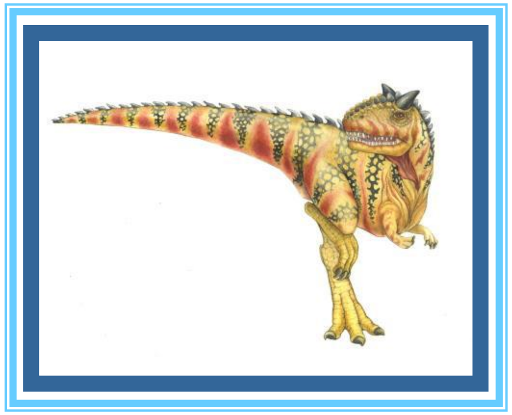
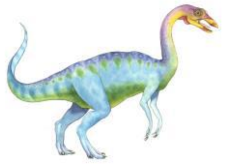
2: Hardware



2: Hardware

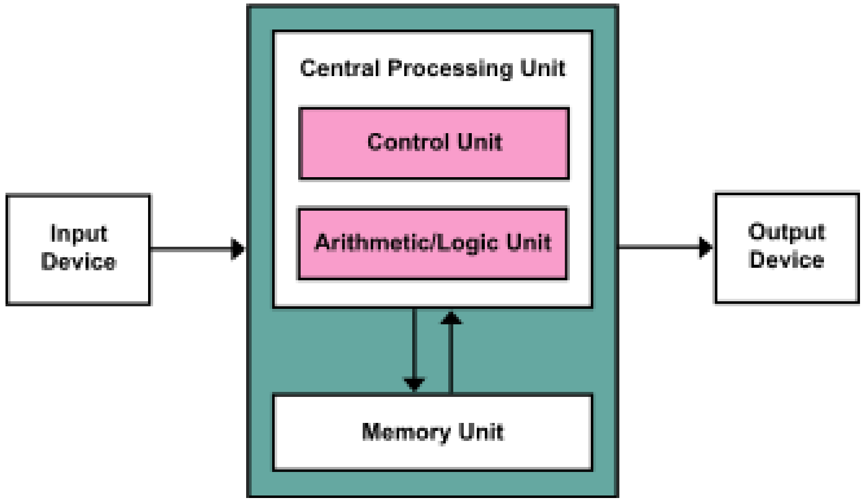
- CPU / Von Neumann Model
- Device and interrupt

