Computer Systems Overview

Dr. Ailbhe Gill

Textbook: Operating Systems: Internals and Design Principles, 9th Edition by William Stallings

Outline

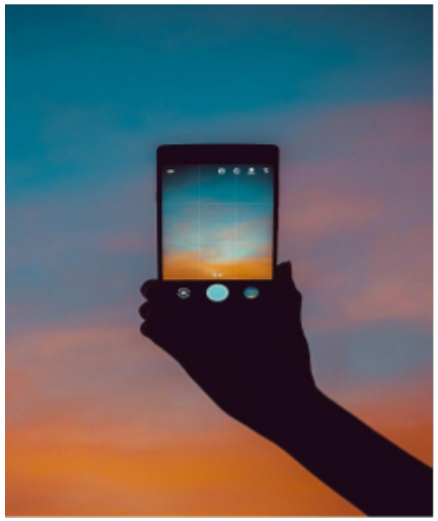
- Basic Elements of a Computer System
- Instruction Execution
- Interrupts
- Memory Hierarchy
- Multiprocessor and Multicore Organization

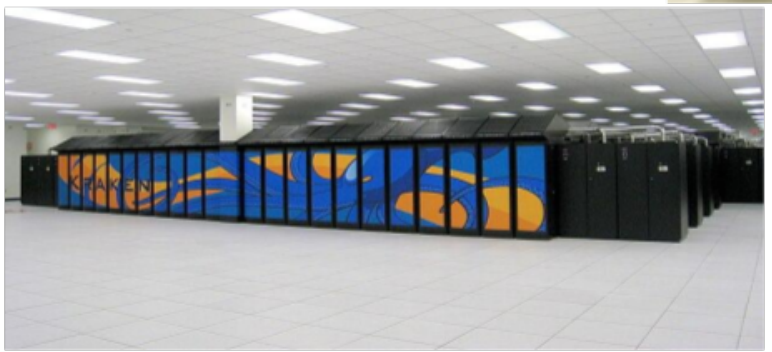
Learning Objectives

- Describe the basic elements of a computer system and their interrelationship.
- Understand the concept of interrupts, and how and why a processor uses interrupts.
- Explain the basic characteristics of multiprocessor systems and multicore
- List and describe the levels of a typical computer memory hierarchy. computers.
- Discuss the concept of locality and analyze the performance of a multilevel memory hierarchy.
- Understand the operation of a stack and its use to support procedure call and return.

Basic Elements of a Computer System

Computing Devices Everywhere





Computer System Structure

Computer system can be divided into four components:

Hardware: Provides basic computing resources

- Processor (CPU), Main memory, I/O module, System bus

Operating system: Controls and coordinates the use of hardware among various applications and users

System and application programs: Define the ways in which the system resources are used to solve the computing problems of the users

- Word processors, compilers, web browsers, database systems, video games

Users:

- People, machines, other computers

Computer System Structure

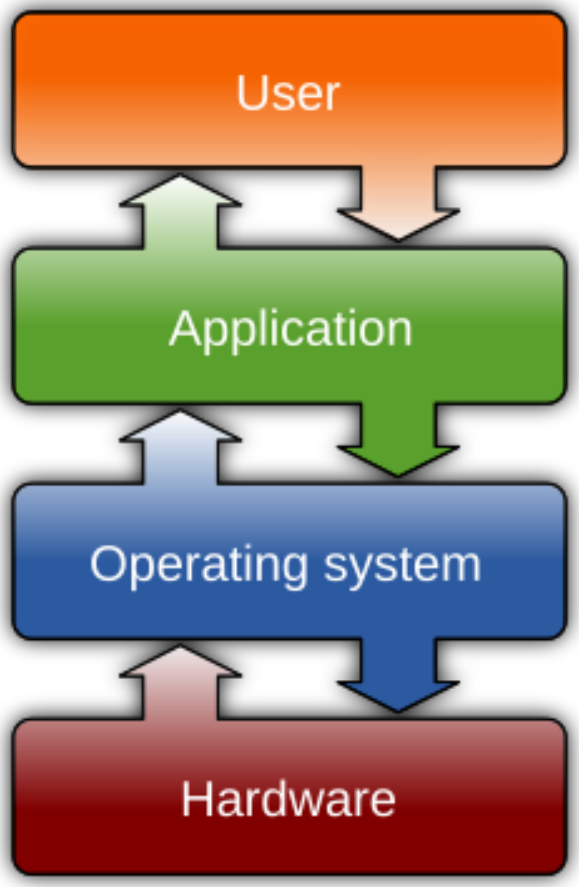


Figure 2: Components of a Computer System

[Golftheman, Operating system placement, CC BY-SA 3.0]

Computer Hardware Components

Let's start at the lowest level with a review of the components of hardware. These are:

- Processor (CPU)
- Main Memory
- I/O Module
- System Bus

Computer Hardware Components

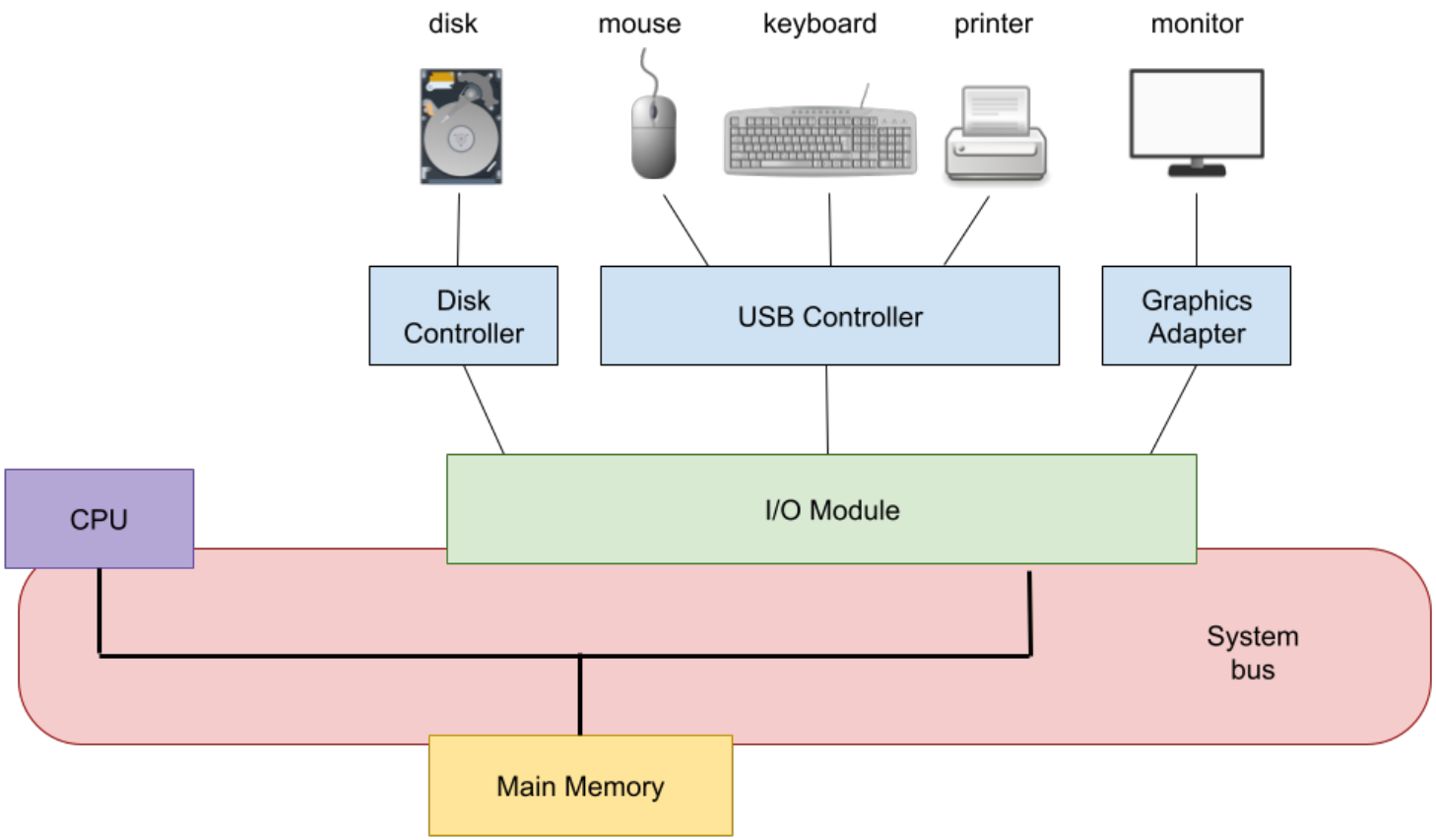


Figure 3: Computer Hardware Components

Processor

Referred to as the Central Processing Unit (CPU) when there is only one processor.

Processor functionality

- Controls the operation of the computer
- Performs the data processing functions

Main Memory

Also referred to as real memory or primary memory

Main Memory functionality

Stores data and programs

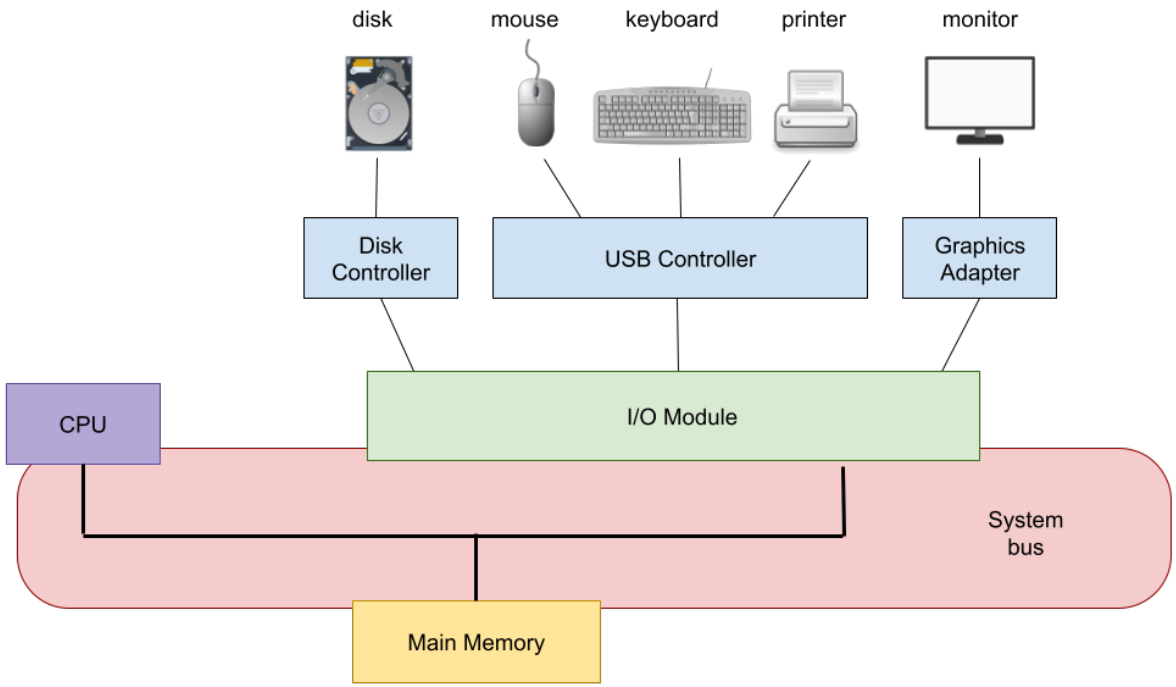
Typically Volatile; Contents of the memory are lost when the computer is shut down.
(In contrast, the contents of disk memory are retained when the computer system is shut down)

