Chapter No. 03 – A Top-Level View of Computer Function and Interconnection

Week – 05

(Part-02)

Topics to Cover

- Interrupts
- I/O Function
- 3.3 Interconnection Structures
- 3.4 Bus Interconnection
- Bus Structure
- Multiple Buses
- Elements of Bus Design

What is an ‘Interrupt’?

- An **interrupt** is a signal to the processor from a device attached to a computer or from a program within the computer indicating an event that needs immediate attention.
- Interrupts provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.
- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.
- The processor responds by suspending its current activities, saving its state, and executing a function called an ISR to deal with the event.
- This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.

Types of Interrupts

- There are two types of interrupts:

1) Hardware interrupts 2) Software interrupts.

1. **Hardware interrupts:** are used by devices to communicate that they require attention from the **operating system**.

- 'Hardware interrupts' are generated when a key is pressed or when the mouse is moved.

2. **Software interrupt:** is caused from executing a program/instruction within the computer that requires the operating system to stop and figure out what to do next. (e.g. int 21h in assembly)

- 'Software interrupts' are generated by a program requiring user input or output.

Classes of Interrupts

- Interrupts can be of any of the four types:

1. **Program interrupt:** Generated by some condition that occurs as a result of an instruction execution e.g. overflow, division by zero.
2. **Timer interrupt:** Generated by a timer within a processor. This allows the operating system to perform certain functions on a regular basis. Used in pre-emptive multi-tasking (time for a task).
3. **I/O interrupt:** Generated by an I/O controller, to signal normal completion of an operation, or to request service from the CPU.
4. **Hardware Failure:** Generated by a failure such as power failure or memory parity error (bit), or to signal a variety of error conditions.

Importance of Interrupts

- The purpose of **interrupts** is to improve processing efficiency.
- For example, most external devices are much slower than the CPU.
- Suppose that the processor is transferring data to a printer using the ‘instruction cycle’ scheme.
- After each ‘write’ operation, the processor must pause and remain idle until the printer catches up.
- The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory.
- Clearly, this is a very wasteful use of the processor.
- During the wait cycles, the CPU can work on other task until interrupt.

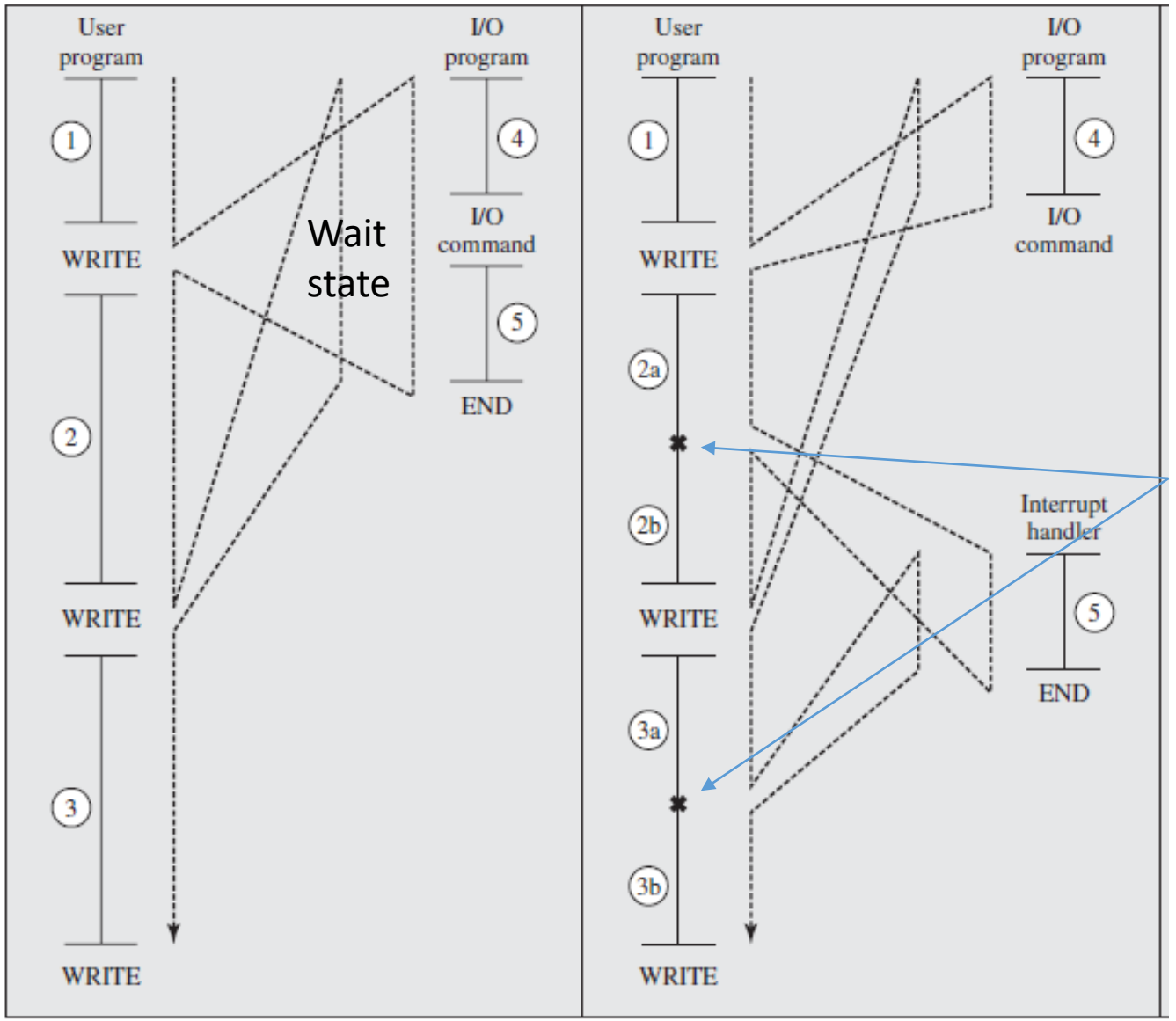
Fig. 3.7 Program Flow Control (Next Slide)

- The user program performs a series of WRITE calls to an I/O device interleaved with processing. Code segments 1, 2 and 3 refer to sequence of instructions that do not involve I/O.
- The WRITE calls are to an I/O program (driver) that is a system utility and that will perform the actual I/O operation (to manage & control computer resources).
- Because the I/O operation may take a relatively long time to complete, the user 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.

Stages of an I/O Write Operation (Figure Next)

- The I/O program consists of three sections:
- Section-1: A sequence of instructions, labelled 4 in the figure, 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.
- Section-2: The actual I/O command. Without the use of interrupts (Fig.a), once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically poll the device). The program may wait to determine if the operation is done.
- Section-3: A sequence of instructions, labelled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.

Program Flow Control without and with Interrupts



Interrupt Cycle added to Instruction Cycle

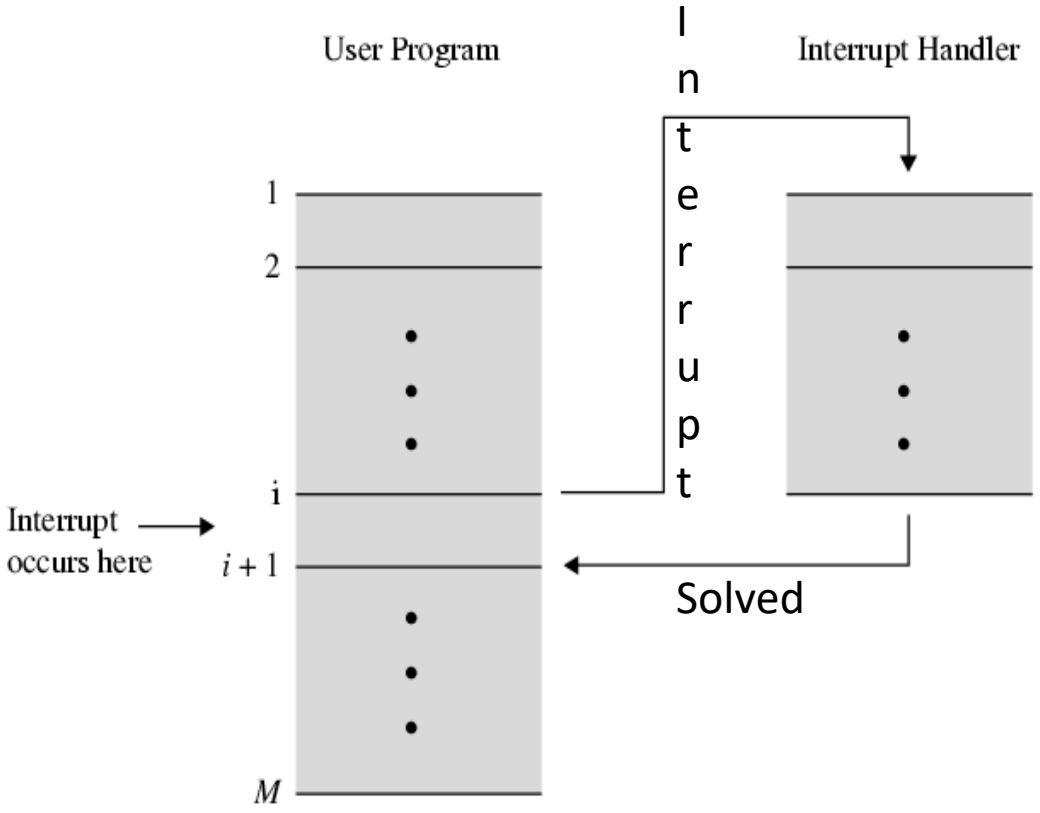
- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- As before, the user program reaches a point at which it makes a system call in the form of a WRITE call.
- The I/O program that is invoked in this case consists only of the preparation 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.
- The I/O operation is carried along with the execution of user program.