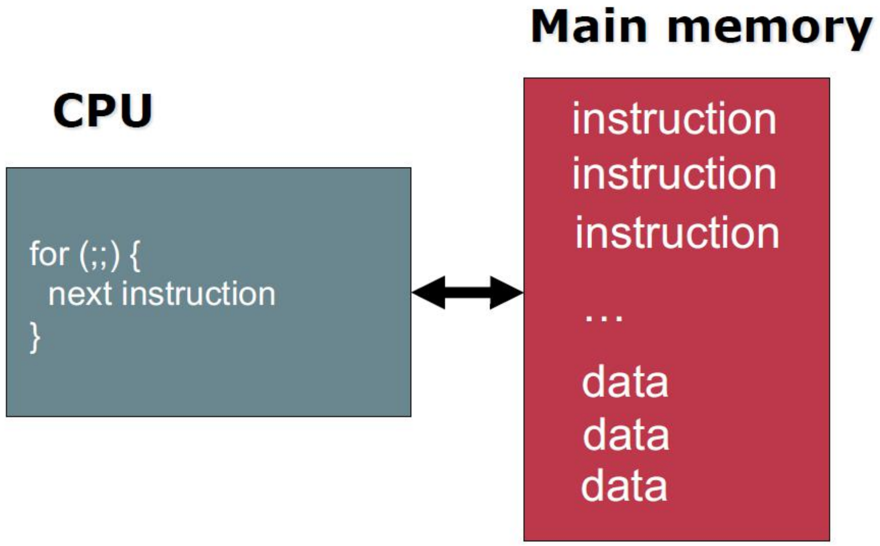


Objectives

- Understand CPU / Von Neumann Model
- Understand Device and interrupt

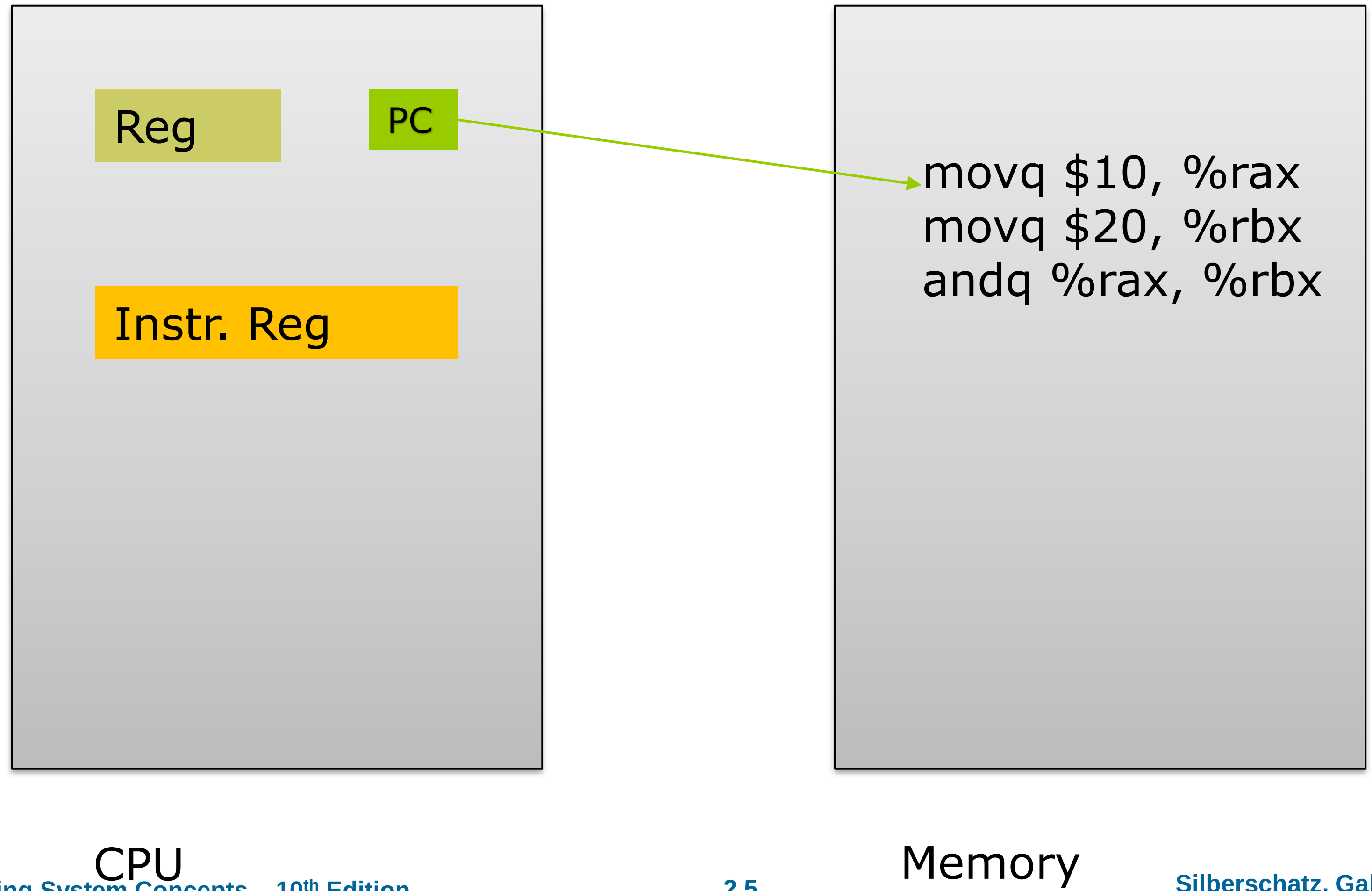
Von Neumann Model





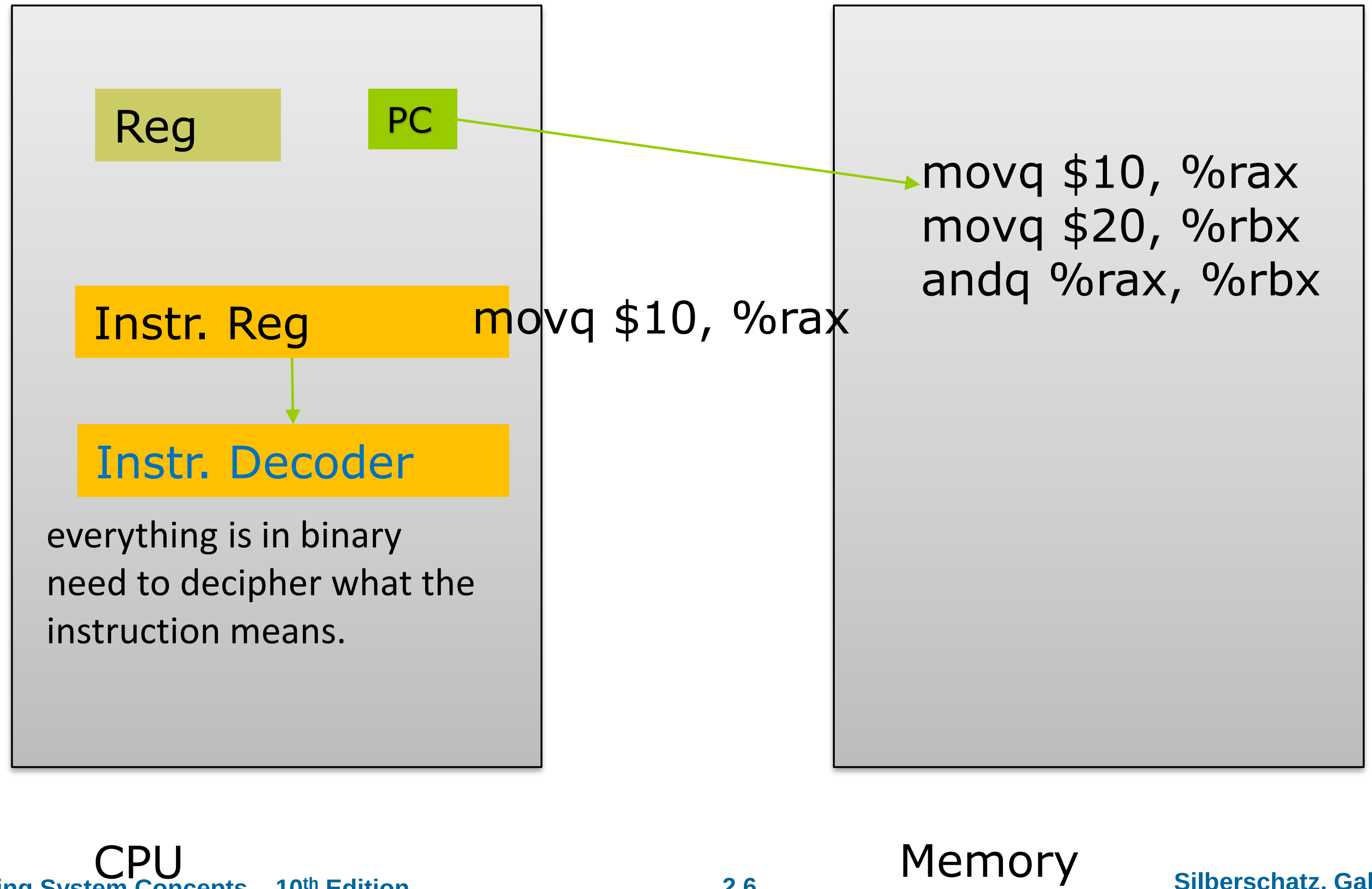
```
for (;;) {
  next instruction
}
```

Fetch



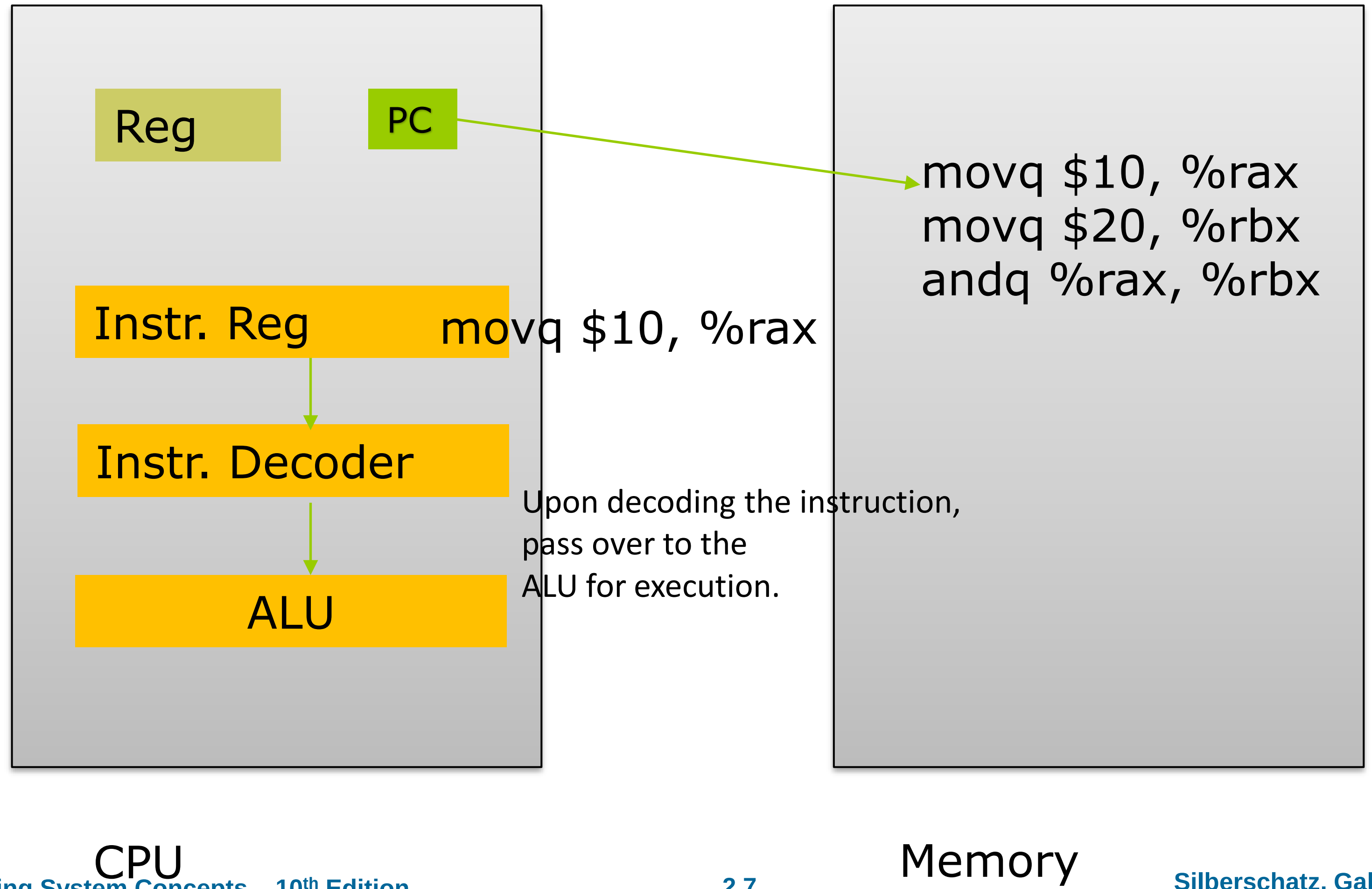
```
movq $10, %rax
movq $20, %rbx
andq %rax, %rbx
```

Decode



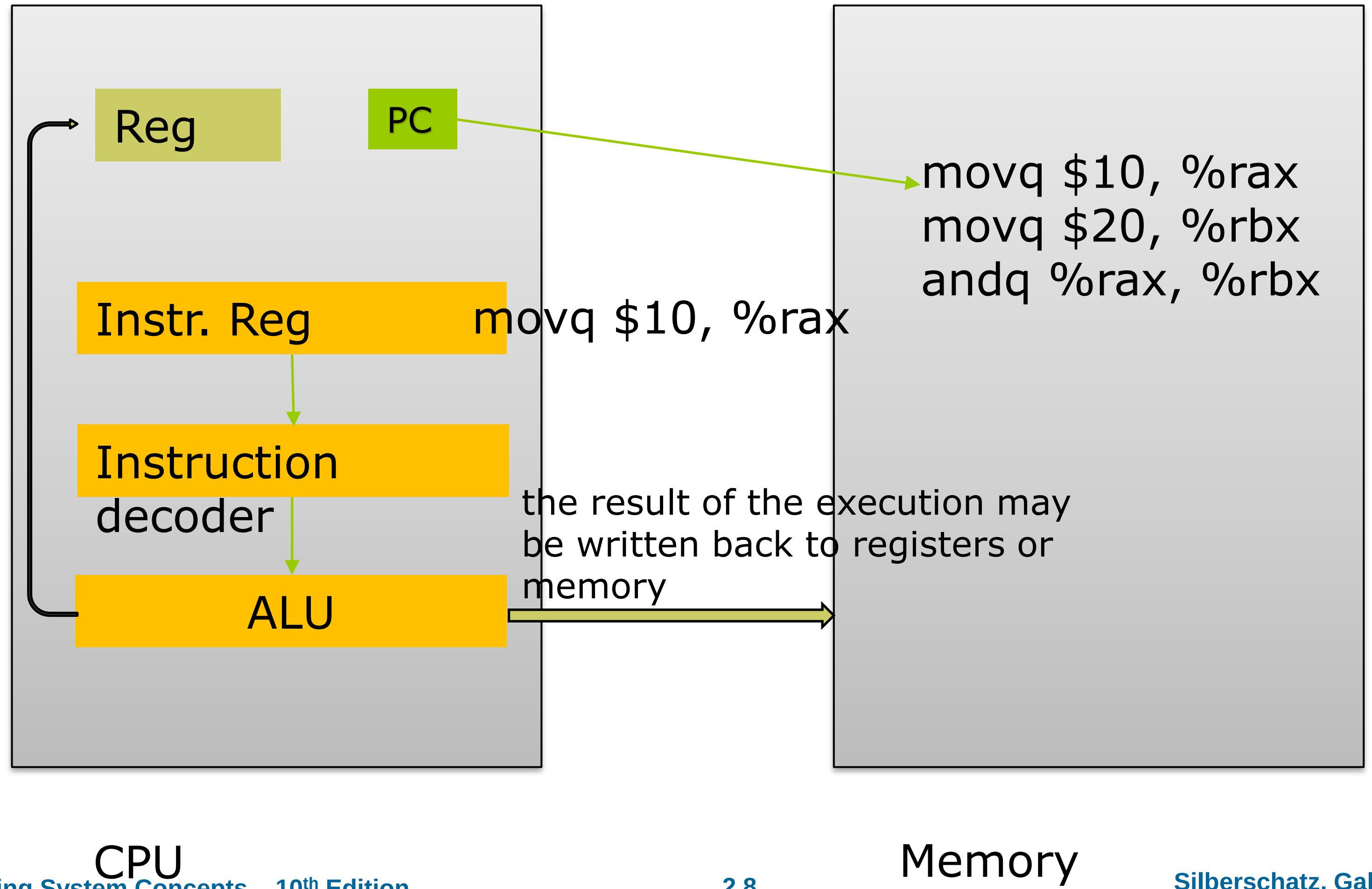
everything is in binary
need to decipher what the
instruction means.

Execute

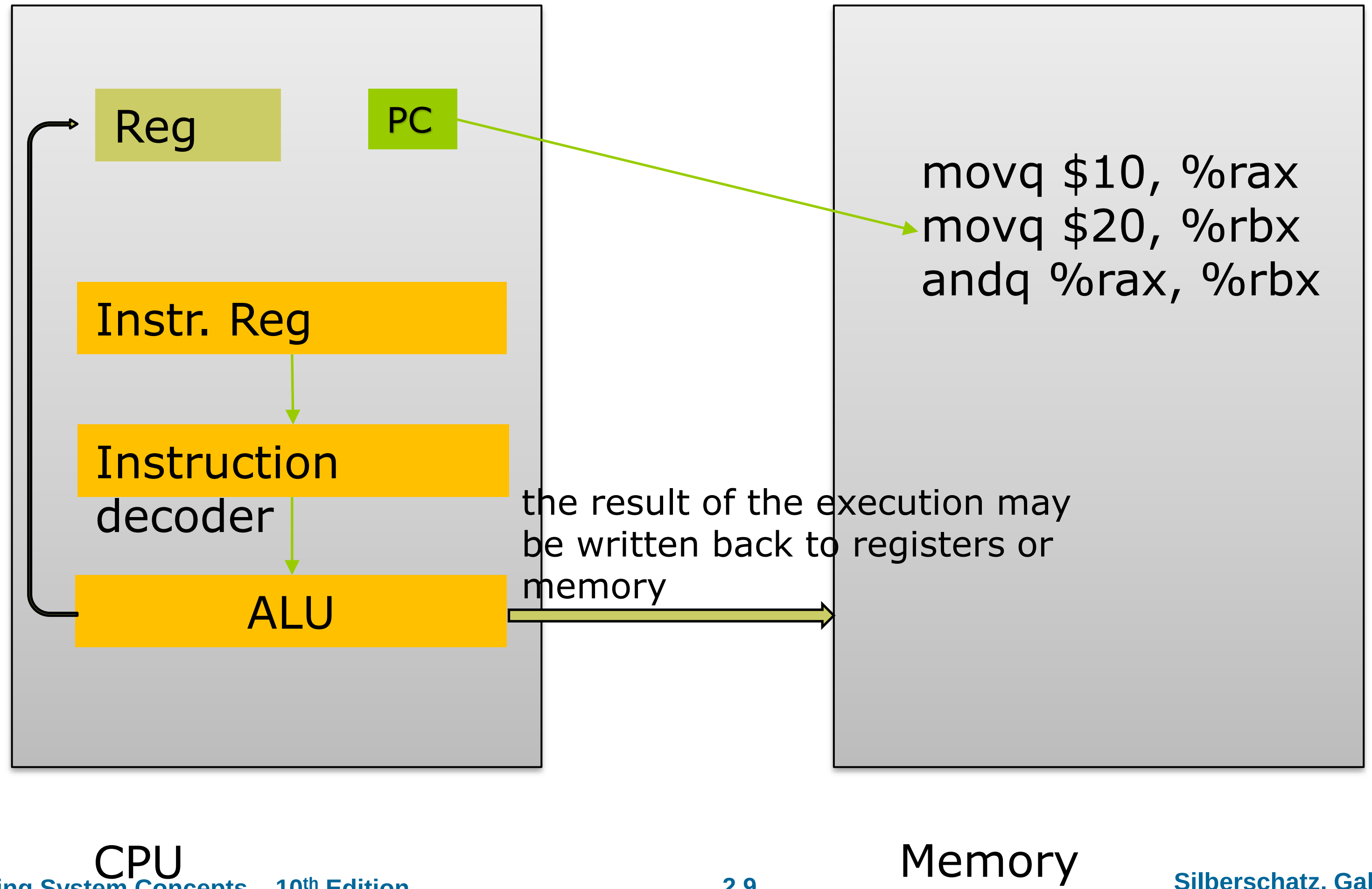


Upon decoding the instruction, pass over to the ALU for execution.

Writeback

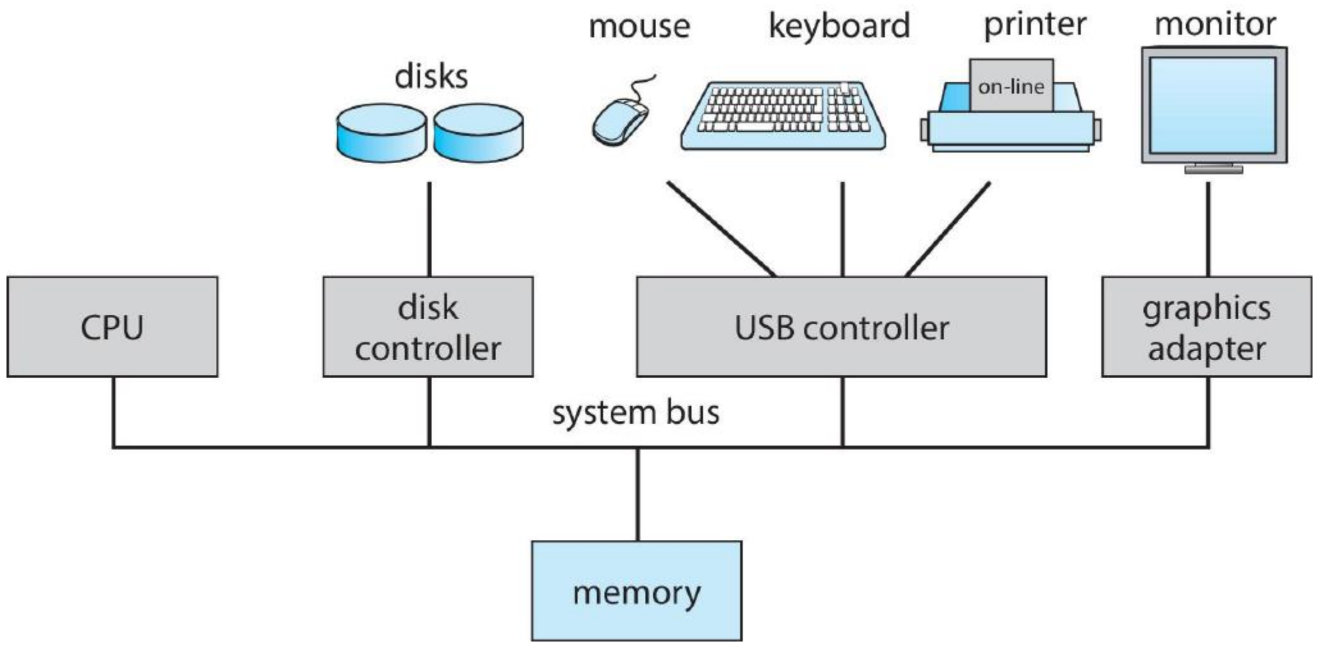

Reg
PC
movq $10, %rax
movq $20, %rbx
andq %rax, %rbx
Instr. Reg
movq $10, %rax
Instruction decoder
the result of the execution may be written back to registers or memory
ALU
CPU
Memory

