UNIVERSITY OF PORTSMOUTH

Operating Systems and Internetworking

OSI – M30233 (OS Theme)

Week 1- Introduction

Dr Tamer Elboghdadly

Module Introduction

Plan of Lecture

- This theme of the module concentrates on Operating Systems (OS) and microprocessor architectures.
- This lecture provides some background material on important topics relating to Operating Systems, including some important *microprocessor instructions* and the role of *interrupt handlers*.

What is an Operating System?

- We are all familiar with various flavours of the Windows operating system from Microsoft.
  - Many will be familiar with the Mac OS operating systems from Apple.
  - Others will know one or more distributions of the Open Source Linux operating system.
  - We perhaps all use either Android or IOS.

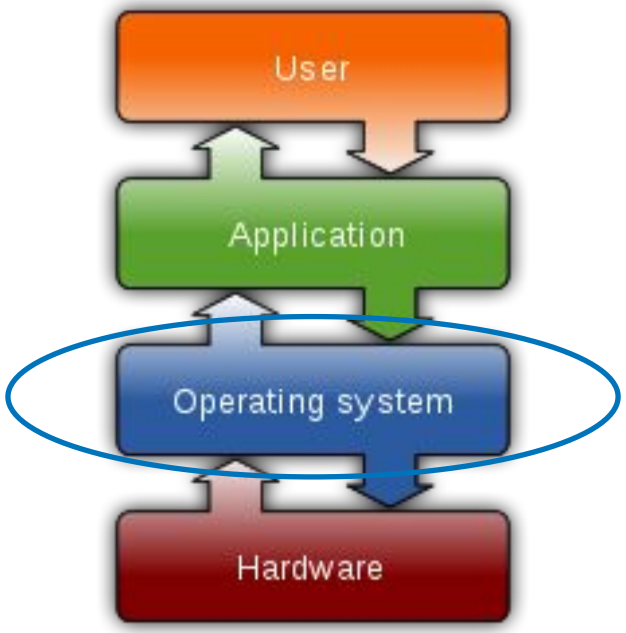
What is an Operating System?

- There are many OS. Some of them for computer devices (Desktop/Laptop) and others for Mobile devices.

- But what exactly *is* an operating system?

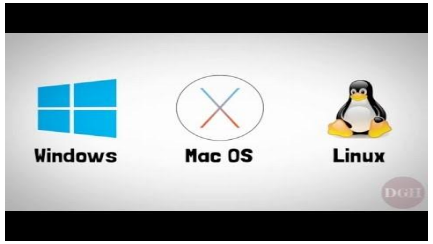
Operating System vs Desktop

- Most *programmers* would agree the OS is *not* the familiar “desktop” (or start screen) one sees on logging in to these systems.
  - This GUI (Graphical User Interface) is typically provided by a suite of “application-level” programmes.
- The operating system itself exists at <u>a lower level - below the “applications”</u>.
  - It is really only directly accessible by programmers.