Interrupt Handler (See Fig. Last Slide)

- 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 the external device sends an *interrupt request signal* to the processor. The processor responds by suspending operation of the current program, branching off to a program to service that particular I/O device, known as **interrupt handler.** (an OS program)
- The processor resumes the original program execution after the device is serviced.
- The points at which such interrupts occur are shown with **X** in figure.

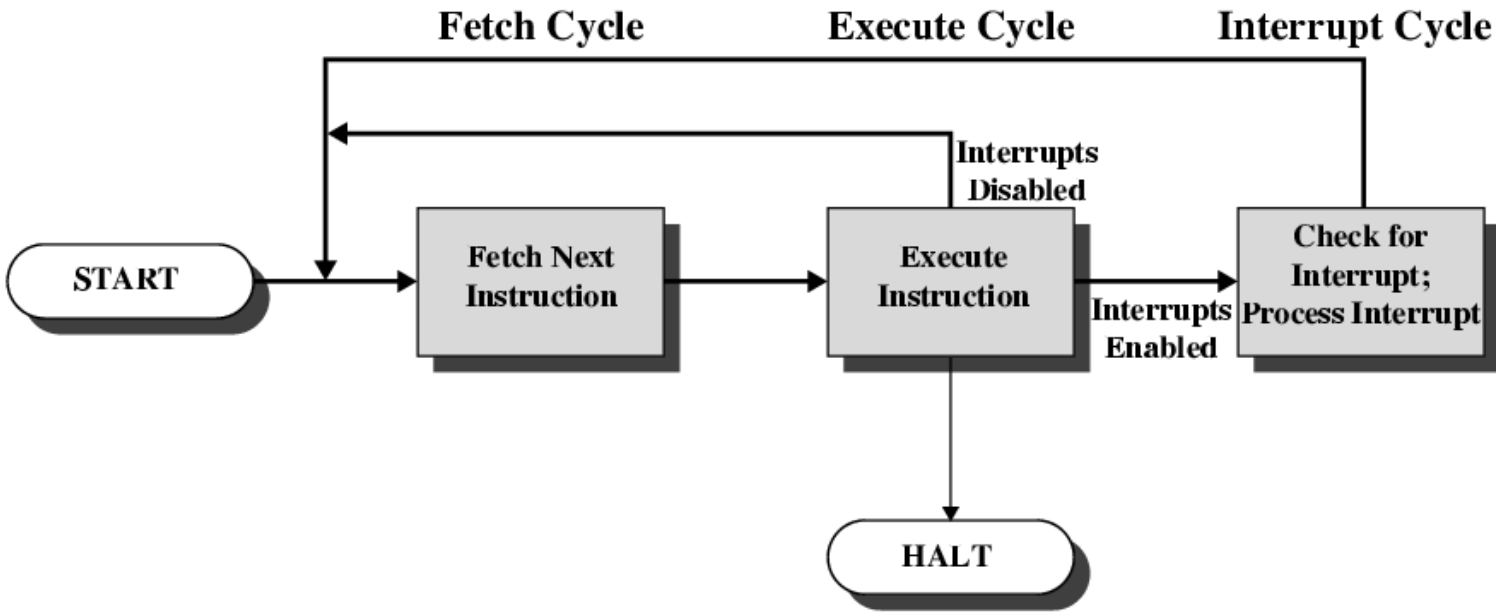
Fig. 3.8 Transfer of Control via Interrupts

- From the point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution.
- When the interrupt processing is completed, execution resumes (Fig. 3.8).
- Thus, the user program does not have to contain any special code to accommodate interrupts;
- The processor and the operating system are responsible for suspending the user program and then resuming it at the same point.



Instruction Cycle with 'Interrupts Cycle'

- To accommodate interrupts, an *interrupt cycle* is added to the 'instruction cycle'. (See Fig. 3.9 below)
- In the interrupt cycle the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program.



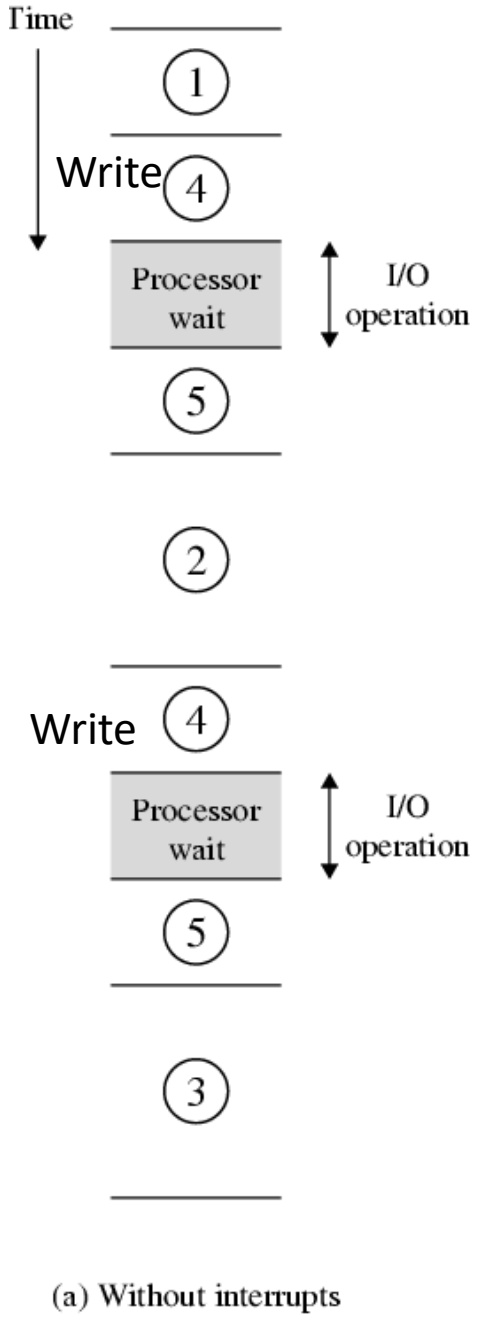
Steps of ‘Interrupt Handler Process’

- If an interrupt is pending, the processor does the following:

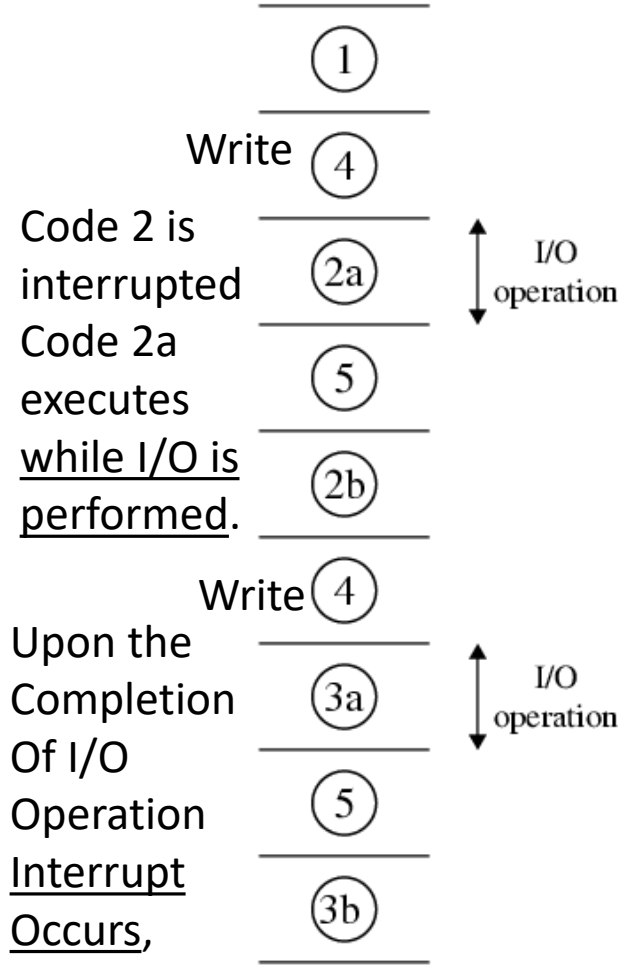
1. It suspends execution of the current program being executed and saves its context to the stack. It also saves the current contents of the PC (program counter) that contains the address of the next instruction to be executed.
2. It sets the program counter to the starting address of the *interrupt handler routine*. (it determines nature of interrupt and does action).
3. The processor now proceeds to the fetch cycle and fetches the first instruction in the ‘interrupt handler program’, which will service the interrupt. (this program is part of the operating system).
4. When the interrupt handler routine is completed, the processor can resume execution of the user program at the point of interruption.

‘Program Timing’, Short I/O Wait

- Timing diagram based on the flow of control.
- In Fig. (a), without interrupts, during I/O operations, the processor waits while an I/O operation is performed.
- In Fig. (b), with interrupts, the I/O operation is carried in concurrent with processor executing.
- This saves valuable processing time.
- Result: Gain in ‘system performance’.

