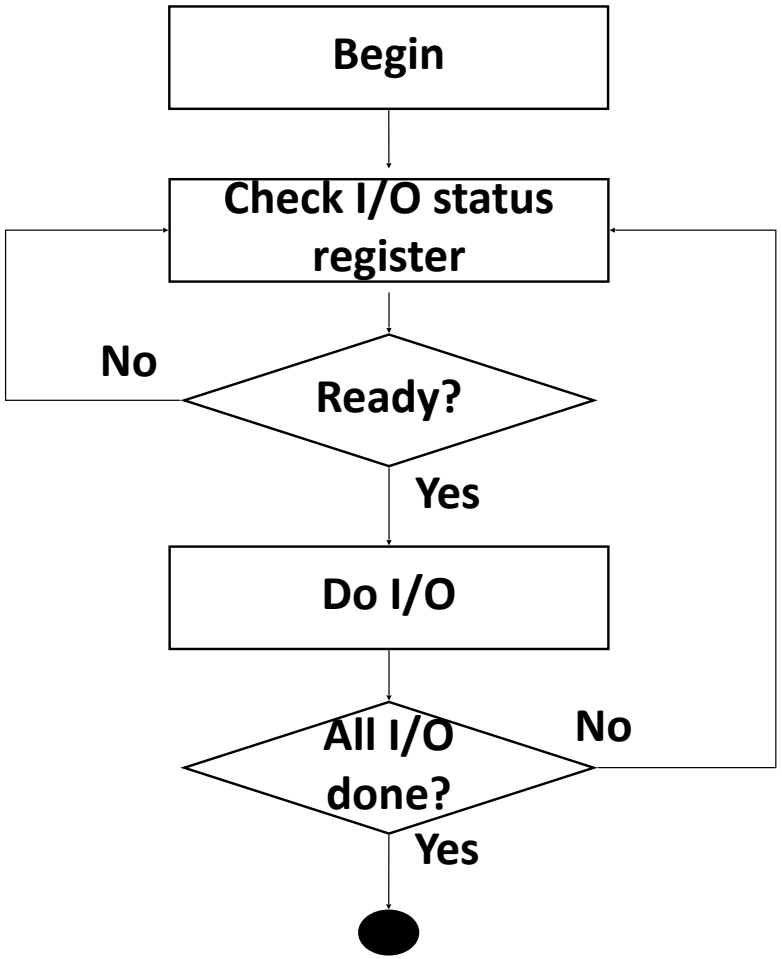
Example: Setting up an alarm for 7:00 AM to interrupt the sleep

Polling Example with Keyboard

- Two I/O registers are needed to facilitate polling for an input device like keyboard:
  - **Status register:** indicates a key struck
  - **Data register:** stores the input data from keyboard

- CPU regularly checks the status register
- If the specific bit of the status register is set:
  - CPU reads from the data register

Polling


Begin
Check I/O status register
No
Ready?
Yes
Do I/O
All I/O done?
No
Yes

6

Polling

- The problem
  - CPU can't do any useful things in the meantime.
- Improvements
  - Check multiple I/O devices rather than one
  - Instead of keep checking the status (busy waiting), do some other useful work and then check again
    - Q. when to check again?
    - Q. what if the I/O is latency critical?
    - The CPU may miss the I/O when its ready and check when its not ready
- Can we do I/O immediately when it is ready?
  - Solution: Interrupt

https://www.youtube.com/watch?v=jP1JymlHUtc&ab_channel=QuantumLeaps%2CLLC

7

Interrupt

- What is an 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, …
  - an unexpected event from within the process

8

Interrupt Operation

- I/O device that may or may not have anything to do with the program that is running can

(1) force the running program to stop,

(2) have the processor execute a program that carries out the needs of the I/O device, and then

(3) have the stopped program resume execution as if nothing had happened.