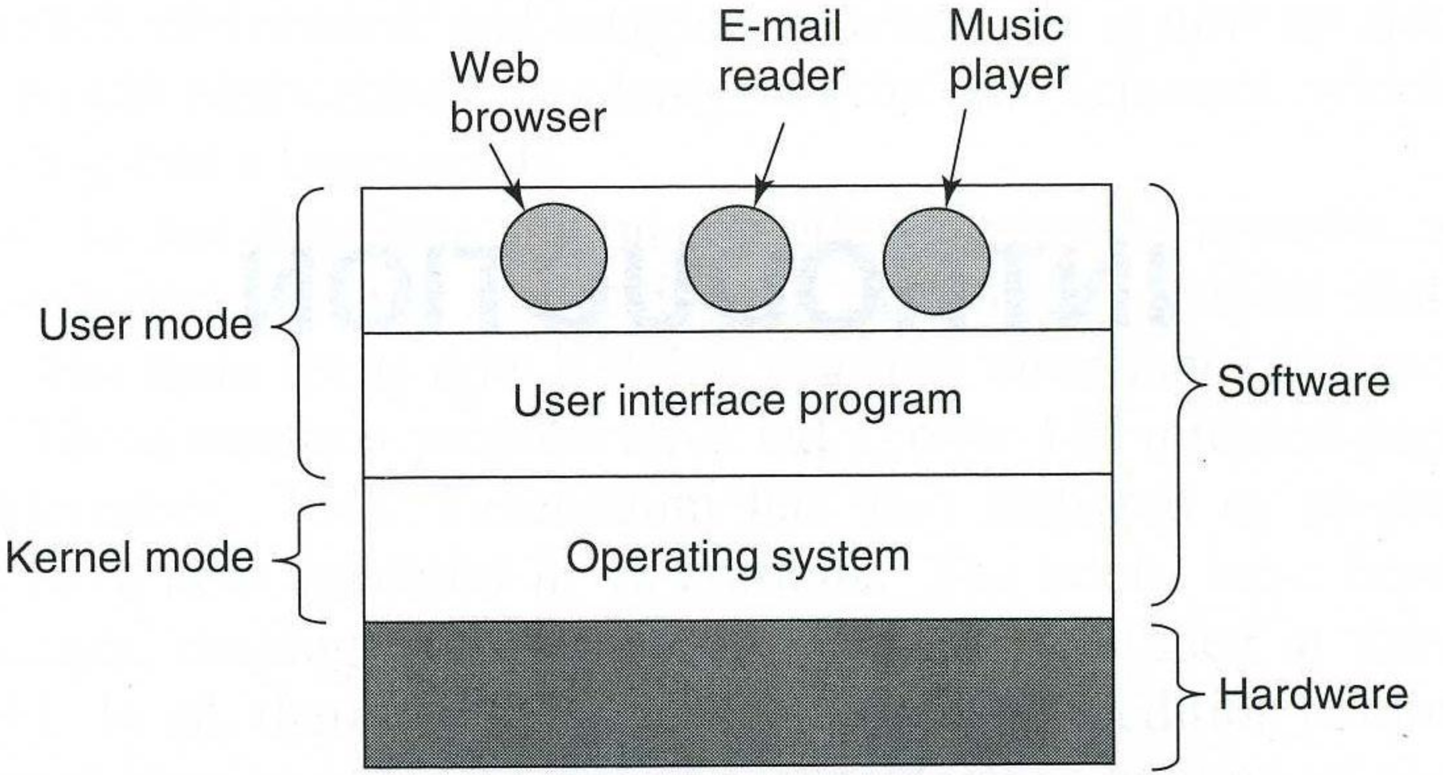
What is an Operating System?

- It is a system software that manage computer hardware resources and control processing.
- It allows multiple computational processes and users to share a processor simultaneously, protect data from unauthorized access and keep independent input/output (I/O) devices operating correctly.
- It provide a common services for application software.
- Users **can not** run any software application without it.



UNIVERSITY OF PORTSMOUTH

Position of Operating System[†]



[†]Tanenbaum, MOS, Fig 1-1