I/O Module

I/O Module functionality

Moves data between the computer and its external environment

- Secondary memory devices (e.g. disks)
- Communications equipment
- Terminals



System Bus

System Bus functionality

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

Hardware Components in Detail

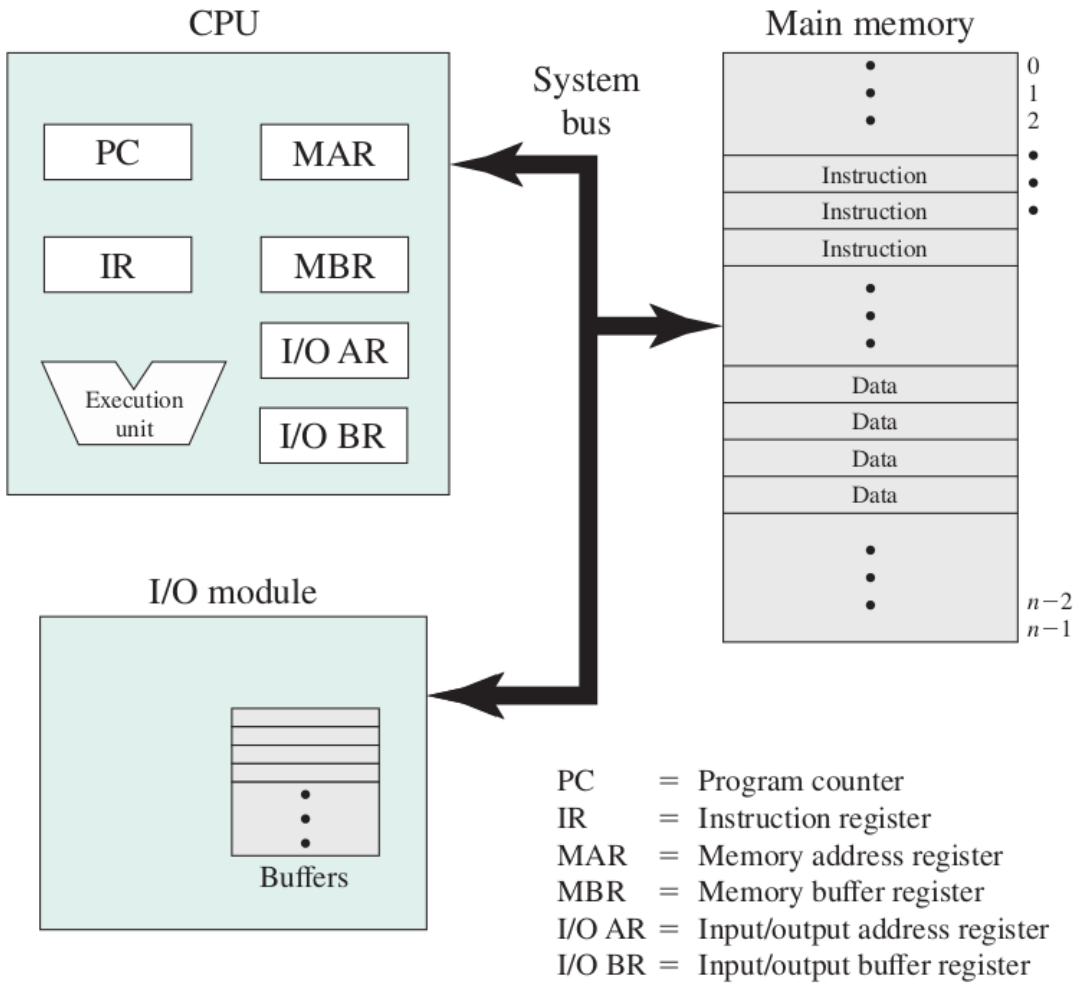


Figure 4: Computer Components: Top-Level View

CPU Registers

To exchange data with memory, the CPU typically makes use of two internal (to the processor) registers:

- A **memory address register (MAR)**, which specifies the address in memory for the next read or write.
- A **memory buffer register (MBR)**, which contains the data to be written into memory, or receives the data read from memory.

It has a further two registers to interact with I/O devices:

- An **I/O address register (I/OAR)** specifies a particular I/O device.
- An **I/O buffer register (I/OBR)** is used for the exchange of data between an I/O module and the processor.

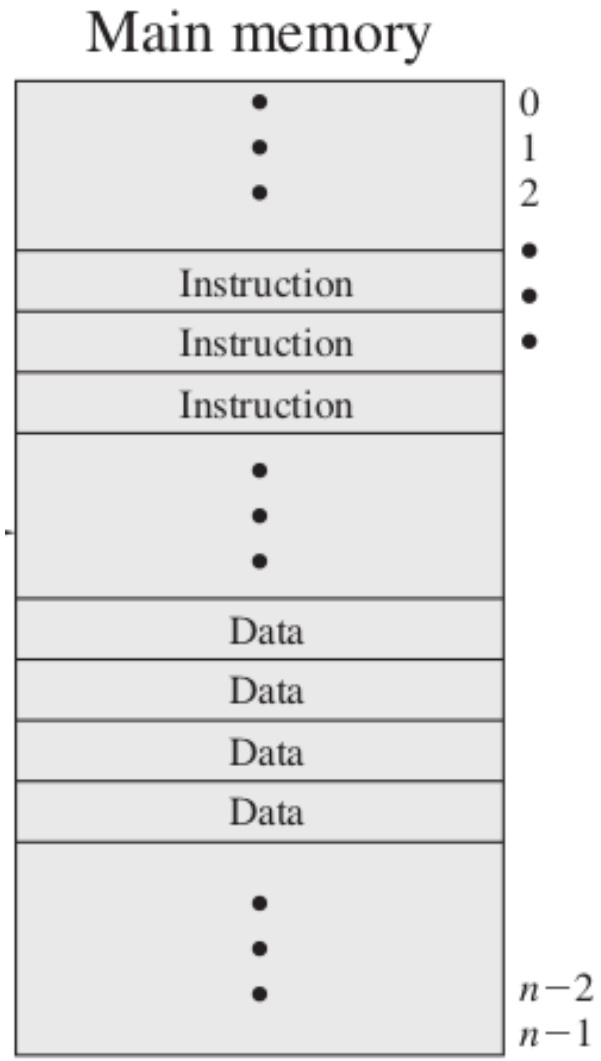
CPU Registers

Important registers for fetching and executing instructions:

- **Program counter**, which contains the address of the next instruction.
- **Instruction Register**, which contains the instruction currently being executed.
- **Accumulator (AC)**, which is used for temporary storage

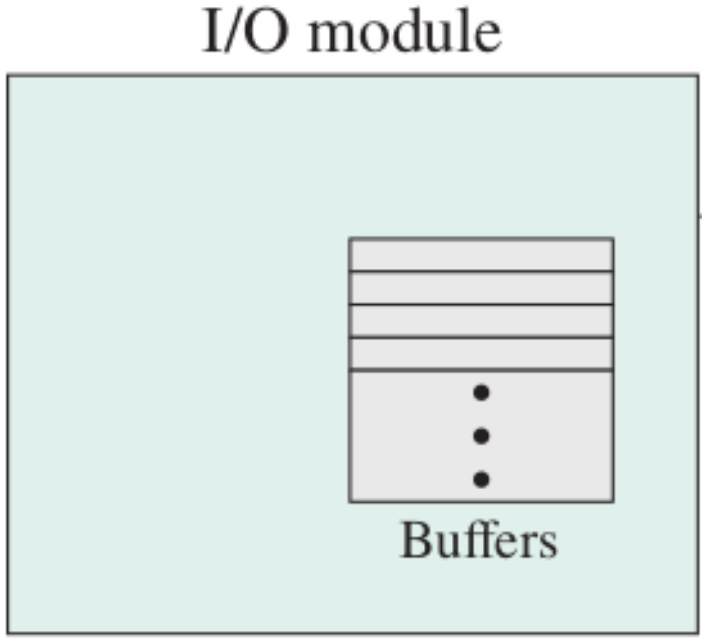
Main Memory

Main Memory consists of a set of locations, defined by sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as either an instruction or data.



I/O Module

An **I/O module** transfers data from external devices to a processor and memory, and vice versa. It contains internal buffers for temporarily storing data until they can be sent on.



Microprocessor, Multiprocessor and Multicore

Microprocessor

Microprocessor

Contains a processor (or at least one) on a single chip

- Invention that brought about desktop and handheld computing
- They are now multiprocessors
  - Each chip (socket) may contain multiple processors (cores)
  - A laptop could have for example 2 or 4 cores, each with 2 hardware threads, for a total of 4 or 8 logical processors. These devices are known as multicore computers.

