Chapter 1

Computer System Overview

Seventh Edition
By William Stallings
To provide a grand tour of the major computer system components:

- Describe the basic elements of computer system and their interrelationship
- Explain the steps taken by a processor to execute an instruction
- Understand the concept of interrupts and how and why a processor uses interrupts.
- Describe the levels of a computer memory hierarchy
- Discuss the concept of locality and analyze the performance of multilevel memory hierarchy
What is an operating system?

- A program that acts as an intermediary between a users/applications and the computer hardware

- A program that controls and manages the execution of application programs issued by user

- A program that manages the hardware

![Diagram of operating system components: User/Application, OS, HW, CPU, Memory, Devices]
Operating System Tasks

- Exploits the **hardware resources** of one or more processors (cores)
- **Manages** main/secondary memory and I/O devices
- Dynamically **allocate** the shared system resources to the executing programs
- Resources in this area is concerned with the management and scheduling of memory, processes, and other devices.
- Provides a set of services (**system calls**) to system users
Operating System Golas

- Operating system functionalities/goals
  - Start/terminate/control **executing** user programs
  - Make the computer system **convenient** to use

- Efficiency: use the computer hardware in an **efficient** manner
  - Control and coordinate use of hardware
    - Perform I/O; setup devices
    - **Allocate resources**
    - Use hardware efficiently
  - Implement **common services**
Place of OS in Computer System

- **Kernel**: running all the time; having most of the functionality of OS
Computer System Organization and Operation
Basic Elements

Processor

Main Memory

I/O Modules

System Bus
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
Top-Level View of computer components

MAR: Specifies the address in memory for the read or write

MBR: contains the data to be written in to memory or read from memory

I/O AR: Specifies a particular I/O device

Figure 1.1 Computer Components: Top-Level View
Processor Registers

★ User-visible registers
- Enable programmer to minimize main-memory references by optimizing register use

★ Control and status registers
- Used by processor to control operating of the processor
- Used by operating-system routines to control the execution of programs
User-Visible Registers

• Available to all programs - application programs and system programs
• Types of registers
  • Data
  • Address
    • Index
    • Segment pointer
    • Stack pointer
Control and Status Registers

- Program Counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction Register (IR)
  - Contains the instruction most recently fetched
- Program Status Word (PSW)
  - condition codes
  - Interrupt enable/disable
  - Supervisor/user mode
Processor

- Controls the operation of the computer
- Performs the data processing functions
- Referred to as the *Central Processing Unit (CPU)*
Main Memory

- Volatile
- Contents of the memory is lost when the computer is shut down
- Referred to as real memory or primary memory
- Why we have two different types of memories?
I/O Modules

Moves data between the computer and external environments such as:

- Storage (e.g. hard drive)
- Communications equipment
- Terminals
System Bus

- Provides communication among processors, main memory, and I/O modules

- What happens if two devices send data into the Bus at the same time?
- Who controls the access to the BUS?
- What is a priority?
- Who determines this priority?
- Is there any rules or algorithm?
OS decides who should use the resources.

It resolves the conflict if several applications are running concurrently in the system if those applications make several conflicting requests for the resources.

OS decides between those conflicting requests considering efficiency (maximum utilization of resources) and fairness.
Instruction Execution
What is a program: A program consists of a set of instructions stored in memory

Program execution (Instruction Processing):

Two steps:
- processor reads (fetches) instructions from memory one at a time
- processor executes each instruction
The processing required for a single instruction is called an instruction cycle.

**Figure 1.2 Basic Instruction Cycle**
Instruction Fetch and Execute

- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
  - PC is incremented after each fetch
Fetched instruction is placed (loaded) into Instruction Register (IR)

- The instruction contains bits that specify the action the processor is to take.
- Processor interprets the instruction and performs required action

- Types of instructions:
  - Processor-memory
  - Processor-I/O
  - Data processing
  - Control
Types of Instructions

- Types of instructions
  - **Processor-memory**
    - transfer data between processor and memory
  - **Processor-I/O**
    - data transferred to or from a peripheral device
  - **Data processing**
    - arithmetic or logic operation on data
  - **Control**
    - alter sequence of execution
Example of Program Execution

load 940

add 941

store 941

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Figure 1.4 Example of Program Execution
(contents of memory and registers in hexadecimal)
Announcements

• **Exercise #1** is already on Piazza!

