Tishk International University
Engineering Faculty
Mechatronics Engineering Department Computer
Microprocessor and Programmer

Top Level View of Computer Function and Interconnection

Dr. Rand Basil Alhashimie

rand.basil@tiu.edu.iq

Lecture Objectives

After studying this lecture, you should be able to:

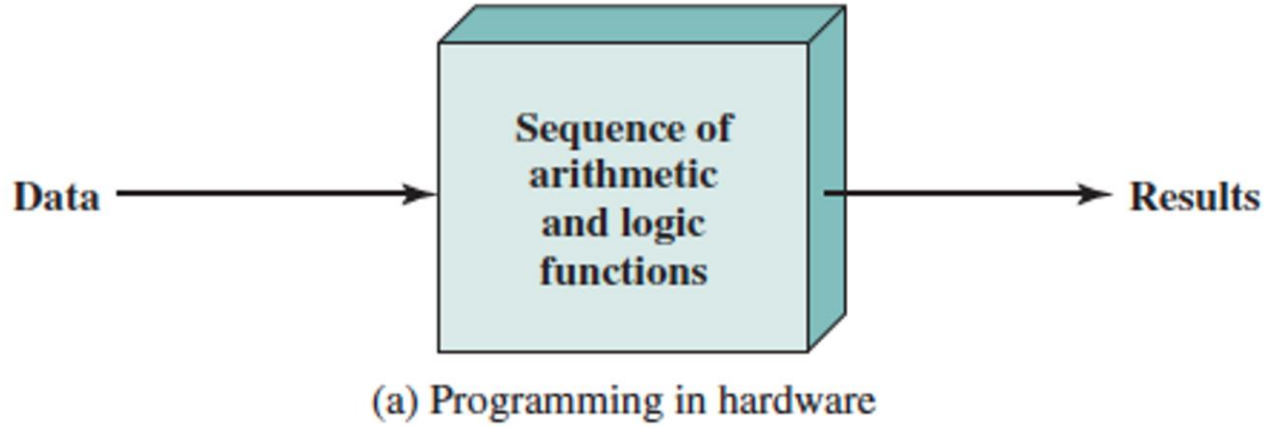
- Understand the basic elements of an instruction cycle and the role of interrupts.
- Describe the concept of interconnection within a computer system.
- Understand the difference between synchronous and asynchronous bus timing.

Lecture Outline

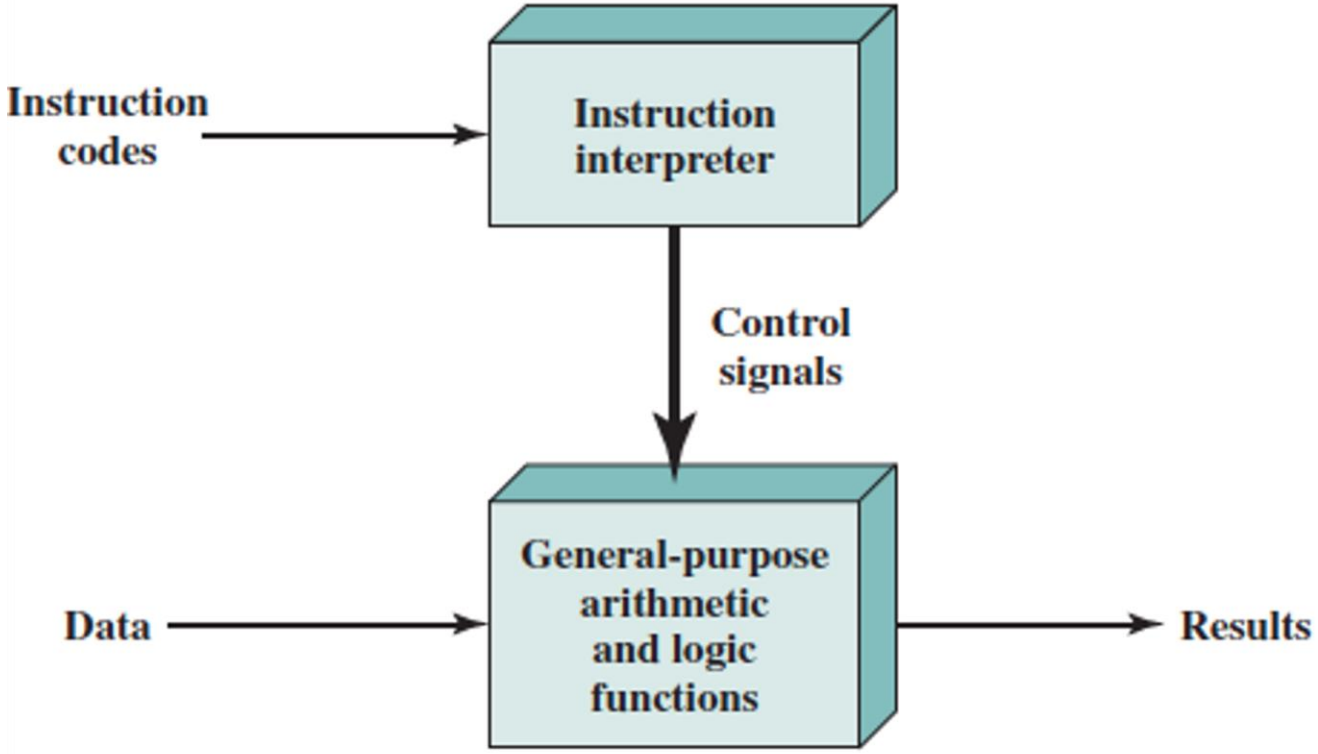
- Program concept
- Computer Top level
  - CPU
  - Memory
  - I/O interconnections