Multicore Computer

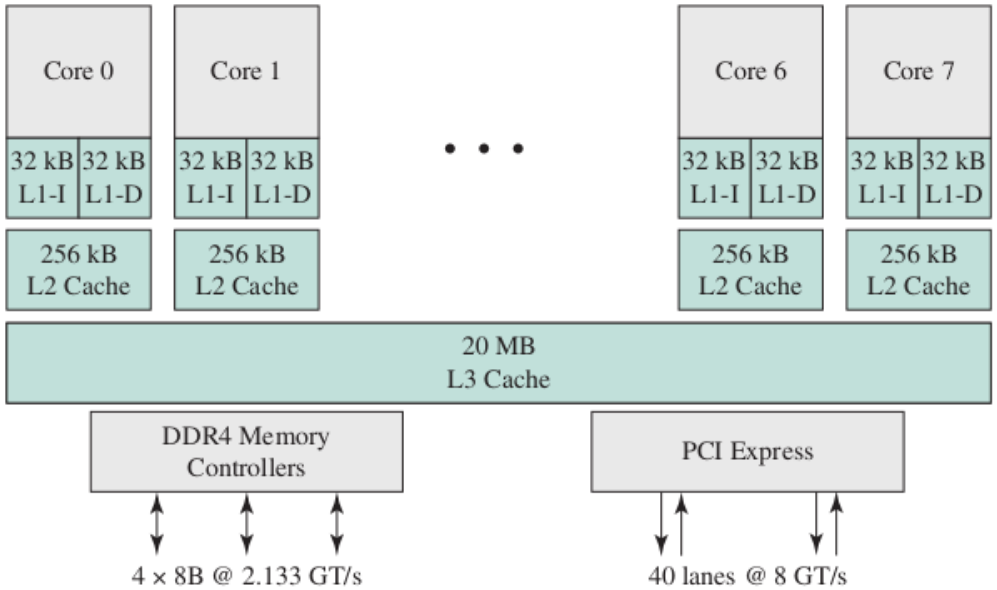
Also known as a "chip multiprocessor"

Multicore Computer properties

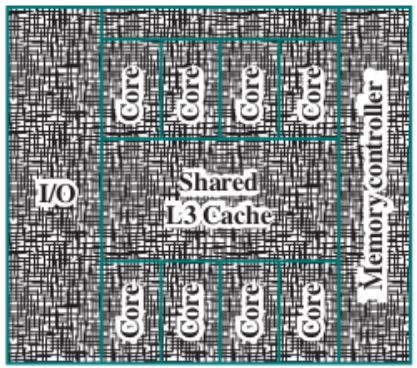
Combines two or more processors (cores) on a single piece of silicon (die)

- Each core consists of all of the components of an independent processor
- Multicore chips also include L2 cache and in some cases L3 cache

Core Block Diagram and Layout on Chip



(a) Block diagram



(b) Physical layout on chip

Figure 5: Intel Core i7-5960X Block Diagram

Symmetric Multiprocessors (SMP)

Symmetric Multiprocessors

Stand-alone computer system with two or more similar processors of comparable capability.

- Processors share
  - the same main memory and are interconnected by a bus or other internal connection scheme
  - access to I/O devices
- All processors can perform the same functions
- The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data elements

Symmetric Multiprocessor Organization

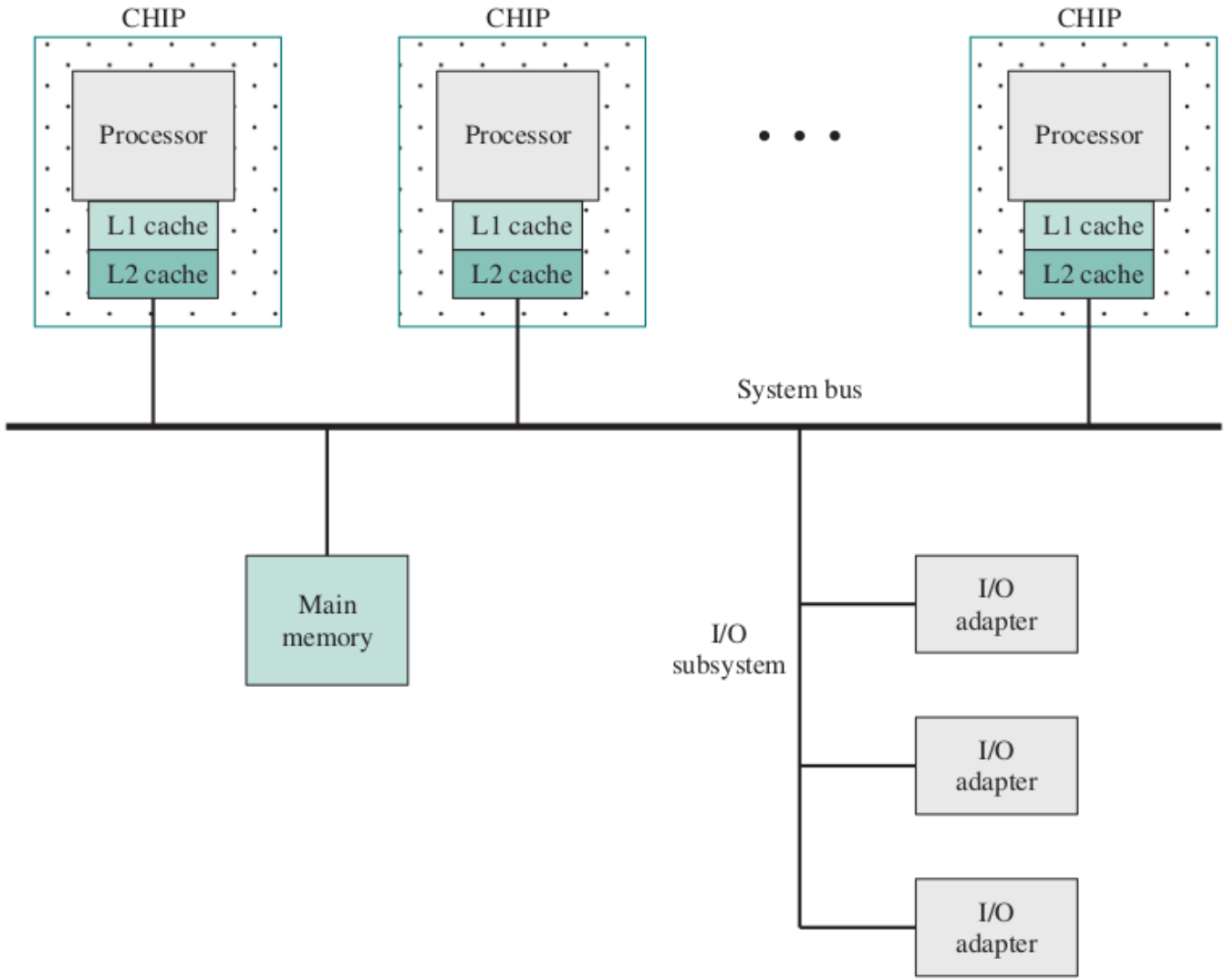


Figure 6: Symmetric Multiprocessor Organization

Multiprocessor Advantages

Advantages of multiprocessors over uniprocessors.

Multiprocessor Advantages

1. **Performance:**

   A system with multiple processors will yield greater performance if work can be done in parallel

2. **Scaling:**

   Vendors can offer a range of products with different price and performance characteristics

3. **Availability:**

   The failure of a single processor does not halt the machine

4. **Incremental Growth:**

   An additional processor can be added to enhance perfromance

Interrupts

Interrupts

Interrupts Definition

Mechanism by which other modules may interrupt the normal sequencing of the processor

Purpose: to improve processor utilization

- Most I/O devices are slower than the processor
- Processor must pause to wait for device
- Wasteful use of the processor

Classes of Interrupts

Program	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, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Flow of Control Without and With Interrupts

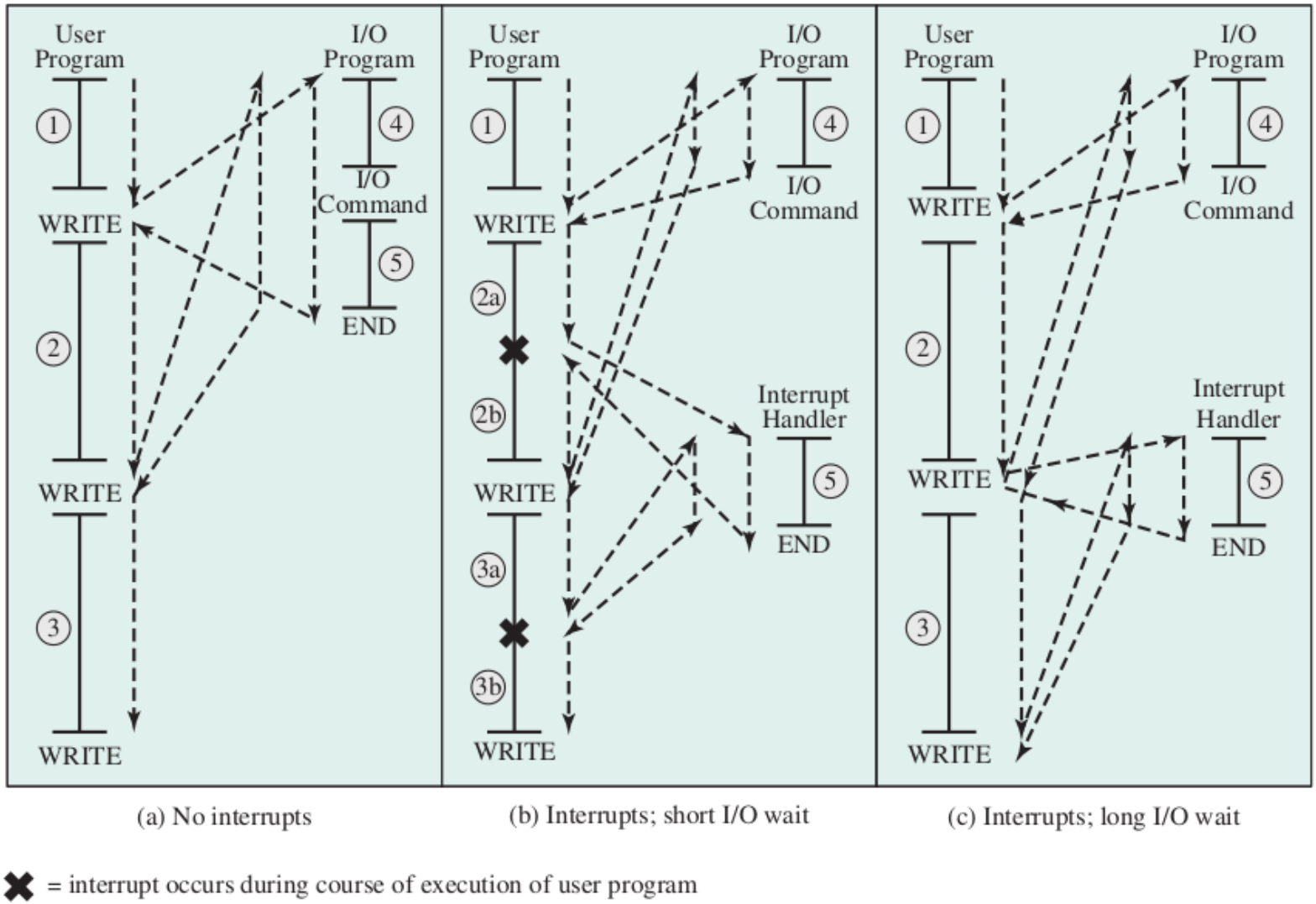


Figure 7: Program Flow of Control Without and With Interrupts

Flow of Control Without and With Interrupts

Consider a program counter (PC) that operates at 1 GHz, which would allow roughly 109 instructions per second. A typical hard disk has a rotational speed of 7200 revolutions per minute for a half-track rotation time of 4 ms, which is 4 million times slower than the processor.