• Please be active on Piazza and share/discuss your thoughts.

• Due date: Next Monday

• Please submit your solutions via Handin.

• **HW#1** will be out today!

• I will be having Office Hours Friday 2-3 p.m.
Overview
Computer System Overview

Diagram showing the components of a computer system, including:
- CPU
- Disk controller
- USB controller
- Graphics adapter
- Memory
- Mouse
- Keyboard
- Printer
- Monitor

Network cable and network adapter connected to a bus.
A program that acts as an intermediary between a user of a computer and the computer hardware.

Operating system goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to us
- Use the computer hardware in an efficient manner
What Operating Systems Do

- Operating system is is a **resource allocator**:
  - Manages all resources
  - Decides between **conflicting requests** for efficient and fair resource use

- Operating system is a **program controller**:
  - Controls execution of programs to prevent errors and improper use of the computer
Interrupts
I/O Devices Interaction

- I/O devices and the CPU can execute **concurrently**

- Each device controller has a **local buffer**
  - Data movement (I/O) between device and local buffer (by device)
  - Data movement between memory and local buffer (by CPU)

- Device controller informs CPU that it has finished its current operation or it has something

- How we can inform CPU?

- by causing an **interrupt**
Interrupts

- **Definition**: Interrupts are signals sent to the CPU by I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system (handler).

- A suspension of a process caused by an event external to that process and performed in such a way that the process can be resumed.

**Why Interrupts?**

Isn’t it expensive to stop a process, switch to another process, and then resume the suspended process?
Goal of Interrupts

- Provided to improve processor **utilization** (processing efficiency)
  - most I/O devices are slower than the processor
  - processor must pause to wait for device
  - wasteful use of the processor

- Allows the processor to execute other instructions while an I/O operation is in progress
Common Classes of Interrupts

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user’s allowed memory space.</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td>Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.</td>
</tr>
<tr>
<td>I/O</td>
<td>Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.</td>
</tr>
<tr>
<td>Hardware failure</td>
<td>Generated by a failure, such as power failure or memory parity error.</td>
</tr>
</tbody>
</table>

Program:

1- Arithmetic overflow
2- Division by zero
3- Execute illegal instruction
4- Reference outside user’s memory space

interrupt happens as a result of executing an instruction.
Hardware v.s Software Interrupts

classify interrupts from another perspective

- **Hardware** may trigger an interrupt at any time by sending a **signal** to the CPU

- **Software** may trigger an interrupt by executing a special operation called a **system call**
When interrupt occurs, hardware does the following:

- CPU is interrupted
  - at that time application code or kernel code might be running
  - registers and the program counter saved in RAM to preserve CPU state
- CPU starts running the respective Interrupt Service Routing (ISR)
  - (kernel routine)
- ISR is found through interrupt vector
  - (table containing addresses of ISRs)
Software Interrupts

• Running application software may generate interrupts as well.
  – They are called software interrupts (also called traps)
    • 1. exceptions (caused by errors)
    • 2. system calls (service request)

• An operating system (kernel) is interrupt-driven (event driven)
- It is clear that there is some **overhead** involved in this process.
- Extra instructions must be executed (in the interrupt handler) to determine the nature of the interrupt and to decide on the appropriate action.
Interrupts

There is an overhead in using interrupts.

CUP stops the running of the current process, runs the handler and then resumes the suspended process.

does it worth? How interrupts improve the efficiency?
Flow of Control Without 
No Interrupts Service

(a) No interrupts

Sequence of the code

Figure 1.8  Program Timing
Flow of Control with Interrupt Services

The CPU is never idle. Interrupts improve CPU utilization.

Sequence of the code
The user program does not have to contain any special code to accommodate interrupts.

- The processor (CPU) and the OS are responsible for suspending the user program and then resuming it at the same point.