Next instruction…Repeat the cycle



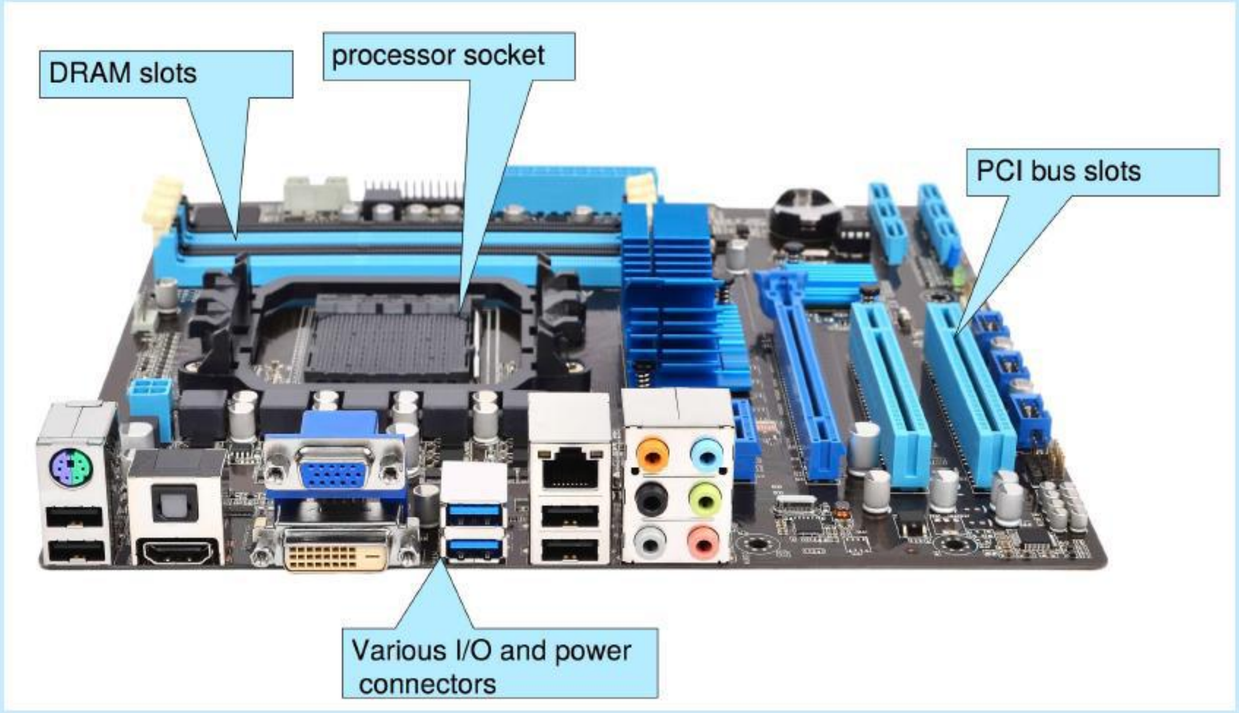
Computer System Organization

- Computer-system operation
  - **One or more CPUs**, device controllers connect through common **bus** providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



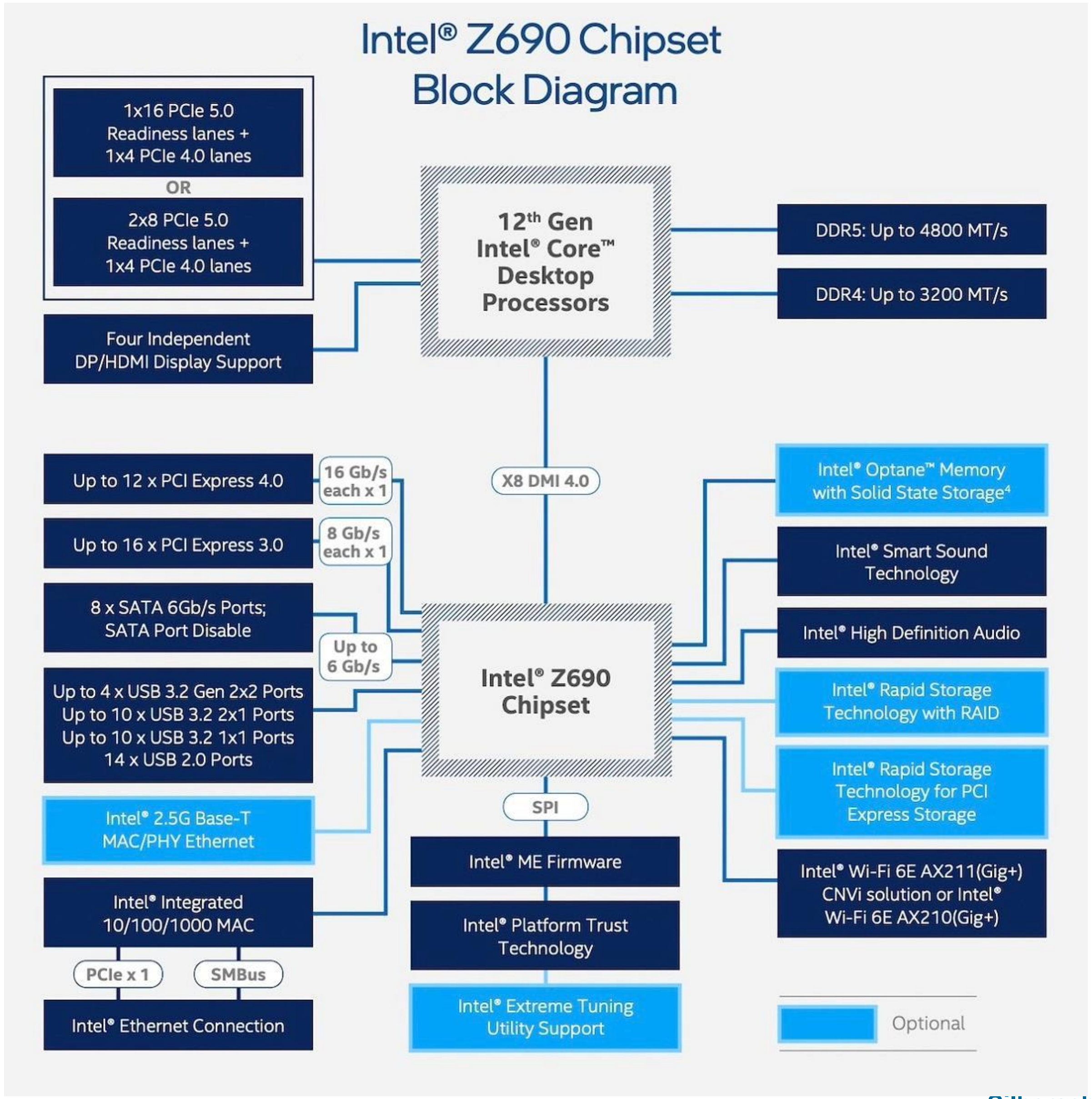
PC Motherboard

Consider the desktop PC motherboard with a processor socket shown below:



This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

Chipset Block Diagram



Intel® Z690 Chipset Block Diagram
1x16 PCIe 5.0 Readiness lanes + 1x4 PCIe 4.0 lanes
OR
2x8 PCIe 5.0 Readiness lanes + 1x4 PCIe 4.0 lanes
Four Independent DP/HDMI Display Support
12th Gen Intel® Core™ Desktop Processors
DDR5: Up to 4800 MT/s
DDR4: Up to 3200 MT/s
X8 DMI 4.0
Up to 12 x PCI Express 4.0
16 Gb/s each x 1
Up to 16 x PCI Express 3.0
8 Gb/s each x 1
8 x SATA 6Gb/s Ports; SATA Port Disable
Up to 6 Gb/s
Up to 4 x USB 3.2 Gen 2x2 Ports Up to 10 x USB 3.2 2x1 Ports Up to 10 x USB 3.2 1x1 Ports 14 x USB 2.0 Ports
Intel® 2.5G Base-T MAC/PHY Ethernet
Intel® Integrated 10/100/1000 MAC
PCIe x 1
SMBus
Intel® Ethernet Connection
Intel® Z690 Chipset
SPI
Intel® ME Firmware
Intel® Platform Trust Technology
Intel® Extreme Tuning Utility Support
Intel® Optane™ Memory with Solid State Storage4
Intel® Smart Sound Technology
Intel® High Definition Audio
Intel® Rapid Storage Technology with RAID
Intel® Rapid Storage Technology for PCI Express Storage
Intel® Wi-Fi 6E AX211(Gig+) CNVi solution or Intel® Wi-Fi 6E AX210(Gig+)
Optional

Low-level architecture affects the OS

- The operating system supports sharing of hardware and protection of hardware
  - Multiple applications can run concurrently, sharing resources
  - A buggy or malicious application can't violate other applications or the system

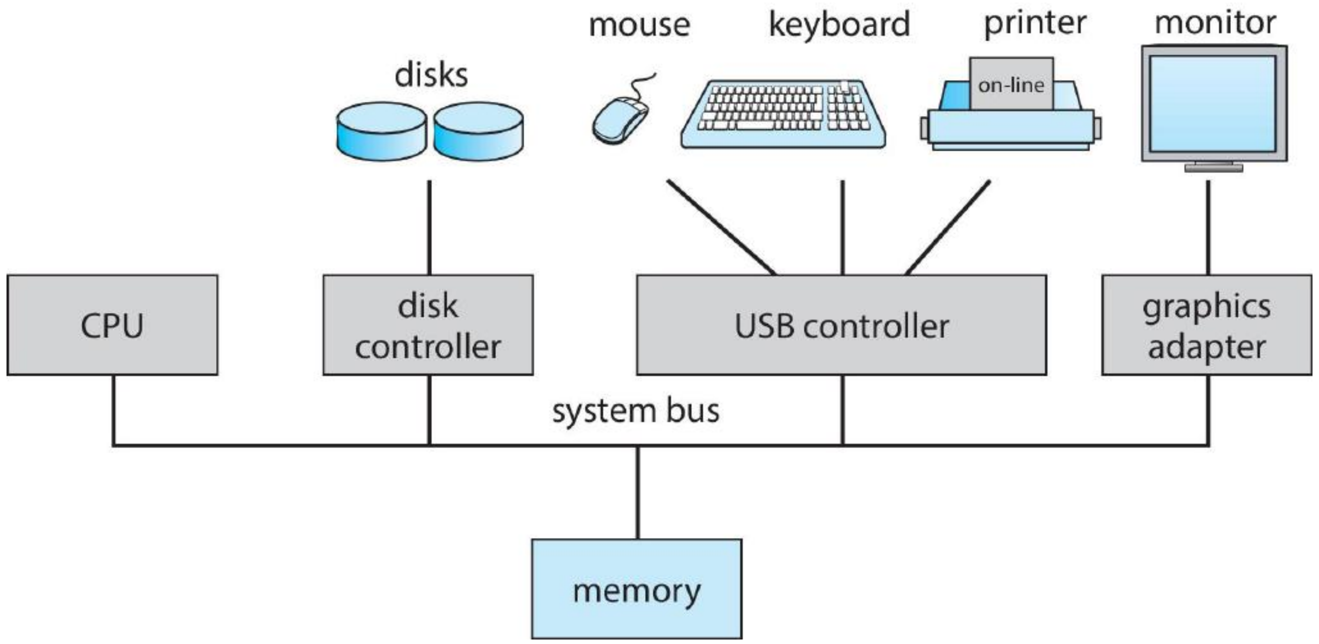
- The architecture determines which approaches are viable (reasonably efficient, or even possible)
  - Instruction set (synchronization, I/O, …)
  - Hardware components like MMU or DMA controllers

Architectural features affecting OS's

- These hardware features were built primarily to support OS's:
  - Timer (clock) operation
  - Synchronization instructions (e.g., atomic test-and-set, mem fence)
  - Memory protection
  - I/O control operations
  - Interrupts and exceptions and system calls (and software interrupts)
  - Protected modes of execution (kernel vs. user)
    - Privileged instructions (only in kernel)
  - Virtualization architectures