Problem

The user program performs a series of "write" calls interleaved with processing. The "write" calls are to an I/O routine that will perform the actual I/O operation.

The figure in the previous slide shows the flow of control with and without interrupts for this problem.

- Solid vertical lines represent segments of code in a program.
- Dashed line represents the path of execution followed by the processor i.e. the sequence in which instructions are executed.
- The points at which such interrupts occur are indicated by X.

Flow of Control Without and With Interrupts

The **user program** consists of code segments 1, 2, and 3. Each of these refer to a sequence of instructions that do not involve I/O.

The **I/O program** consists of three parts:

1. Code segment 4 refers to a sequence of instructions to prepare for actual I/O operation
   - e.g. copying data to a buffer, preparing parameters for a device command
2. The I/O command execution
3. Code segment 5 refers to a sequence of instructions to complete the I/O operation.
   - e.g. setting a flag indicating success or failure of the operation.

Flow of Control Without Interrupts

Flow of Control **Without** Interrupts

1. After the first "write" instruction is encountered, the user program is interrupted and execution continues with the I/O program.
2. After the I/O program execution is complete, execution resumes in the user program immediately following the "write" Instruction.

Because the I/O operation may take a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence, the user program is stopped at the point of the "write" call for some considerable period of time.

Flow of Control With Interrupts

Flow of Control **with** Interrupts

1. After I/O prep instructions (code segment 4) and the I/O command have been executed, control returns to the user program. Meanwhile, the external device is busy completing the actual I/O operation e.g. accepting data from computer memory and printing it. With interrupts, this I/O operation is conducted concurrently with the execution of instructions in the user program.
2. When the external device becomes ready to be serviced, e.g. when it is ready to accept more data from the processor, the I/O module for that external device sends an interrupt request signal to the processor. The processor responds by suspending operation of the current program; branching off to a routine to service that particular I/O device, known as an interrupt handler; and resuming the original execution after the device is serviced.

Note that an interrupt can occur at any point in the main program, not just at one specific instruction.

Transfer of Control via Interrupts

- For the user program, an interrupt suspends the normal sequence of execution.
- When the interrupt processing is completed, execution resumes.
- Thus, the user program does not have to contain any special code to accommodate interrupts
- The **processor and the OS are responsible** for suspending the user program and then resuming it at the same point.

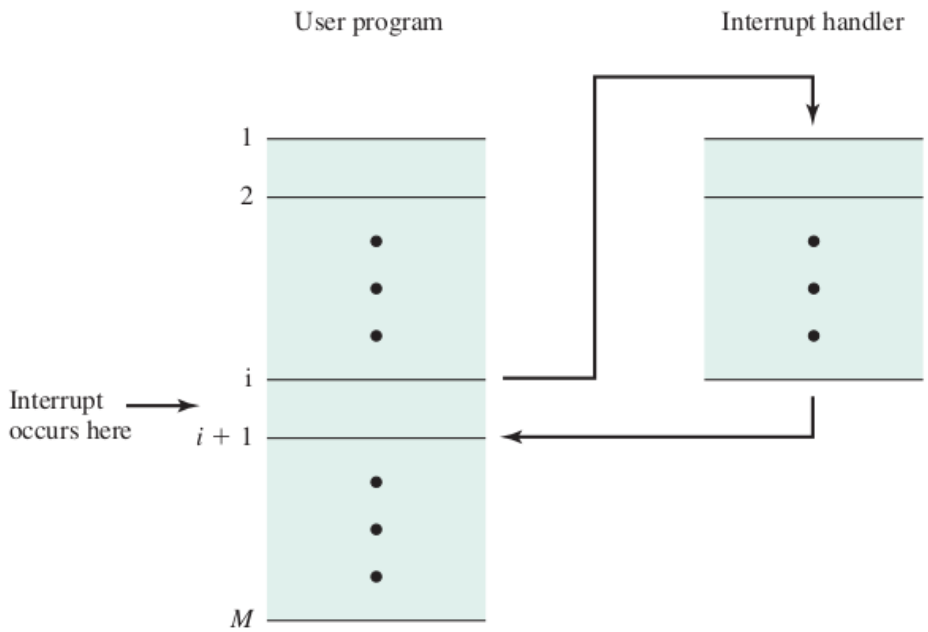


Figure 8: Transfer of Control via Interrupts

Instruction Cycle with Interrupts

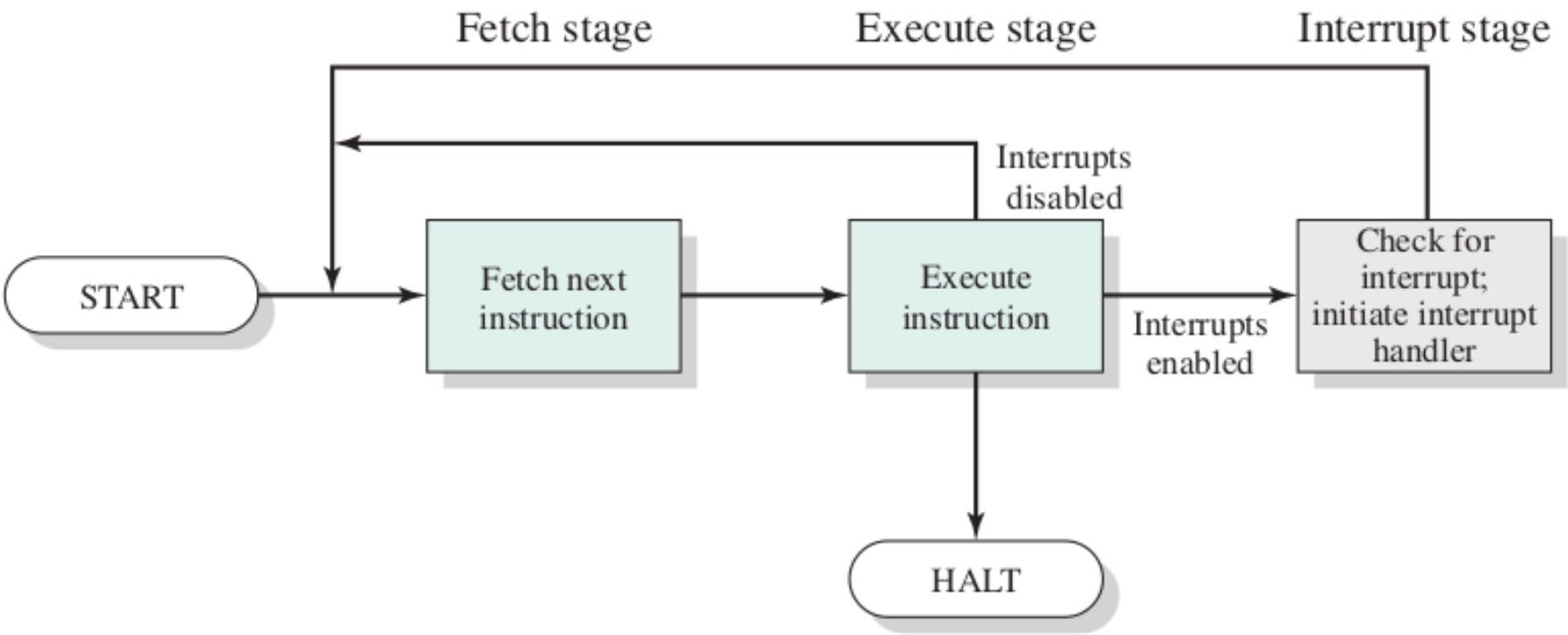


Figure 9: Instruction Cycle with Interrupts

Short vs Long I/O Wait

Short I/O wait

- Assume that the time required for the I/O operation is relatively short: less than the time to complete the execution of instructions between write operations in the user program.

Long I/O wait

- The more typical case, especially for a slow device such as a printer, is that the I/O operation will take much more time than executing a sequence of user instructions.
- In this case, the user program reaches the **second WRITE** call before the I/O operation spawned by the first call is complete. So, the user program is hung up at that point.
- There is still a gain in efficiency because part of the time during which the I/O operation is underway overlaps with the execution of user instructions.