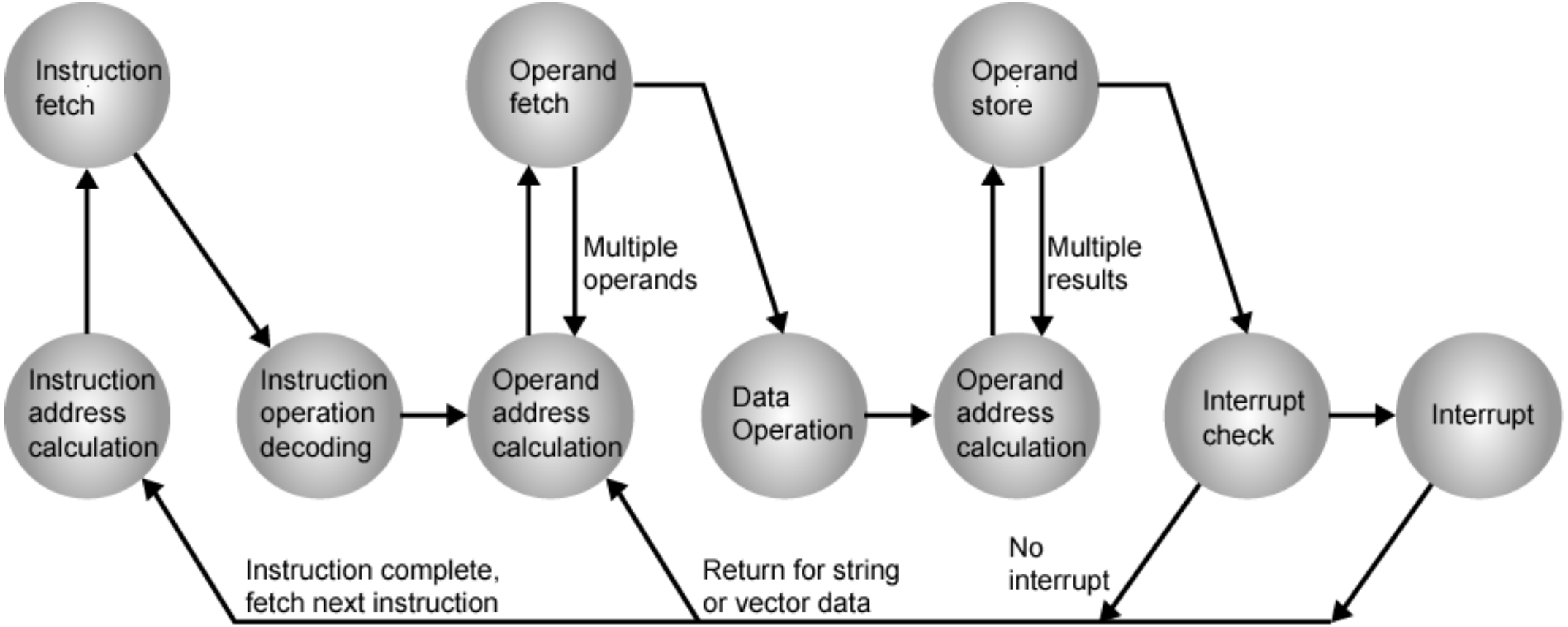


(a) Without interrupts



(b) With interrupts

Instruction Cycle, ‘State Diagram’ with Interrupts



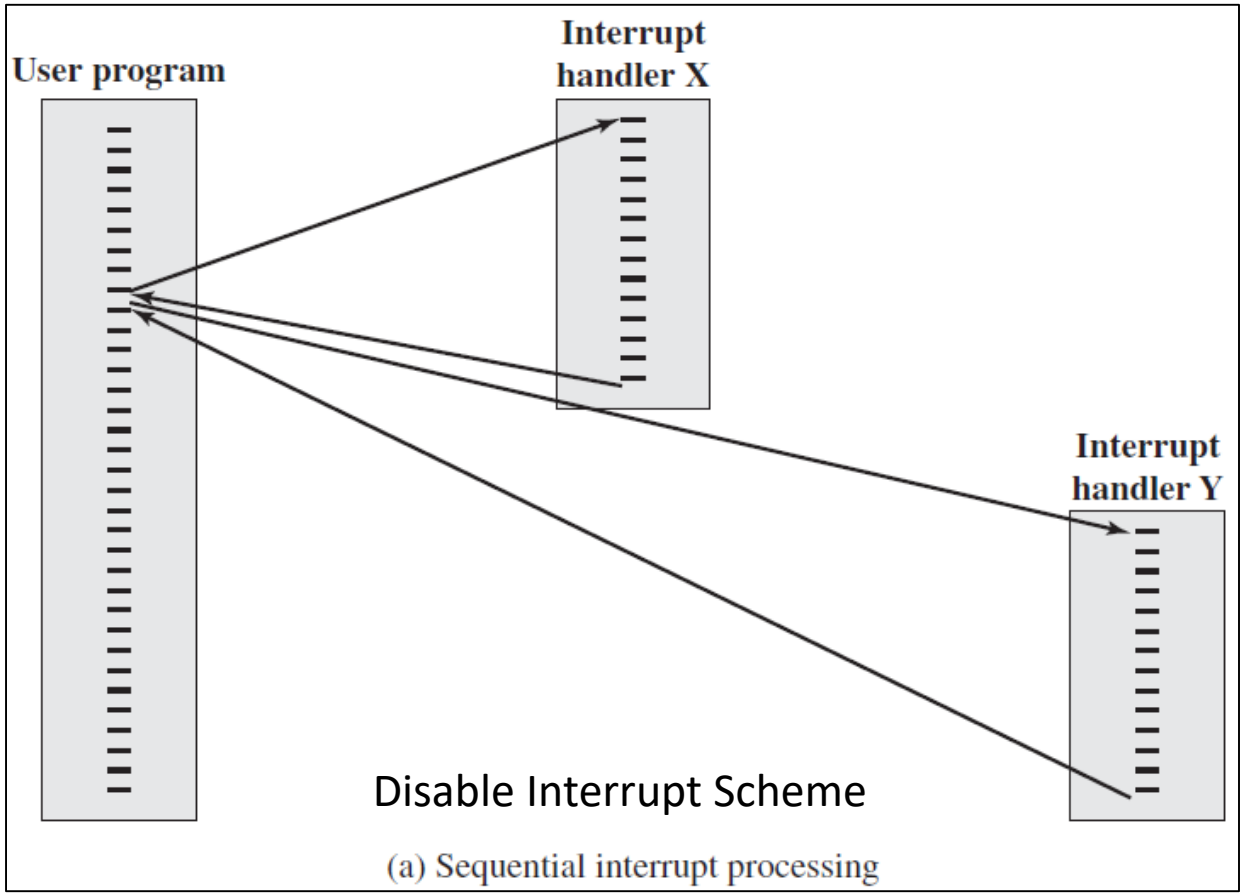
Multiple Interrupts

- There is a possibility that multiple interrupts can occur at a time.
- For example, a program may be receiving data from communication line and printing results.
- The printer will generate interrupt every time it completes a printer operation.
- The communication line controller will generate an interrupt every time a unit of data arrives. (e.g. a character or a block of data)
- In any case, it is possible for a communications interrupt to occur while a printer interrupt is being processed.

Two Approaches to Dealing with ‘Multiple Interrupts’

1. **Disable interrupts (Maskable interrupt)**
   - Processor will ignore further interrupts whilst processing one interrupt.
   - Interrupts remain pending and are checked after first interrupt (ISR) has been processed.
   - Interrupts handled in sequence as they occur. (e.g. data may be lost on a line)
   - Drawback, it does not take into account relative priority, or time critical needs.
2. **Define priorities (Non-Maskable interrupt)**
   - Low priority interrupts can be interrupted by higher priority interrupts.
   - When higher priority interrupt has been processed, processor returns to previous interrupt.