Computer System Organization

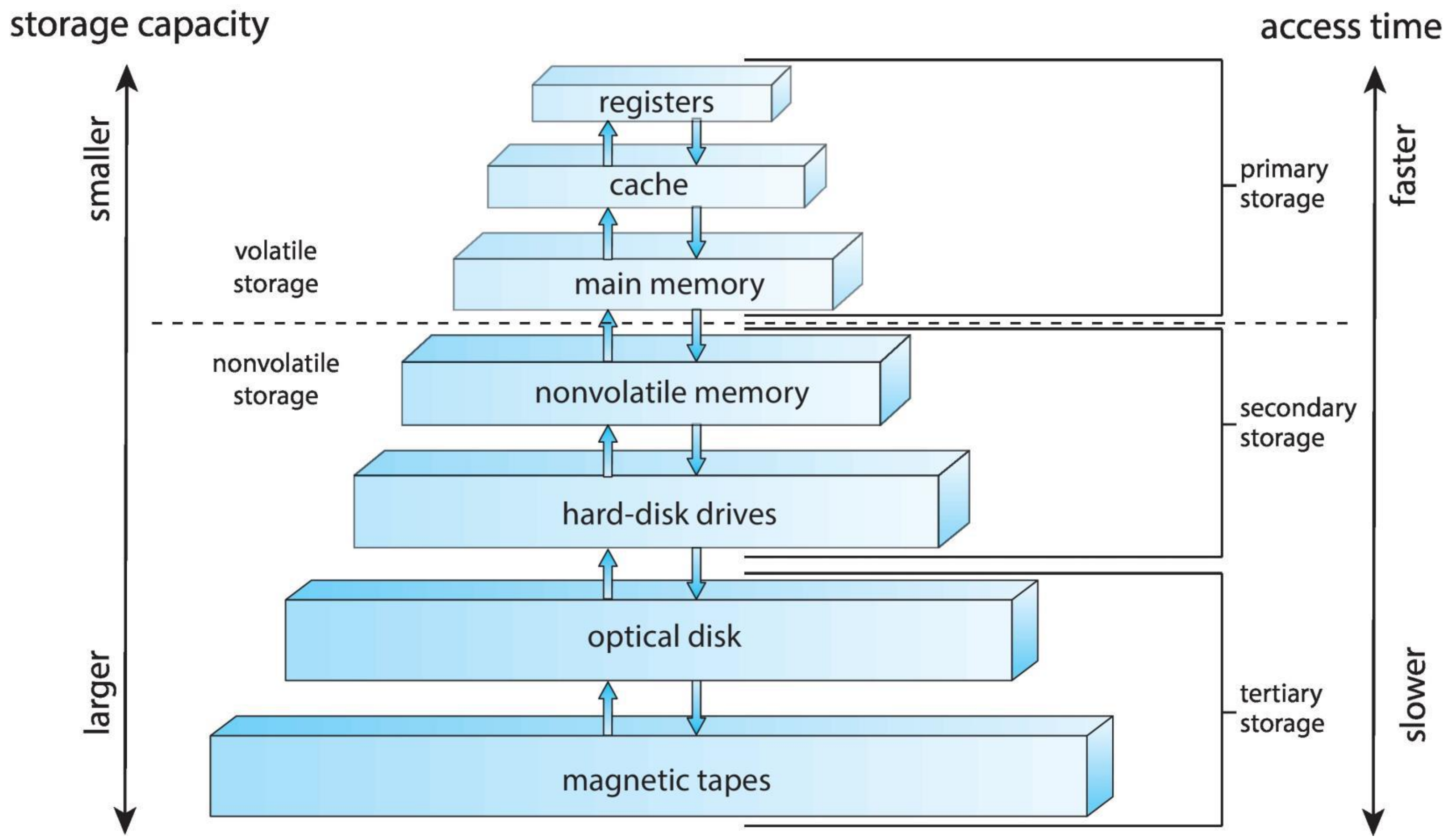
- Computer-system operation
  - One or more CPUs, device controllers connect through common **bus** providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**

Storage-Device Hierarchy

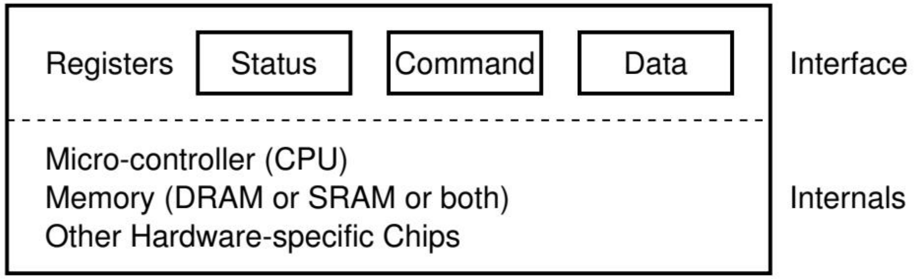


I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

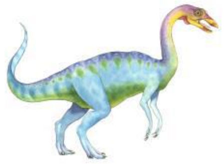
I/O Device

- It is "several sets of wires with agreed functions" (RTFM)
- Read/write data from/to the wire via handshake signals
- Each group of wires has its own address
  - The CPU can directly use instructions (in/out/MMIO) to exchange data with the device (regardless of how the device is implemented)



```
Example 1:
outb(COM1, send->data); //write
recv->data = (inb(COM1 + 5) & 0x1) ? inb(COM1) : -1; //read

Example 2:
IBM PC/AT 8042 PS/2 (Keyboard) Controller
“hardcoded” to I/O port: 0x60 (data), 0x64 (status/command)
```

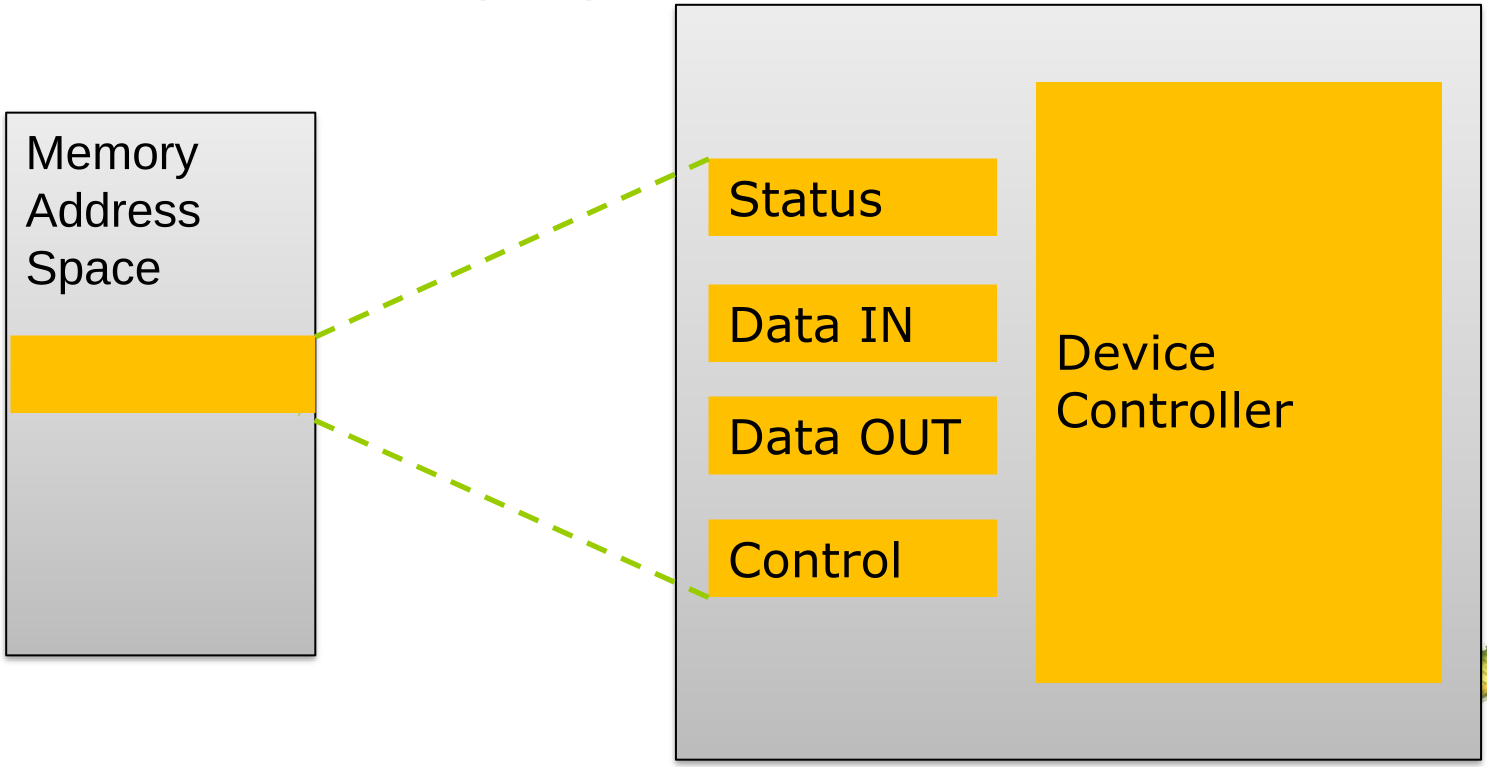


Bus and Driver

- Bus
  - Provides device registration and address-to-device forwarding
  - Forward the received address (bus address) and data to the corresponding device
    - Example: port I/O, port is the address on the bus
  - The CPU of the IBM PC actually only sees this one I/O device
  - In this way, the CPU only needs to be directly connected to one bus!
  - Today the PCI bus handle this task
- Device driver
  - “Translate" system calls (read/write/ioctl/...) into interactions with device registers
  - Just a piece of normal kernel code, but may sleep
  - E.g., the objects in /dev/,
  - Numerous and lowest quality codes

Memory-mapped I/O

- Device data and command registers mapped to processor address space, e.g. intel MOV instructions
  - Useful for large address spaces (e.g., graphics)
  - Specific addresses mapped to I/O devices
  - Fast response time (pros)
  - Low CPU utilization (cons)



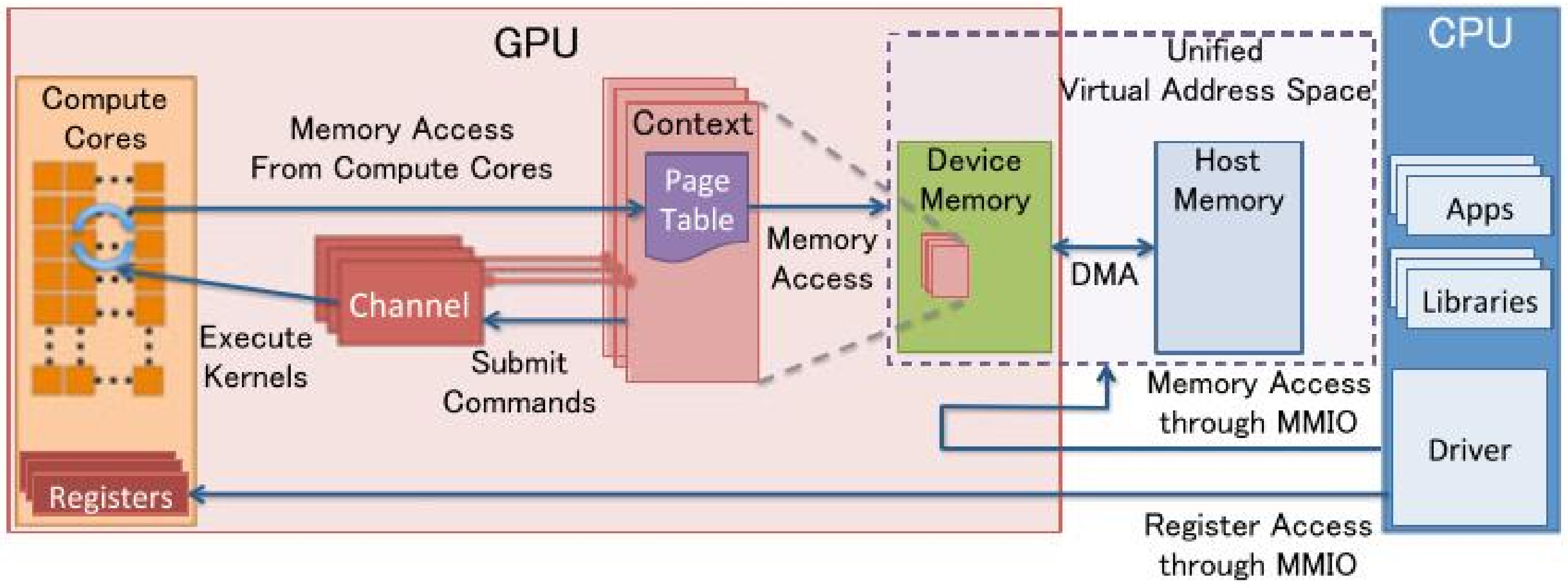
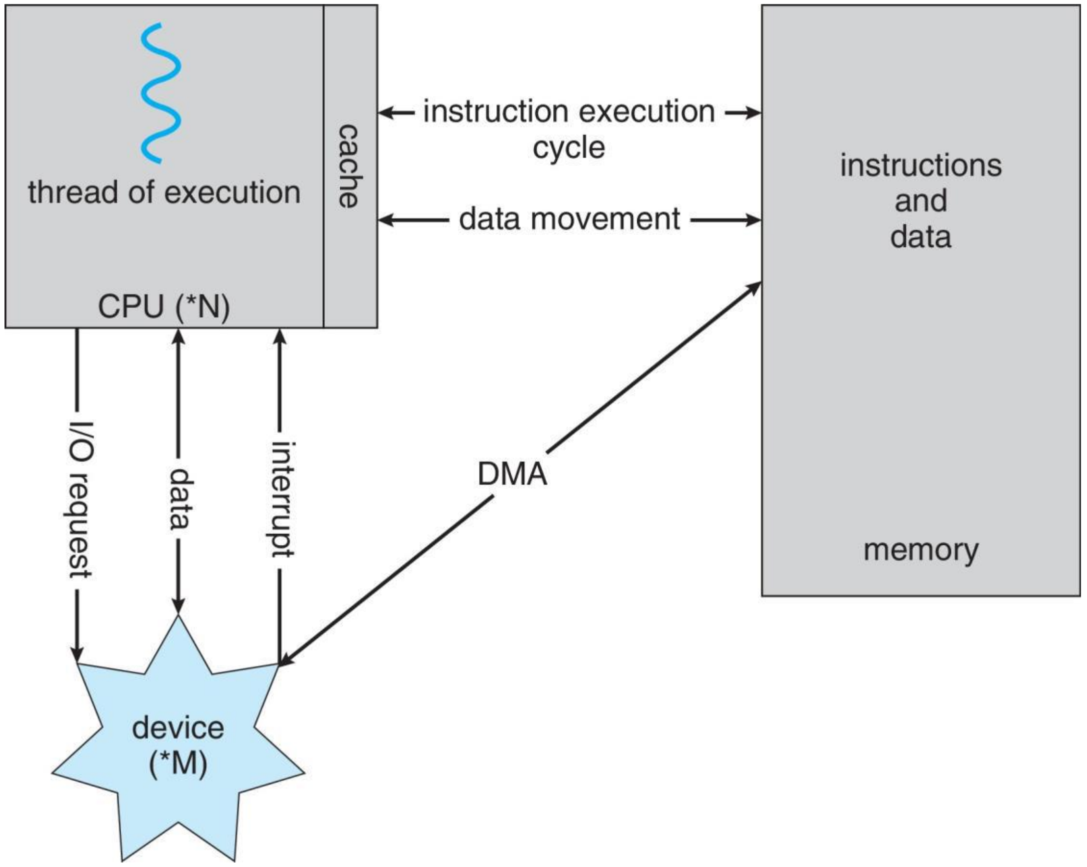