System Software vs. Application Software

System Software	Application Software
The operating system and utility programs that control a computer system and allow you to use your computer	Programs that allow a user to perform specific tasks on a computer
Enables the boot process, launches applications, transfers files, controls hardware configuration, manages files on the hard drive, and protects from unauthorized use	Word processing, playing games, browsing the Web, listening to music, etc.

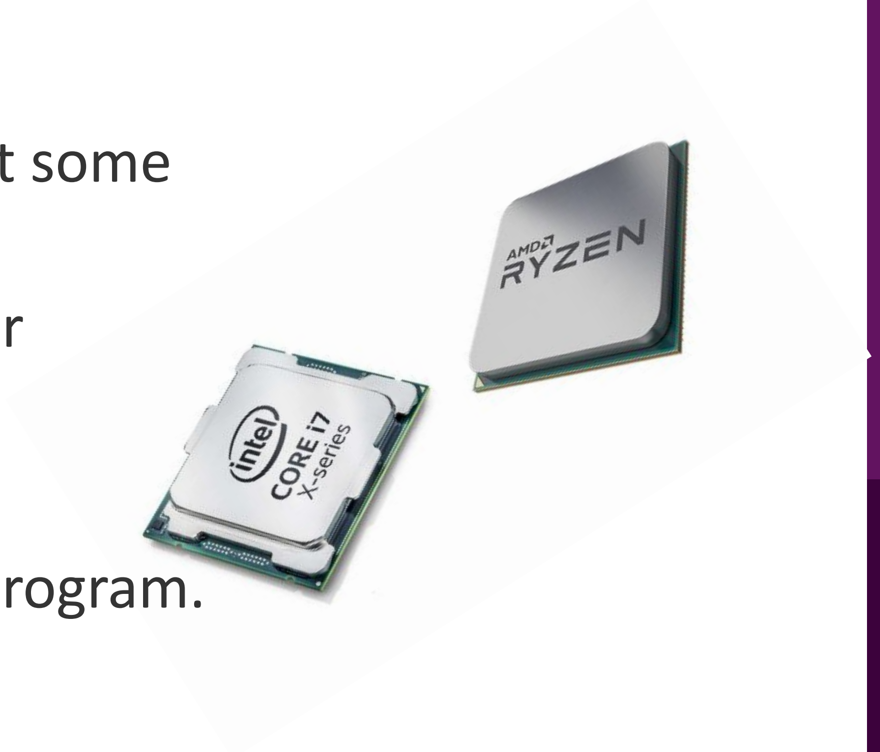
Characteristics of the OS[†]