9

Interrupt Operation

- Let us assume that program A is running on the CPU
- What happens when an interrupt occurs after instruction n+2 occurs?

```
.
Program A is executing instruction n
Program A is executing instruction n+1
Program A is executing instruction n+2
Program A is executing instruction n+3
Program A is executing instruction n+4
.
```


10

Interrupt Operation

(1) force the running program to stop,

(2) have the processor execute a program that carries out the needs of the I/O device, and then

(3) have the stopped program resume execution as if nothing had happened.

```
                    .
  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
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
                    .
                    .
```

- *As far as Program A is concerned, the work carried out and the results computed are no different from what would have been the case if the interrupt had never happened;*

11

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

13

Conditions for Having an Interrupt

- Several things must be true for an I/O device to actually interrupt the program that is running:
  1. The I/O device **must want** service.
  2. The device must **have the right** to request the service.
  3. The device request **must be more urgent** than what the processor is currently doing.

- If all three elements are present, the processor stops executing the program that is running and takes care of the interrupt.

14

I/O device must want service

Examples:

- An ADC finished analog to digital conversion
- A UART has received a byte
- A keystroke in a keyboard

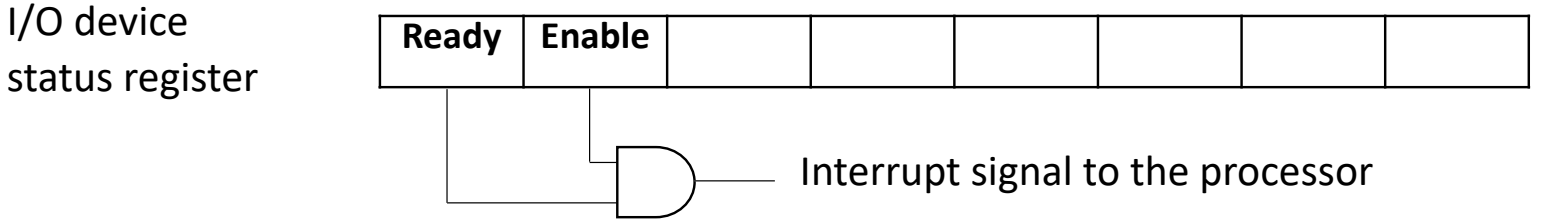
Just like polling, to request a service a bit is set in a IO device register

I/O device status register

15

I/O device has the right to request the service

- There is an **interrupt enable** bit in each I/O device
- Can be set/reset by the processor depending on whether or not the processor wants to give the I/O device the right to request service.
- In most I/O devices, this interrupt enable (IE) bit is part of the device status register.



