Operating Systems: Internals and Design Principles

Chapter 1 Computer System Overview

Ninth Edition

By William Stallings

Operating System

- Exploits the hardware resources of one or more processors
- Provides a set of services to system users
- Manages secondary memory and I/O devices

Basic Elements

- Processor
- I/O Modules
- Main Memory
- System Bus

Processor

Controls the operation of the computer

Performs the data processing functions

Referred to as the *Central Processing Unit* (CPU)

Processor

- Arithmetic/logic unit (ALU) performs arithmetic and logic operations
- Control unit (CU) directs the operation of the processor
  - It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor

https://en.wikipedia.org/wiki/Central_processing_unit

Main Memory

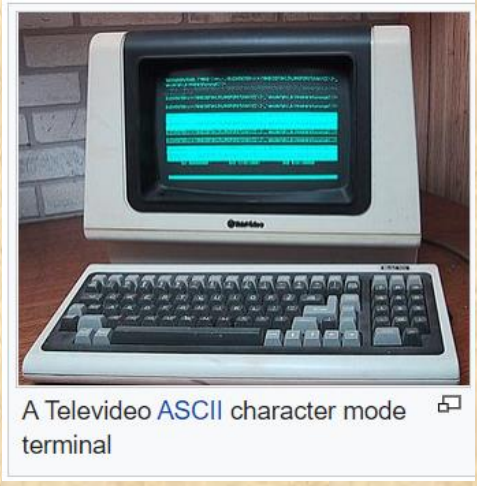
- Stores data and programs
- Typically volatile
  - Contents of the memory is lost when the computer is shut down
- Referred to as *real memory* or *primary memory*

I/O Modules

Move data between the computer and its external environment

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

A Teletype Model 33 ASR teleprinter, usable as a terminal



A Televideo ASCII character mode terminal

https://en.wikipedia.org/wiki/Computer_terminal

System Bus

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

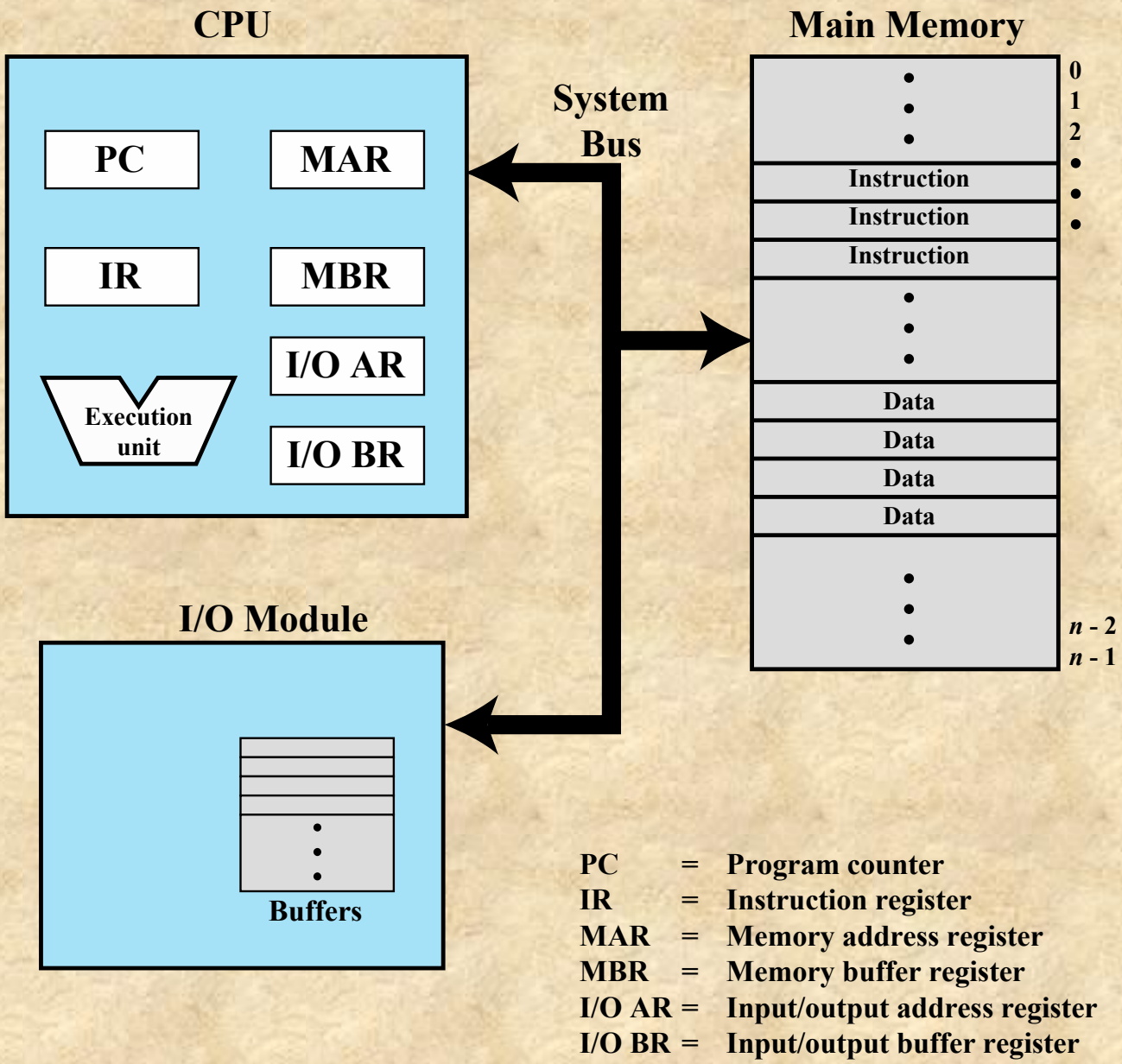


Figure 1.1 Computer Components: Top-Level View

Microprocessor

- Invention that brought about desktop and handheld computing
- Contains a processor on a single chip
- Fastest general-purpose processors
- Multiprocessors
  - Each chip (socket) contains multiple processors (cores)

Processor Registers

- A processor register is a quickly accessible location available to a computer's processor
  - A small amount of fast storage
  - normally at the top of the memory hierarchy and provide the fastest way to access data

https://en.wikipedia.org/wiki/Processor_register

Processor Registers

- A memory address register (MAR) specifies the address in memory for the next read or write
- A memory buffer register (MBR) contains the data to be written into memory or which receives the data read from memory and also called Memory Data Register (MDR)
- 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.

https://en.wikipedia.org/wiki/Processor_register

Graphical Processing Units (GPUs)

- Provide efficient computation on arrays of data using Single-Instruction Multiple Data (SIMD) techniques pioneered in supercomputers
- No longer used just for rendering advanced graphics
  - Also used for general numerical processing
    - Physics simulations for games
    - Computations on large spreadsheets
    - Crypto-mining
    - Artificial Intelligence
- GPUs cannot replace CPUs

Digital Signal Processors (DSPs)

- Deal with streaming signals such as audio or video
- Used to be embedded in I/O devices like modems
  - Are now becoming first-class computational devices, especially in handhelds
- Encoding/decoding speech and video (codecs)
- Provide support for encryption and security

Digital Signal Processors (DSPs)

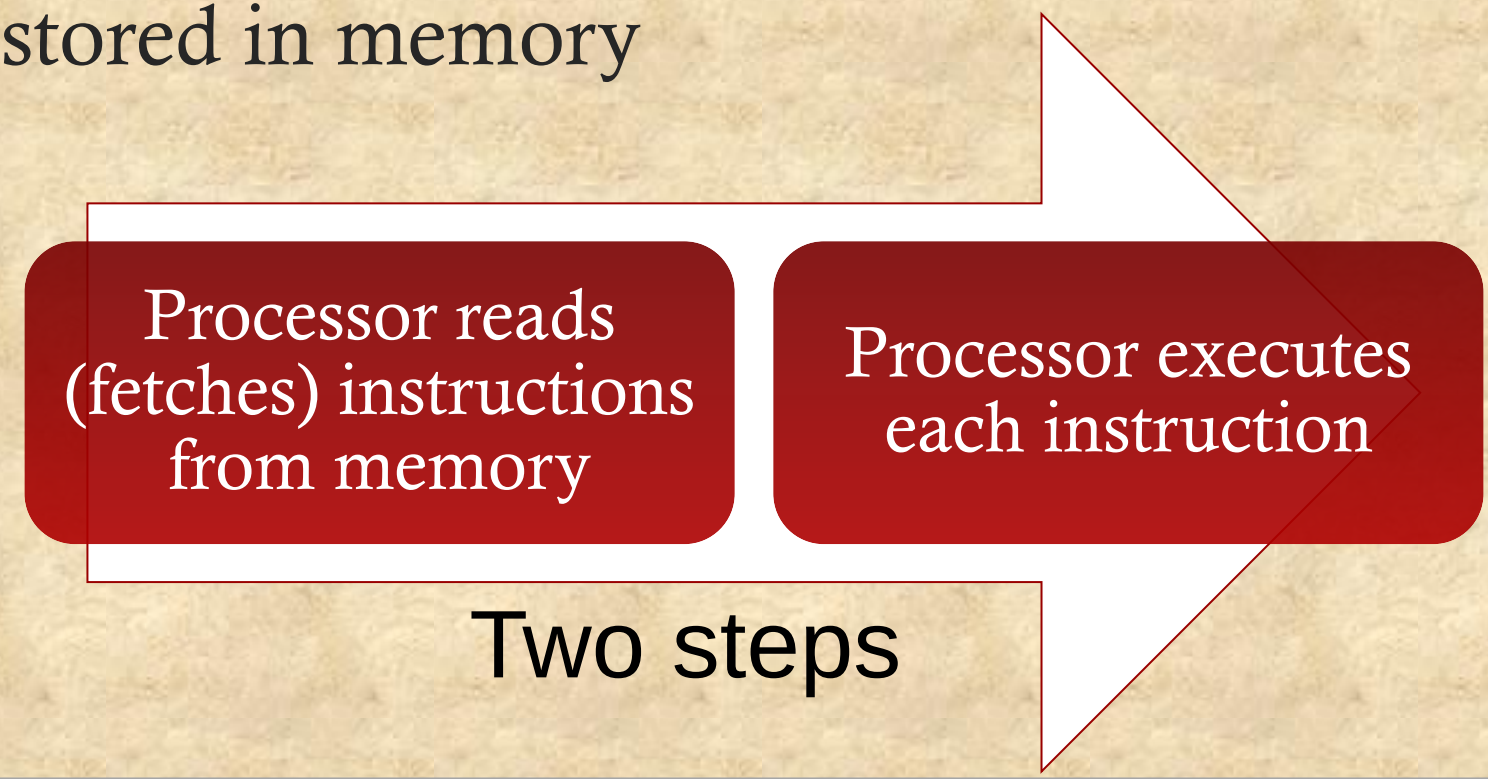
A TMS320 digital signal processor chip found in a guitar effects unit. A crystal oscillator may be seen above.
https://en.wikipedia.org/wiki/Digital_signal_processor

System on a Chip (SoC)

- To satisfy the requirements of handheld devices, the classic microprocessor is giving way to the SoC
  - Other components of the system, such as DSPs, GPUs, I/O devices (such as codecs and radios) and main memory, in addition to the CPUs and caches, are on the same chip

Instruction Execution

- A program consists of a set of instructions stored in memory



The processing required for a single instruction is called an *instruction cycle*

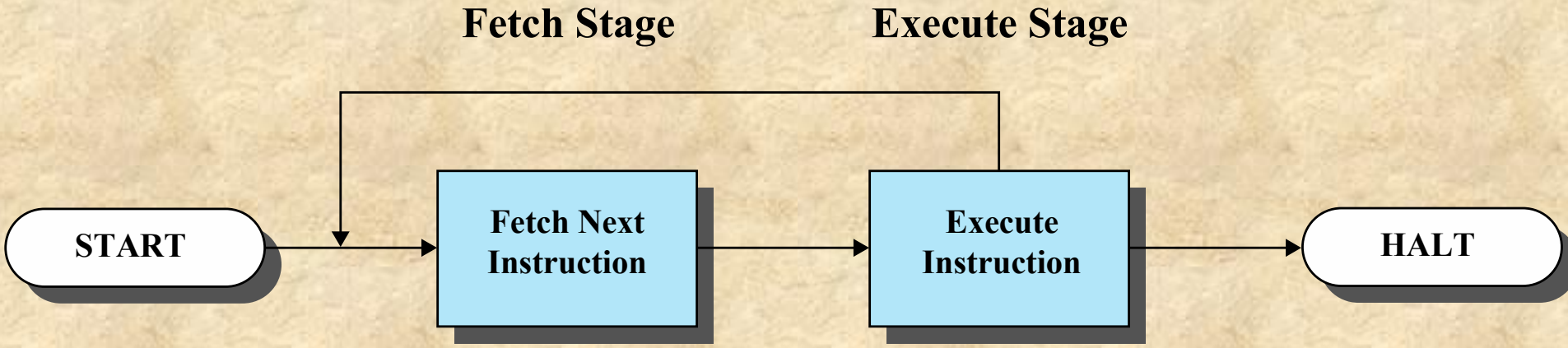


Figure 1.2 Basic Instruction Cycle

Instruction Fetch and Execute

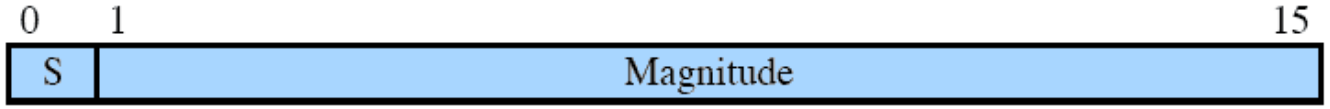
- The processor fetches an instruction from memory (where?)
- Typically the program counter (PC) holds the address of the next instruction to be fetched
  - PC is incremented after each fetch