Fig. 1. GPU resource management model.

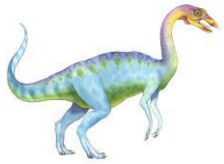
How a Modern Computer Works



A von Neumann architecture

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte
- Can be considered a “CPU” dedicated to executing "memcpy" programs
  - Wasteful to add a general-purpose processor, better to add a simple one
- Several supported memcpy
  - memory → memory, memory → device (register), device (register) → memory
  - Practical implementation: directly connect the DMA controller to the bus and memory
  - PCI bus supports DMA

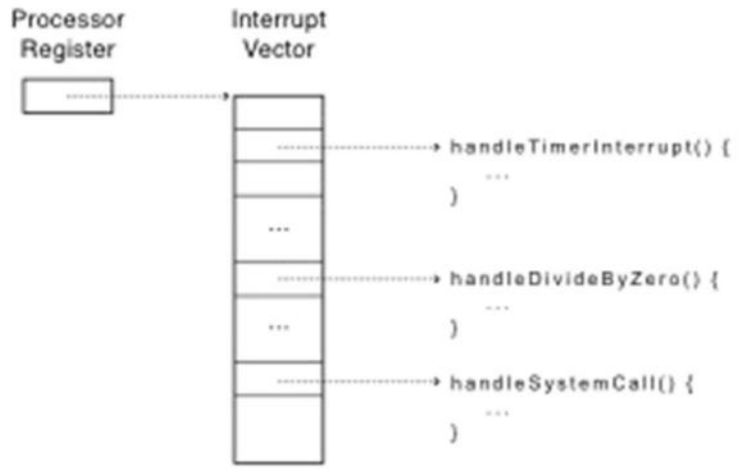


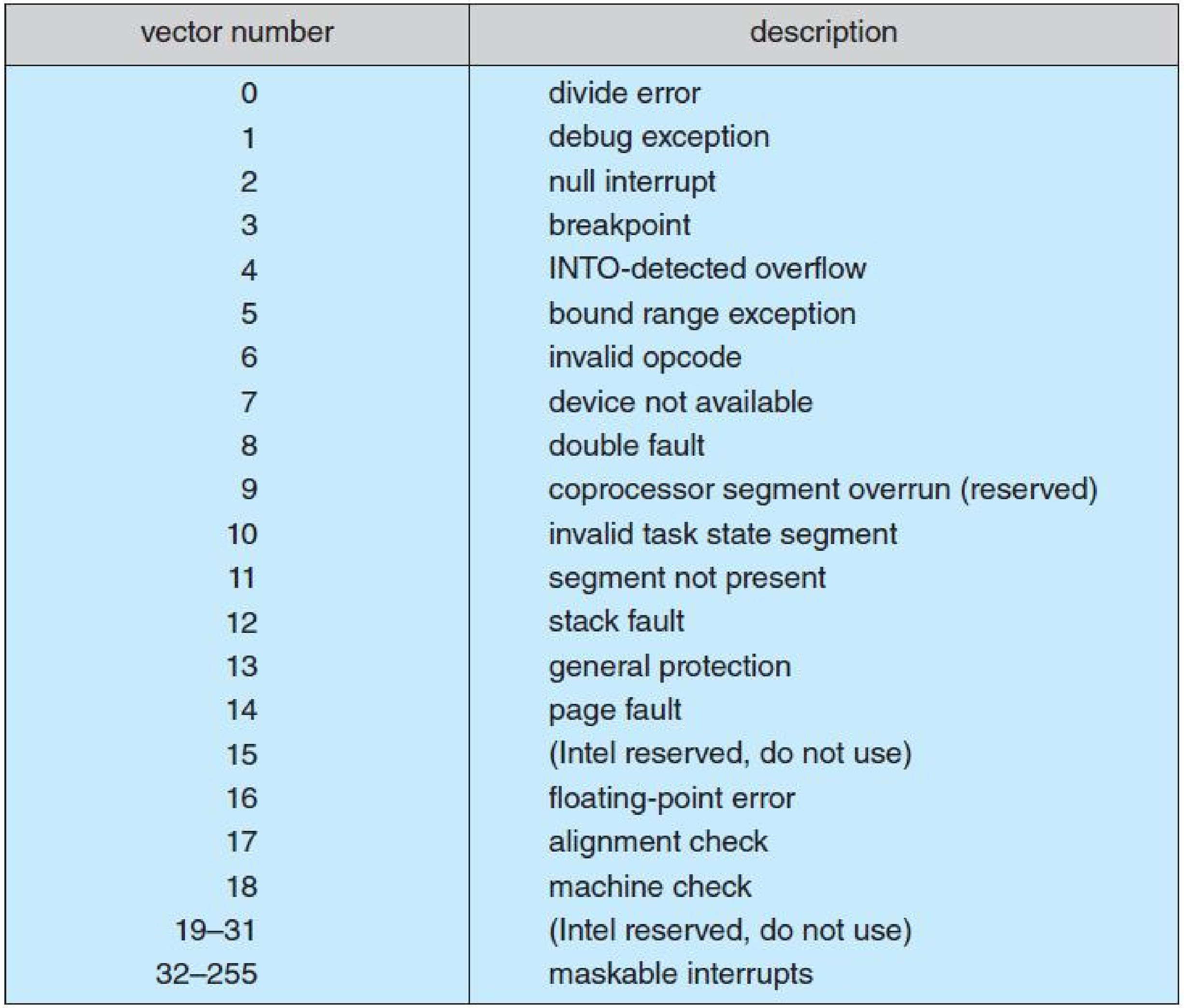
Computer-System Operation

- I/O device such as keyboards have controllers and buffers that store input into the buffers
- Possible ways for the OS to get this input
  - Polling: The OS keeps reading the value in the buffer of devices
  - Interrupt: When the controller received input, alerts the OS/CPU, interrupt the normal execution of the CPU
- **Questions**
  - For keyboards, which way is better? Why?
  - For which scenario, it is better to use "polling"?

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**





vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 1.5 Intel processor event-vector table.

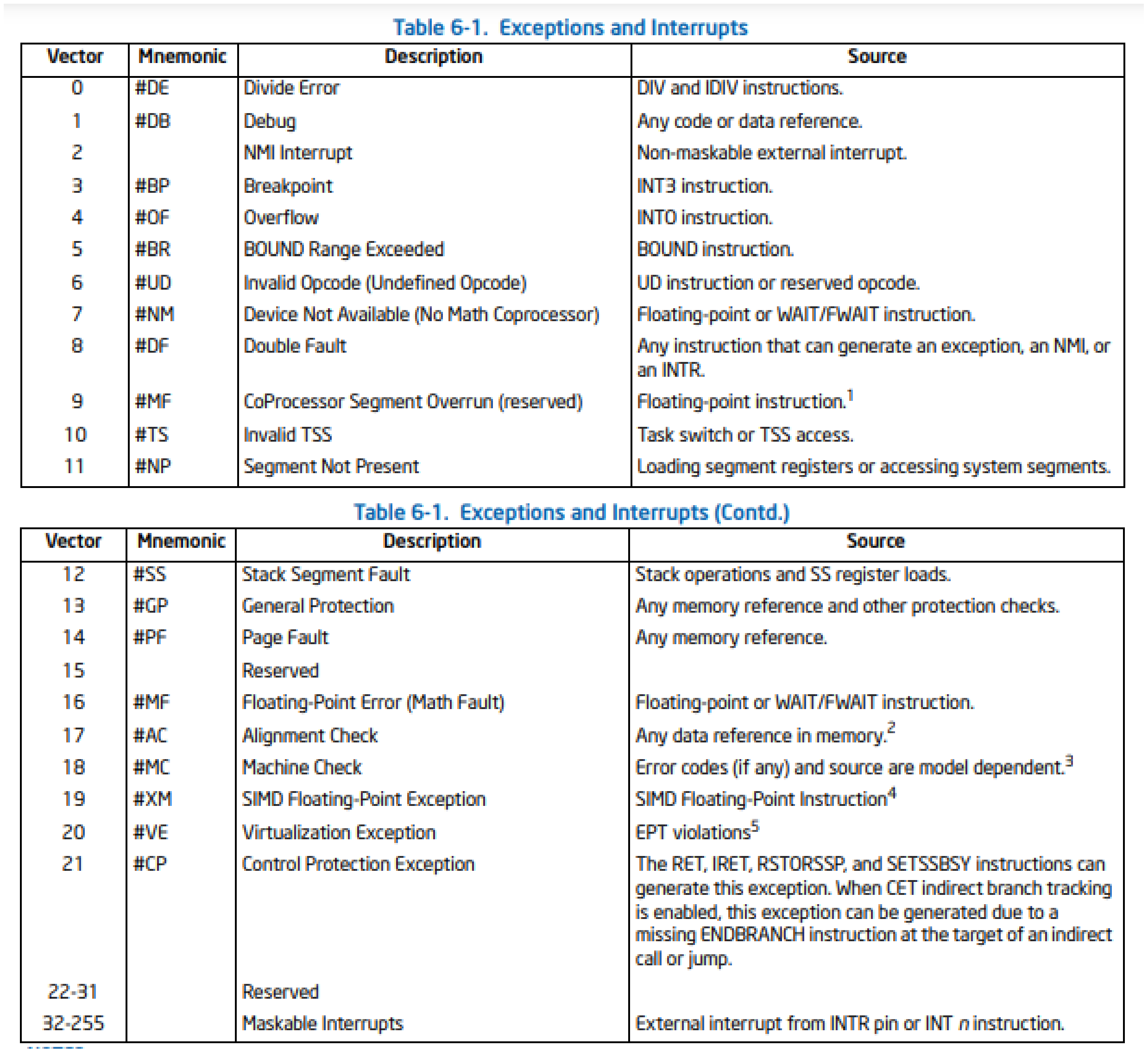
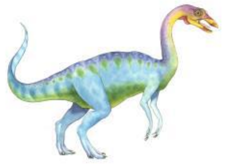


Table 6-1. Exceptions and Interrupts

Vector	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2		NMI Interrupt	Non-maskable external interrupt.
3	#BP	Breakpoint	INT3 instruction.
4	#OF	Overflow	INTO instruction.
5	#BR	BOUND Range Exceeded	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Floating-point instruction.[1]
10	#TS	Invalid TSS	Task switch or TSS access.
11	#NP	Segment Not Present	Loading segment registers or accessing system segments.

Table 6-1. Exceptions and Interrupts (Contd.)