Fig. 3.13 Transfer of Control with Multiple Interrupts



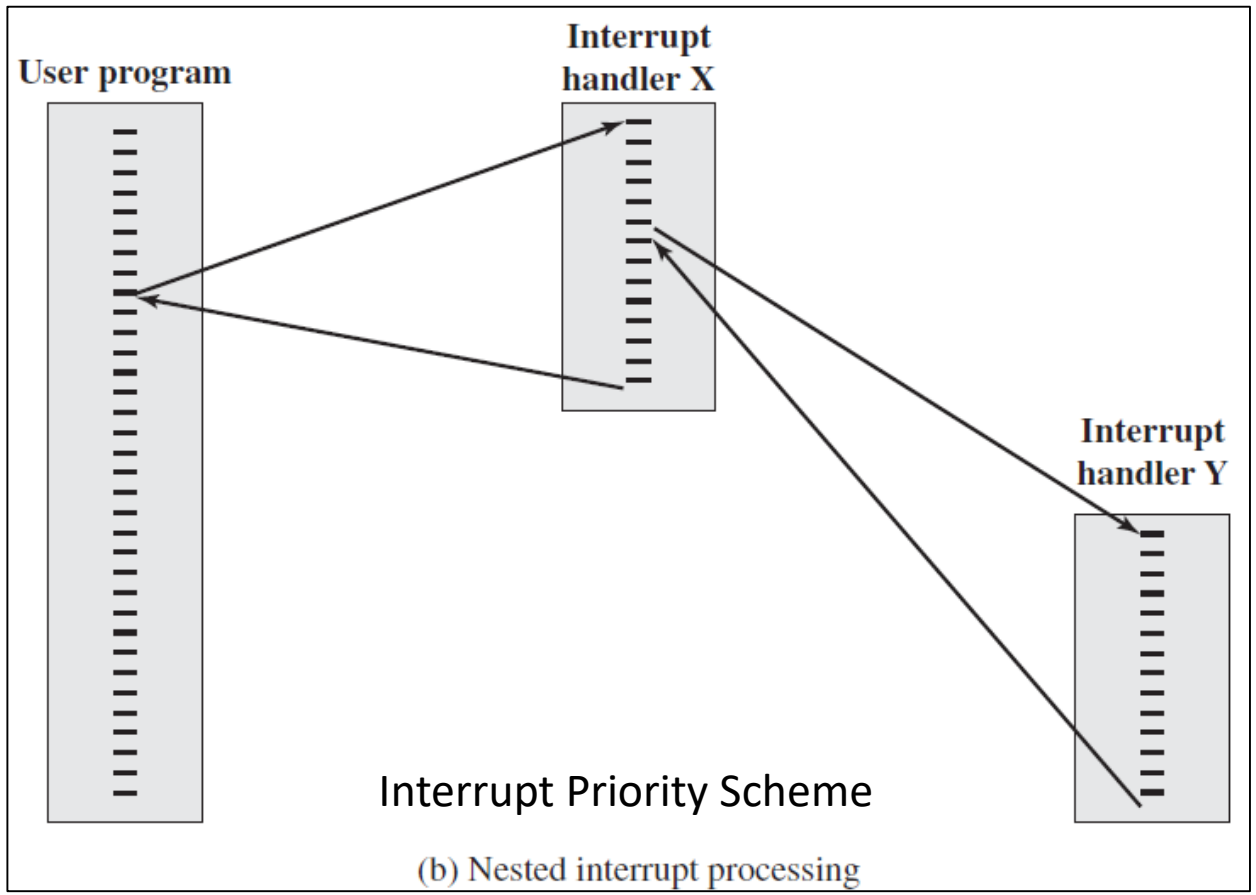


Fig. 3.14 Example Time Sequence of Multiple Interrupts

- This is an example of <u>interrupt priority</u> scheme.
- Consider a system with three I/O devices: a printer, a disk and a communication line.
- With increasing priorities of 2,4 & 5.
- Program begins t=0.

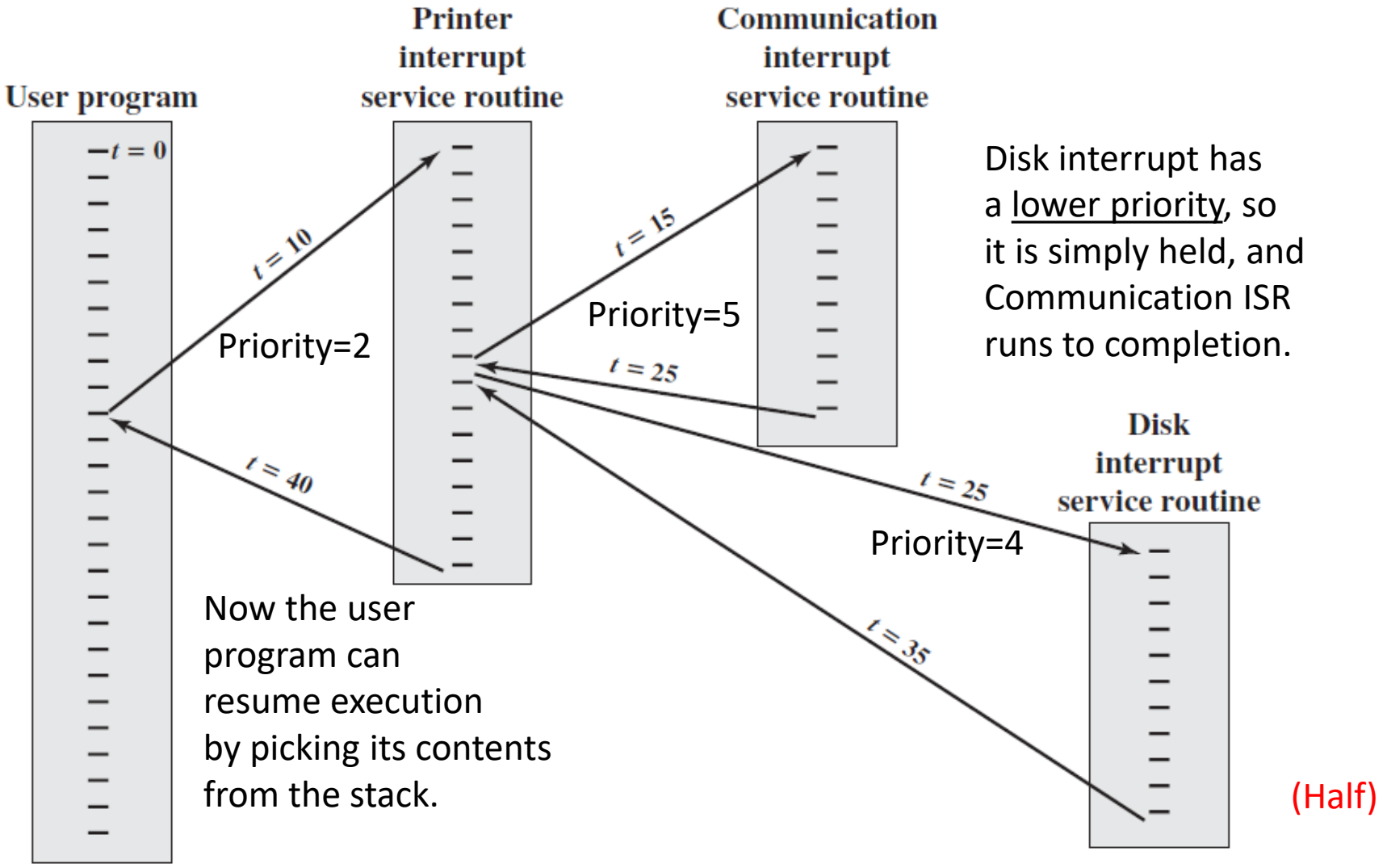


Figure 3.14 Example Time Sequence of Multiple Interrupts

(Half)

3.3 Interconnection Structure

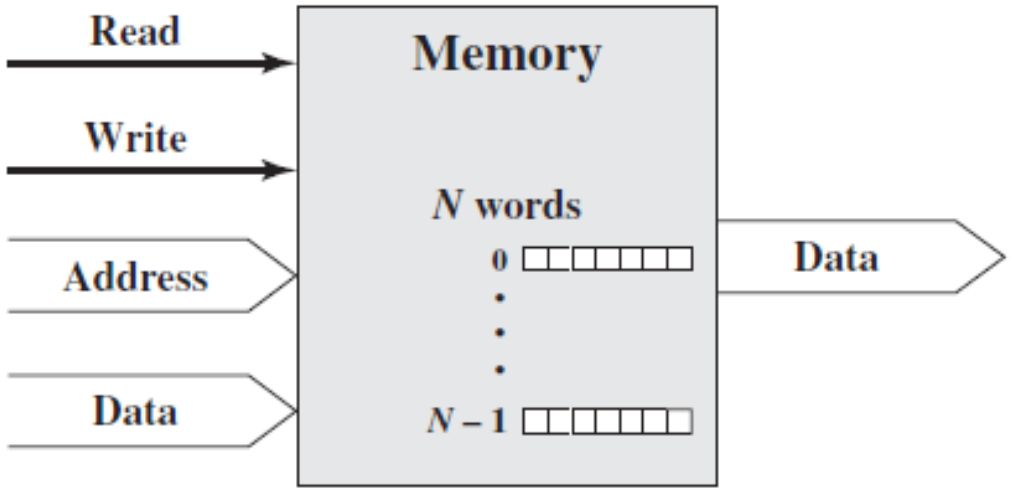
Skip

- A computer consists of a set of components or modules of three types:
  - Processor
  - Memory
  - Input/output (I/O)
- These modules need to communicate with each other, so there must be paths for connecting the modules.
- The collection of paths connecting the various modules (CPU, memory & I/O) is called the *interconnection structure*.
- For each module type has its own forms of input and output.