Instruction Register (IR)

Fetched instruction is loaded into Instruction Register (IR)

- Processor interprets the instruction and performs required action:
  - Processor-memory
  - Processor-I/O
  - Data processing
  - Control

(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 1.3 Characteristics of a Hypothetical Machine

Program counter (PC) holds the address of the next instruction to be fetched

Instruction register (IR) holds the instruction being executed

Accumulator (AC): Temporary storage

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

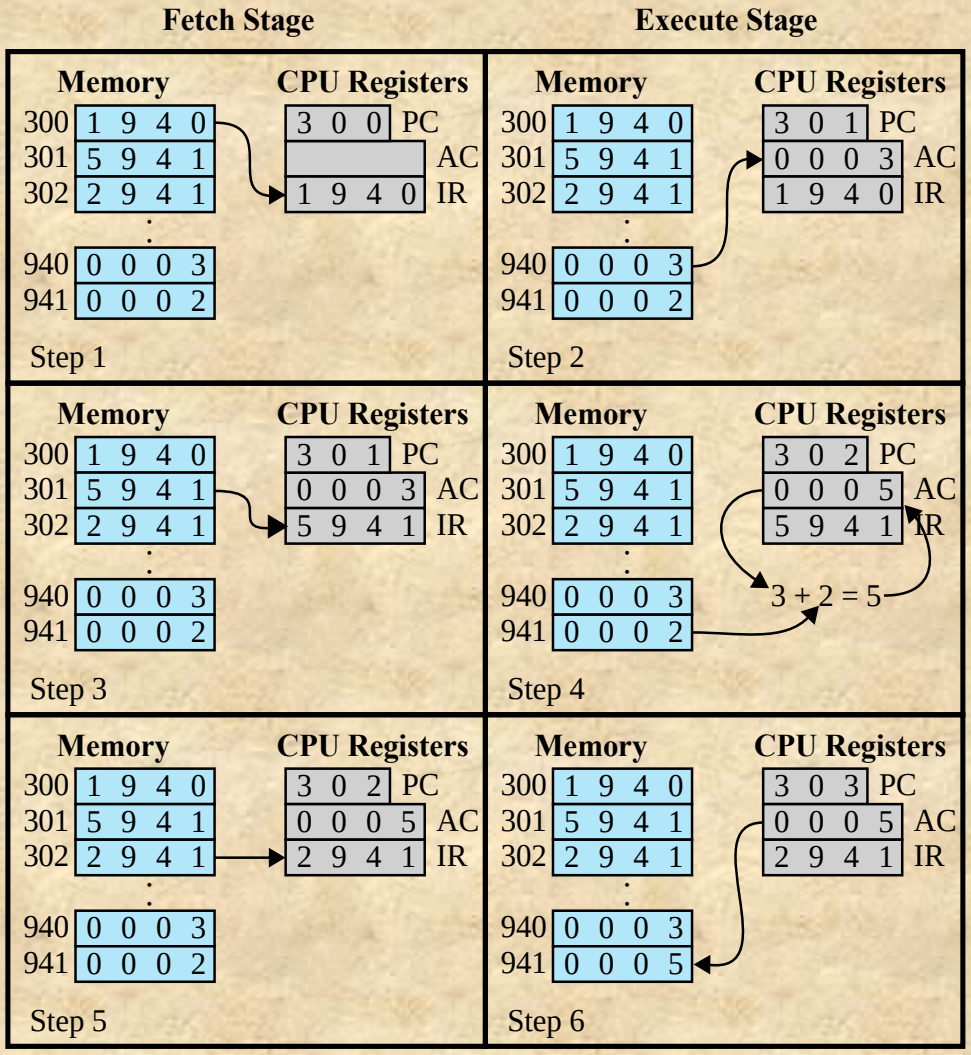


Figure 1.4 Example of Program Execution (contents of memory and registers in hexadecimal)

MAR and MBR (MDR)

1. The memory address held in the PC is copied into the MAR.
2. The address in the PC is then incremented (increased) by one. The PC now holds the address of the next instruction to be fetched.
3. The processor sends a signal along the address bus to the memory address held in the MAR (activate the control signal to perform the Read operation)
4. The instruction/data held in that memory address is sent along the data bus to the MBR.
5. The instruction/data held in the MBR is copied into the IR.
6. The instruction/data held in the IR is decoded and then executed. Results of processing are stored in the AC.
7. The cycle then returns to step 1.

https://www.bbc.co.uk/bitesize/guides/zbfny4j/revision/3

More on Instruction Fetch

http://theteacher.info/index.php/fundamentals-of-cs/1-hardware-and-communication/topics/2599-registers-and-the-fetch-decode-execute-cycle

https://www.robots.ox.ac.uk/~dwm/Courses/2CO_2014/2CO-N2.pdf

https://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Machine_Level_Architecture/The_Fetch%E2%80%93Execute_cycle_and_the_role_of_registers_within_it

Interrupts

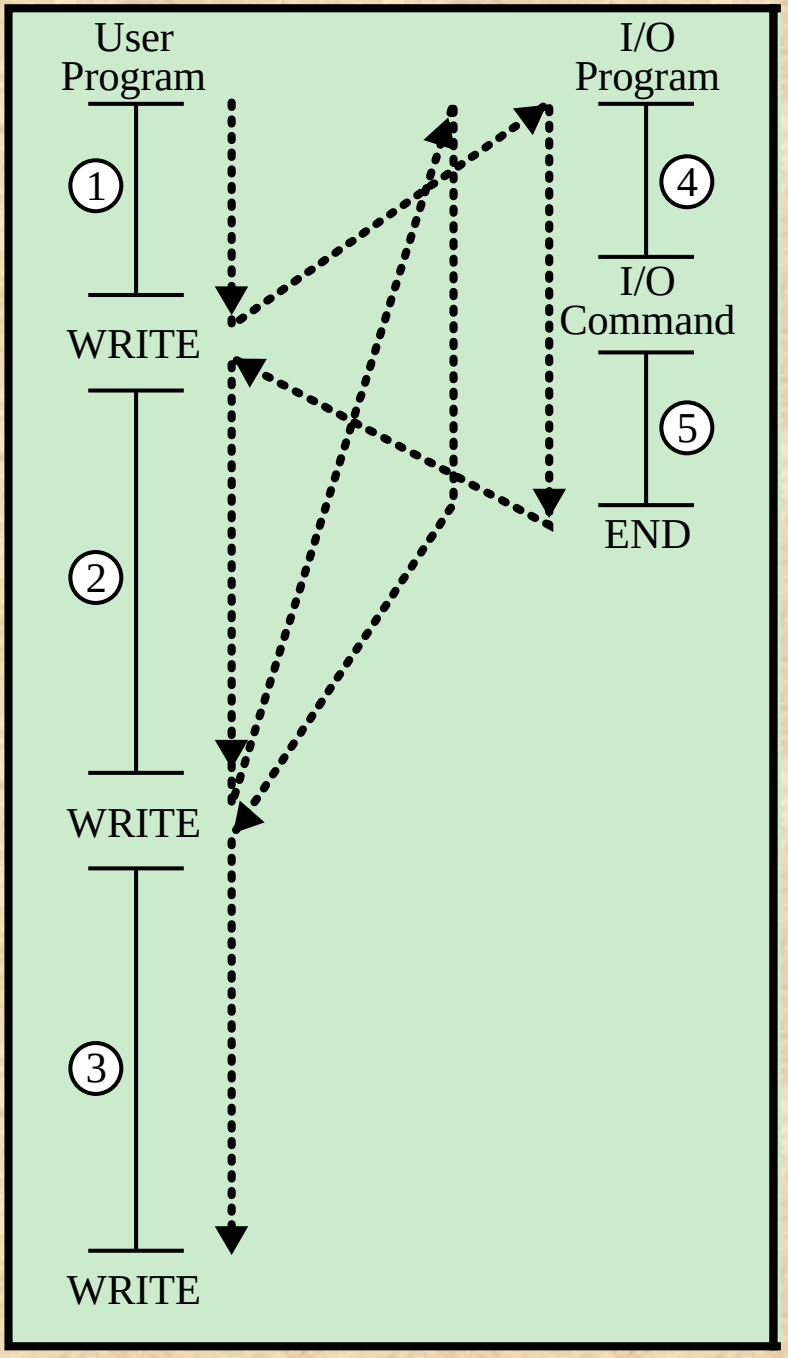
- Mechanism by which other modules may interrupt the normal sequencing of the processor
- Provided to improve processor utilization
  - Most I/O devices are slower than the processor
  - Processor must pause to wait for device
  - Wasteful use of the processor

Table 1.1 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, and 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.

Figure 1.5a

Flow of Control Without Interrupts



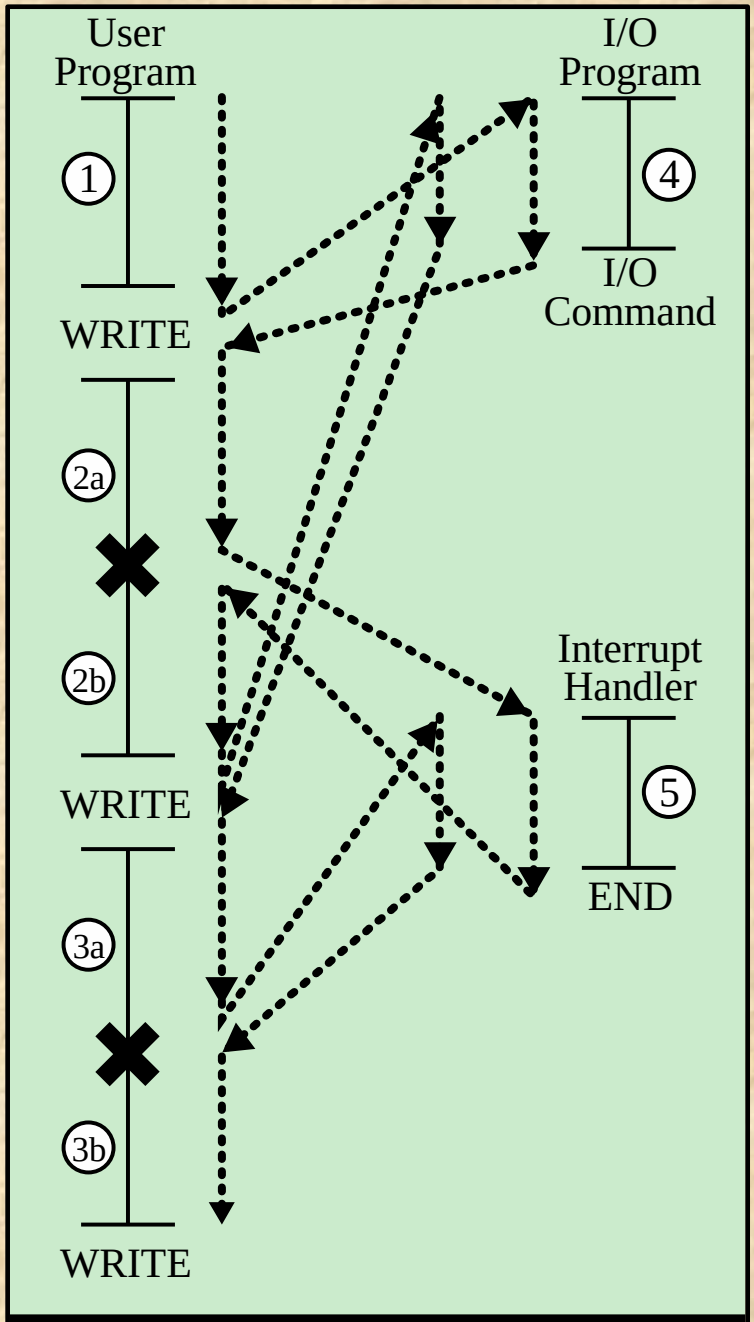
(a) No interrupts

- The user program performs a series of WRITE calls interleaved with processing. Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O. The WRITE calls are to an I/O routine that is a system utility and that will perform the actual I/O operation. The I/O program consists of three sections:
  - A sequence of instructions to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command (labeled 4 in the figure)
  - The actual I/O command. **Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function** (or periodically check the status, or poll, the I/O device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.
  - A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.
- After the first WRITE instruction is encountered, the user program is paused, and execution continues with the I/O program. After the I/O program execution is complete, execution resumes in the user program immediately following the WRITE Instruction

- The I/O program consists of three sections:
  - A sequence of instructions to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command (labeled 4 in the figure)
  - The actual I/O command. **Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function** (or periodically check the status, or poll, the I/O device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.
  - A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.
- What's the problem here?