- OS as “Extended Machine”
  - I/O (for example) involves reading and writing control registers, handling interrupts, …
  - Mistakes will crash whole computer.
  - OS provides a cleaner, safer, higher level set of operations for doing these things.
- OS as “Resource Manager”
  - In a modern OS, many different processes are going on *simultaneously*; these must *share* resources.
  - The OS arbitrates between the requests these processes make to I/O subsystems, memory, etc, to *ensure smooth functioning of the system*.

[†] Tanenbaum, MOS, 1.1

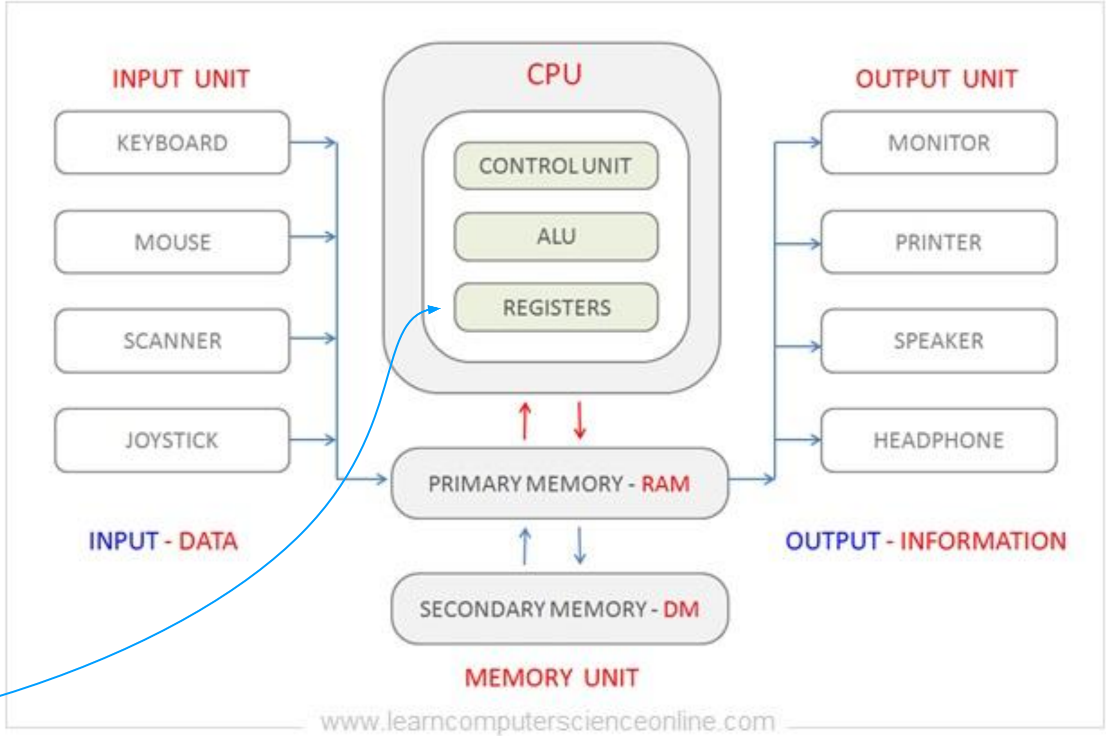
Central Processing Unit (CPU)

- For further clues about the role of the OS, let’s look at some aspects of a modern *Central Processing Unit*.
- CPU is also referred to as *processor*, *microprocessor* or *processing unit*.
- The CPU is the heart of the computer.
- It reads a program from memory, and executes that program.