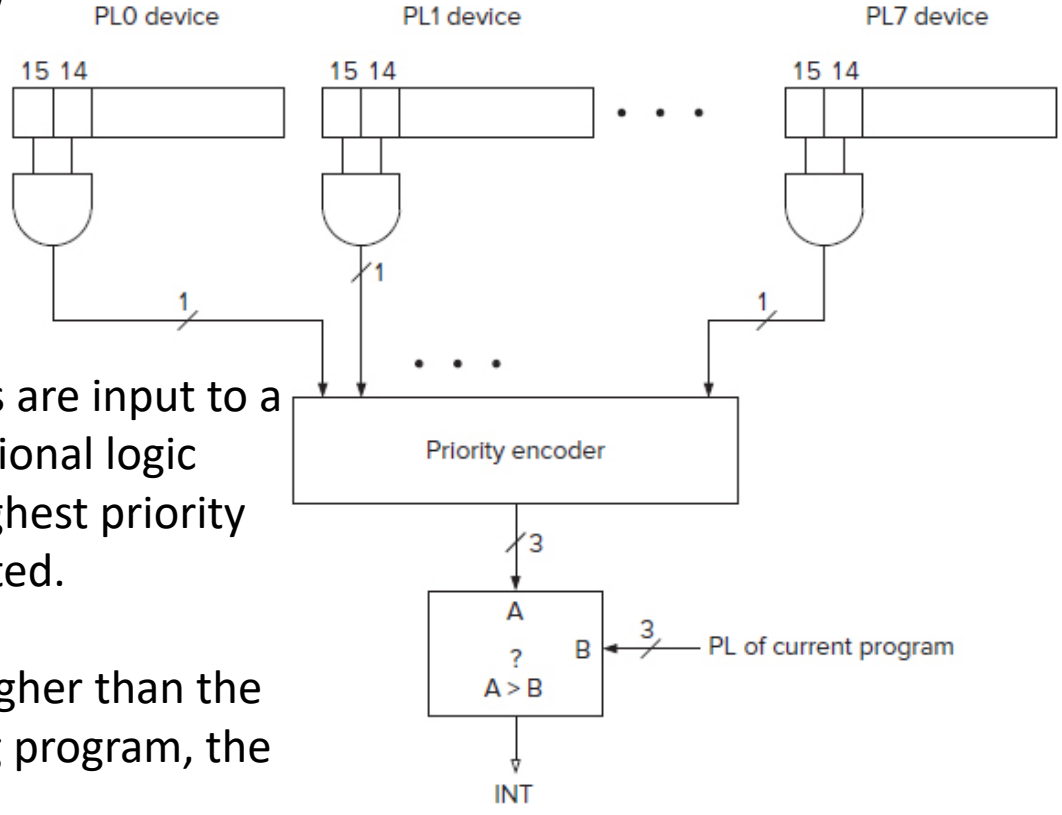
The Urgency of the Interrupt

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

17

Generation of an Interrupt Signal

- Figure shows the status registers of several devices operating at various priority levels (PL).
- Any device that has bits [14] and [15] both set asserts its interrupt request signal. (ready and enable bit)