Figure 1.5b

Short I/O Wait



(b) Interrupts; short I/O wait

X = interrupt occurs during course of execution of user program

- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress
  - Consider the user program reaches a point at which it makes a system call in the form of a WRITE call. The I/O program that is invoked in this case consists only of the preparation code and the actual I/O command. After these few instructions have been executed, control returns to the user program. Meanwhile, the external device is busy accepting data from computer memory and printing it. This I/O operation is conducted concurrently with the execution of instructions in the user program.
  - When the external device becomes ready to be serviced, that is, 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.

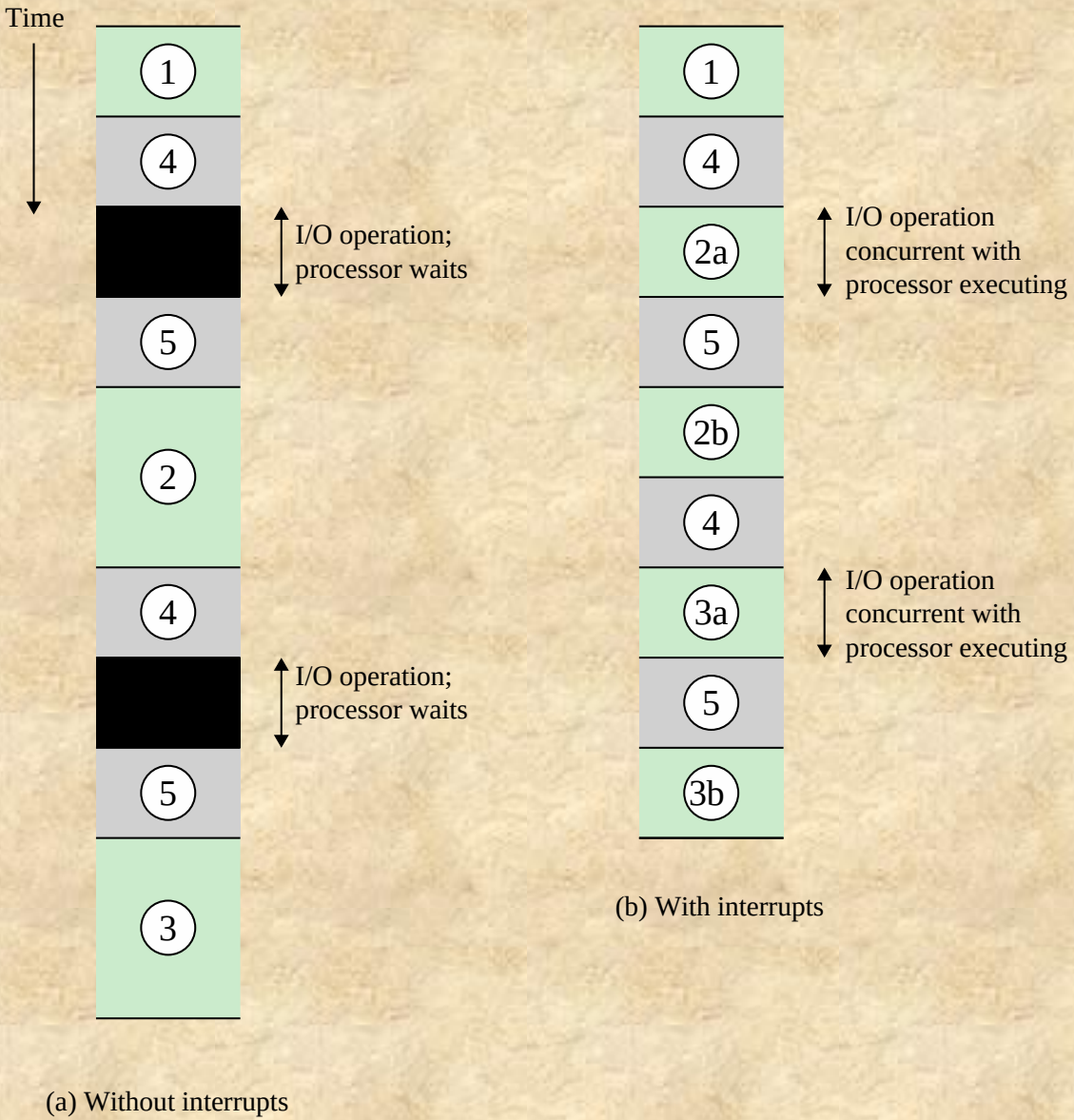
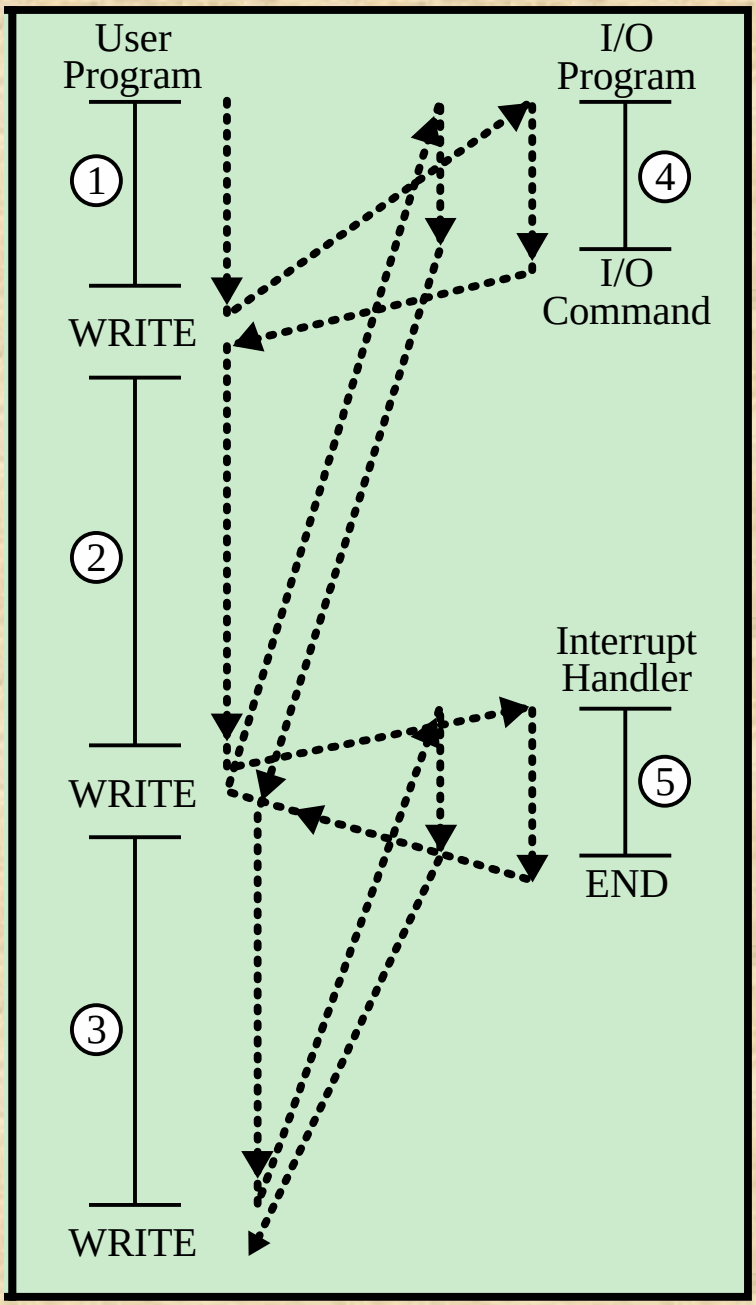


Figure 1.8 Program Timing: Short I/O Wait

Figure 1.5c

Long I/O Wait

- In cases with slow I/O, the user program reaches the second WRITE call before the I/O operation spawned by the first call is complete. The user program is hung up at that point.
- When the preceding I/O operation is completed, this new WRITE call may be processed, and a new I/O operation may be started.