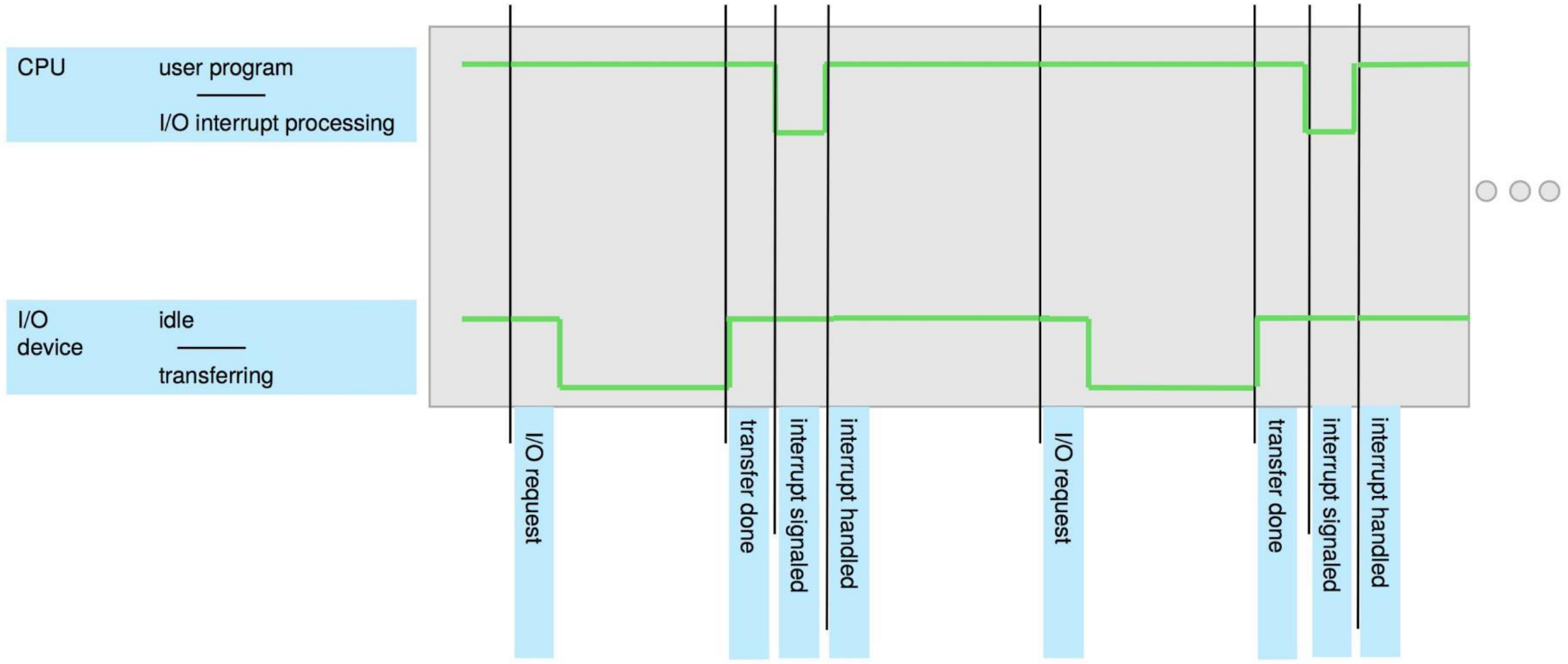
Vector	Mnemonic	Description	Source
12	#SS	Stack Segment Fault	Stack operations and SS register loads.
13	#GP	General Protection	Any memory reference and other protection checks.
14	#PF	Page Fault	Any memory reference.
15		Reserved	
16	#MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Any data reference in memory.[2]
18	#MC	Machine Check	Error codes (if any) and source are model dependent.[3]
19	#XM	SIMD Floating-Point Exception	SIMD Floating-Point Instruction[4]
20	#VE	Virtualization Exception	EPT violations[5]
21	#CP	Control Protection Exception	The RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at the target of an indirect call or jump.
22-31		Reserved	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT *n* instruction.



x86-64 case

- **Interrupt**: Signals generated by external **hardware** devices
  - **Asynchronous**: The cause is not related to the current instruction, such as the program is interrupted by disk read, timer
  - **Maskable**: The signal from the device, connected to the processor through the interrupt controller, can be temporarily blocked (e.g., keyboard, network events)
  - **Unmaskable**: some critical hardware crashes (e.g., memory checksum errors)
- **Exception** : Events generated by program execution of **software**
  - **Fault**: such as page fault exception (recoverable), segmentation fault (unrecoverable), etc.
  - **Trap**: No need to resume, e.g., breakpoint (int 3), system call (int 80 or syscall or…, user programs request services from the operating system)
  - **Abort**: critical error, unrecoverable (machine check)
  - **Synchronous**: The cause is related to the current instruction

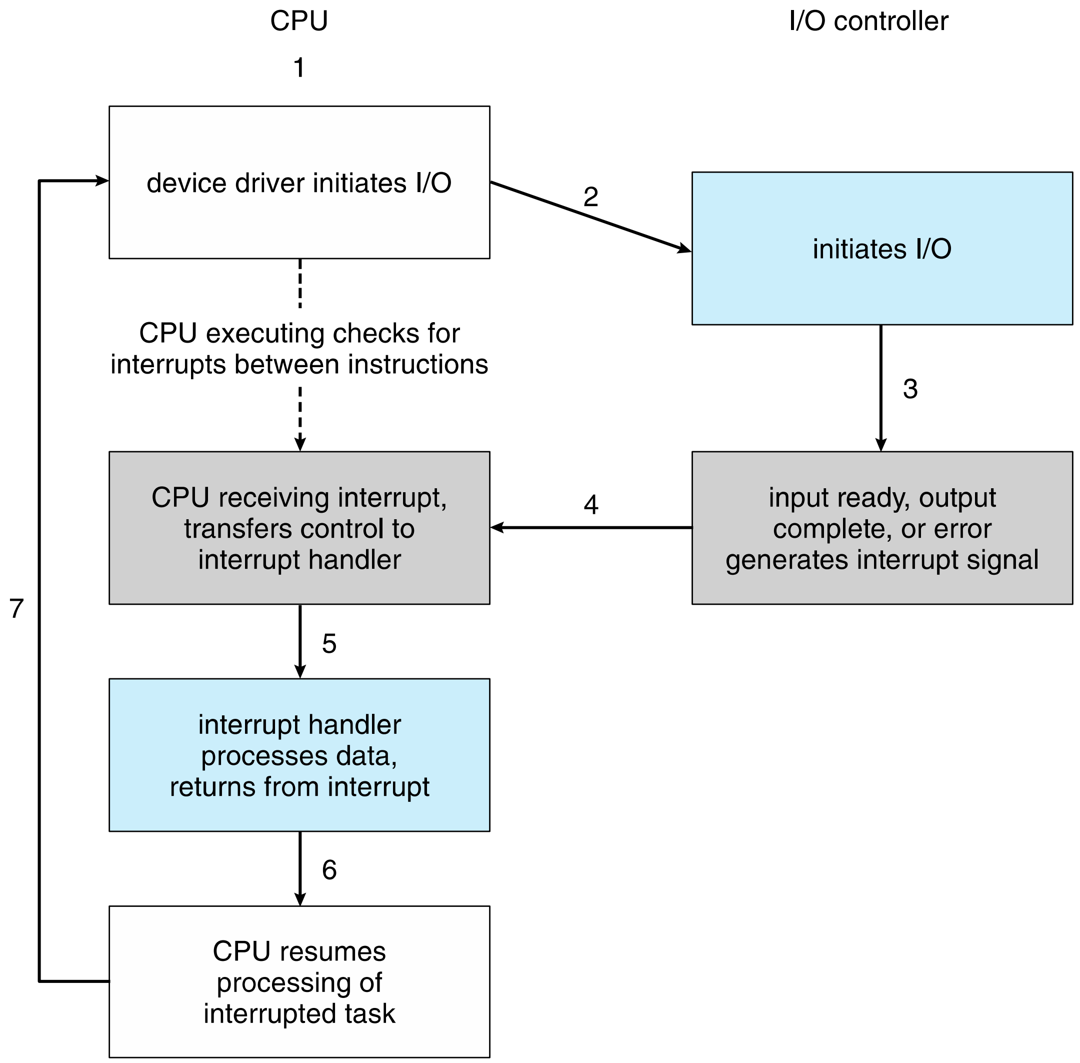
Interrupt Timeline



Interrupt Handling

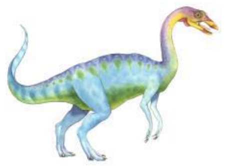
- When there is an interrupt, the operating system preserves the state of the CPU by storing **registers** and **the program counter** for the software that was just interrupted (so that the same software can be restarted later).
- **Determine which type of interrupt** has occurred:
  - **polling**: ask each controller one by one if it is the controller that sent the interrupt (simpler hardware for interrupts but slow);
  - **vectored interrupt system**: each interrupt has a number associated with it that identifies the controller that sent the interrupt. This is the method used on all modern computers.
- Separate segments of code (**interrupt service routines**) inside the kernel determine what action should be taken for each type of interrupt.

Interrupt-drive I/O Cycle


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
4
CPU receiving interrupt, transfers control to interrupt handler
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

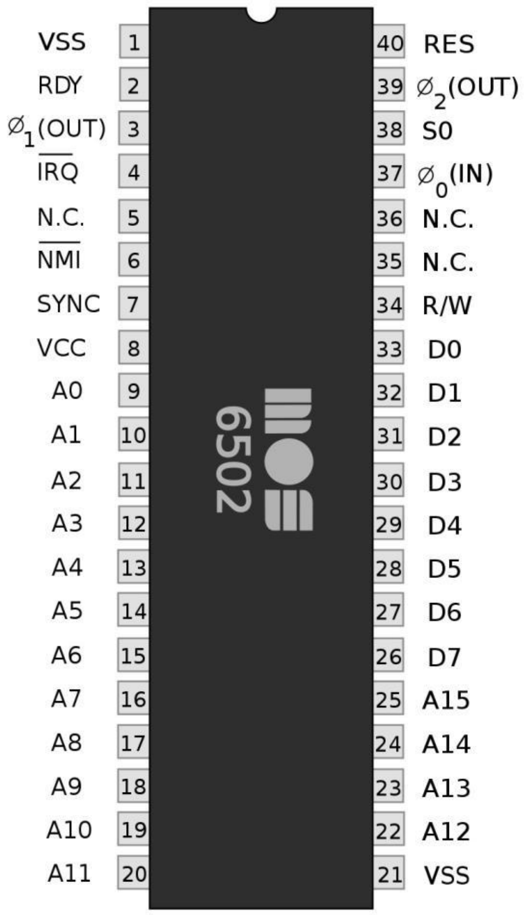
I/O Structure

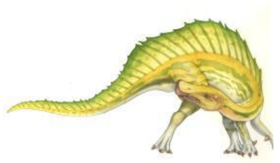
- **Synchronous I/O:** After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- **Asynchronous I/O:** After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS to allow user to wait for I/O completion
  - **Device-status table** contains entry for each I/O device indicating its type, address, and state
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt



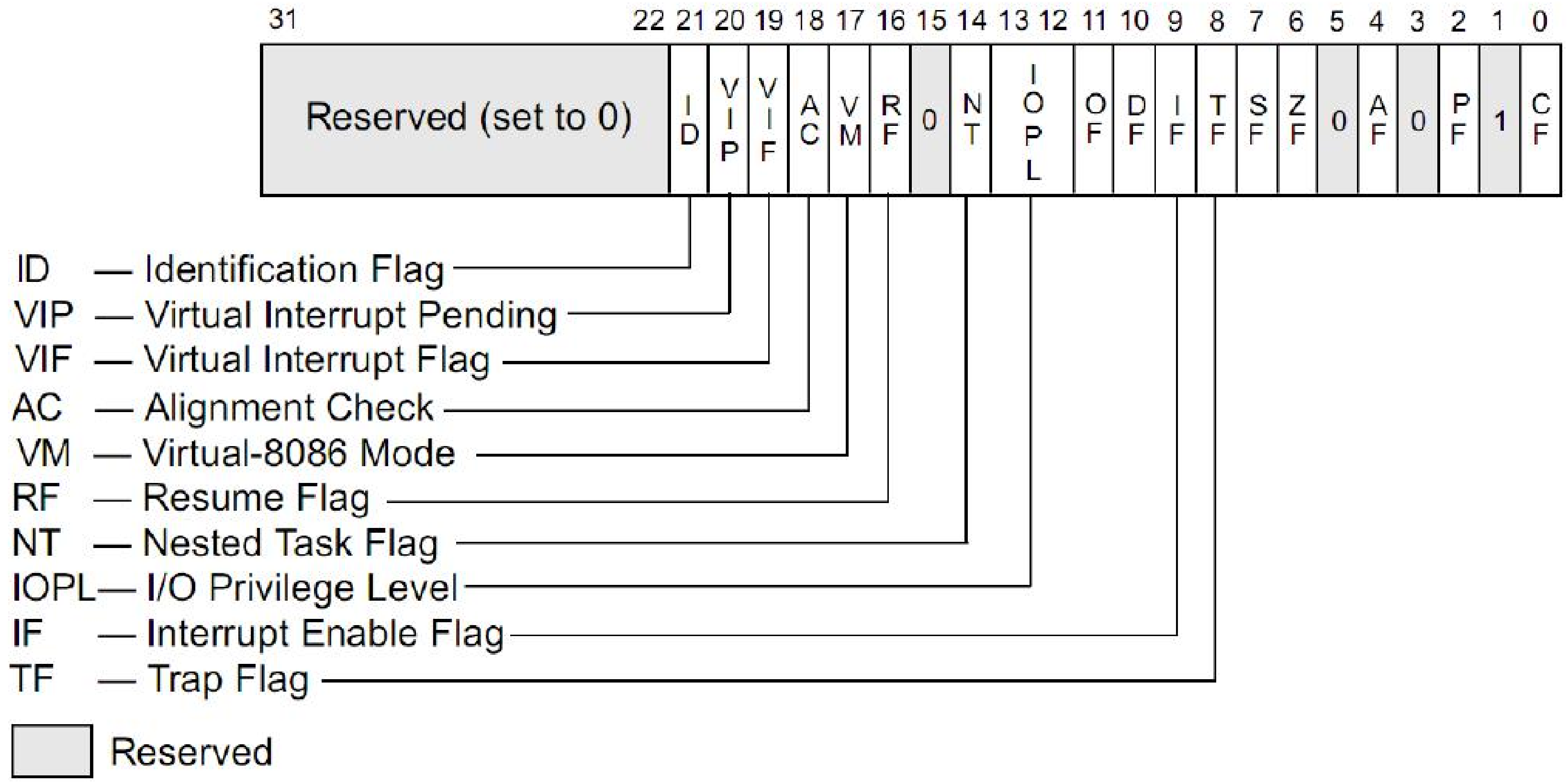
Interrupt Controller

- CPU has an interrupt pin
  - Receiving a specific electrical signal will trigger an interrupt
    - Save 5 registers (cs, eip, eflags, ss, esp)
    - Jump to the corresponding entry in the interrupt vector table to execute
- Other devices can wire to the interrupt controller
  - Intel 8259 PIC
    - Can set interrupt mask, interrupt trigger, etc...
  - APIC (Advanced PIC)
    - Local APIC: interrupt vector table, IPI, clock, …
    - I/O APIC: Other I/O Devices