What is the interrupt handler?

Figure 1.6 Transfer of Control via Interrupts
A program that determines nature of the interrupt and performs whatever actions are needed (Determines which types of interrupts has occurred)

Control is transferred to this program

Generally part of the operating system

The OS preserves the state of the CPU by storing registers and the program counter.

[Application in cpu] - interrupt - [service routine in cpu]
Simple Interrupt Processing

Figure 1.10 Simple Interrupt Processing
Interrupt-Driven OS

Applications or System Programs running in CPU

- software interrupt / trap
  (due to system service requests or errors)

Kernel Code

- hardware interrupt

Devices
  disk, keyboard, timer, network adapter...

When OS supports interrupts:
1) implement specific interrupt handler for different type
2) implement module for suspending the current application and resume it
Multiple Interrupts

- An interrupt occurs while another interrupt is being processed.
- A program may be receiving data from a communications line and printing results at the same time:
  - The printer will generate an interrupt every time that it completes a print operation.
  - The communication line controller will generate an interrupt every time a unit of data arrives.

Two approaches:

- Disable interrupts while an interrupt is being processed (sequential)
- Prioritize interrupts: use a priority scheme (nested)
Multiple Interrupts: Sequential Order

- Disable interrupts so processor can complete task
- Interrupts remain pending until the processor enables interrupts
- After interrupt handler routine completes, the processor checks for additional interrupts

Question: any drawback?
Multiple Interrupts: Priorities

- Higher priority interrupts cause lower-priority interrupts to wait.
- Example: when input arrives from communication line, it needs to be absorbed quickly to make room for more input.
Example Time Sequence of Multiple Interrupts

Figure 1.13  Example Time Sequence of Multiple Interrupts
I/O Techniques
Application programs can request I/O (read from a device or write to a device) via the help of OS

- The request is done by calling a System Call (OS routine)

- System call routine in OS performs the I/O via the help of device driver in OS.

- After issuing a system call, an application may wait for the call to finish (blocking call) or may continue to do something else (non-blocking call)

- The I/O module informs the OS when the task is done!

```c
printf("Hello World!\n");
```
Three techniques are possible for I/O operations:

- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access (DMA)
Programmed I/O

- The I/O module performs the requested action (not the processor) then sets the appropriate bits in the I/O status register.
- The I/O module takes NO further action to alert the CPU.
- The processor **periodically checks** the status of the I/O module until it determines the instruction is complete.
- With programmed I/O the performance level of the entire system is **severely degraded**.
- No interrupts occur.
Interrupt-Driven I/O

Processor issues an I/O command to a module and then goes on to do some other useful work.

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor.

The processor executes the data transfer and then resumes its former processing.

More efficient than Programmed I/O but still requires active intervention of the processor to transfer data between memory and an I/O module.
Interrupt-Driven I/O

• Processor is interrupted when I/O module ready to exchange data

• Processor is free to do other work

• No needless waiting

• Consumes a lot of processor time because every word read or written passes through the processor
**Interrupt-Driven I/O**

**Drawbacks**

- I/O Transfer rate is limited by the speed with which the processor can test and service a device
- The processor is tied up in managing an I/O transfer
  - a number of instructions must be executed for each I/O transfer
Direct Memory Access (DMA)

* Performed by a separate module on the system bus or incorporated into an I/O module

* Processor is involved only at the beginning and end of the transfer

* An interrupt is sent when the task is complete

When the processor wishes to read or write data it issues a command to the DMA module containing:

- whether a read or write is requested
- the address of the I/O device involved
- the starting location in memory to read/write
- the number of words to be read/
• With DMA, device controller **transfers blocks of data** from buffer storage directly to main memory **without CPU intervention** so I/O exchanges occur directly with memory.