(c) Interrupts; long I/O wait

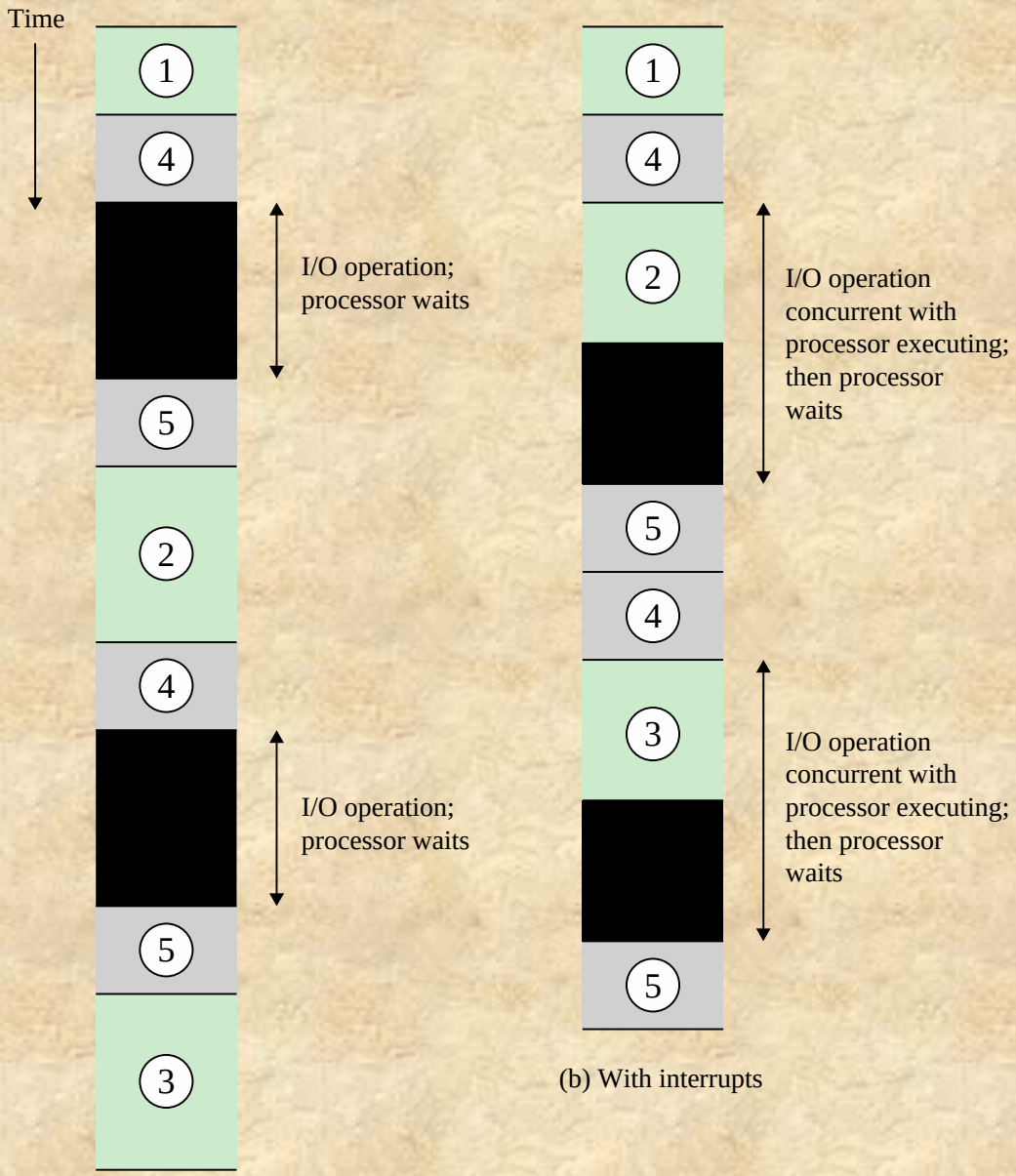


Figure 1.9 Program Timing: Long I/O Wait

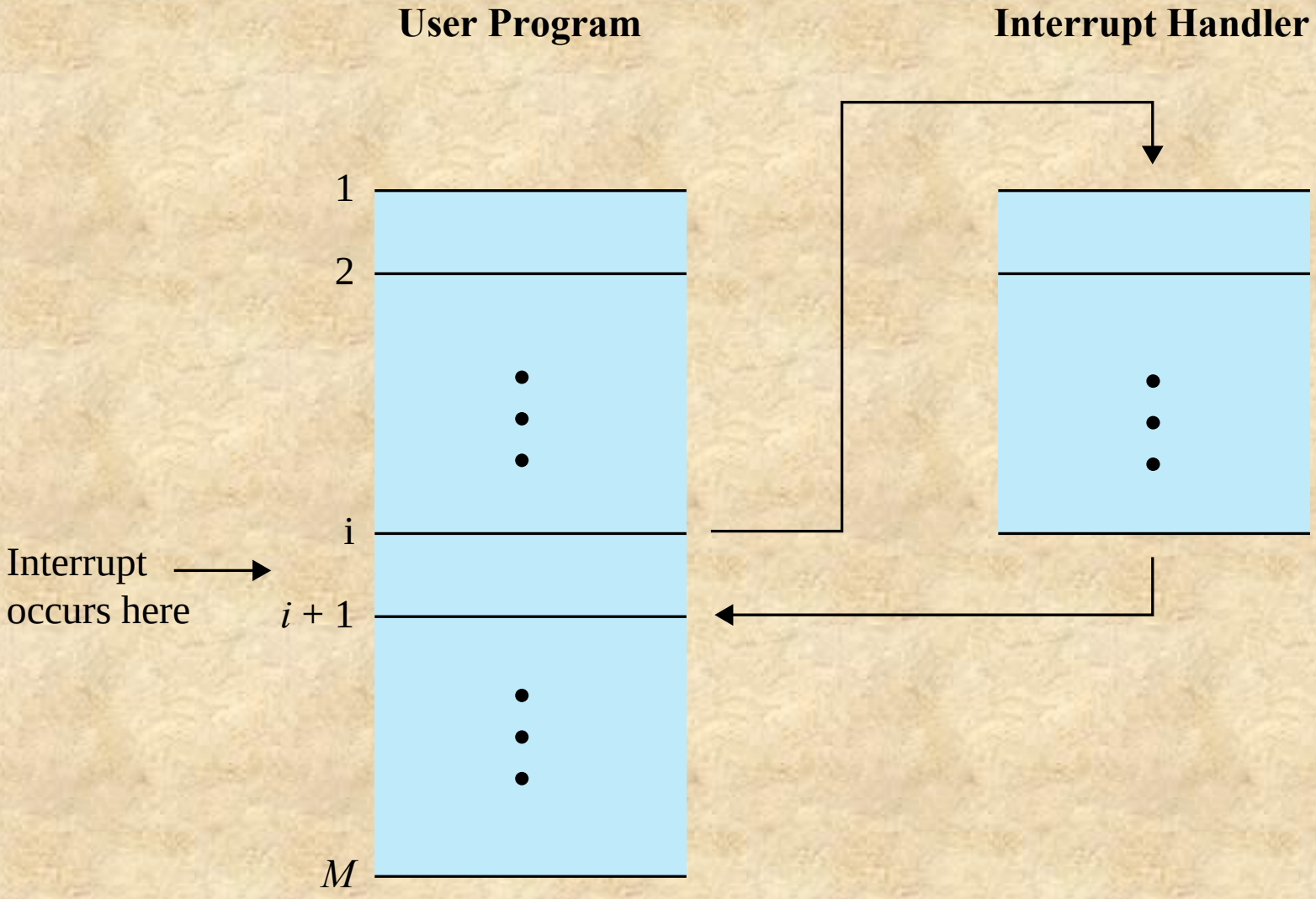


Figure 1.6 Transfer of Control via Interrupts

An interrupt triggers a number of events, both in the processor hardware and in software.

Program status word (PSW) contains status information about the currently running process, including memory usage information, condition codes, and other status information