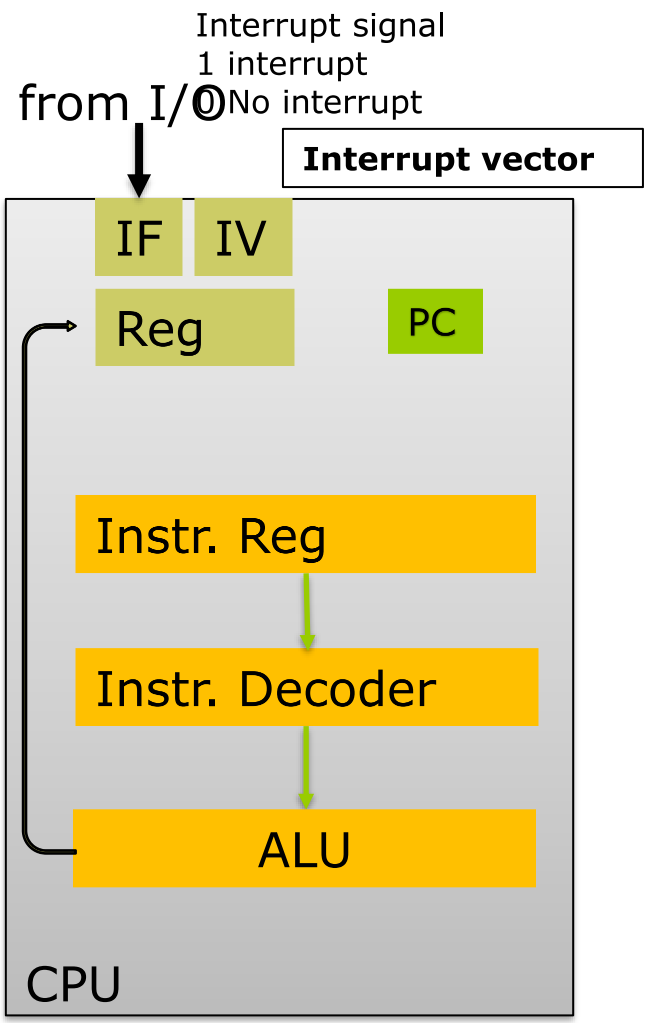


System Flags in the EFLAGS

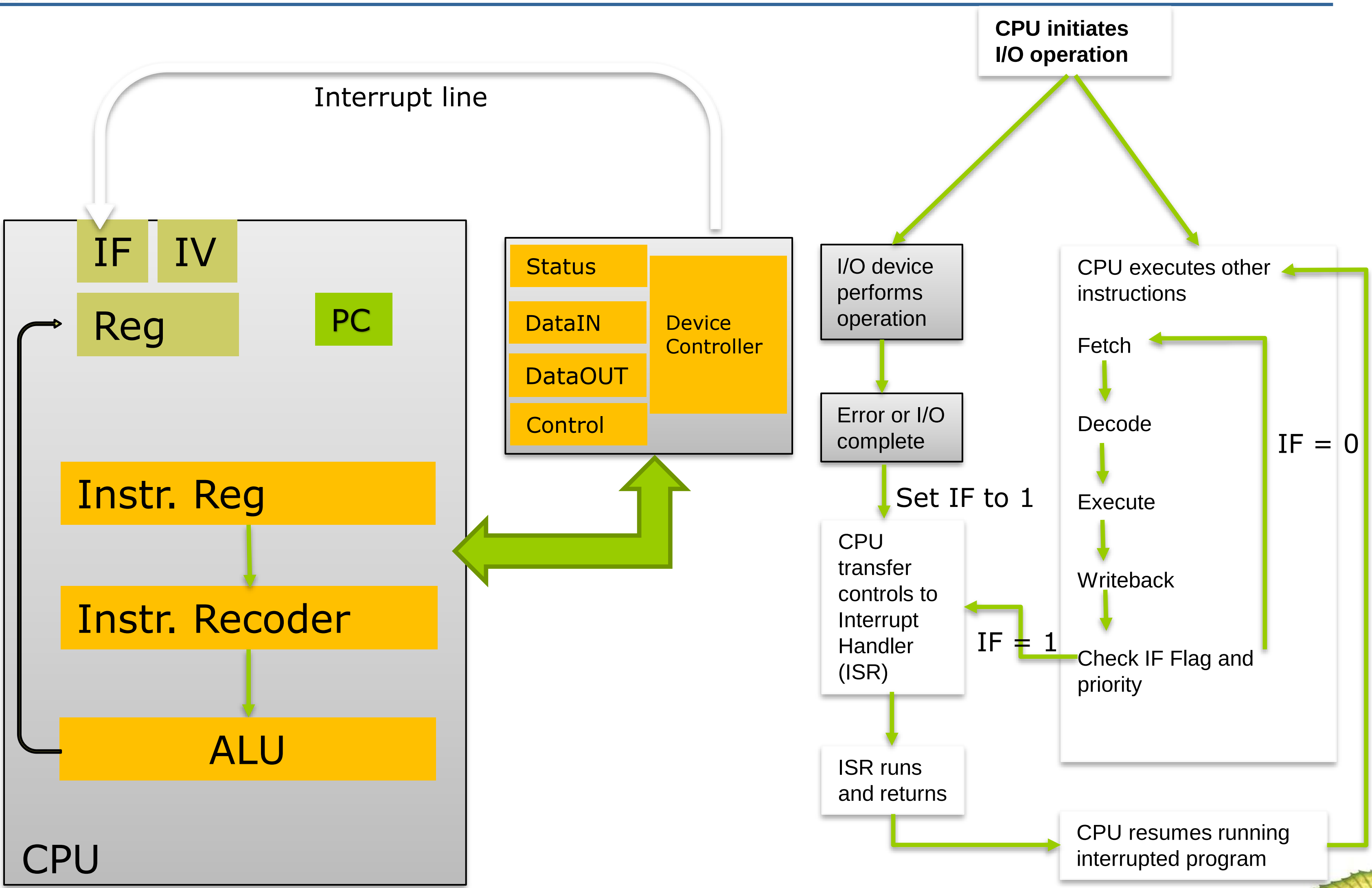


HW Setup for Interrupt Handling

- While waiting for I/O, do other jobs first
- Wait for I/O device to assert interrupt signal
- IF and IV register
  - Interrupt flag
  - Interrupt vector
    - Interrupt **type number**



Interrupt Handling


CPU initiates I/O operation
Interrupt line
IF
IV
Reg
PC
Instr. Reg
Instr. Recoder
ALU
CPU
Status
DataIN
DataOUT
Control
Device Controller
I/O device performs operation
Error or I/O complete
Set IF to 1
CPU transfer controls to Interrupt Handler (ISR)
IF = 1
ISR runs and returns
CPU resumes running interrupted program
CPU executes other instructions
Fetch
Decode
Execute
Writeback
Check IF Flag and priority
IF = 0

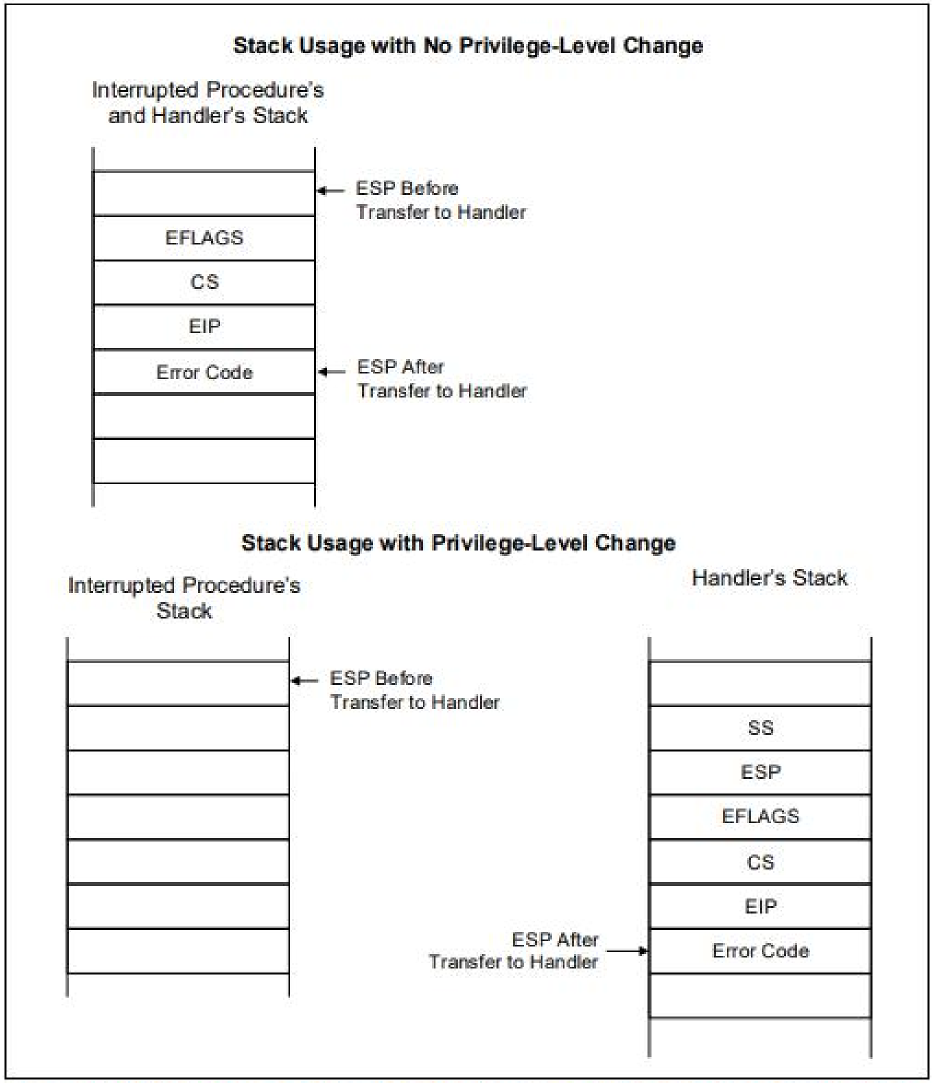
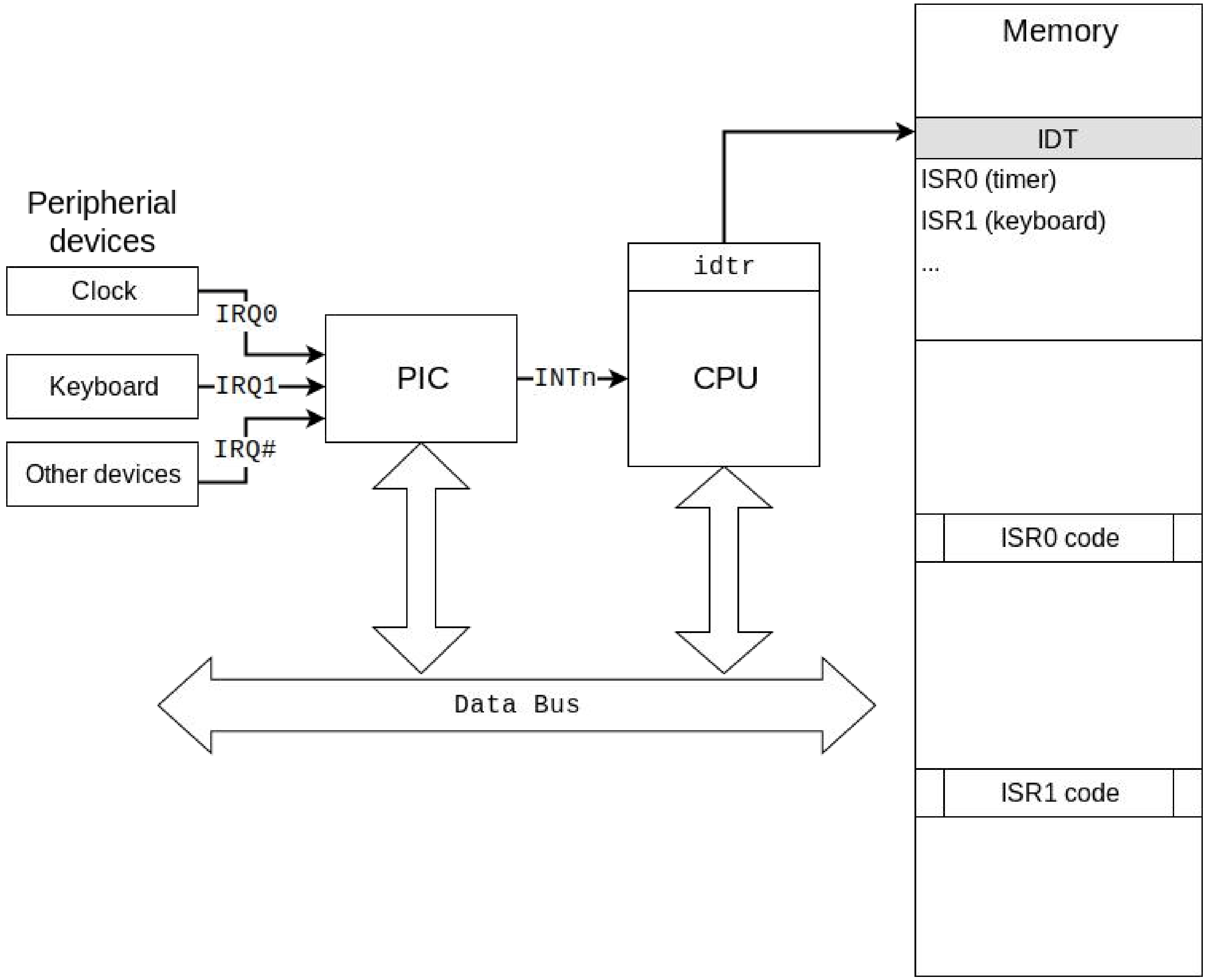
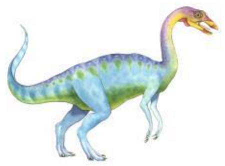