Memory Module and its Data Exchange**Skip**

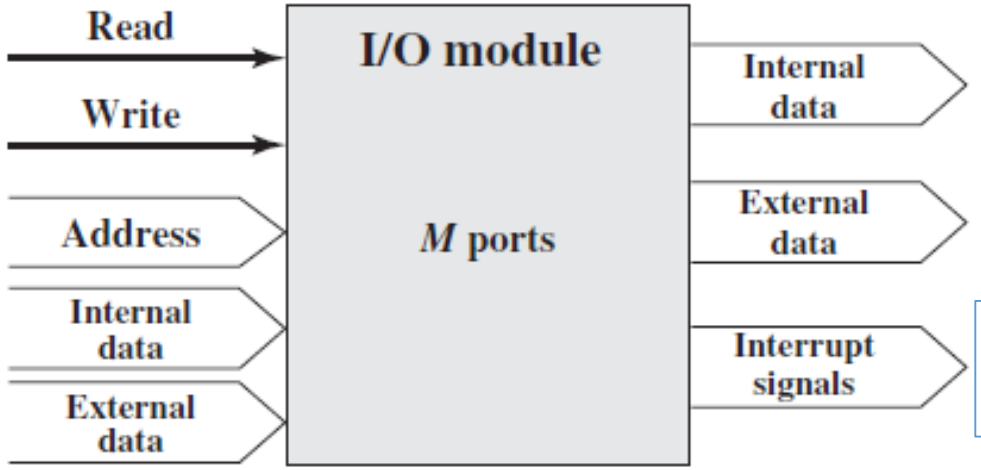
- A memory module consists of N words of equal length.
- Each word is assigned a unique numerical address from (0, 1, …, N-1).
- A word of data can be read from or written into the memory.
- The nature of operation is indicated by read and write control signals.
- The location for the operation is specified by an address.

Note: The wide arrows represent multiple signal lines, carrying multiple bits of information in parallel.



I/O Module and its Data Exchange **Skip**

- From the computer's point of view, I/O is exactly similar to memory.
- There are two operations read and write.
- An I/O module may control more than one external device.
- We refer to each of the interfaces to an external device as a **port** and give each a unique address (e.g., 0, 1, …, M-1).
- An I/O module may be able to send interrupt signals to the processor.



Input

Receive data from peripheral
Receive data from computer

Output

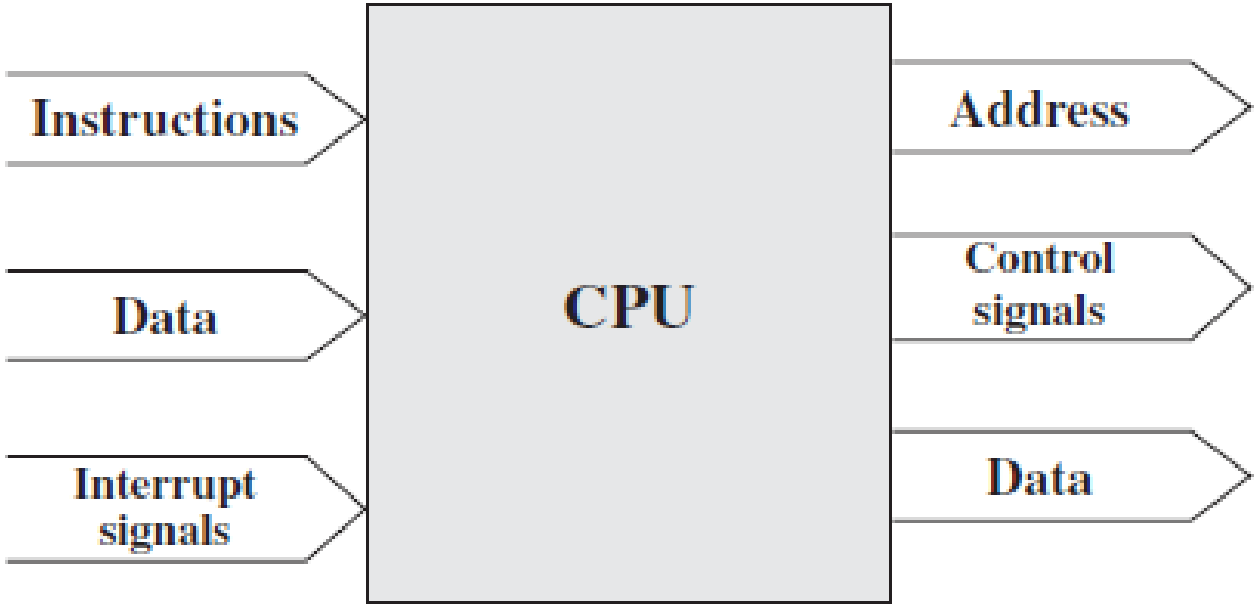
Send data to computer
Send data to peripheral

Internal Data: Data inside system.
External Data: Data outside system.

Processor Module and its Data Exchanges

Skip

- The processor reads in instructions and data, writes out data after processing.
- It uses control signals to control the overall operation of the system.
- It also receives interrupt signals and acts on them.



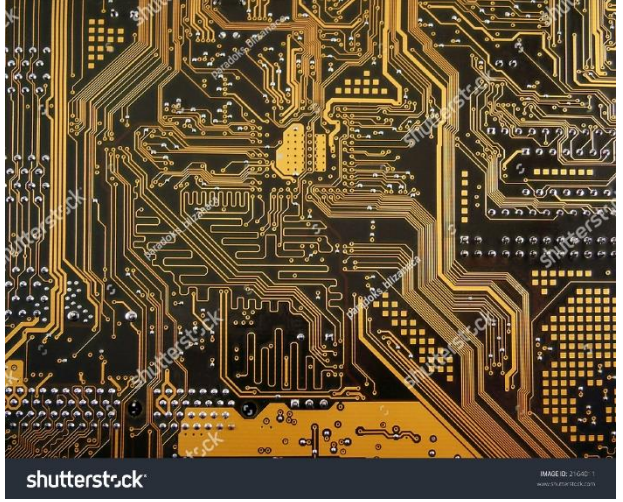
Types of Data Transfer b/w Computer Modules

Skip

- The interconnection structure must support the following types of transfer:

1. **Memory to processor:** The processor reads an instruction or a unit of data from memory.
2. **Processor to memory:** The processor writes a unit of data to memory.
3. **I/O to processor:** The processor reads data from an I/O device via an I/O module.
4. **Processor to I/O:** The processor sends data to the I/O device.
5. **I/O to or from memory:** Direct memory access (DMA) via I/O.

3.4 Bus Interconnection



- A bus is a shared transmission medium.
- A **Bus** is a communication pathway connecting two or more devices.
- A bus consists of multiple communication pathways, or lines, often grouped. A **System Bus** connects major computer components.
  - A number of line/channel in one bus, each capable of carrying one bit.
  - Over time, a sequence of binary digits can be transmitted over a single line.
  - Taken together, several lines of a bus can be used to transmit binary data simultaneously (in parallel).
  - e.g. 32 bit data bus is 32 separate single bit channels.
  - For example, an 8-bit unit of data can be transmitted over eight bus lines.

Clock Cycles

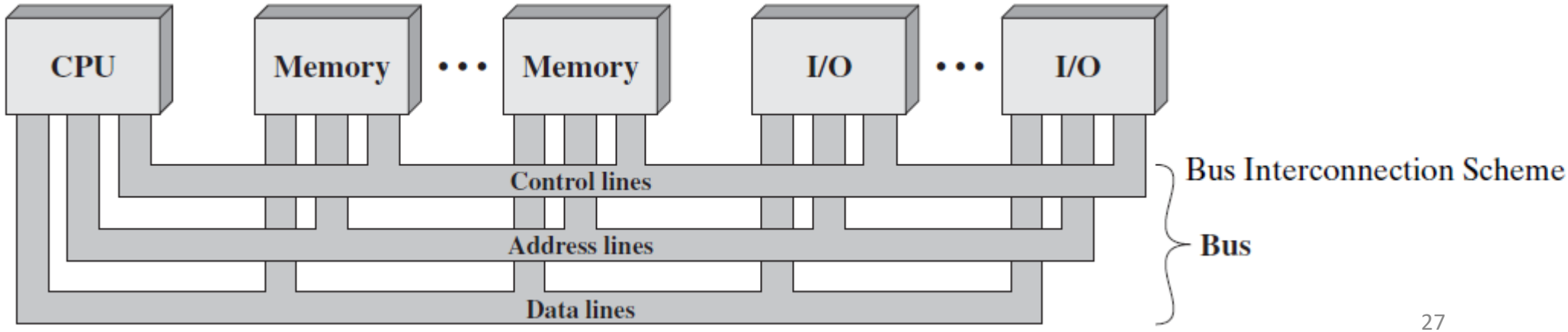
Types of Buses

- A system bus consists typically, of from about fifty to hundreds of separate lines.
- Each line is assigned a particular meaning or function.
- The bus lines can be classified into three groups:

1) Address lines (bus) 2) Data lines (bus) 3) Control lines (bus)

- In addition, there may be power distribution lines to attached module.

Two way Communication; Read & Write



1. Data Bus

- The **data lines** provide a path for moving data among system modules.
- These lines, collectively, are called the **data bus**.
- The data bus may consist of 32, 64, 128 or even more separate lines.
- Each line can carry only one bit at a time, the number of lines determine how many bits can be transferred at a time.
- The number of these data lines are called the **width** of the data bus.
- The width of the data bus determines the overall *system performance*.
- For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle. (e.g. two fetch operations in each line)

2. Address Bus

Skip

- The **address lines** are used to designate the source or destination of the data on the data bus.
- If the processor wishes to read a word (8, 16 or 32 bits) of data from memory, it puts the address of the desired word on the address lines.
- The width of the **address bus** determines the maximum 'memory capacity' of the system. (i.e. more bits mean more memory locations)
- E.g. 8080 has 16 bit address bus giving 2^16 = 64k address space.
- Furthermore, the address lines are also used to address I/O ports.
- The higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port.