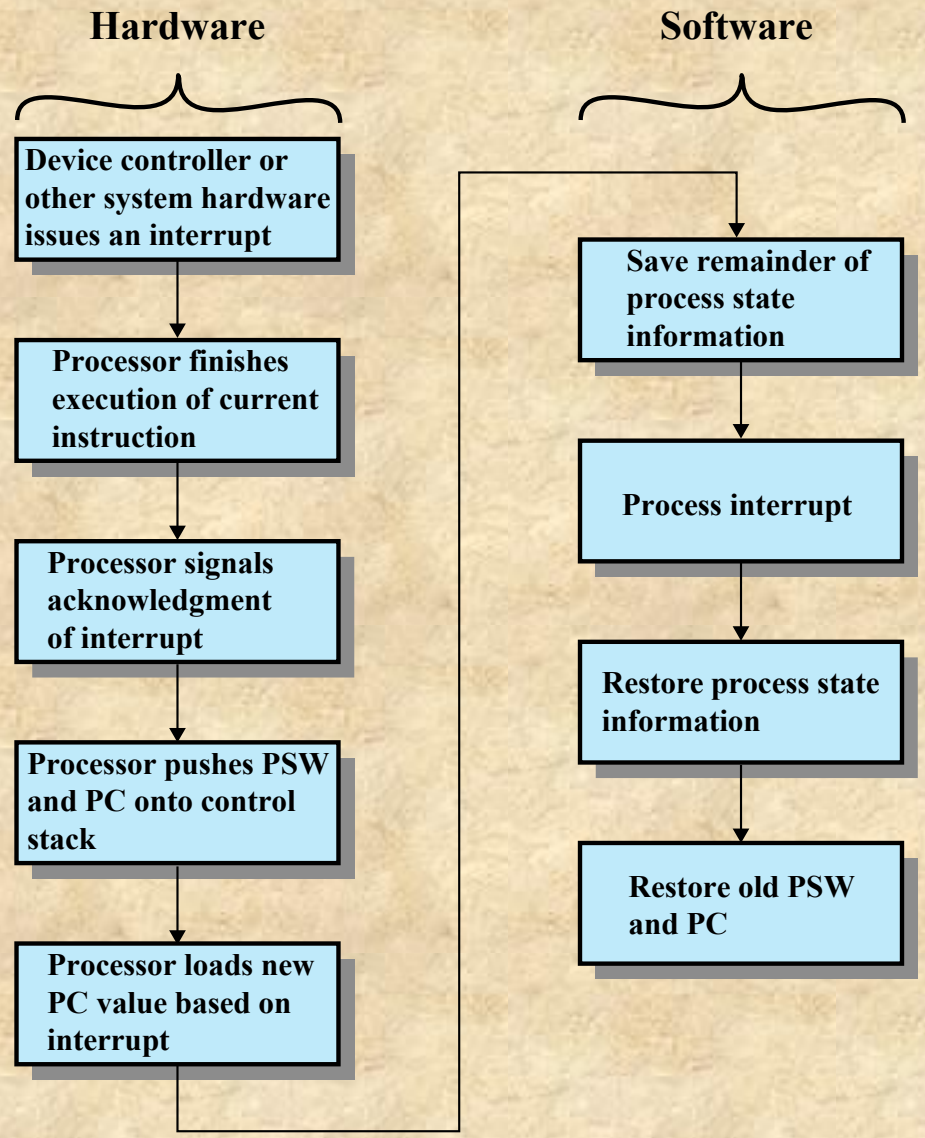


Figure 1.10 Simple Interrupt Processing

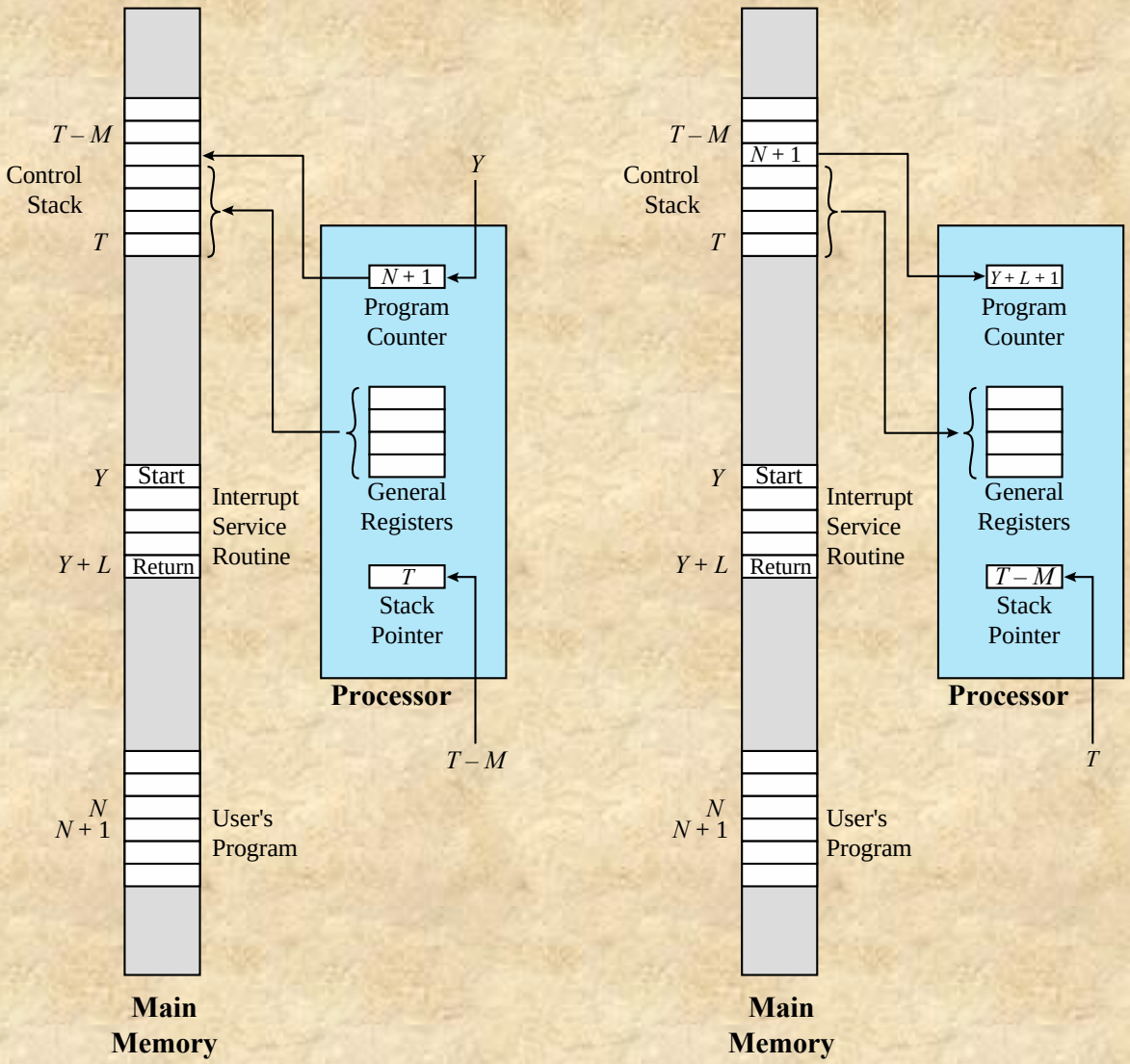


Figure 1.11 Changes in Memory and Registers for an Interrupt

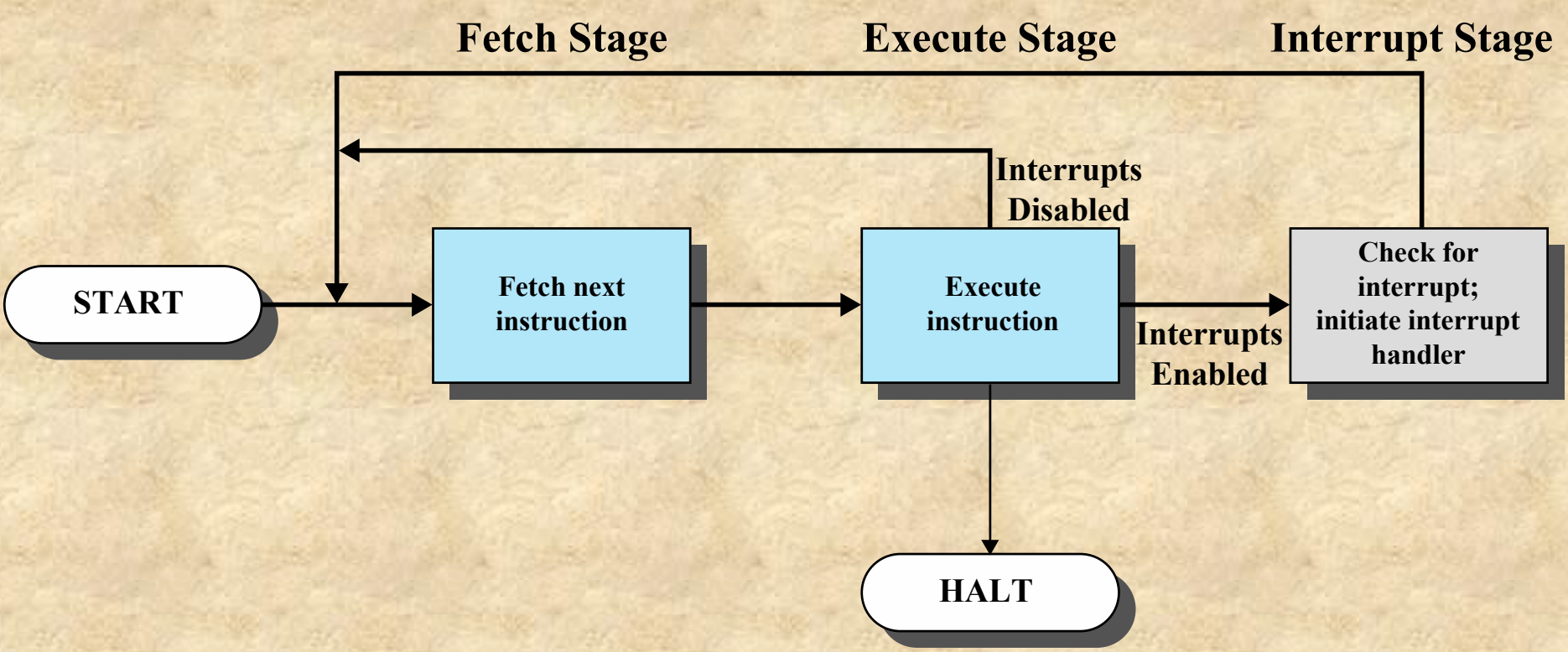


Figure 1.7 Instruction Cycle with Interrupts

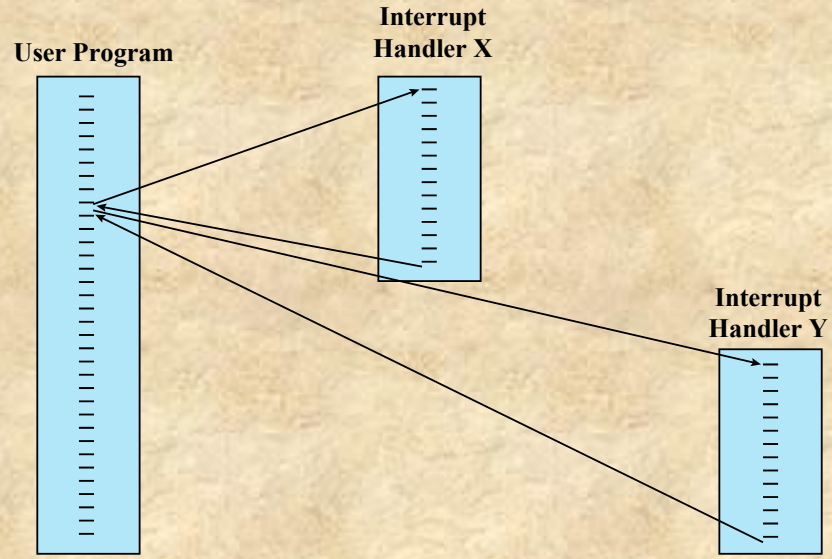
Multiple Interrupts

An interrupt occurs while another interrupt is being processed

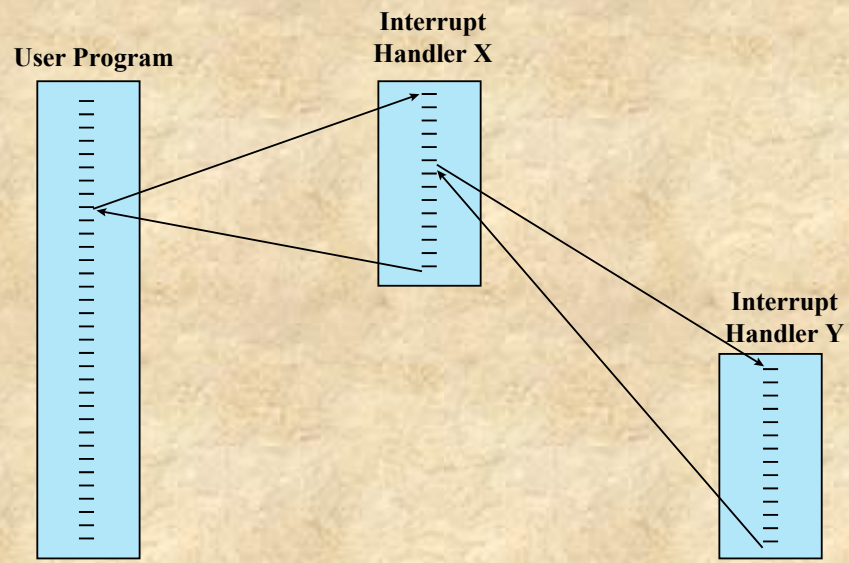
- e.g., receiving data from a communications line and printing results at the same time

Two approaches:

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

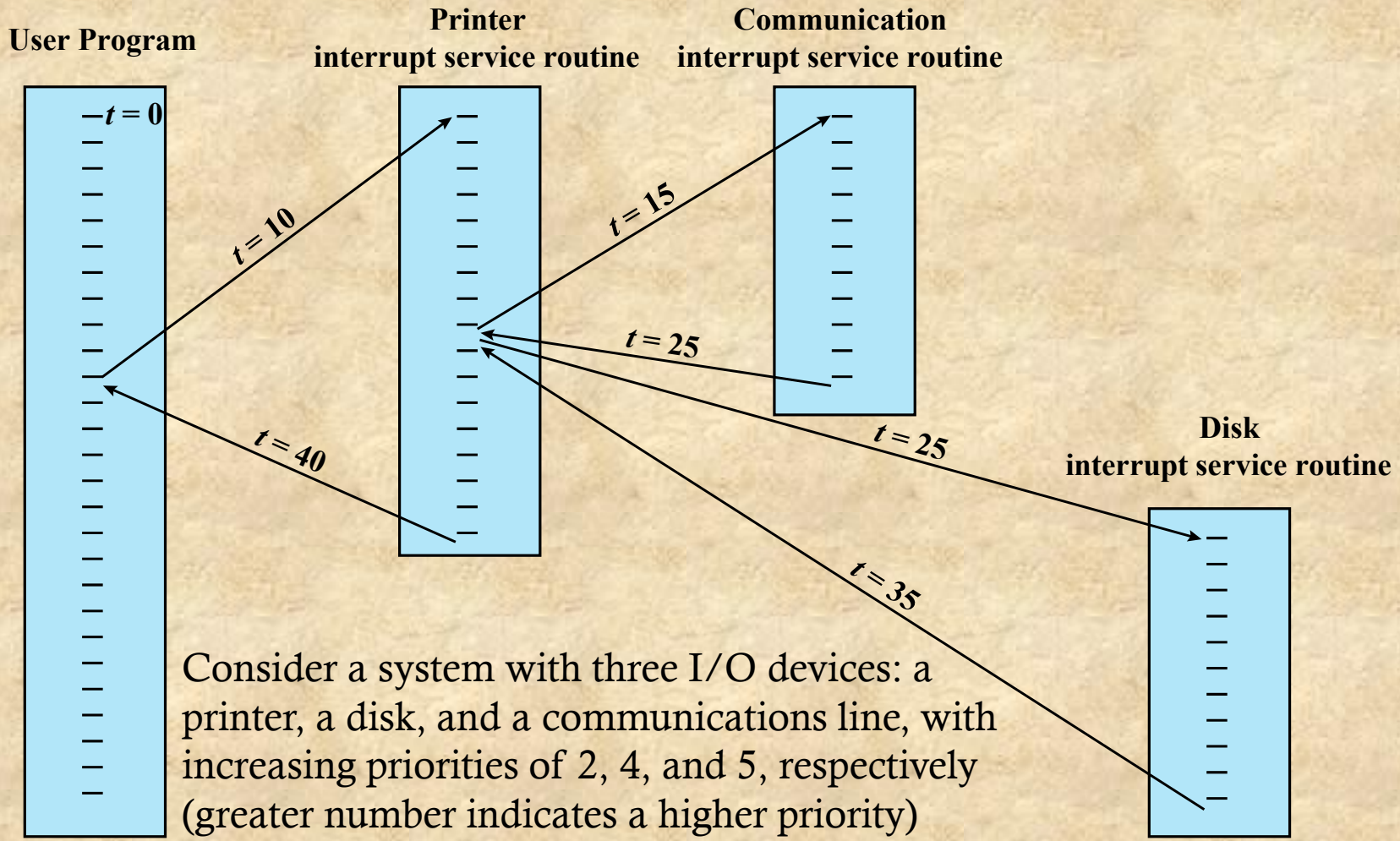


(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 1.12 Transfer of Control with Multiple Interrupts



Consider a system with three I/O devices: a printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respectively (greater number indicates a higher priority)

A printer interrupt occurs at t = 10; A communication interrupt occurs at t = 15; A disk interrupt occurs at t = 20

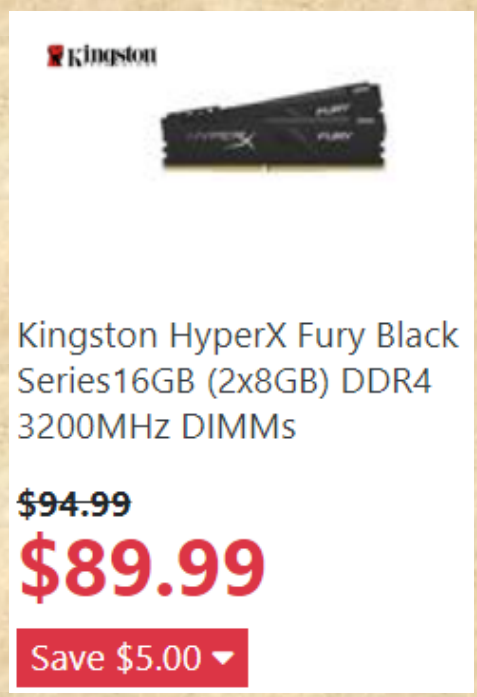
Figure 1.13 Example Time Sequence of Multiple Interrupts

Memory Hierarchy

- Design constraints on a computer's memory
  - How much? (capacity)
  - How fast? (access time)
  - How expensive? (cost)
- If the capacity is there, applications will likely be developed to use it
- Memory must be able to keep up with the processor
- Cost of memory must be reasonable in relationship to the other components

Memory Relationships

- Across different technologies
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access speed
- Need trade-off