Program Timing: Short I/O Wait

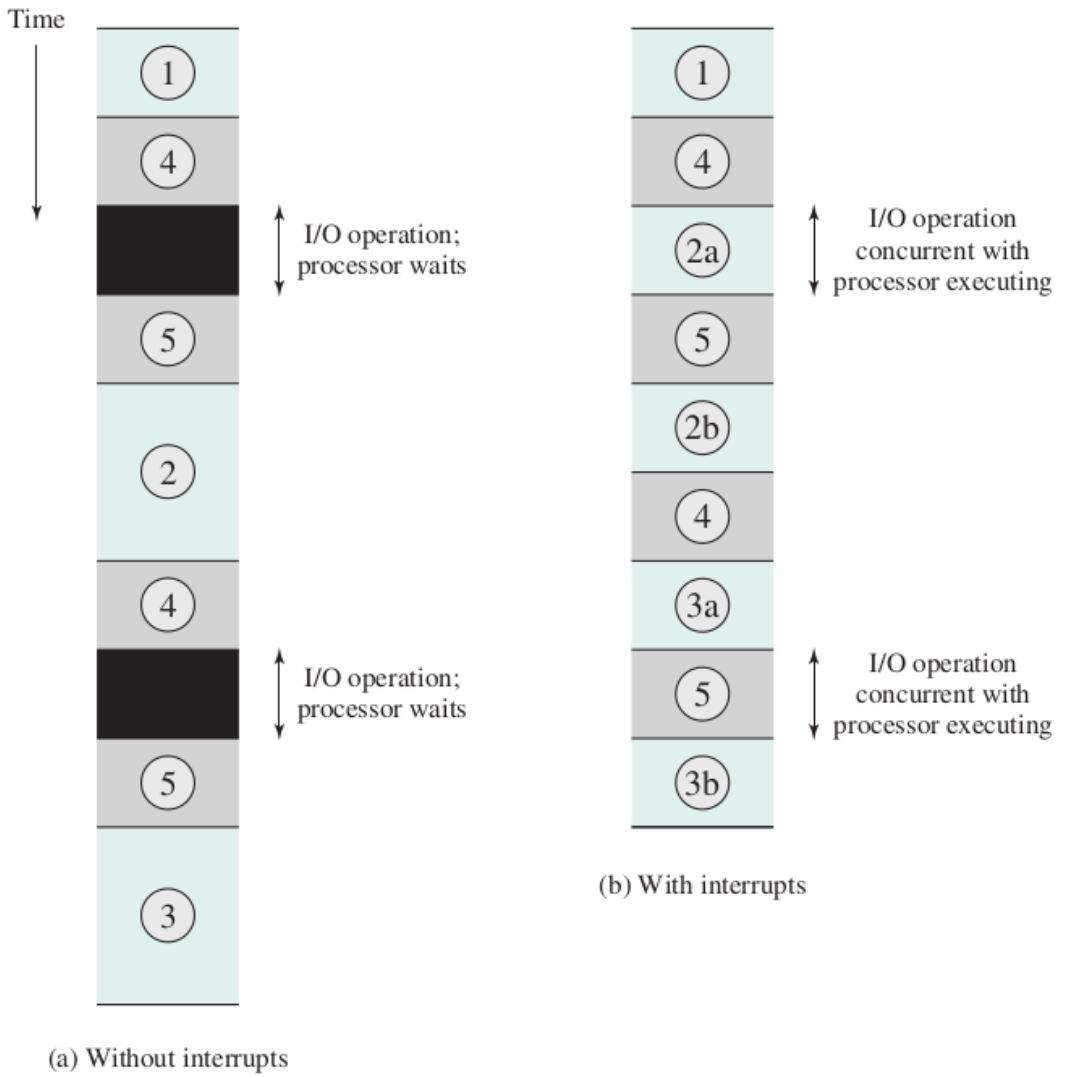


Figure 10: Program Timing: Short I/O Wait

Program Timing: Long I/O Wait

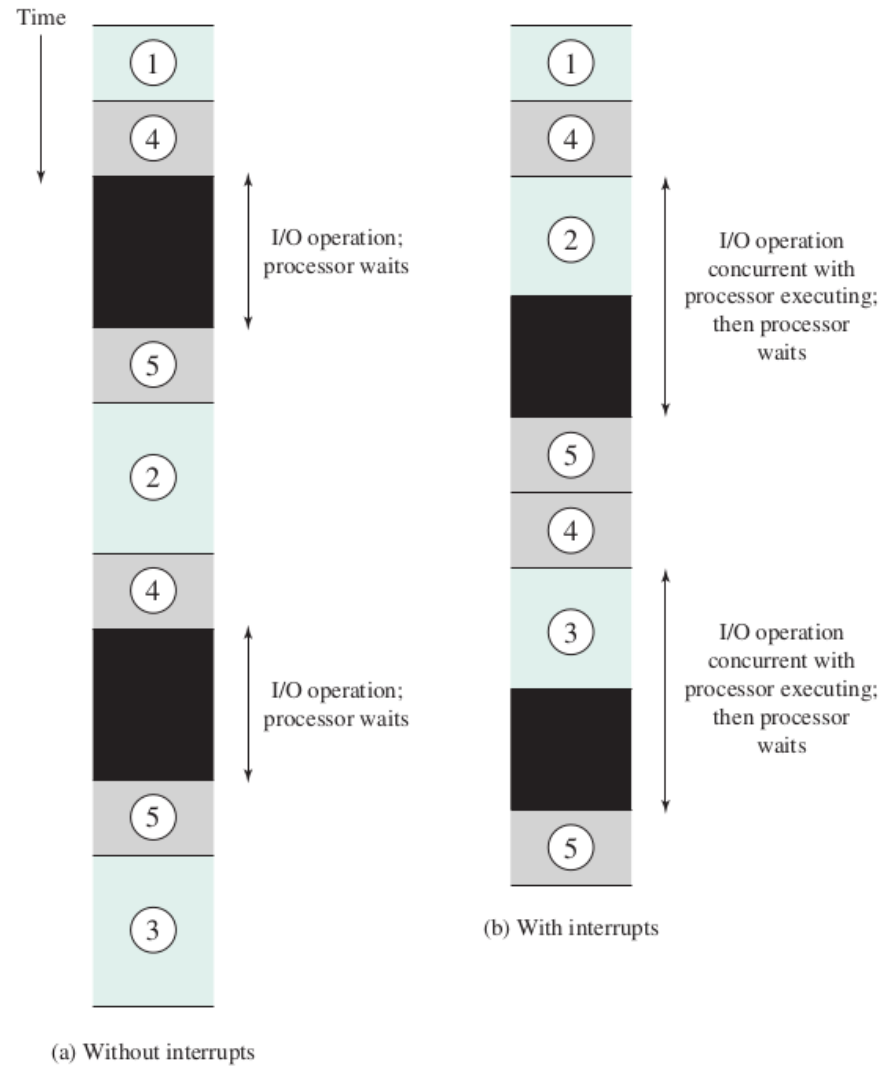


Figure 11: Program Timing: Long I/O Wait

Simple Interrupt Processing

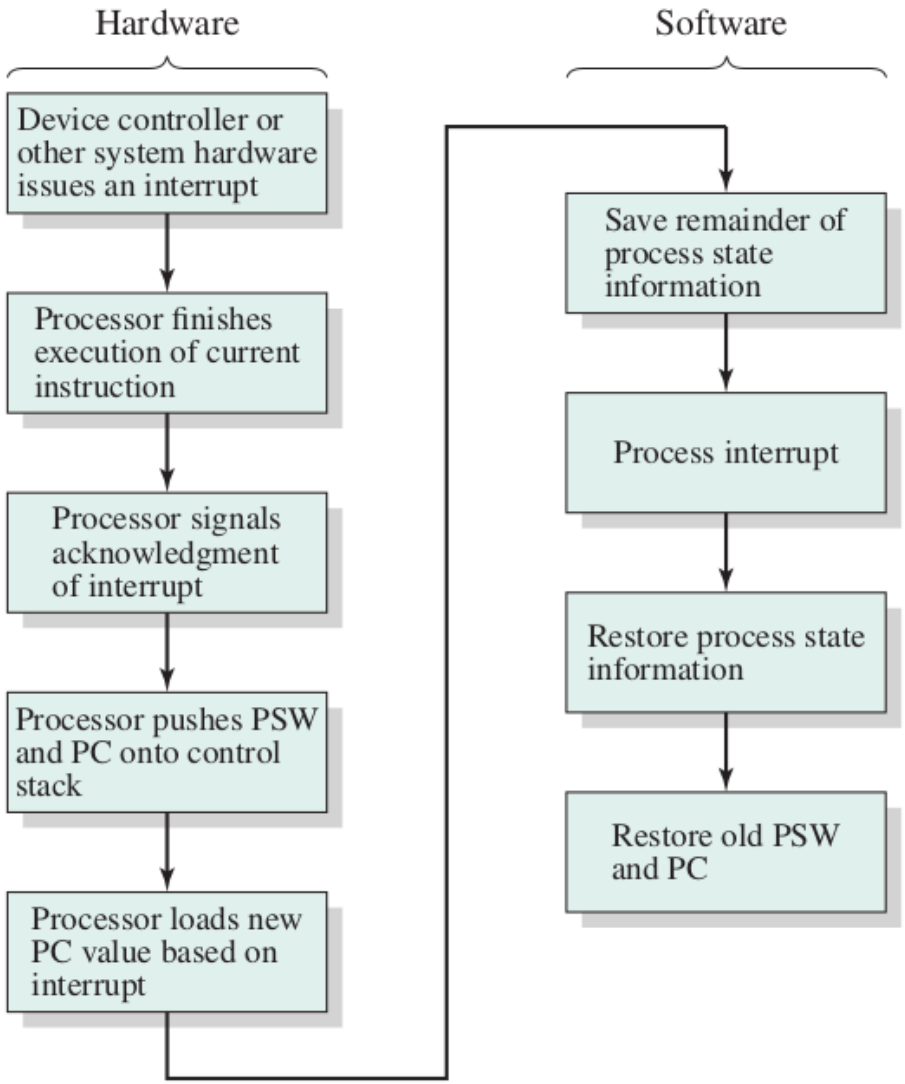


Figure 12: Simple Interrupt Processing

Simple Interrupt Processing

The processor loads the PC with the entry location of the interrupt-handling routine that will respond to this interrupt.

Interrupt-handling routine

(software section in previous slide)

- Information considered part of the state of the executing program needs to be saved for later resumption, in particular, the contents of the processor registers, because these registers may be used by the interrupt handler.
- This is because the interrupt is not a routine called from the program, it can occur at any time (unpredictable).

PSW is the program status word a system status register.

Changes in Memory and Registers for an Interrupt

- A user program is interrupted after the instruction at location N.
- The contents of all of the registers plus the address of the next instruction (N + 1), a total of M words, are pushed onto the **control stack**.
- The stack pointer is updated to point to the new top of stack, and the program counter is updated to point to the beginning of the interrupt service routine.
- Once the interrupt-handler routine completes, the saved register values can be retrieved from the stack and restored to the registers.