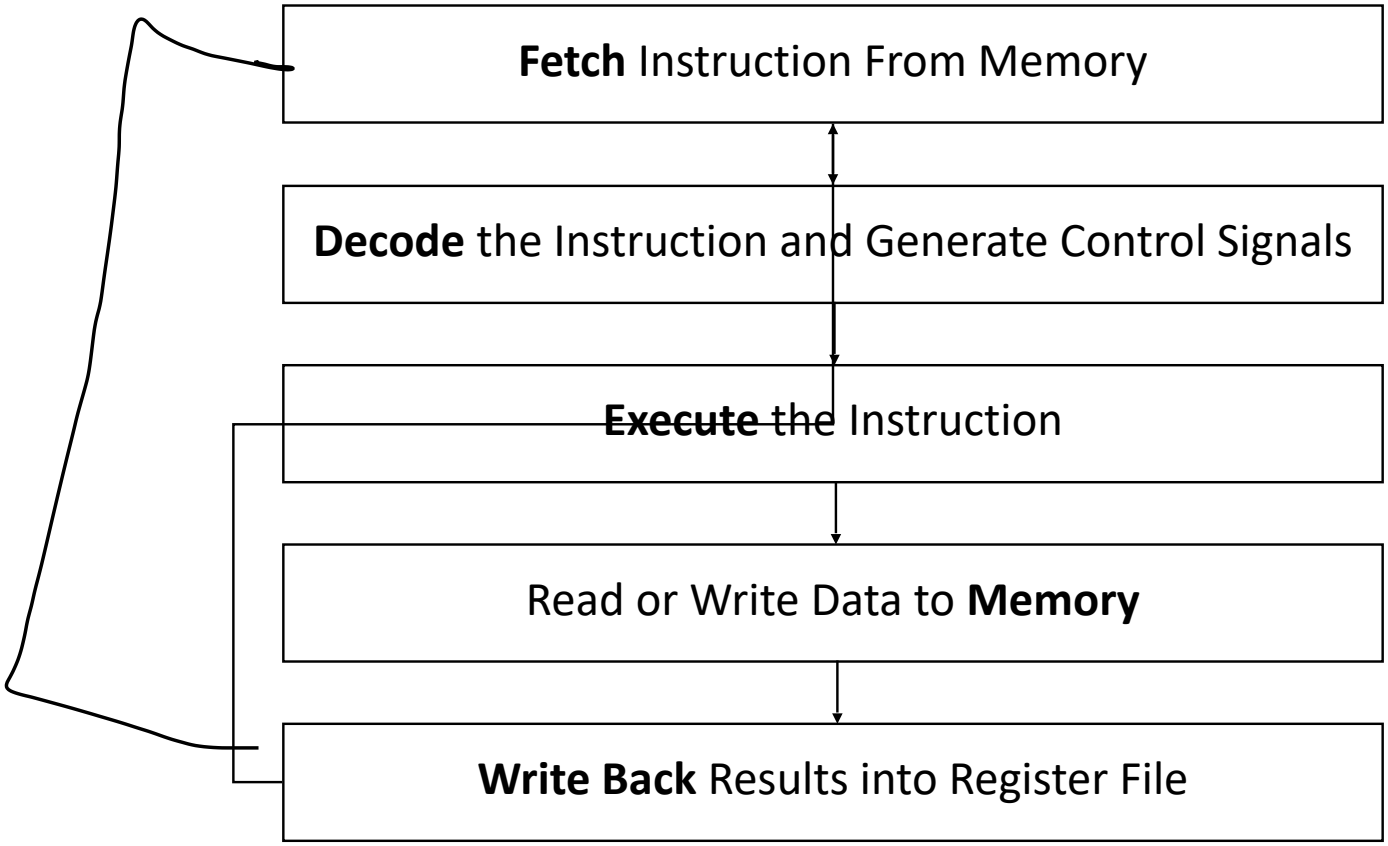


- The interrupt request signals are input to a priority encoder, a combinational logic structure that selects the highest priority request from all those asserted.

- If the PL of that request is higher than the PL of the currently executing program, the INT signal is asserted.

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

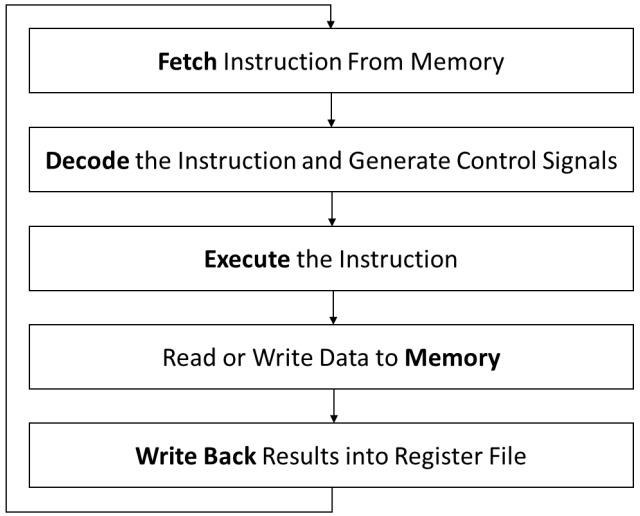
Interrupts can happen at anytime!



19

Interrupt Signal

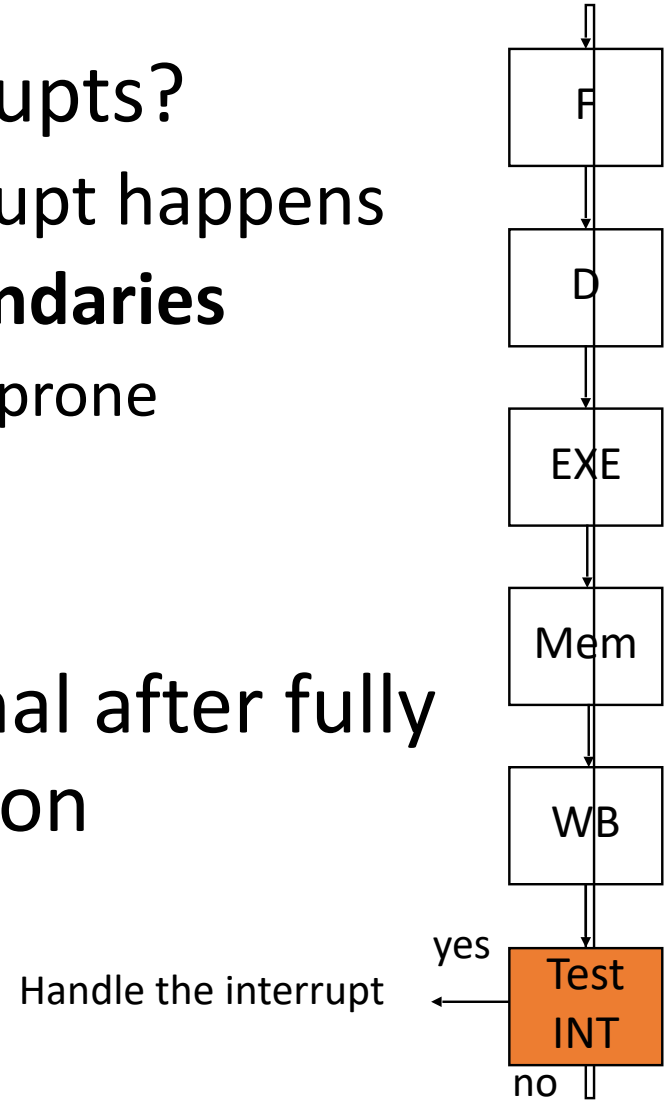
- If we stopped the currently executing program when the instruction cycle was in its FETCH OPERAND phase:
  - we would have to keep track of what part of the current instruction has executed
- It makes much more sense to ignore interrupt signals except when we are at an instruction boundary;