Transfer rate up to 25600 MB/s

Read:2400 MB/s
Write:1750 MB/s

Transfer rate up to 180 MB/s

Pictures and information from Canada Computers as of Jan 22, 2021

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

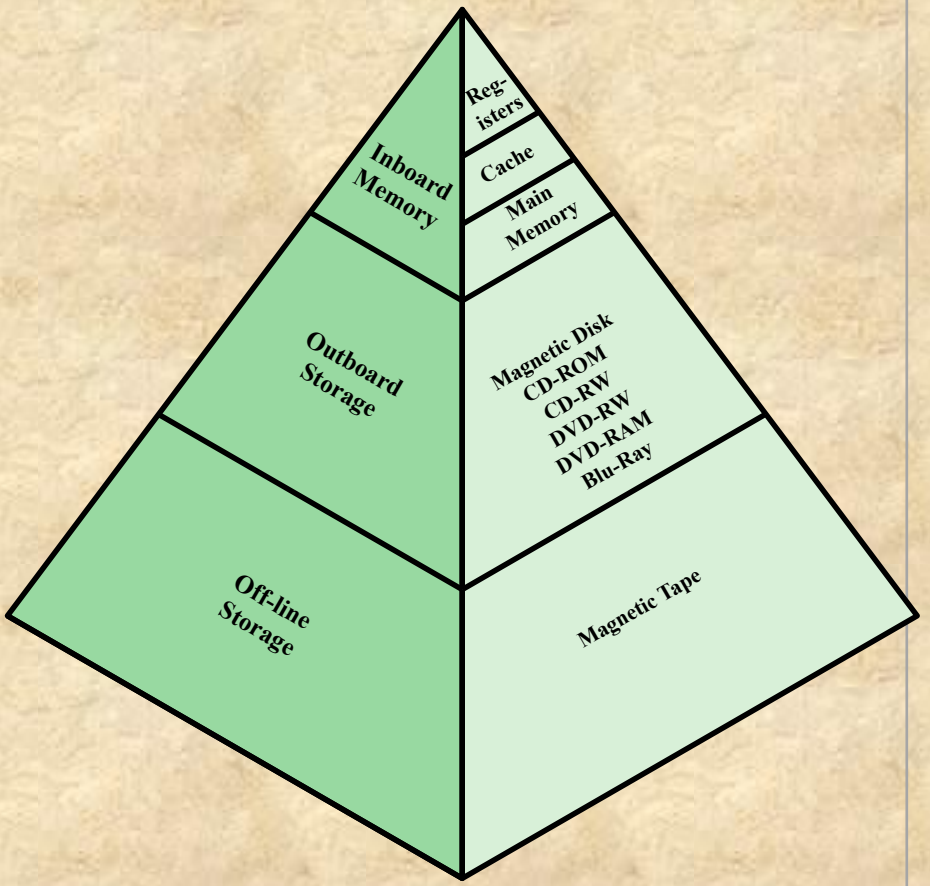
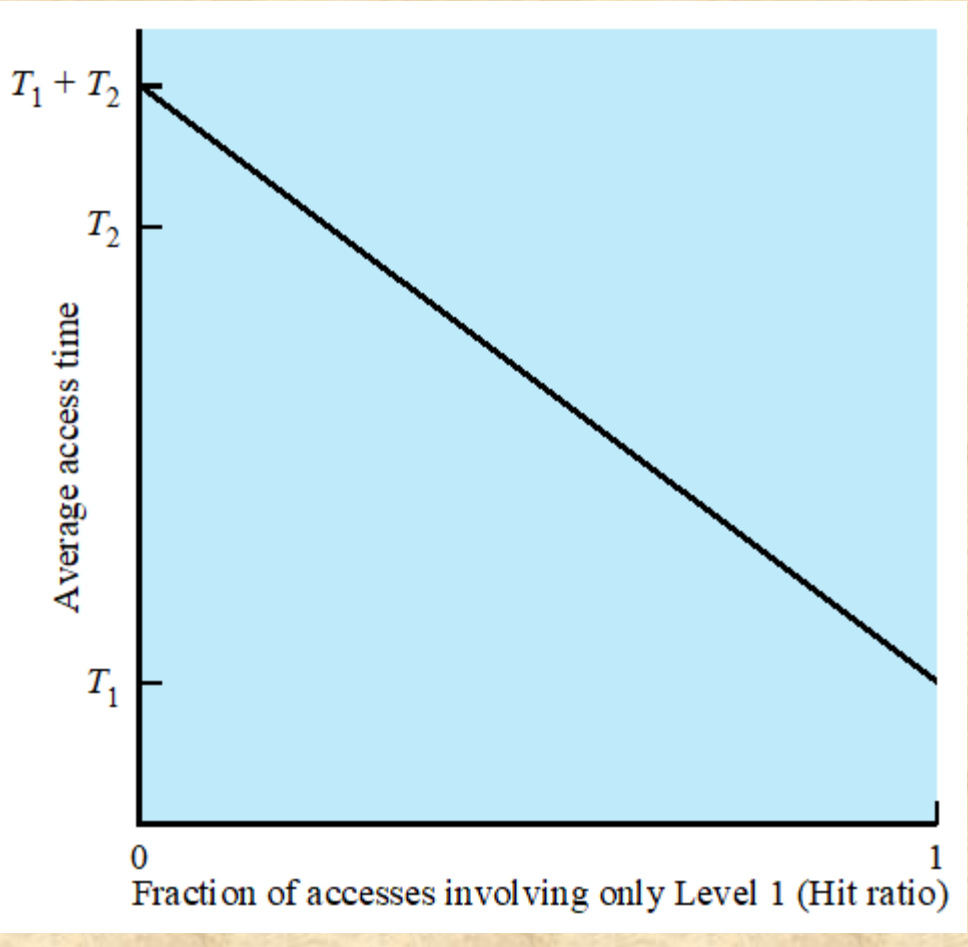


Figure 1.14 The Memory Hierarchy

Figure 15 Performance of a Simple Two-Simple Two-Level Memory



The figure shows the average access time to a two-level memory as a function of the **hit ratio** H

- H is defined as the fraction of all memory accesses that are found in the faster memory (e.g., the cache)
- T_1 is the access time to level 1, and T_2 is the access time to level 2
- Assume that if a byte to be accessed is in level 1 (a *hit*), then the processor accesses it directly. Otherwise (a *miss*) the byte is first transferred to level 1 and then accessed by the processor

Cache Example

- Suppose 95% of the memory accesses are found in the cache (H=0.95), T_1 = 0.1 μs, and T_2 = 1 μs. Then, what is the average time to access a byte?

Cache Example

- Suppose 95% of the memory accesses are found in the cache (H=0.95), T_1 = 0.1 μs, and T_2 = 1 μs. Then, what is the average time to access a byte?

(0.95)*(0.1 μs) + (0.05)*(0.1 μs + 1 μs) = 0.15 μs

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

Temporal and Spatial Locality

- **Temporal locality**: Recently accessed items will be accessed in the near future (e.g., code in loops)
- **Spatial locality**: Items at addresses close to the addresses of recently accessed items will be accessed in the near future (sequential code, elements of arrays)

https://courses.cs.washington.edu/courses/cse471/07sp/lectures/Lecture7.pdf

Locality Example

```
sum = 0;

for (int i=0; i<array.length; i++)

{

    sum = sum + array[i];

}
```

Spatial locality: array elements are referenced in succession; instructions are referenced in sequence

Temporal locality: the **sum** variable is referenced in each iteration; the instruction is repeated through the cycle

Program counter (PC) holds the address of the next instruction to be fetched

Instruction register (IR) holds the instruction being executed

Accumulator (AC): Temporary storage

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

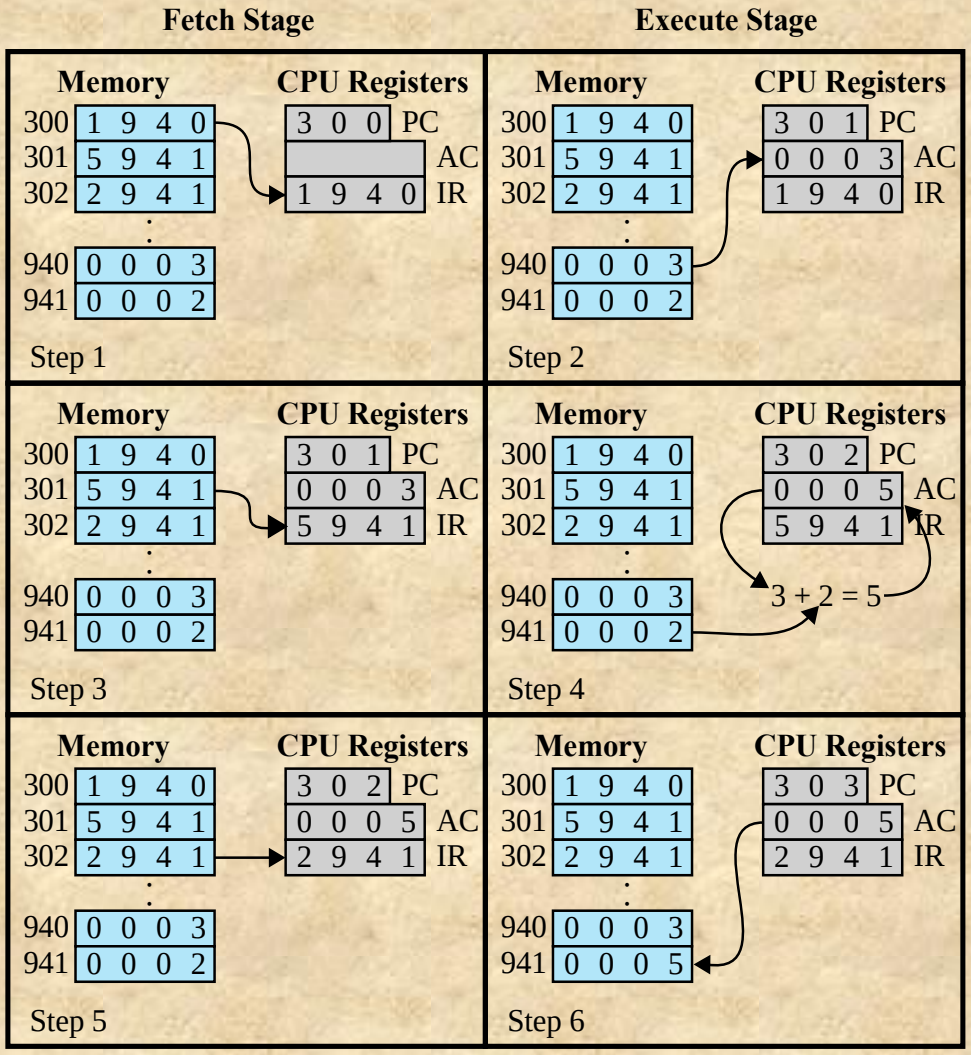


Figure 1.4 Example of Program Execution (contents of memory and registers in hexadecimal)

Secondary Memory

Also referred to as auxiliary memory

- External
- Nonvolatile
- Used to store program and data files

Cache Memory

- Invisible to the OS
- Interacts with other memory management hardware
- Processor must access memory at least once per instruction cycle
- Processor execution is limited by memory cycle time
- Exploit the principle of locality with a small, fast memory