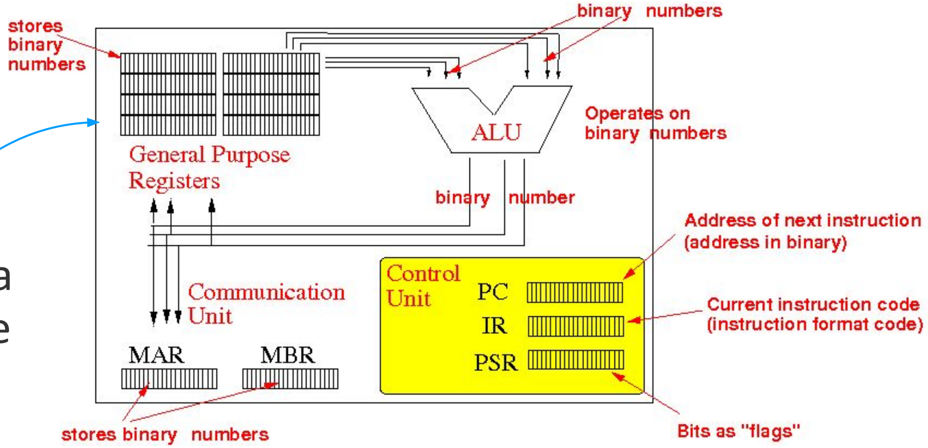
CPU Organization

- A modern CPU is complex, containing control unit, Arithmetic Logic Units, cache memory, memory management unit, etc, etc.
    - Will discuss in greater detail later in the module.
- Functionally, an engine for processing a sequence of *machine instructions*.
    - A single instruction might perform simple arithmetic on data values – typically individual *words* - or move data between memory and/or *registers*.



Registers?

- Very fast storage built into the CPU, typically for individual *words* of data.
  - Nowadays 32 bits or 64-bits.
  - Small amounts of high-speed memory contained within the CPU
  - They are used by the processor to store small amounts of data that are needed during processing, such as: the address of the next instruction to be executed, the current instruction being decoded.
  - CPU has many registers and they play a key role in **OS design** because they form part of state of a computation.
  - Most computer architecture provide a small set of **General-Purpose Registers (GPR)**.



stores binary numbers
binary numbers
General Purpose Registers
ALU
Operates on binary numbers
binary number
Address of next instruction (address in binary)
Communication Unit
Control Unit
PC
IR
PSR
Current instruction code (instruction format code)
MAR
MBR
stores binary numbers
Bits as "flags"

MAR: Memory Address Register
MBR: Memory Buffer Register

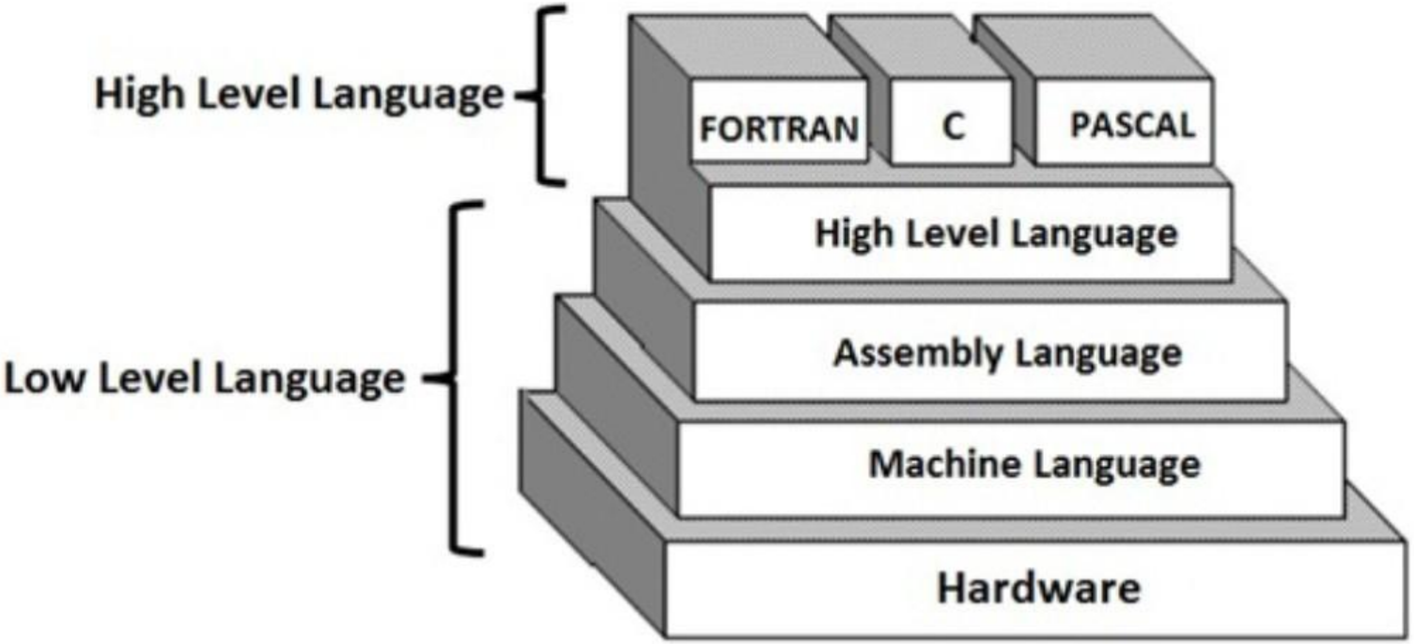
PC: Program Counter
IR: Instruction Register
PSR:Processor State Register

GPR - Registers?