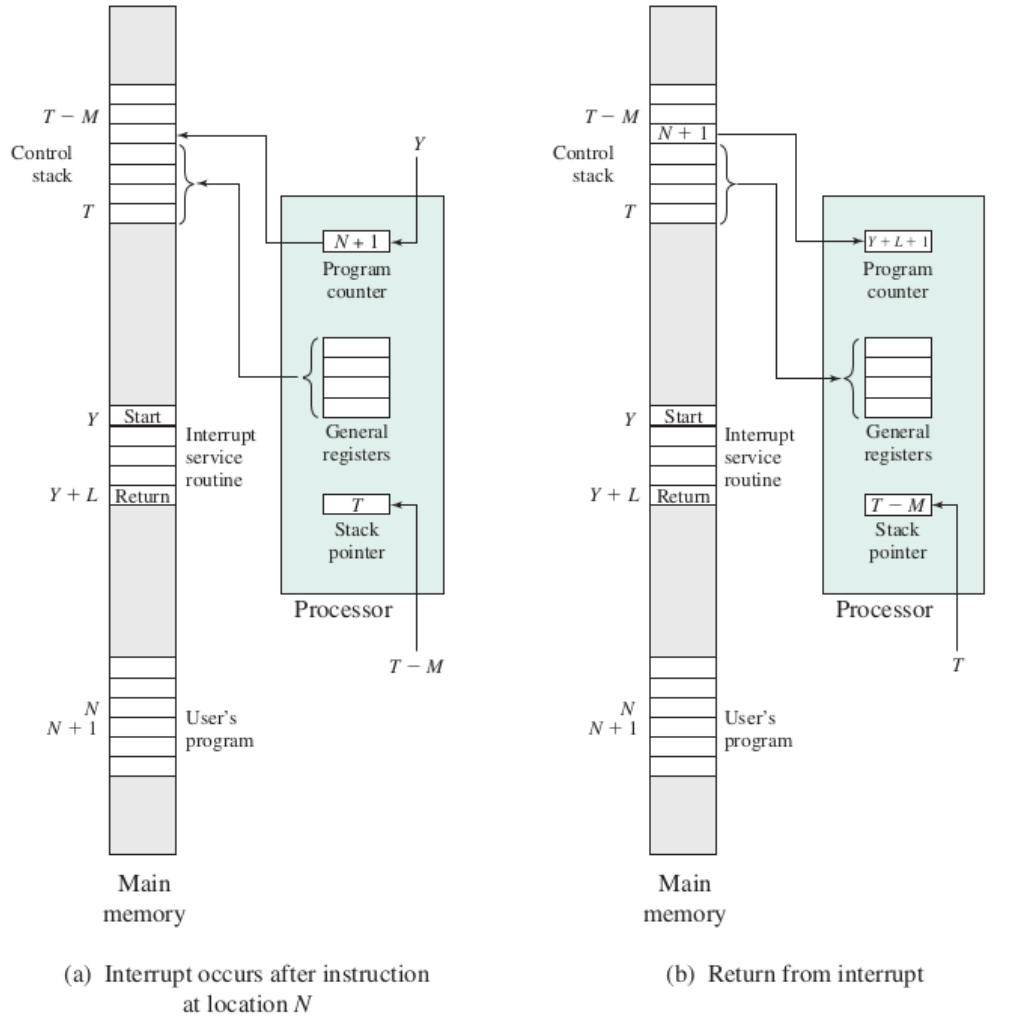


Figure 13: Changes in Memory and Registers for an Interrupt

Multiple Interrupts

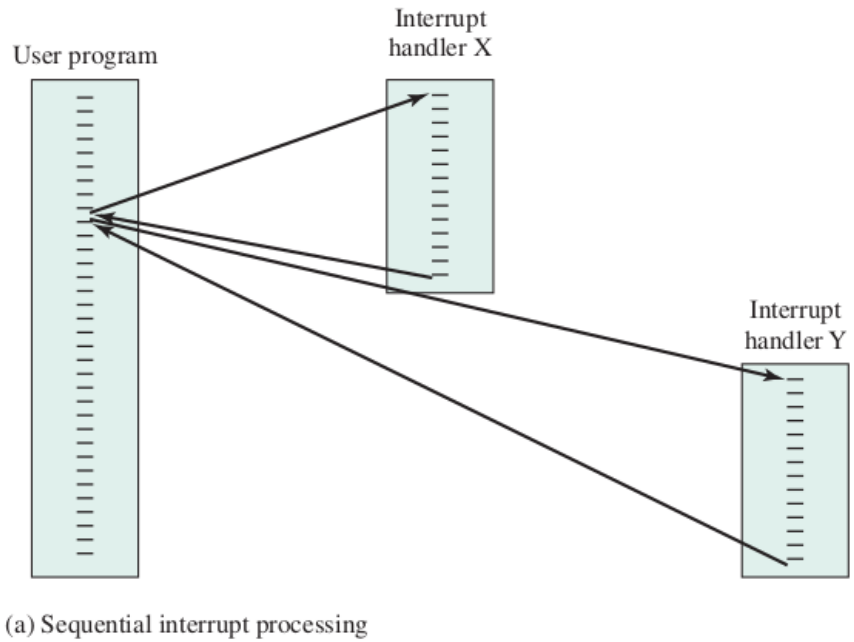
An interrupt occurs while another interrupt is being processed

- e.g. receiving data from a communications line and printing results at the same time

Two approaches:

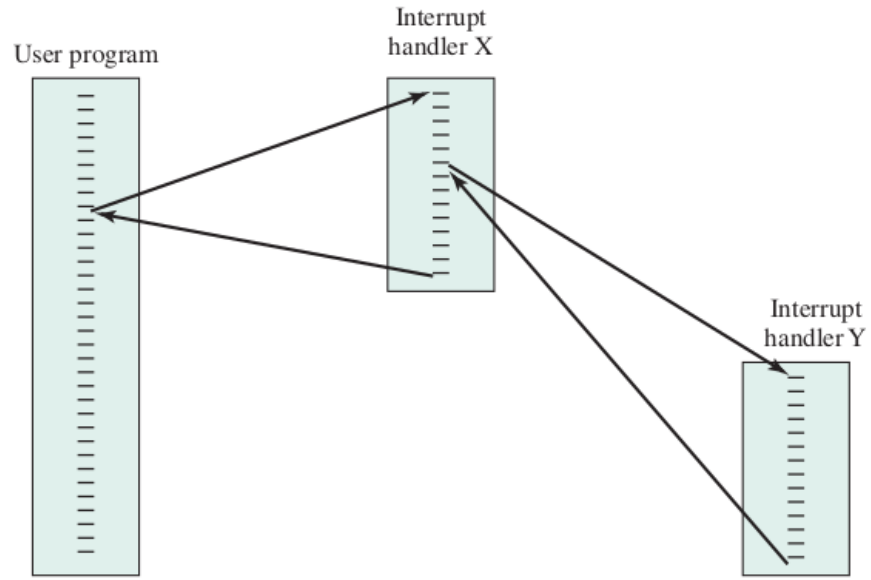
- Disable interrupts while an interrupt is being processed
- Use a priority scheme

Transfer of Control with Multiple Interrupts



(a) Sequential interrupt processing

Figure 14: Sequential interrupt processing



(b) Nested interrupt processing

Figure 15: Nested interrupt processing

Approaches to Handling Multiple Interrupts

Two approaches to multiple interrupts:

1. **Disable Interrupts During Processing** (Fig. 14)
   - Flow:
     1. If an interrupt occurs when a user program is executing → Disable interrupts.
     2. Processor ignores new interrupt requests while processing the current interrupt
     3. When the interrupt-handler routine is complete for current interrupt → Re-enable interrupts
     4. Processor checks for pending interrupts.
   - **Advantages:** Simple; handles interrupts in strict sequential order.
   - **Drawback:** Does not account for priorities or time-critical needs.
     - Example: Communication line input may need rapid processing to prevent buffer overflow.
2. **Priority-Based Interrupts** (Fig. 15)
   - Define priorities for interrupts.
   - Higher-priority interrupts can interrupt lower-priority interrupt handlers.

Approaches to Handling Multiple Interrupts

Example: Priority-Based Interrupt Handling (Figure on next slide)

- System with three I/O devices:
  Printer: Priority 2 Disk: Priority 4 Communications Line: Priority 5
- Sequence:
  - **t = 0:** User program begins.
  - **t = 10:** Printer interrupt occurs, and Printer ISR starts.
  - **t = 15:** Communications interrupt occurs.
    - Higher priority than Printer → Printer ISR is interrupted, Communications ISR begins. (ISR: *interrupt service routine*)
  - **t = 20:** Disk interrupt occurs.
    - Lower priority than Communications → Held until Communications ISR completes.
  - **t = 25:** Communications ISR completes.
    - Processor restores state of Printer ISR.
    - Before executing Printer ISR, Disk ISR (higher priority) starts.
  - **t = 25 to t = 35:** Disk ISR executes and completes.
  - **t = 35:** Printer ISR resumes and completes.
  - **t = 40:** User program resumes.

NB: Everytime a program/routine is interrupted its info/state is pushed onto the control stack