Stack Usage with No Privilege-Level Change
Interrupted Procedure's and Handler's Stack
ESP Before Transfer to Handler
EFLAGS
CS
EIP
Error Code
ESP After Transfer to Handler
Stack Usage with Privilege-Level Change
Interrupted Procedure's Stack
Handler's Stack
ESP Before Transfer to Handler
SS
ESP
EFLAGS
CS
EIP
ESP After Transfer to Handler
Error Code

Figure 6-7. Stack Usage on Transfers to Interrupt and Exception Handling Routines


Memory
IDT
ISR0 (timer)
ISR1 (keyboard)
...
ISR0 code
ISR1 code
Peripherial devices
Clock
Keyboard
Other devices
IRQ0
IRQ1
IRQ#
PIC
INTn
idtr
CPU
Data Bus



Interrupt Controller

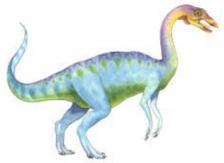
- How to specify the priority of different interrupts
  - A high-priority interrupt occurs during low priority interrupt handling
- Who handles the interruption
- How to work with the software/OS
- For ARM (Aarch 64)
  - IRQ (Interrupt Request)
    - Ordinary interrupt, low priority, slow processing
  - FIQ (Fast Interrupt Request)
    - There can only be one FIQ at a time
    - Fast interrupt, high priority, fast processing
    - Often reserved for trusted interrupt sources
  - SError (System Error)
    - Exceptions whose causes are difficult to locate and are difficult to handle ar
      mostly caused by asynchronous abort (Abort)
    - Such as exceptions that occur when writing back from the cache line to memory

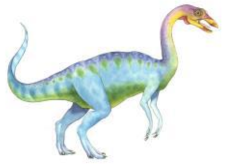
Connect to different CPU pins

Available in the interrupt controller configuration

Operating-System Operations

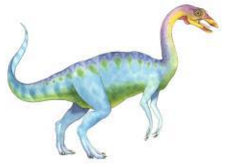
- Bootstrap program – simple code to initialize the system, load the kernel
- Kernel loads
- Starts **system daemons** (services provided outside of the kernel)
- Kernel **interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap**):
    - Software error (e.g., division by zero)
    - Request for operating system service – **system call**
    - Other process problems include infinite loop, processes modifying each other or the operating system





Multiprogramming (Batch system)

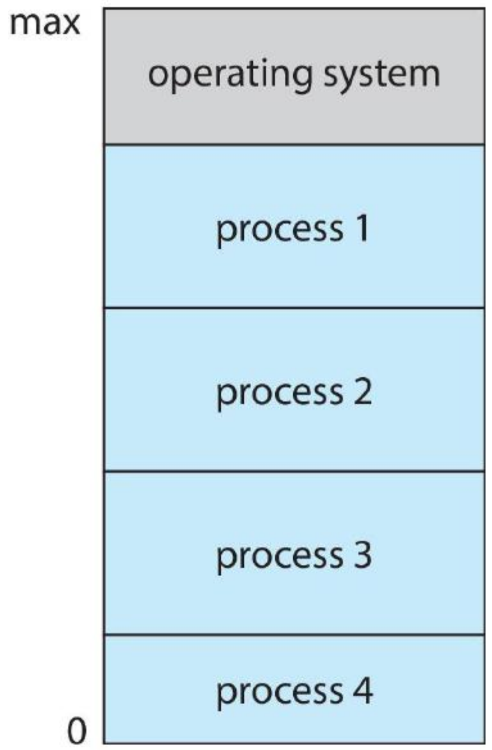
- Single user cannot always keep CPU and I/O devices busy
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When job has to wait (for I/O for example), OS switches to another job



Multitasking (Timesharing)

- A logical extension of Batch systems– the CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨ **process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

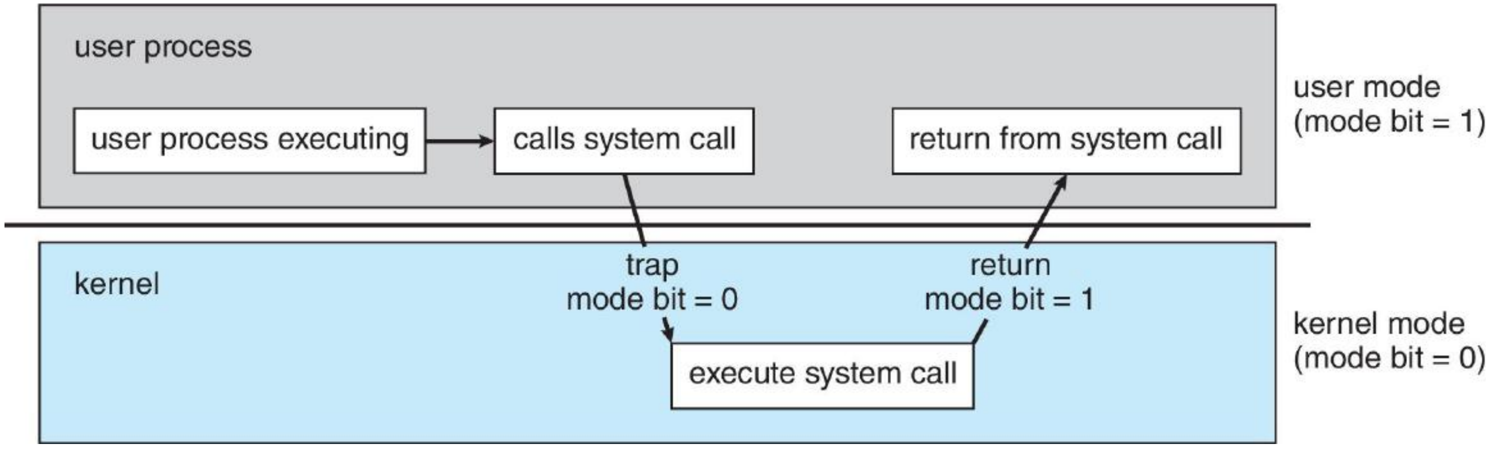
Memory Layout for Multiprogrammed System

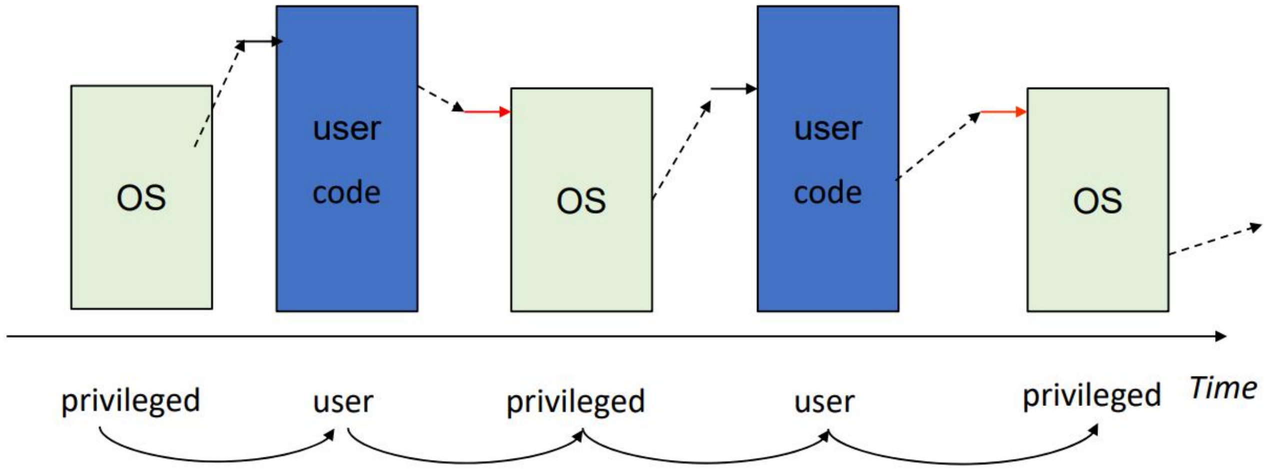


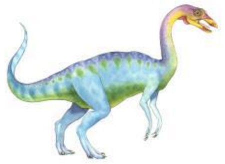
Dual-mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code.
  - When a user is running ⇨ mode bit is “user”
  - When kernel code is executing ⇨ mode bit is “kernel”
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
  - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as **privileged**, only executable in kernel mode

Transition from User to Kernel Mode

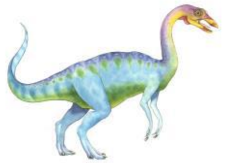






Timer

- Timer to prevent infinite loop (or process hogging resources)
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



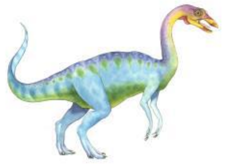
Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity;*** process is an ***active entity***.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

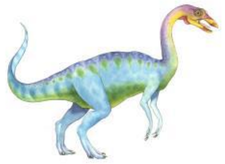
The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling



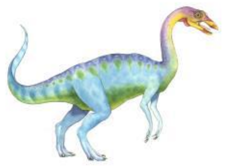
Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed



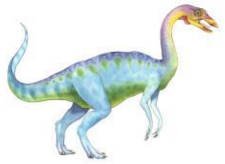
File-system Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media



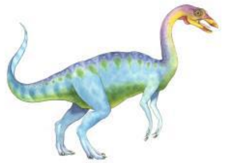
Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Mounting and unmounting
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Partitioning
  - Protection



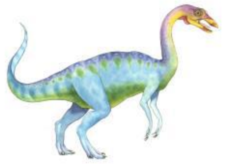
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy



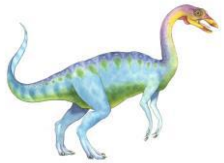
Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights



Virtualization

- Allows operating systems to run applications within other OSes
  - Vast and growing industry
- **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
  - Generally slowest method
  - When computer language not compiled to native code – **Interpretation**
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
  - Consider VMware running WinXP guests, each running applications, all on native WinXP **host** OS
  - **VMM** (virtual machine Manager) provides virtualization services



Computing Environments

- Traditional
- Mobile
- Client Server
- Pear-to-Pear
- Cloud computing
- Real-time Embedded - Vary considerable, special purpose, limited purpose OS, **real-time OS**
  - Use expanding
  - Many other special computing environments as well
    - Some have OSes, some perform tasks without an OS
  - Real-time OS has well-defined fixed time constraints
    - Processing ***must*** be done within constraint
    - Correct operation only if constraints met