20

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

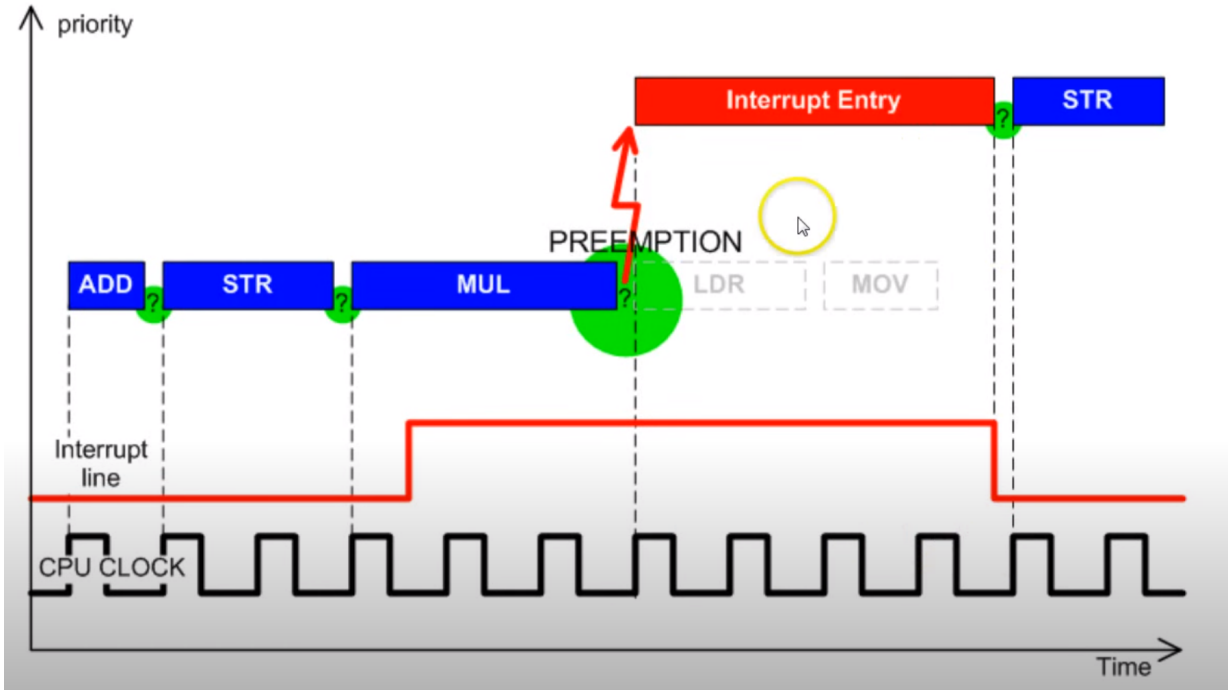
Check the Interrupt signal after fully executing each instruction