Time Sequence of Multiple Interrupts

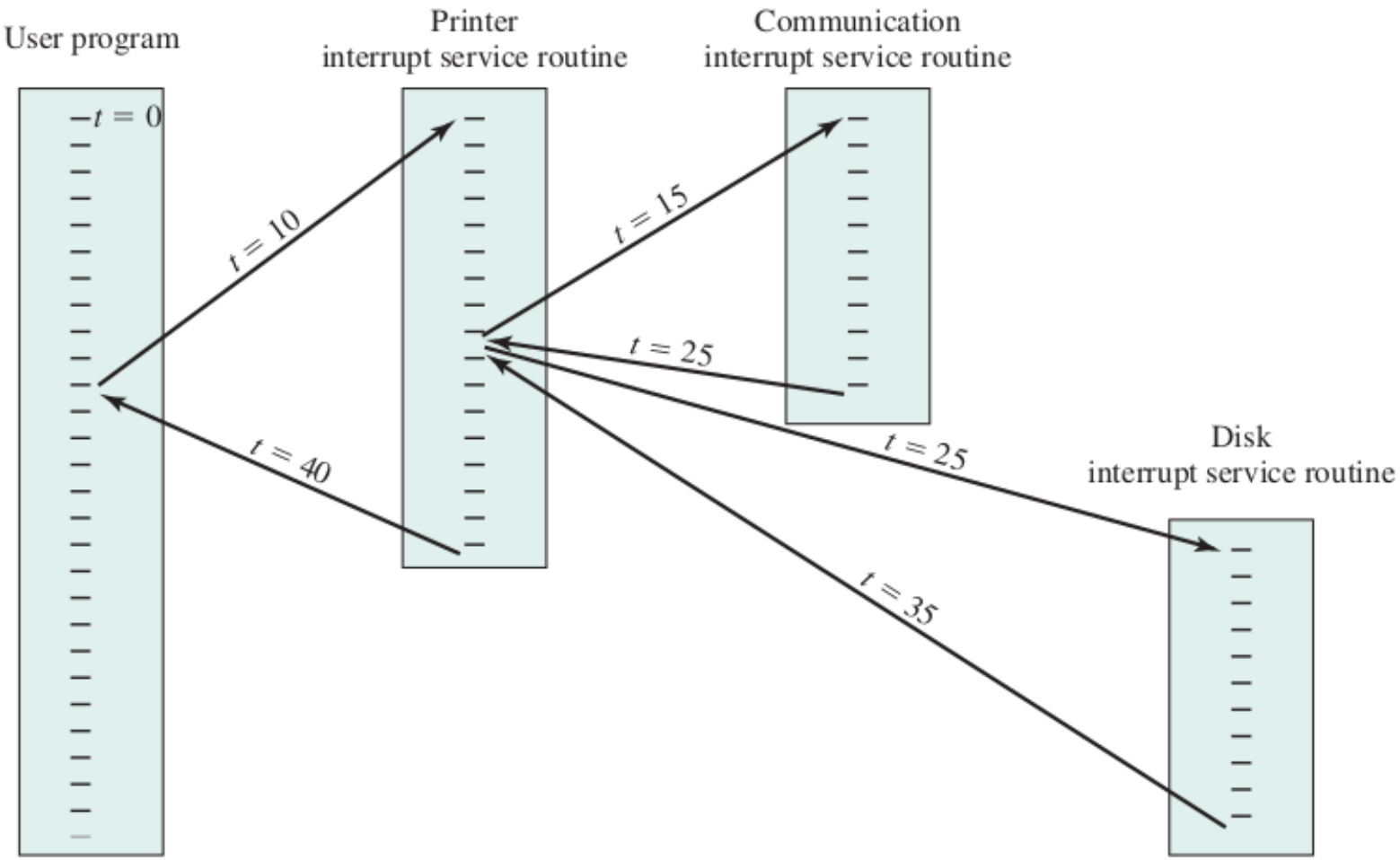


Figure 16: Time Sequence of Multiple Interrupts

Memory Hierarchy, Cache Memory and I/O

Memory Hierarchy

Design constraints on a computer's memory

- How much?
- How fast?
- How expensive?

Examples of things we need to consider in relation to memory, when designing a computer system:

- If the memory capacity is there, applications will likely be developed to use it
- Memory should be able to keep up with the processor
- Cost of memory should be reasonable in relation to the other components

Memory Relationships

Characteristics of memory

Trade-off among the three key characteristics of memory:

1. Capacity
2. Access time
3. Cost

Faster access time = Greater cost per bit.
Greater capacity = Smaller cost per bit.
Greater capacity = Slower access speed.

The Memory Hierarchy

The way out of this dilemma is to not rely on a single memory component or technology, but to employ a memory hierarchy.

Going down the memory hierarchy:

- Decreasing cost per bit
- Increasing capacity
- Increasing access time
- Decreasing frequency of access to the memory by the processor

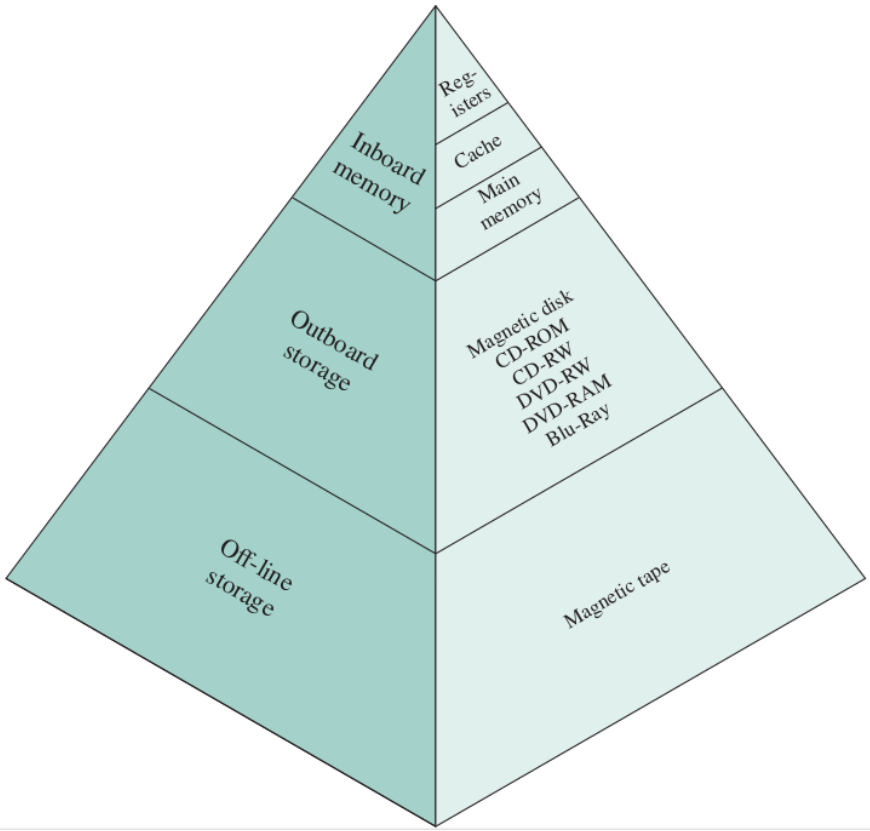


Figure 17: The Memory Hierarchy

Principle of Locality

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
- The principle of locality be applied across more than two levels of memory

Secondary Memory

Also referred to as auxiliary memory.

Secondary Memory function and properties

Used to store program and data files

- External
- Nonvolatile

Cache Memory

Motivation for cache memory:

- Processor must access memory at least once per instruction cycle
- Processor execution is limited by memory cycle time

Cache Memory functionality

Exploits the principle of locality by providing a small, fast memory between the processor and main memory

- Invisible to the OS
- Interacts with other memory management hardware

Cache and Main Memory

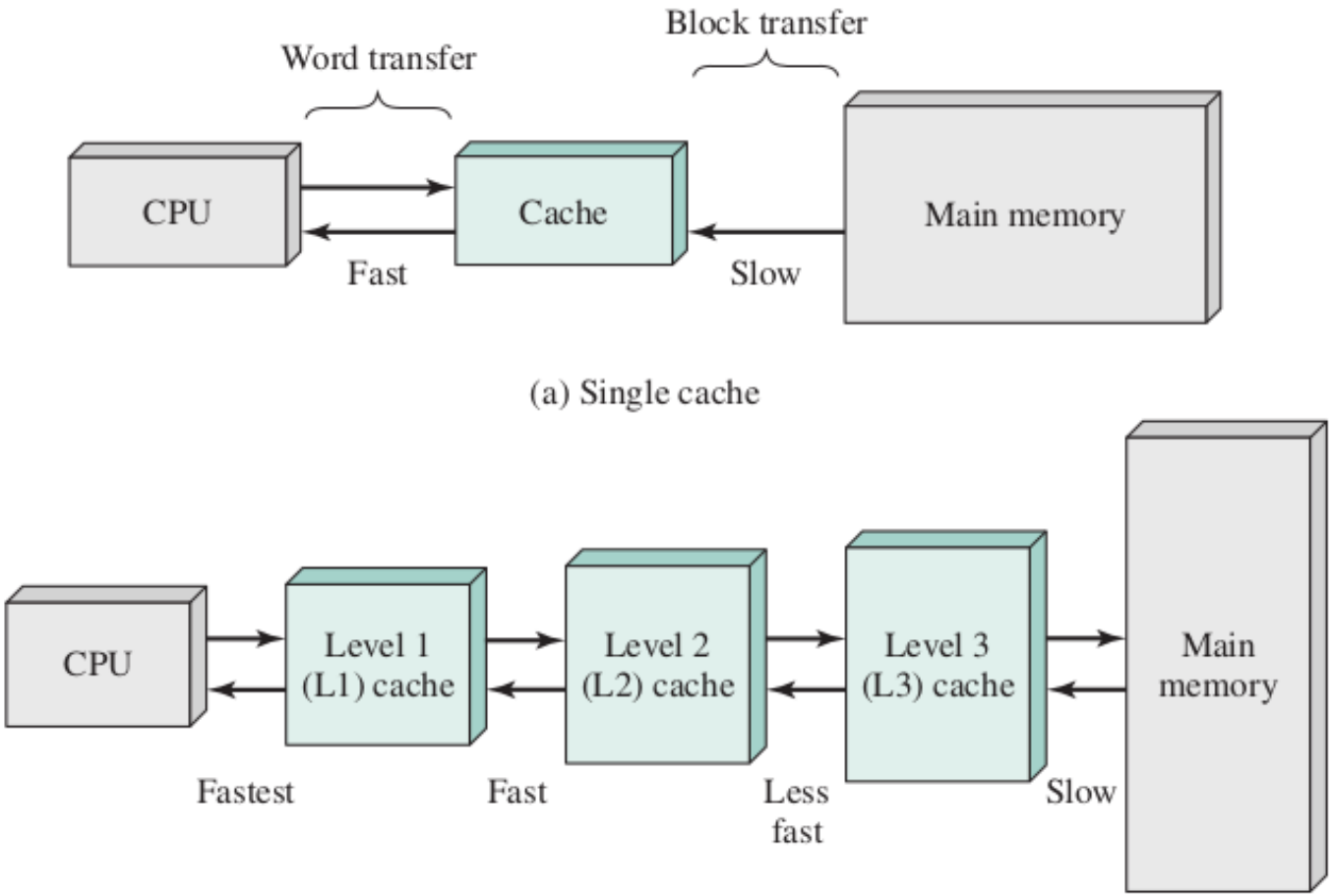


Figure 18: Cache and Main Memory

Computer System Overview Summary

- Basic Elements of a Computer System
- Multiprocessor and Multicore Organization
- Interrupts
- The memory hierarchy
- Cache memory: principles and design

Supplemental Material

Cache Read Operation

The figure to the right illustrates the read operation.

- The processor generates the address, RA, of a word to be read.
- If the word is contained in the cache, it is delivered to the processor.
- Otherwise, the block containing that word is loaded into the cache and the word is delivered to the processor.