- An X86, for example, has around eight **GPR** registers with names like *EAX*, *EBX*, *ECX*, *EDX* and *EBP*, *ESI*, *EDI*, *ESP*.
    - These are supplemented by various “special purpose” registers – *program counter*, etc, etc.
    - One very important special purpose register is the *program status word*, which sets the *mode* the CPU is operating in (see later).

Name	Use	Description
EAX	**Accumulator**	The default register for many addition and multiplication instructions
EBX	**Base**	Stores the base address during memory addressing.
ECX	**Count**	The default counter for repeat (REP) prefix instructions and LOOP instructions.
EDX	**Data**	Used for multiply and divide operations
ESI	**Source Index**	Store source index
EDI	**Destination Index**	Store destination index
EBP	**Base Pointer**	Mainly helps in referencing the parameter variables passed to a subroutine.
ESP	**Stack Pointer**	Provides the offset value within the program stack.

Classification of Programming languages



Computer Language and its Types

Assembly Language

- Assembly language is a symbolic form of machine code, used by system programmers.
- Two instructions in Intel assembler:

```
MOV EBX, EAX
ADD EBX, 4
```

  - The first instruction copies the contents of register *EAX* to register *EBX*.
  - The second instruction increases the value in register *EBX* by 4.
  - Think: *b = a + 4*

Effect of Instructions on Registers

```
MOV EBX, EAX
ADD EBX, 4
```

EAX	27
EBX	31
ECX	
EDX	

Instructions for Accessing Memory

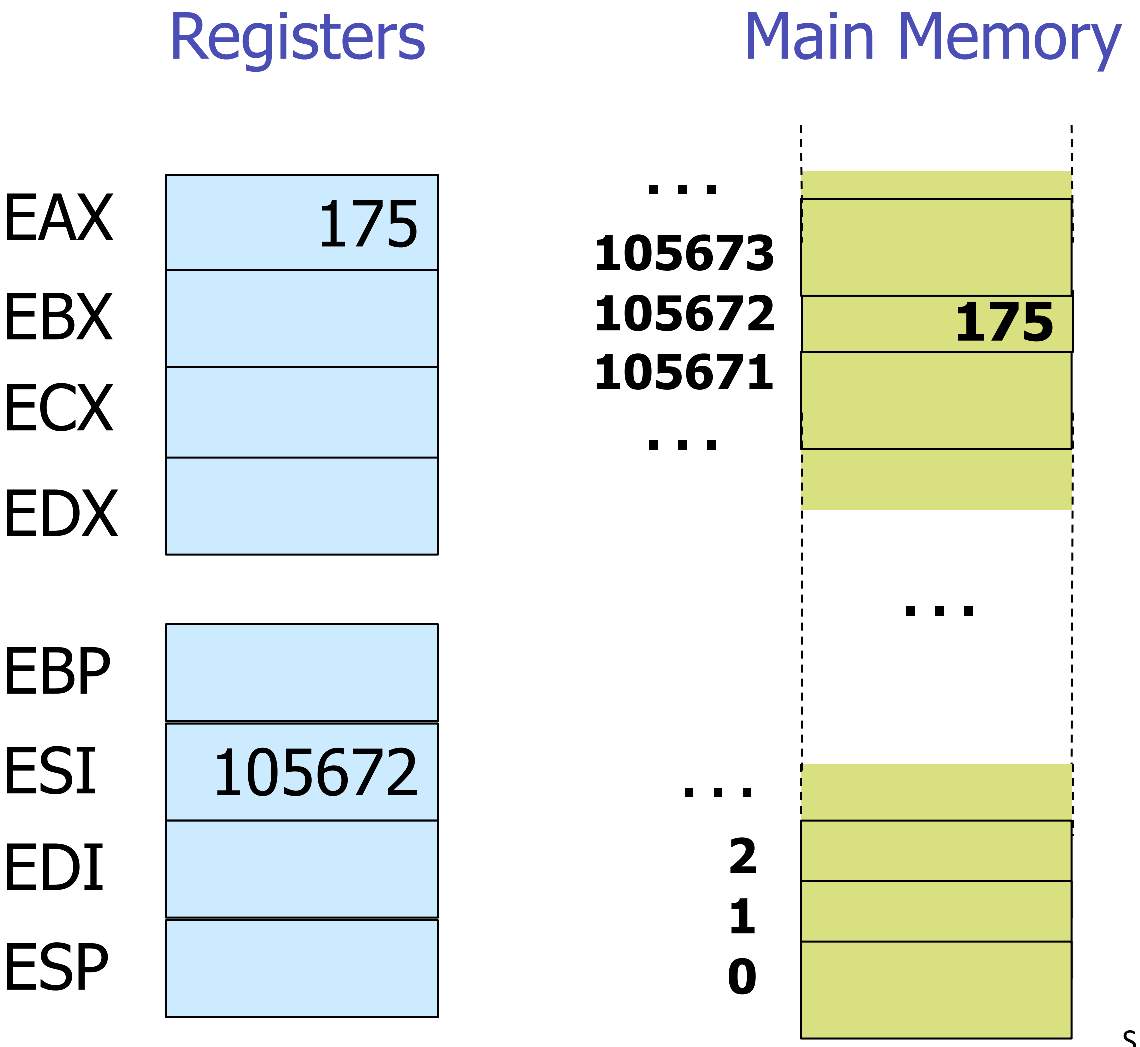
- Pentium MOV instruction can also move a value between *main memory* and register.
- Simple example:

```
MOV ESI, 105672
MOV EAX, [ESI]
```