21

Interrupt Signal

- As long interrupt signal is low, the CPU fetches the next instruction in the pipeline
- When its high, the CPU performs the interrupt entry instructions



22

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
  - Do we need to save the content of the register file?
- Load the state of the interrupting program

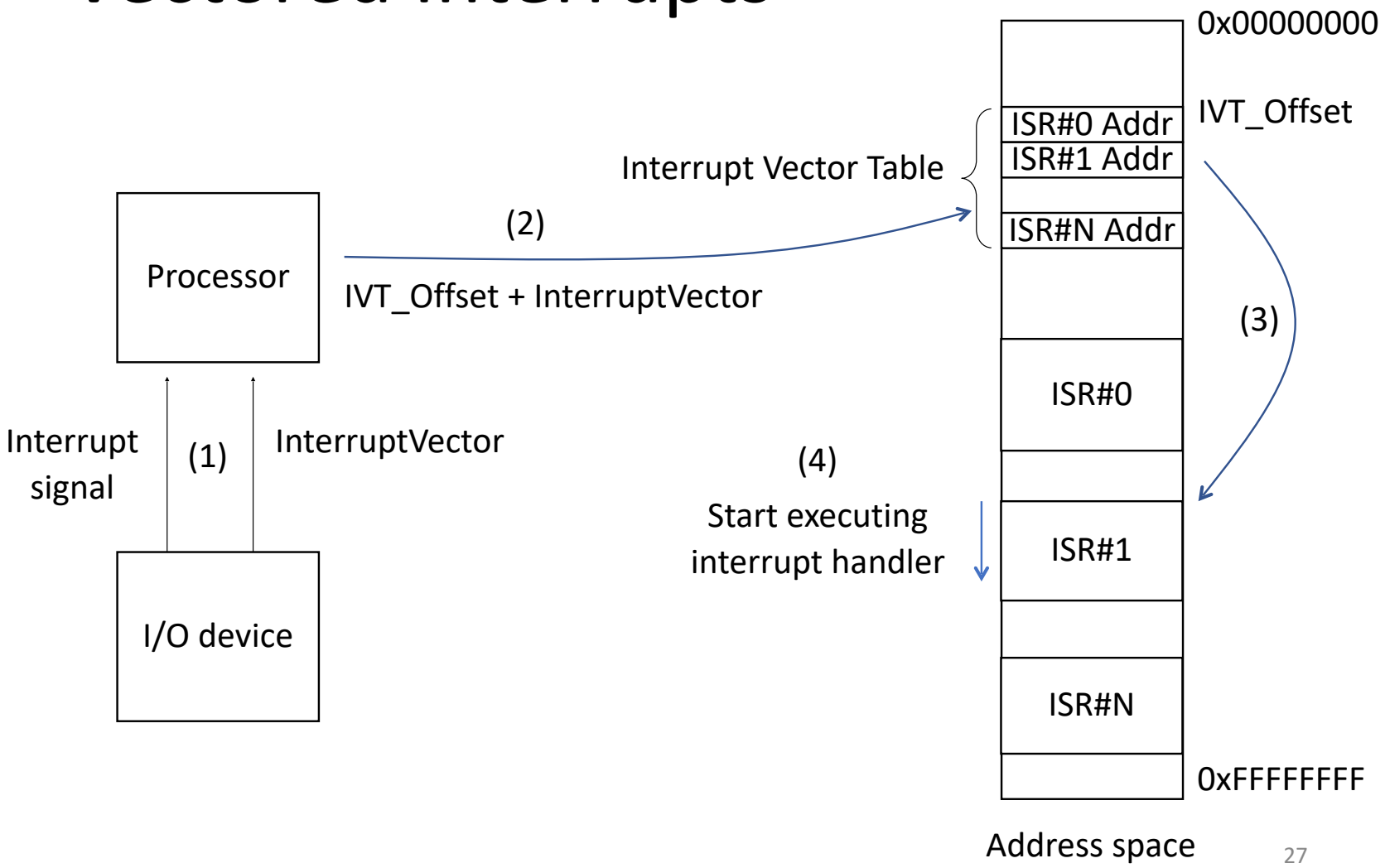
25

Vectored Interrupts

- An ***interrupt vector*** is provided to the processor by the interrupting I/O device
- The *interrupt vector* is used 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