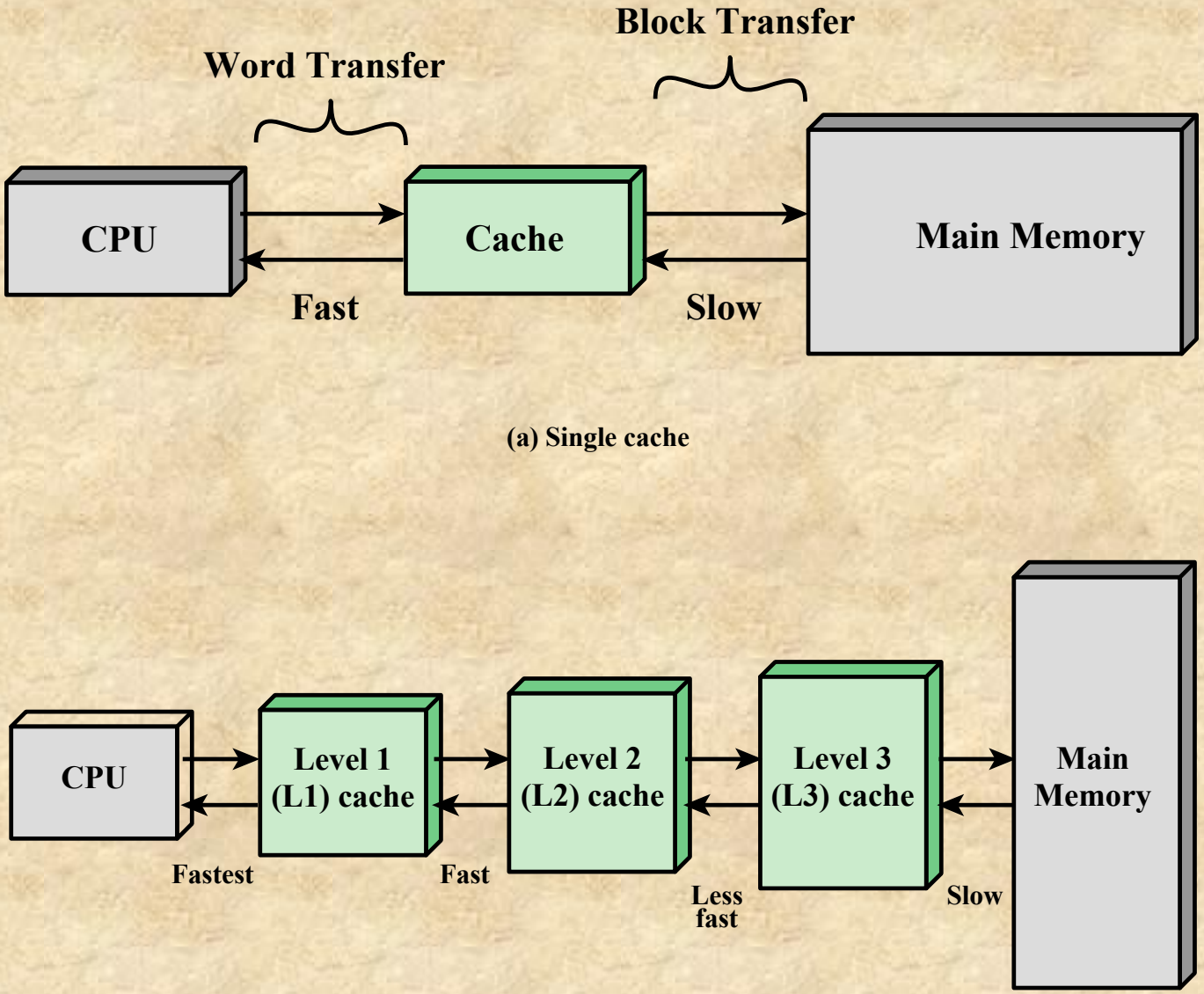
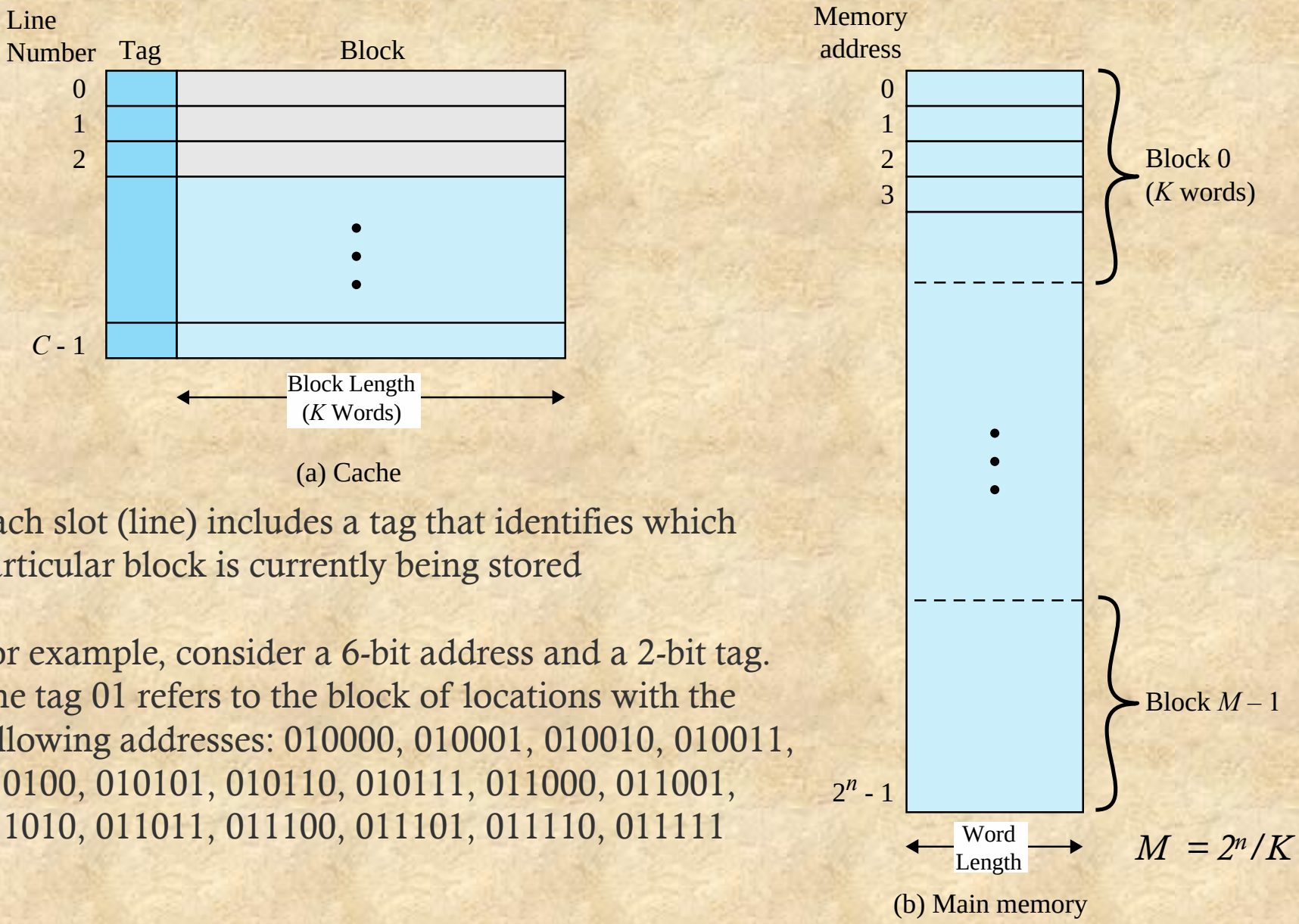


Figure 1.16 Cache and Main Memory



Each slot (line) includes a tag that identifies which particular block is currently being stored

For example, consider a 6-bit address and a 2-bit tag. The tag 01 refers to the block of locations with the following addresses: 010000, 010001, 010010, 010011, 010100, 010101, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, 011111

Figure 1.17 Cache/Main-Memory Structure

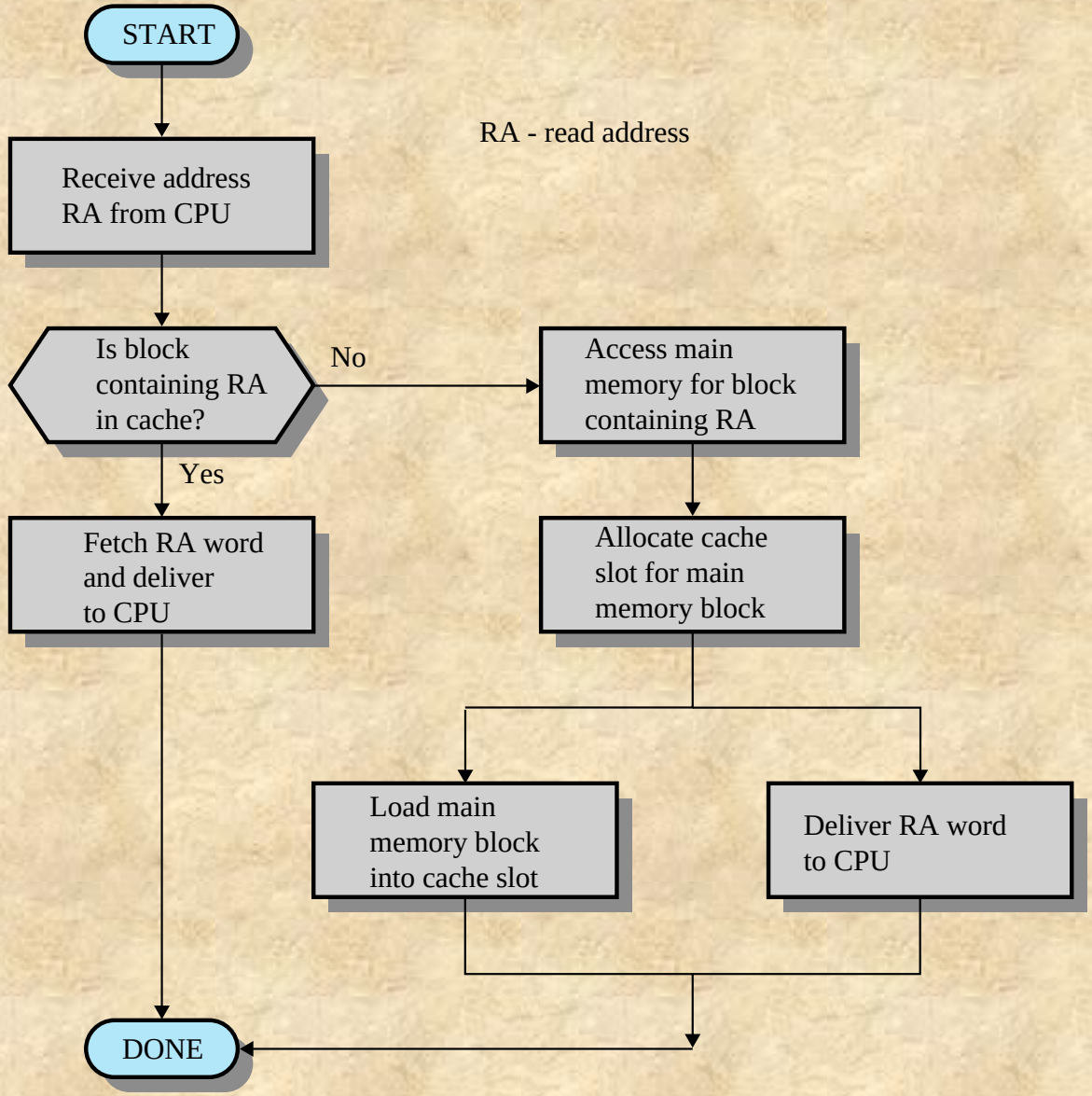
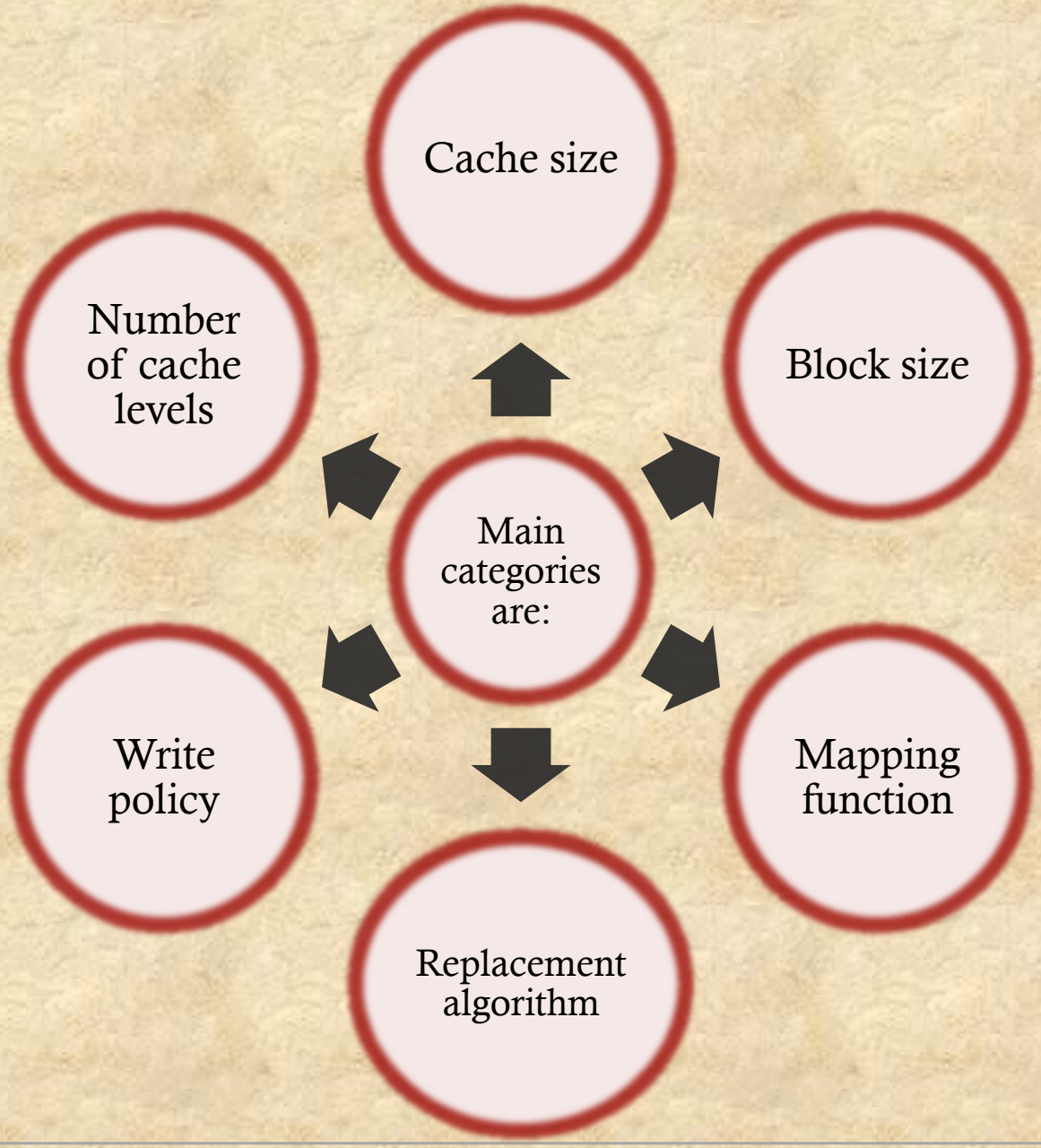


Figure 1.18 Cache Read Operation

Cache Design



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

Mapping Function

- Determines which cache location the block will occupy

Two constraints affect design:

When one block is read in, another may have to be replaced

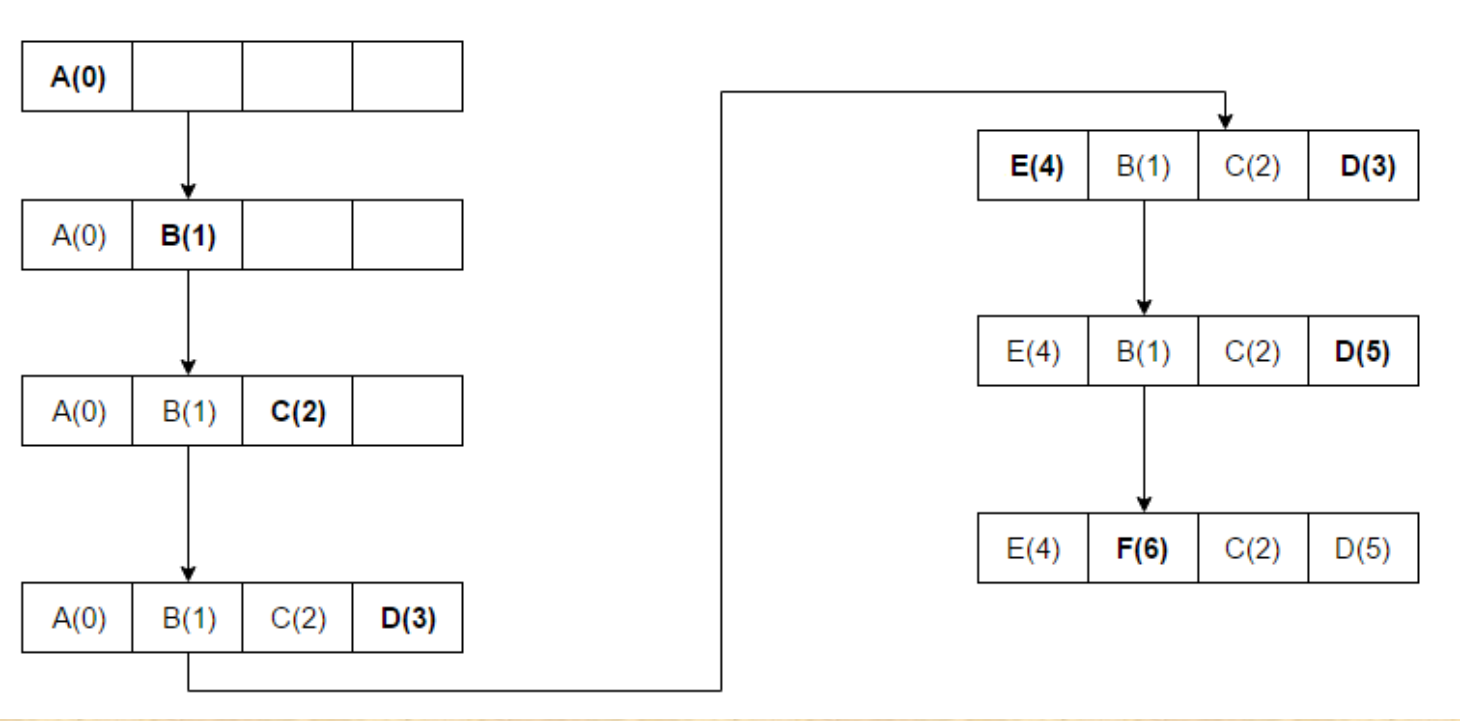
The more flexible the mapping function, the more complex is the circuitry required to search the cache

Replacement Algorithm

- The **replacement algorithm** chooses, within the constraints of the mapping function, which block to replace when a new block is to be loaded into the cache and the cache already has all slots filled with other blocks
- Least Recently Used (LRU) Algorithm
  - Replace a block that has been in the cache the longest with no references to it
  - Hardware mechanisms are needed to identify the least recently used block

LRU Example

The access sequence is A B C D E D F



https://en.wikipedia.org/wiki/Cache_replacement_policies

Picture By Advaitjavadekar - https://en.wikipedia.org/wiki/File:Lruexample.png, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=53645703

Write Policy

Dictates when the memory write operation takes place

- Can occur every time the block is updated (write-through)
- Can occur when the block is replaced (write-back)
  - Minimizes write operations
  - Leaves main memory in an obsolete state
    - Cache and main memory have different data

I/O Techniques

- When the processor encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module

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 then sets the appropriate bits in the I/O status register
- 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

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 Drawbacks

- 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

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/written

Direct Memory Access

- Transfers the entire block of data directly to and from memory without going through the processor
  - Processor is involved only at the beginning and end of the transfer
  - Processor executes more slowly during a transfer when processor access to the bus is required
- More efficient than interrupt-driven or programmed I/O

Symmetric Multiprocessors (SMP)

- A stand-alone computer system with the following characteristics:
  - 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
  - Processors share 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 element levels

SMP Potential Advantages

Performance

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

Scaling

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

Availability

- The failure of a single processor does not halt the machine

Incremental Growth

- An additional processor can be added to enhance performance

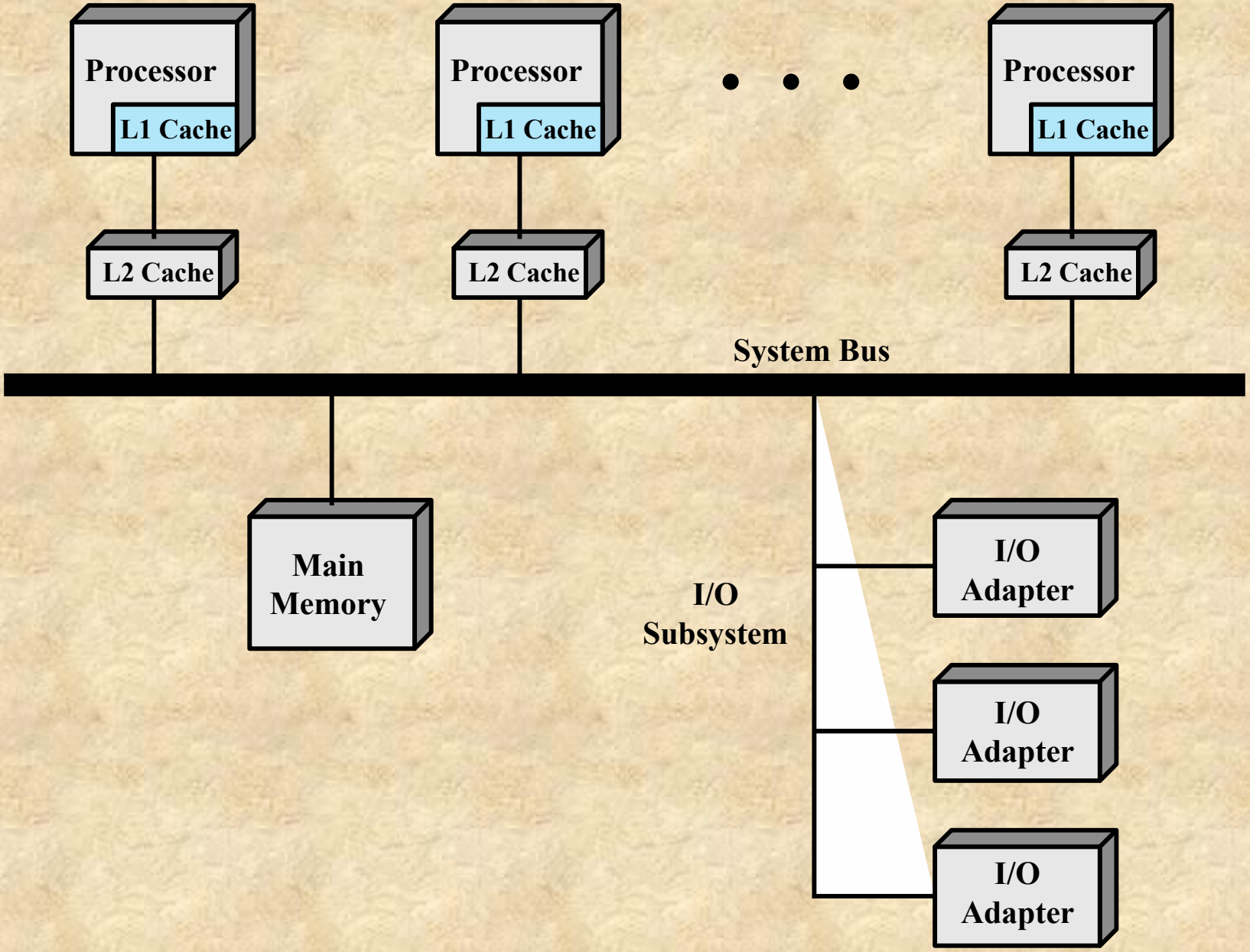
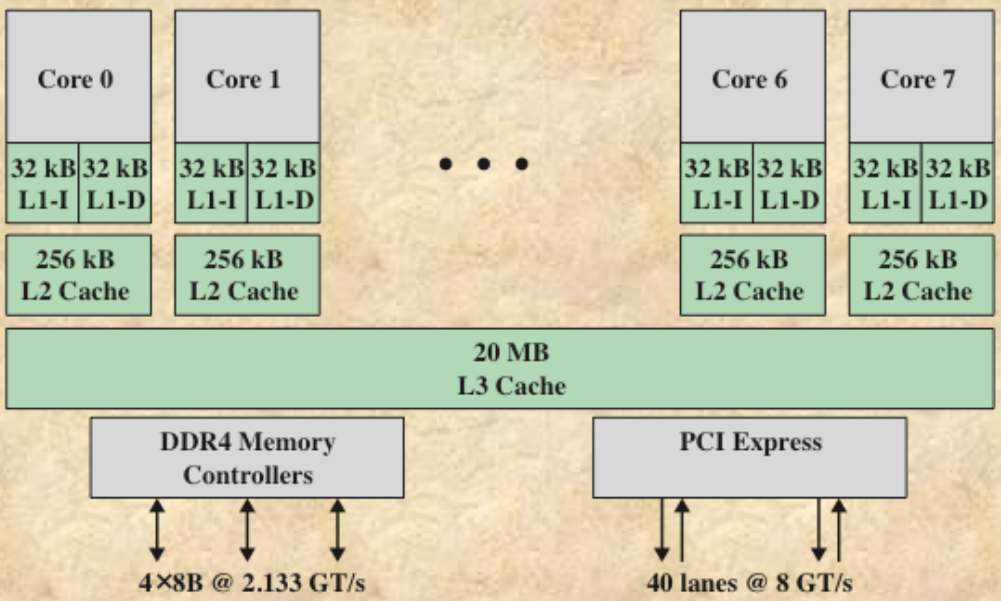


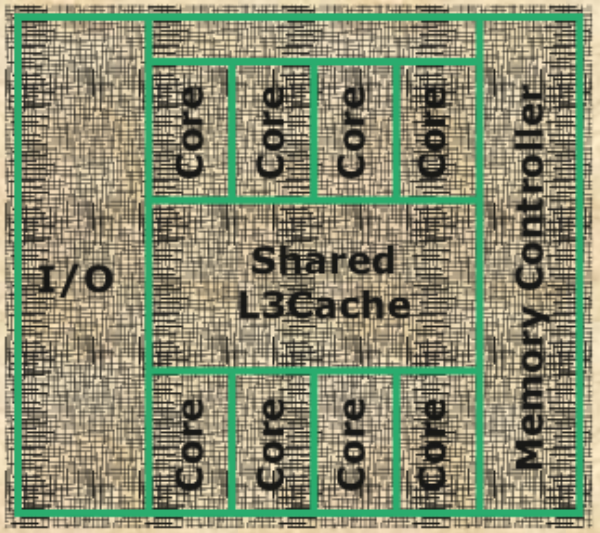
Figure 1.19 Symmetric Multiprocessor Organization

Multicore Computer

- Also known as a chip multiprocessor
- 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
- In addition, multicore chips also include L2 cache and in some cases L3 cache



(a) Block diagram



(b) Physical layout on chip

Figure 1.20 Intel Core i7-5960X Block Diagram

Summary

- Basic Elements
- Evolution of the microprocessor
- Instruction execution
- Interrupts
  - Interrupts and the instruction cycle
  - Interrupt processing
  - Multiple interrupts
- The memory hierarchy
- Cache memory
  - Motivation
  - Cache principles
  - Cache design
- Direct memory access
- Multiprocessor and multicore organization
  - Symmetric multiprocessors
  - Multicore computers

Quiz 1

- Up to 15 minutes, in class from 10:30 AM, Jan 29, Blackboard online test
- Don't be late, otherwise you may not be allowed to finish/attend
- Coverage: Introduction to Hardware, Chapter 1 of the textbook, lab1, and lab2
- True/False, MCQ, Filling in the blank, up to 20 Questions
- Open book