  - Save constant 105672 in register ESI; then load contents of memory at this address to register EAX.

Effect of Instructions on Registers

```
MOV ESI, 105672
MOV EAX, [ESI]
```



Registers

EAX 175
EBX
ECX
EDX

EBP
ESI 105672
EDI
ESP

Main Memory

. . .
105673
105672 175
105671
. . .

. . .

. . .
2
1
0

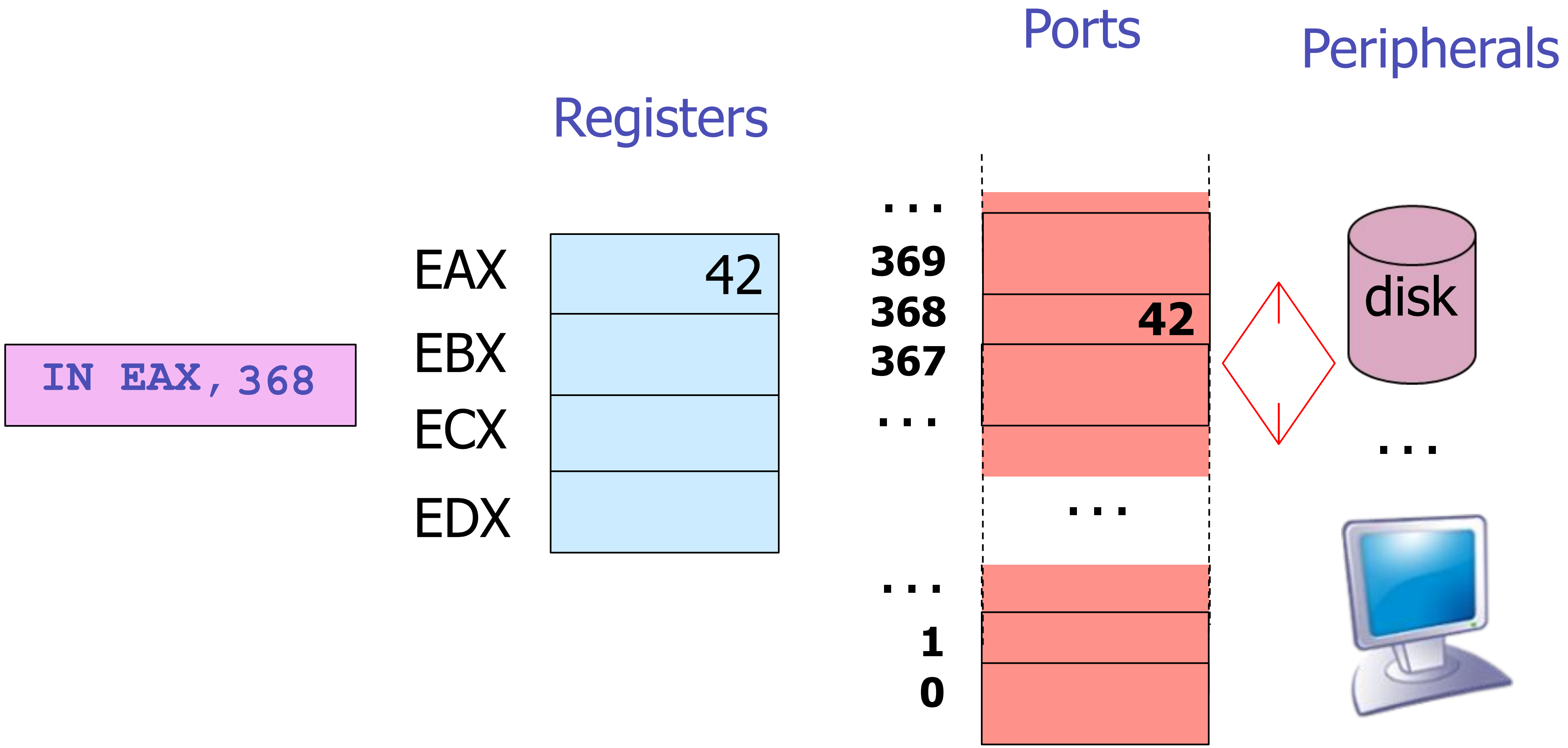
Low Level I/O

- An I/O device like a hard disk will have an associated a set of *ports*, through which device is controlled, and data transferred.
  - A range of ports will be associated with each device.
- Special instructions IN and OUT are used to read or write ports. For example:

```
IN EAX,368
```

If port number 368 corresponds to the “data word” in the disk controller (say), value of requested data is copied to CPU register *EAX*.

Effect of Instruction on Registers



IN EAX, 368

Registers

EAX 42

EBX

ECX

EDX

Ports

...

369

368 42

367

...

...

...

1

0

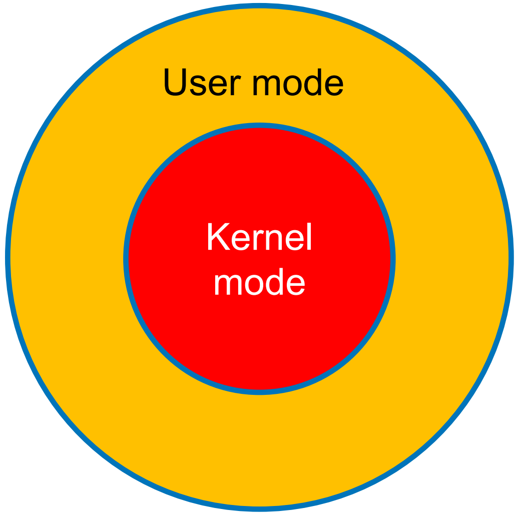
Peripherals

disk

...

User and Kernel modes (Important!)

- Typical CPUs support different *modes* of operation controlled by a register we will refer to as the *Program Status Word* †.
- Machine code running while CPU is in *user mode* can only use *limited* instructions – *not*, e.g., IN or OUT instructions.
- Only code running in *kernel mode* can use *privileged instructions* (e.g. IN, OUT).



  - †On recent X86 processors, it is actually controlled by bit 0 of the *Control Register,* CR0. If this bit is set, we are in user mode (or "protected mode").
  - The *kernel is a computer program at the core of a computer's operating system and has complete control over everything in the system*