26

Vectored Interrupts



27

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:

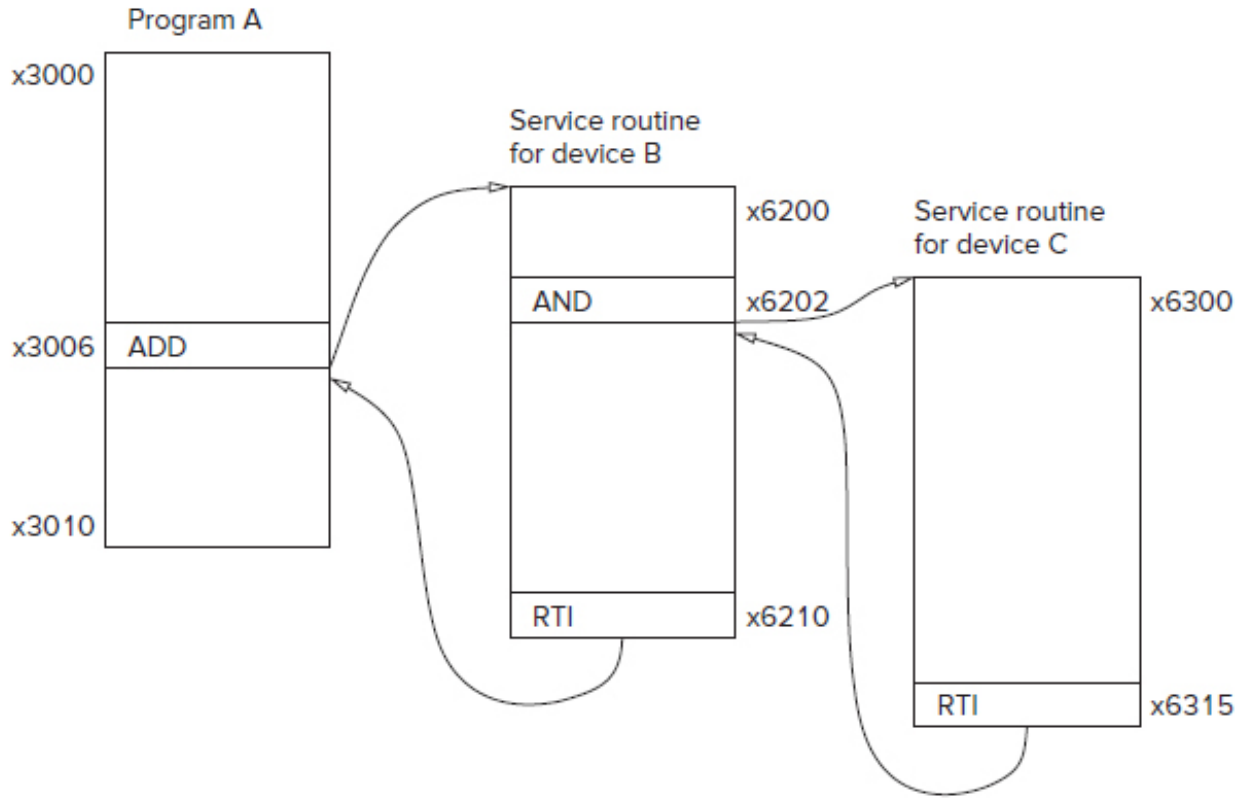
1. Initiate the interrupt
2. Service the interrupt
3. **Return from the interrupt**

Restoring the state of the interrupted program and bring it back to execution

29

Example: execution flow for Interrupt-driven I/O

In class quiz: Can you figure out the priority levels?



References

- Patel, S., and Yale Patt. *Introduction to Computing Systems: from bits & gates to C & beyond*. McGraw-Hill Professional, 2019.

31