• Processor grants I/O module authority to read from or write to memory.
• Relieves the processor responsibility for the exchange.
• Processor is free to do other things.
• Transfers the block of data directly to and from memory without going through the processor.
  – Only **one interrupt is generated per block**, rather than the one interrupt per byte.
Memory Hierarchy and Caching
• **Main memory** – memory that the CPU can access directly

• **Secondary storage** – extension of main memory that provides large nonvolatile memory capacity
Trade-offs

Memory considerations:

- how much?  => as much as possible 😊
- how fast?  => to keep up with the CPU 😊
- how expensive?  => reasonable considering other two 😐

Trade-off:

capacity
access time
cost

Question?
what would be a good memory configuration?
Memory Hierarchy

- Why Memory systems organized in hierarchy?
  - Memory must be able to keep up with the processor
  - Cost of memory must be reasonable in relationship to the other components
The Memory Hierarchy

- Going down the hierarchy:
  - decreasing cost per bit
  - increasing capacity
  - increasing access time
  - decreasing frequency of access to the memory by the processor

Major constrains in memory

- Speed
- Expense Cost
- Volatility
- Amount
There is a trade-off among the three key characteristics of memory, namely **capacity**, **access time**, and **cost**.

- Faster access time = greater cost per bit
- Greater capacity = smaller cost per bit
- Greater capacity = slower access speed
• **Caching** – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage
  – Results from tradeoff between size and speed
• performed at **many levels** in a computer
  (in hardware, operating system, software)
• Cache is checked first for an item cached
Why Caching?

- Processor must access memory at least once per instruction cycle
- Processor execution is limited by memory cycle time
- Mismatch between processor and main memory speeds
- Processor speed has consistently increased more rapidly than memory access speed
- Main memory should be built with the same technology as that of the processor
- Expensive strategy
- Exploit the principle of locality with a small, fast memory between CPU and main memory: **Cache idea**
Cache Memory

- Increase the speed of memory
- Processor speed is faster than memory

small, fast memory
Cache Principles

- Contains a copy of a portion of main memory
- Processor first checks cache
- If not found, a block of memory is read into cache
- Because of locality of reference, it is likely that many of the future memory references will be to other bytes in the block
Cache Read Operation

Figure 1.18 Cache Read Operation
Principle of Locality

- Memory references by the processor tend to cluster
- Data is organized so that the percentage of accesses to each successively lower level is substantially less than that of the level above
- Can be applied across more than two levels of memory
Types of Locality

Processor access instructions/data sequentially...

- **Spatial locality:** tendency of execution to involve a number of memory locations that are clustered

- **Temporal locality:** tendency for a processor to access memory locations that have been used recently
Locality

When an iteration (for) loop is executed...

A. Spatial locality: tendency of execution to involve a number of memory locations that are clustered

B. Temporal locality: tendency for a processor to access memory locations that have been used recently
How to Exploit Locality?

- Spatial locality: use larger cache and pre-fetching
- Temporal locality: keep recently used instruction/data in cache and exploit cache hierarchy
**Cache Design**

<table>
<thead>
<tr>
<th>Key elements are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache size</td>
</tr>
<tr>
<td>block size</td>
</tr>
<tr>
<td>mapping function</td>
</tr>
<tr>
<td>replacement algorithm</td>
</tr>
<tr>
<td>write policy</td>
</tr>
<tr>
<td>number of cache levels</td>
</tr>
</tbody>
</table>

**Table:**

- **cache size**
- **block size**
- **mapping function**
- **replacement algorithm**
- **write policy**
- **number of cache levels**
Cache and Block Size

**Cache Size**
- Small caches have significant impact on performance

**Block Size**
- The unit of data exchanged between cache and main memory

- **Hit** means the information was found in the cache

- Larger block size yields more hits until probability of using newly fetched data becomes less than the probability of reusing data that has been moved out of cache
Mapping Function

* When a new block of data is read in to the cache which cache location the block will occupy?

Two constraints affect design:

- When one block is read in, another may have to be replaced
- The more flexible the mapping function, the more complex is the circuitry required to search the cache
Summary

- Basic Elements
  - processor, main memory, I/O modules, system bus
  - GPUs, SIMD, DSPs, SoC
  - Instruction execution
    - processor-memory, processor-I/O, data processing, control
  - Interrupt/Interrupt Processing
  - Memory Hierarchy
  - Cache/cache principles and designs
Computer System Architecture