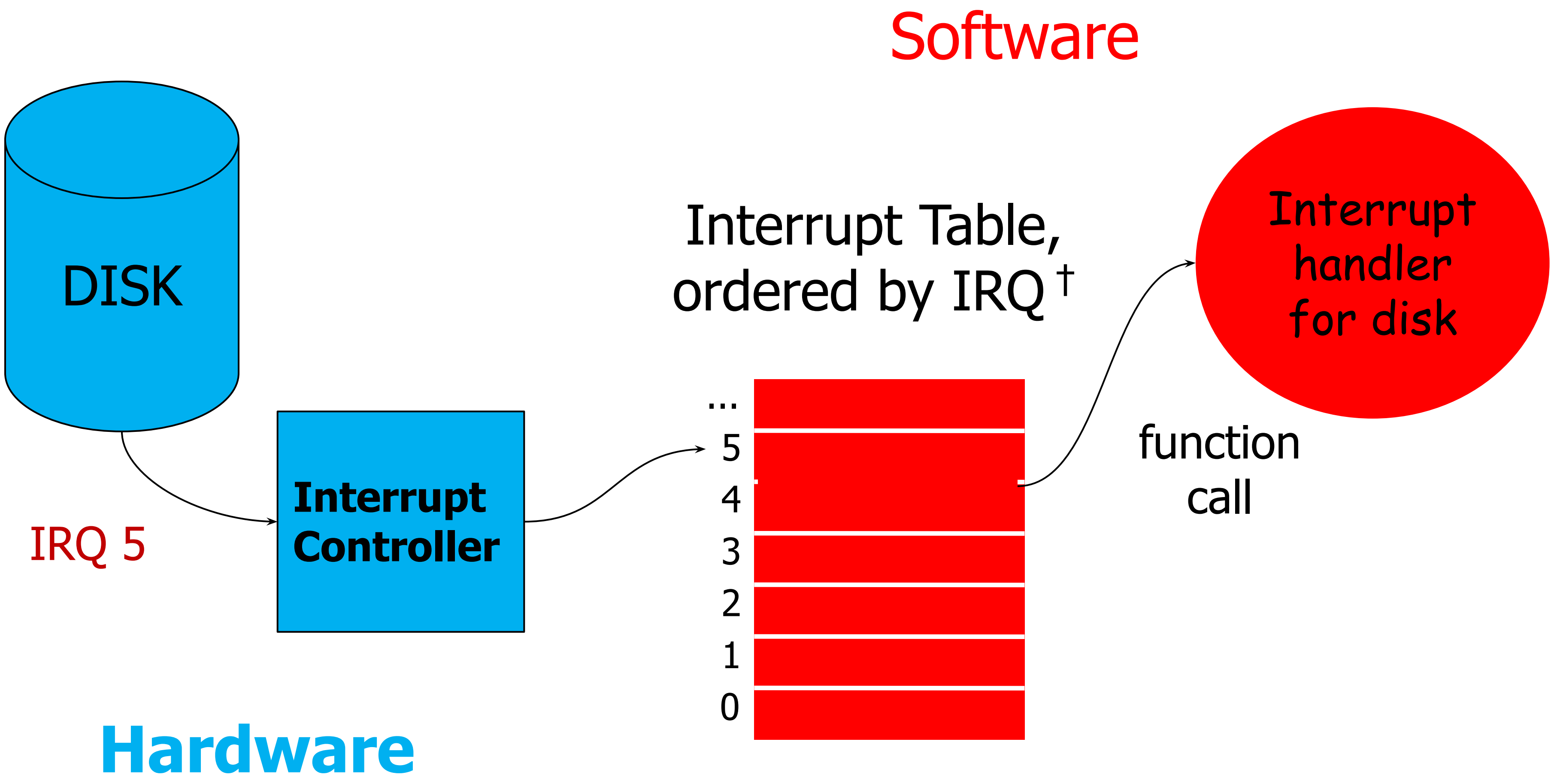
OS as “Kernel” Code

- One possible characterisation of OS is as “the code that runs in *kernel mode*”.
- Thus I/O operations (for example) can only be performed directly *by* the OS, *on behalf of* “application” programs.

Interrupts

- When an I/O controller (e.g. on a disk card) has requested data available, it must gain the attention of the CPU.
- This is done by asserting an electrical signal called an *interrupt*.
- The CPU must (temporarily) abandon whatever program it is executing, and instead execute specialized code to deal with the new event.
- Specialized code takes form of *interrupt handlers*, typically installed at boot time.
    - Importantly, interrupt handlers run in *kernel mode*.

Interrupt Handling (Example)



†*IRQ* stands for Interrupt Request – its number defines kind of request.

Wider Role of Interrupt Handlers

- We will see that interrupt handlers have a wide significance in operating systems - beyond their original role in processing data received from I/O controllers.
  - For example they may have a role in *process scheduling*, and in the implementation of *system calls* (see later lectures).
- In some sense the whole operating system is driven by variations on the theme of “interrupt handler”.

Summary: What is an OS?

- Maybe…
  - It is the subset of software on the computer that runs in *kernel mode*?
  - As a reactive system, it is essentially defined by its interrupt handlers, and code these invoke?

Summary

- We reviewed various components of computer hardware and operating systems.
- *Next lecture*: Threads and concurrency

Further Reading

- Andrew S. Tanenbaum, “*Modern Operating Systems*”, 4th Edition, Pearson, 2015 (MOS)
- Andrew S. Tanenbaum and Albert S. Woodhull “Operating Systems Design and Implementation”, 3rd Edition, 2009 (MODI)
- “Architecture & Operating Systems” notes from the first year.

UNIVERSITY OF PORTSMOUTH

Questions?