3. Control Bus

Skip

- The **control lines** are used to control the access to and the use of the data and address lines.
- Because data and address lines are shared by all components, there must be a means of controlling their use.
- Control signals transmit both command and timing information among system modules. (Control Unit)
- Timing signals indicate the validity of data and address information.
- Command signals specify operations to be performed.
- Typical control line signals are given in the next slide.

Control Line Signals

Skip

- **Memory write:** Causes data on the bus to be written into the addressed location
- **Memory read:** Causes data from the addressed location to be placed on the bus
- **I/O write:** Causes data on the bus to be output to the addressed I/O port
- **I/O read:** Causes data from the addressed I/O port to be placed on the bus
- **Transfer ACK:** Indicates that data have been accepted from or placed on the bus
- **Bus request:** Indicates that a module needs to gain control of the bus
- **Bus grant:** Indicates that a requesting module has been granted control of the bus
- **Interrupt request:** Indicates that an interrupt is pending
- **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized
- **Clock:** Is used to synchronize operations
- **Reset:** Initializes all modules

Limitations of Bus Design

- A bus is a shared transmission medium.
- Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and will become garbled (Make false by subtract or addition).
- Thus, only one device at a time can successfully transmit.
- We need to set up rules so that only one device can take control of the bus at a time.

Bus Operation Explained

- If one module wishes to send data to another, it must do two things:

1. Obtain the use of the bus, using ‘bus request’ control signal.
2. Transfer data via the bus, using memory write or I/O write signal.

- If one module wishes to request data from another module, it must:

1. Obtain the use of the bus, using ‘bus request’ control signal.
2. Transfer a request to the other module over the appropriate control and address lines, using memory read or I/O read control signal.
3. It must then wait for the second module to send the data.

Single-Bus Problems

- If a great number of devices are connected to a single bus, performance will suffer. There are two main causes:

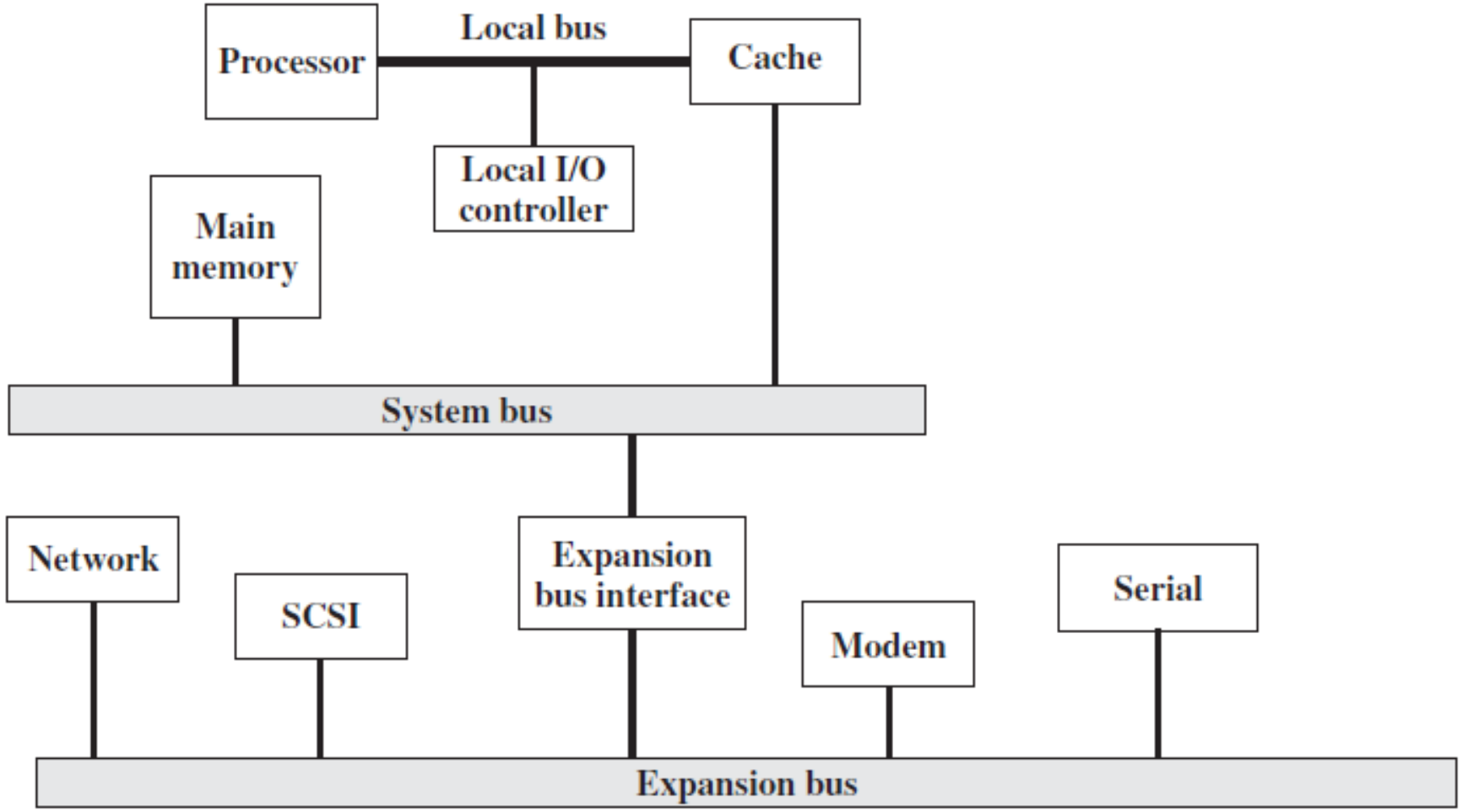
1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay.

- Delay in co-ordination of bus use can adversely affect performance.

2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

- This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses.
- **Solution**: Instead of a single –bus, use multiple buses in a hierarchy.

Traditional Bus Architecture



Break

Elements of Bus Design

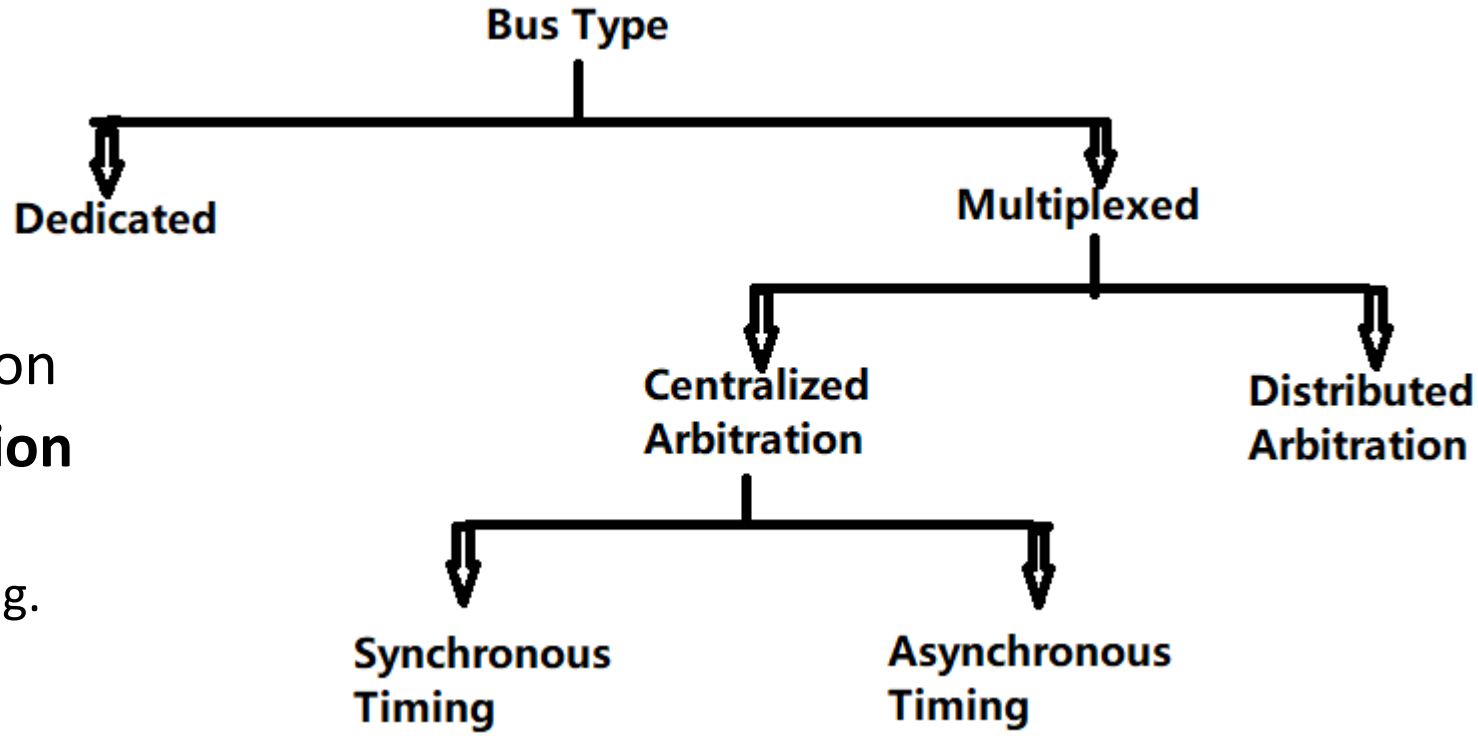
Table 3.2 Elements of Bus Design

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

Hierarchy – Elements of Bus Design

Types of buses

- Dedicated
- **Multiplexed**
  - Distributed Arbitration
  - **Centralized Arbitration**
    - Synchronous Timing
    - Asynchronous Timing.