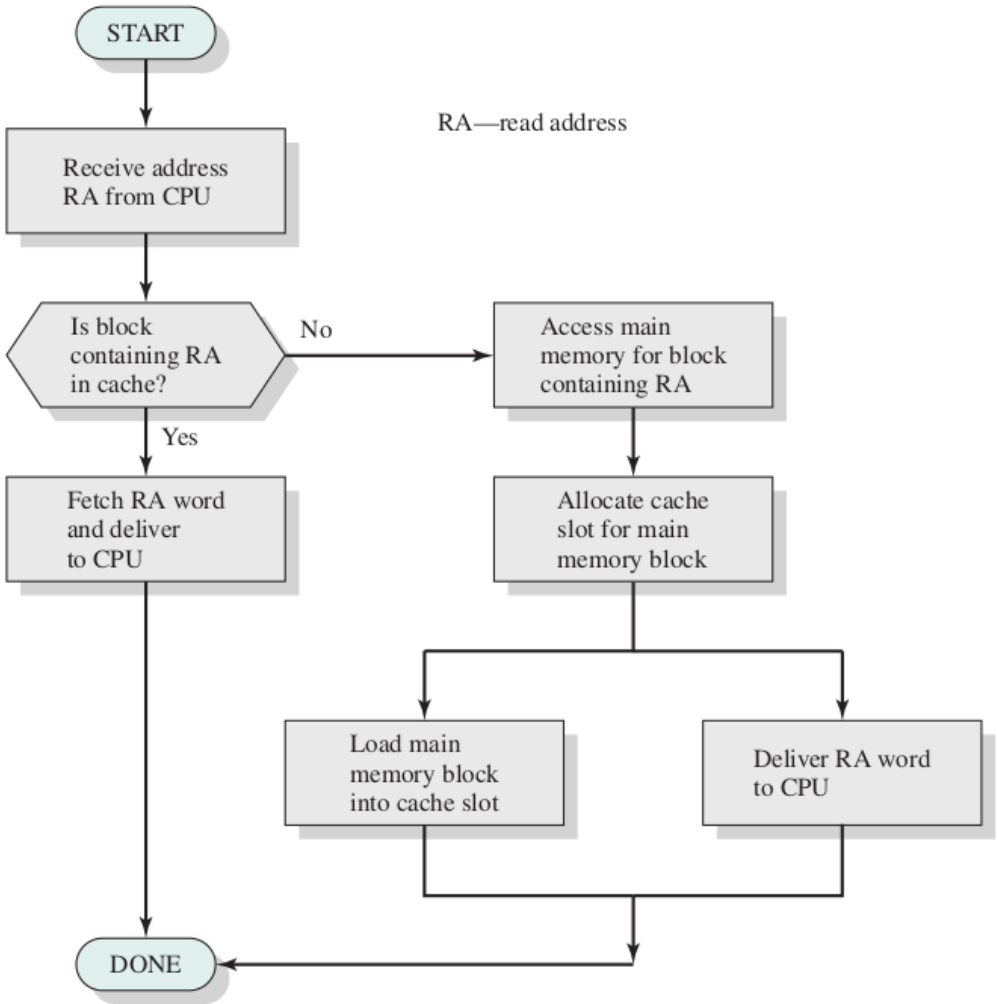


Figure 19: Cache Read Operation

Instruction Execution

A program consists of a set of instructions stored in memory

TWO STEPS

Processor reads (fetches) instructions from memory

Processor executes each instruction

Basic Instruction Cycle

Instruction Cycle

The processing required for a single instruction

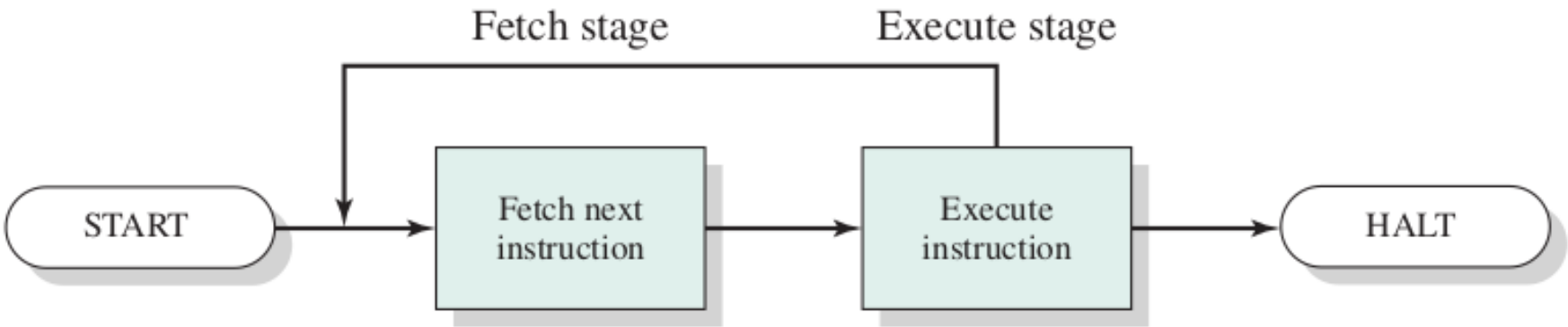


Figure 20: Simplified Two-step Instruction Cycle

Program execution halts only if the processor is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the processor is encountered.

Instruction Fetch and Execute

- At the beginning of each instruction cycle, the processor fetches an instruction from memory.
- Typically, the **Program Counter** (PC) holds the address of the next instruction to be fetched.
  - ➤ PC is incremented after each fetch by the processor, to ensure it will fetch the next instruction in sequence
    i.e. the instruction located at the next higher memory address

Instruction Fetch and Execute

Example

Consider a simplified computer in which each instruction occupies one 16-bit word of memory.
Assume that the program counter is set to location 300. The processor will next fetch the instruction at location 300.
On subsequent instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on.
This sequence may be altered e.g. the processor may fetch an instruction from location 303, which specifies that the next instruction be from location 327. The processor sets the PC to 327. On the next fetch, the instruction will be fetched from location 327 rather than 304.

Instruction Register (IR)

1. The fetched instruction is loaded into the Instruction Register (IR).
2. The instruction contains bits that specify the action the processor is to take.
3. Processor interprets the instruction and performs required actions
   - Data transfer between the processor and memory or I/O device
   - The processor performs arithmetic or logic operation on data
   - Sequence of execution is to be altered

Characteristics of a Hypothetical Machine

Hypothetical Processor

Consider a hypothetical processor (figure in next slide) where the processor contains a single data register, called the accumulator (AC), both instructions and data are 16 bits long, and memory is organized as a sequence of 16-bit words.

Also the instruction format provides 4 bits for the opcode, allowing as many as $2^4 = 16$ different opcodes (represented by a single hexadecimal digit). The **opcode** defines the operation the processor is to perform.

The remaining 12 bits of the instruction format, up to $2^{12} = 4096$ ($\sim$4K) words of memory (denoted by three hexadecimal digits) can be directly addressed.

Characteristics of a Hypothetical Machine

(a) Instruction format

(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

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

(d) Partial list of opcodes

Figure 21: Characteristics of a Hypothetical Machine

Program Execution Example

Example

The figure on the next slide illustrates a partial program execution, showing the relevant portions of memory and processor registers.
The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.
Three instruction cycles, each consisting of a fetch stage and an execute stage, are needed to add the contents of location 940 to the contents of 941.

Note:

With a more complex set of instructions, fewer instruction cycles would be needed.
Most modern processors include instructions that contain more than one address.
Thus the execution stage for a particular instruction may involve more than one reference to memory.
Also, instead of memory references, an instruction may specify an I/O operation.

Program Execution Example

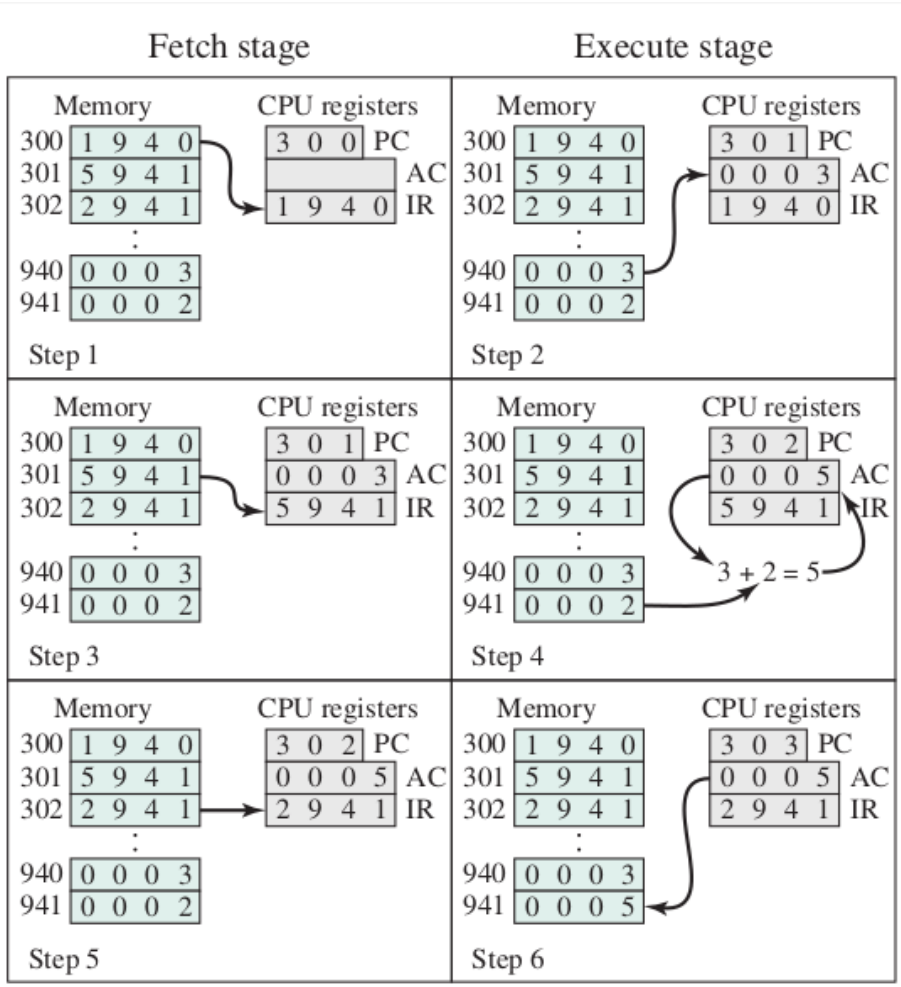


Figure 22: Example of Program Execution

Program Execution Example

Example

The instructions for this example are as follows:

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are not shown. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded from memory. The remaining 12 bits (three hexadecimal digits) specify the address, which is 940.
2. The next instruction (5941) is fetched from location 301 and the PC is incremented. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
3. The next instruction (2941) is fetched from location 302 and the PC is incremented. The contents of the AC are stored in location 941.