Bus Types

- Bus lines can be separated into two generic types:

1) Dedicated 2) Multiplexed

1. **Dedicated bus line:** is permanently assigned to one physical subset of computer components. E.g. processor and cache.

- It refers to the use of multiple buses, each of which connects only a subset of modules.
- An example is the use of separate dedicated address and data lines.
- The potential advantage of physical dedication is high throughput (data rate), because there is less bus contention.
- A disadvantage is the increased size and cost of the system.

Multiplexed Bus

2. **Multiplexed bus line:** Using the same set of lines for multiple purposes and multiple modules is known as *time multiplexing*.

- For example, address and data information may be transmitted over the same set of lines using an Address Valid control line.
- First place address, each module copies address and determines if it's the addressed module. The address is removed from the bus and the same bus is used for the subsequent read or write data transfer.
- The advantage is the use of fewer lines, which saves space and cost.
- Disadvantage, more complex circuitry is needed within each module.
- There is a reduction in performance because certain events that share the same line cannot take place in parallel.

Method of Arbitration & Its Types

- In multiplexed, more than one module may need control of the bus.
- The purpose of arbitration is to designate one device as master/slave.
- Because only one unit can successfully transmit over the bus, some method of **arbitration** is needed (to gain control of the bus).
- The two arbitration methods can be classified as:

1) Centralized arbitration (e.g. CPU) 2) Distributed arbitration

1. **Centralized arbitration:** a single hardware device, called an *arbiter* or *bus controller*, is responsible for allocating time on the bus. CPU.
2. **Distributed arbitration:** there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus. (e.g. they sense if line is available or not)

Bus Timing & Its Types

- In centralized arbitration, each device is allocated time on the bus.
- **Timing** refers to the way in which events are coordinated on the bus.
- Buses use two types of timing:

1) Synchronous timing 2) Asynchronous timing

1. **Synchronous timing:** The occurrence of events on the bus is determined by a clock. It uses 'Time multiplexing' technique.

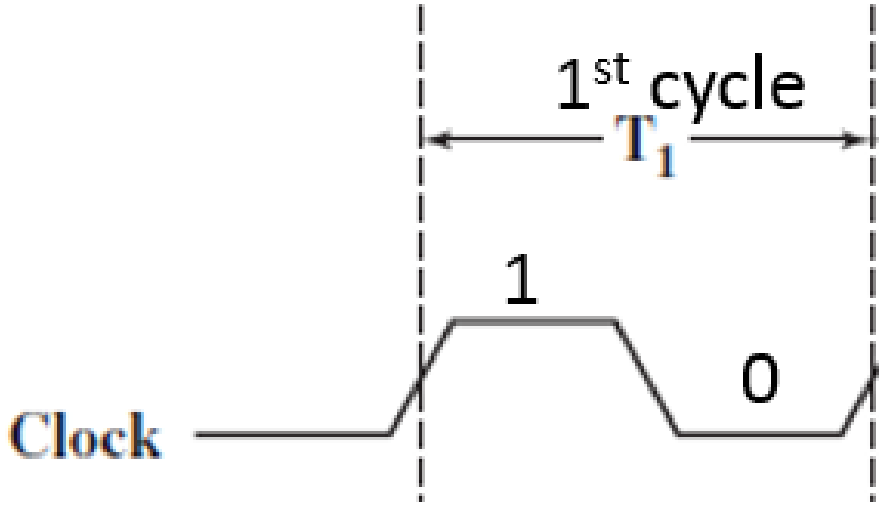
- It is simpler to implement and test, however it is less flexible (fixed Cl)

2. **Asynchronous timing:** the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

- Advantage, a mixture of slow and fast devices, can share a bus.

Timing of Synchronous Bus Operation

- With Synchronous bus timing
  - Events determined by clock signals
  - Control Bus includes clock line
  - A single 1-0 is a bus/clock cycle, ‘time slot’.
  - All devices can read clock line
  - Usually sync on leading edge
  - Usually a single cycle for an event



- In a simple example, the processor places a memory address on the address line during the first clock cycle, it may assert status symbols.
- Once the address lines have stabilized, the processor issues an address enable signal.

See Synchronous Timing Figure (Next Slide)

- For read operation, the processor issues a read command at the start of the second cycle.
- A memory module recognizes the address and, after a delay of one cycle, places the data on the data lines.
- The processor reads the data from the data lines and drops the read signal.
- For a write operation, the processor puts the data on the data lines at the start of the second cycle and issues a write command.
- The memory module copies the information from the data lines during the third clock cycle.

Synchronous Bus Timing

Read Operation Sequence:

1. Place memory address on address line.
2. Issues 'address enable' signal.
3. Issues 'read' single in 2nd cycle.
4. Memory module recognizes address.
5. Places data on 'data lines', 3rd cycle.

'Address' is placed on Address bus.

Write Operation Sequence:

1. Processor puts 'data', 2nd cycle.
2. Then issues a 'write' command.
3. Memory module copies data from 'data lines' in 3rd cycle.

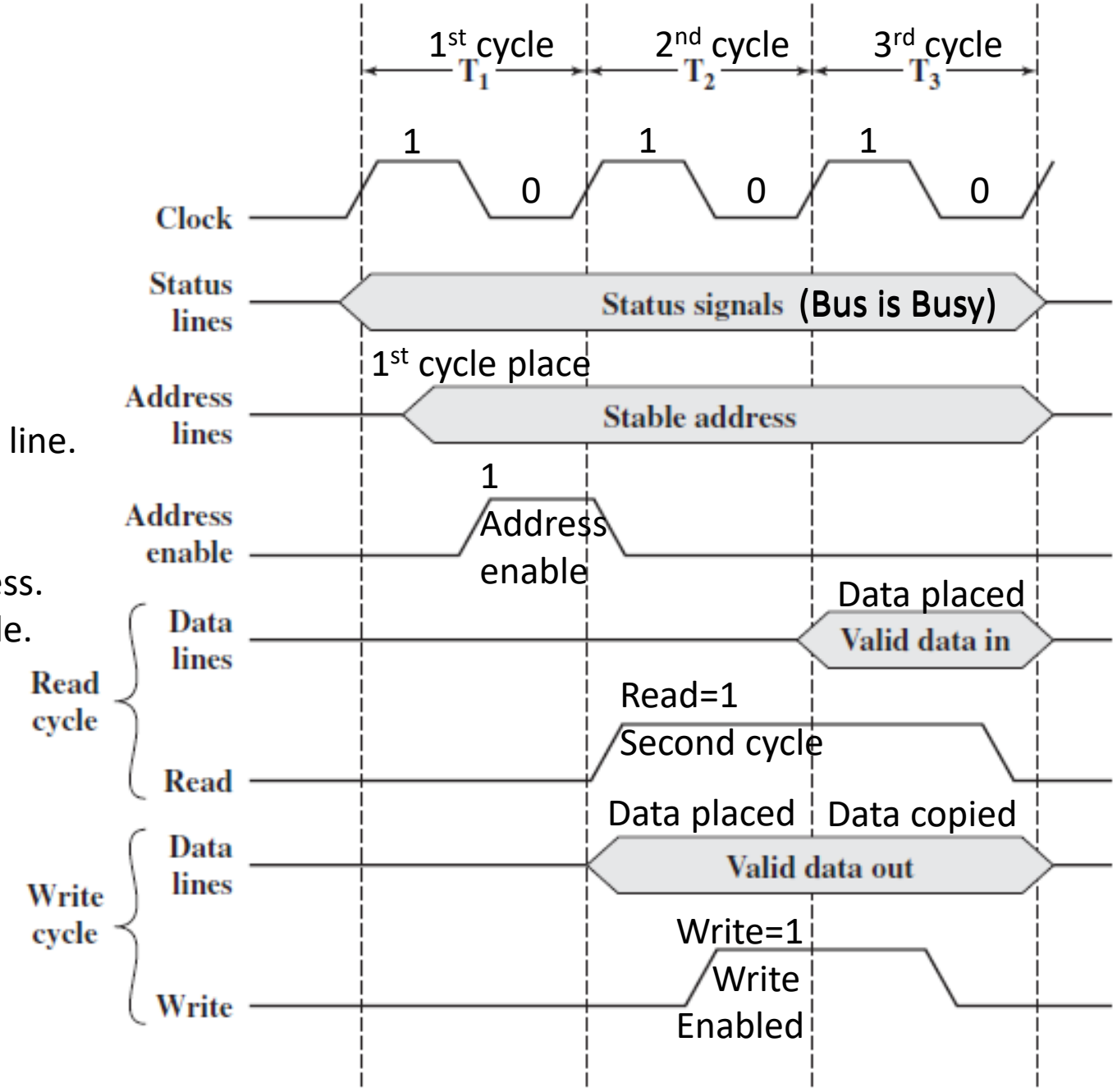
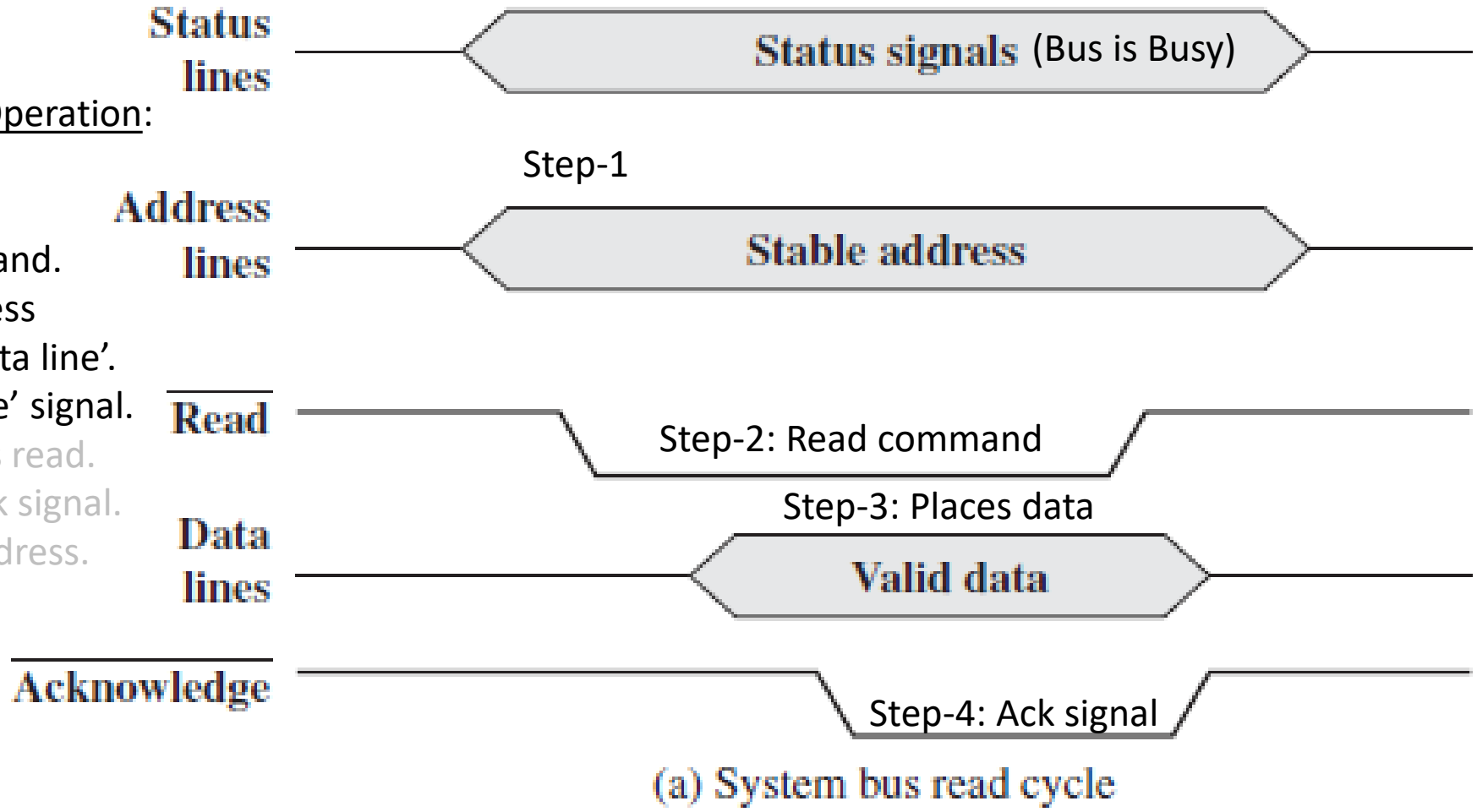


Figure 3.19 Timing of Synchronous Bus Operations

Asynchronous Read Operation

Steps of Asynchronous Read Operation:

1. The processor places 'address' and 'status' signals.
2. Then it issues a read command.
3. The memory decodes address and places the data on the 'data line'.
4. Memory gives 'acknowledge' signal.
5. Master reads data, removes read.
6. Memory drops data and Ack signal.
7. The master removes the address.

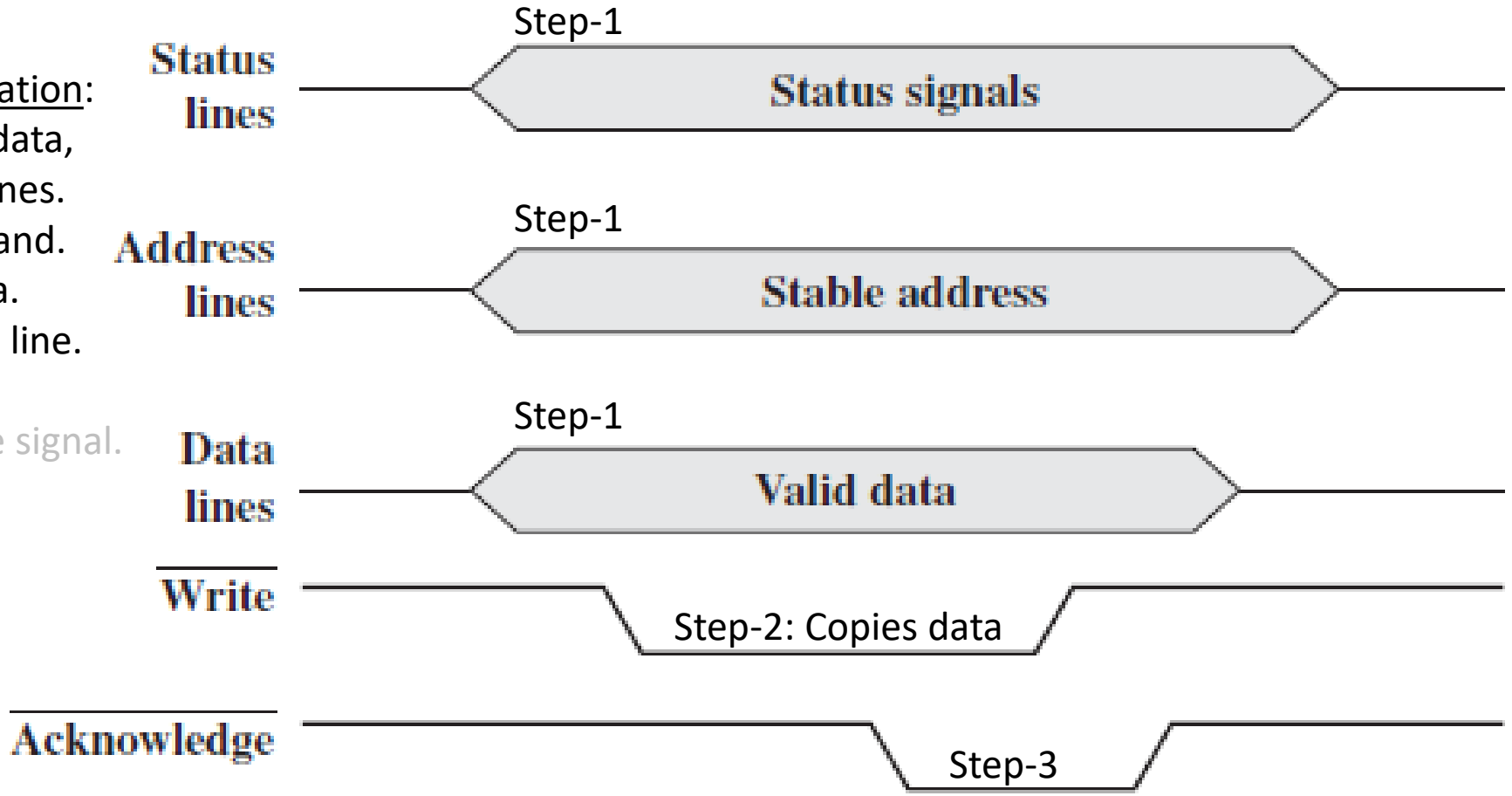


(a) System bus read cycle

Asynchronous Write Operation

Steps of Asynchronous Write Operation:

1. Master simultaneously places data, status and address on respective lines.
2. Then it issues the 'Write' command.
3. Memory module copies the data.
4. Then it asserts the acknowledge line.
5. Master drops the write signal.
6. Memory drops the acknowledge signal.



(b) System bus write cycle

Read for Knowledge

- Hardware interrupts are asynchronous and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an interrupt request (IRQ).
- Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.

Preparatory Questions

1. **What are the two ‘types of interrupts’? Explain.**
2. **What are the different ‘classes of interrupts’?**
3. **Write down the steps of ‘interrupt handler process’.**
4. **List and briefly define two approaches to dealing with ‘multiple interrupts’.**
5. **What is the ‘limitation’ of a bus design?**
6. **Explain the ‘bus operation’ for a read & write operation, using ‘bus request’ signal.**
7. **What is a ‘system bus’? Explain its three constituent buses.**
8. **List the different ‘elements of a bus design’. (Bus-type, method of arbitration, timing).**

Quiz 02 – From Chapter 03 – Next Class

- Homework: Read point-to-point interconnection structure. (Topic 3.5)

- This topics is not included in Exam.

Quiz-2 (Ch-3) Next Class
Assignment-2 (Submission) Next Week