Program Concept

Hardware and Software Approaches



(a) Programming in hardware



(b) Programming in software

Program Concept

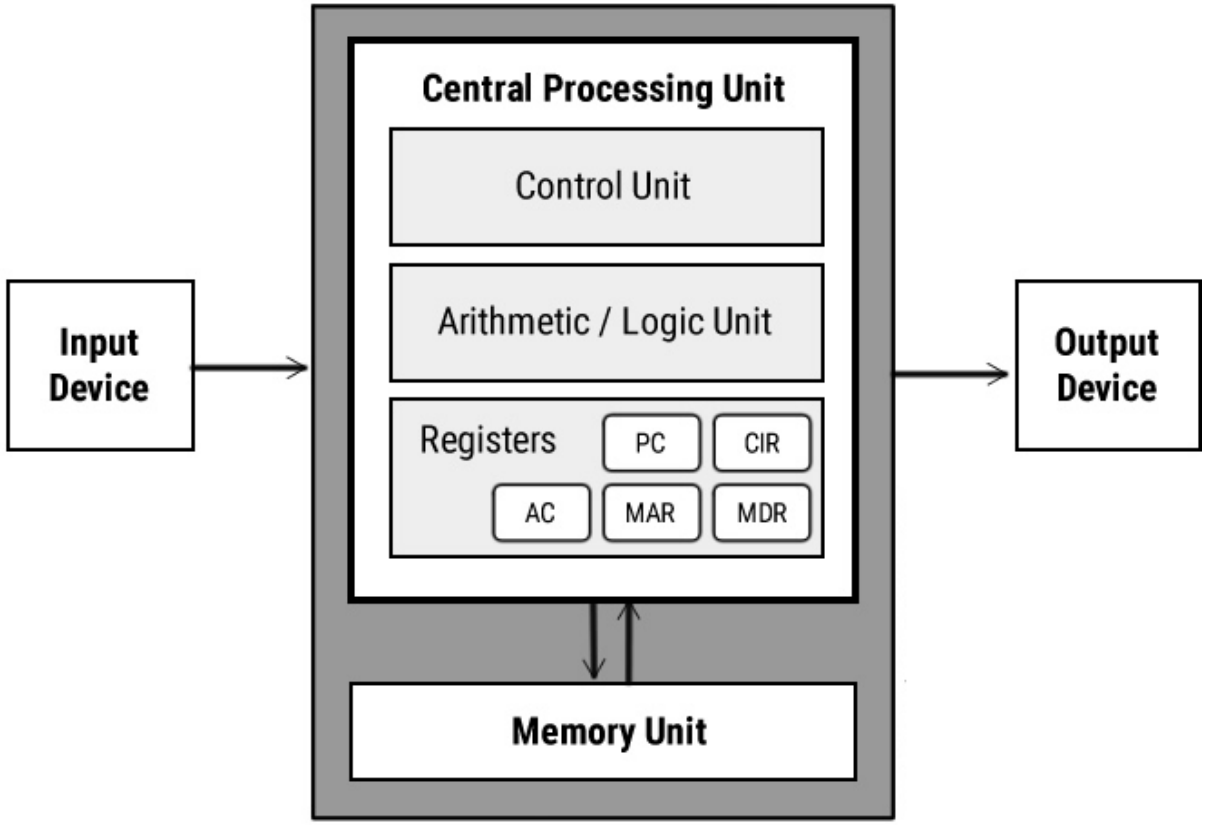
- Hardwired systems are inflexible
- General purpose hardware can do different tasks, given correct control signals.
- Instead of re-wiring, supply a new set of control signals

What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation:
  - a different set of control signals is needed
  - a unique code is provided (e.g. ADD, MOVE)

Von Neumann Architecture

- Von Neumann architecture was first published by John von Neumann in 1945.
- His computer architecture design consists of a **Control Unit, Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/Outputs**.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today.



Von Neumann Architecture

- Data and instructions are stored in a single read–write memory.
- The contents of this memory are addressable by location, without regard to the type of data contained there.
- Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.
- A particular set of hardware will perform various functions on data depending on control signals applied to the hardware.

Von Neumann Architecture

How shall control signals be supplied?

- The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data.
- Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes.
- Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals.
- To distinguish this new method of programming, a sequence of codes or instructions is called *software*.

Computer Function: Top Level View

- Computer consists of:
  - CPU
  - Memory
  - I/O interconnections
- These components are connected to achieve the main function of the computer, which is to execute program.

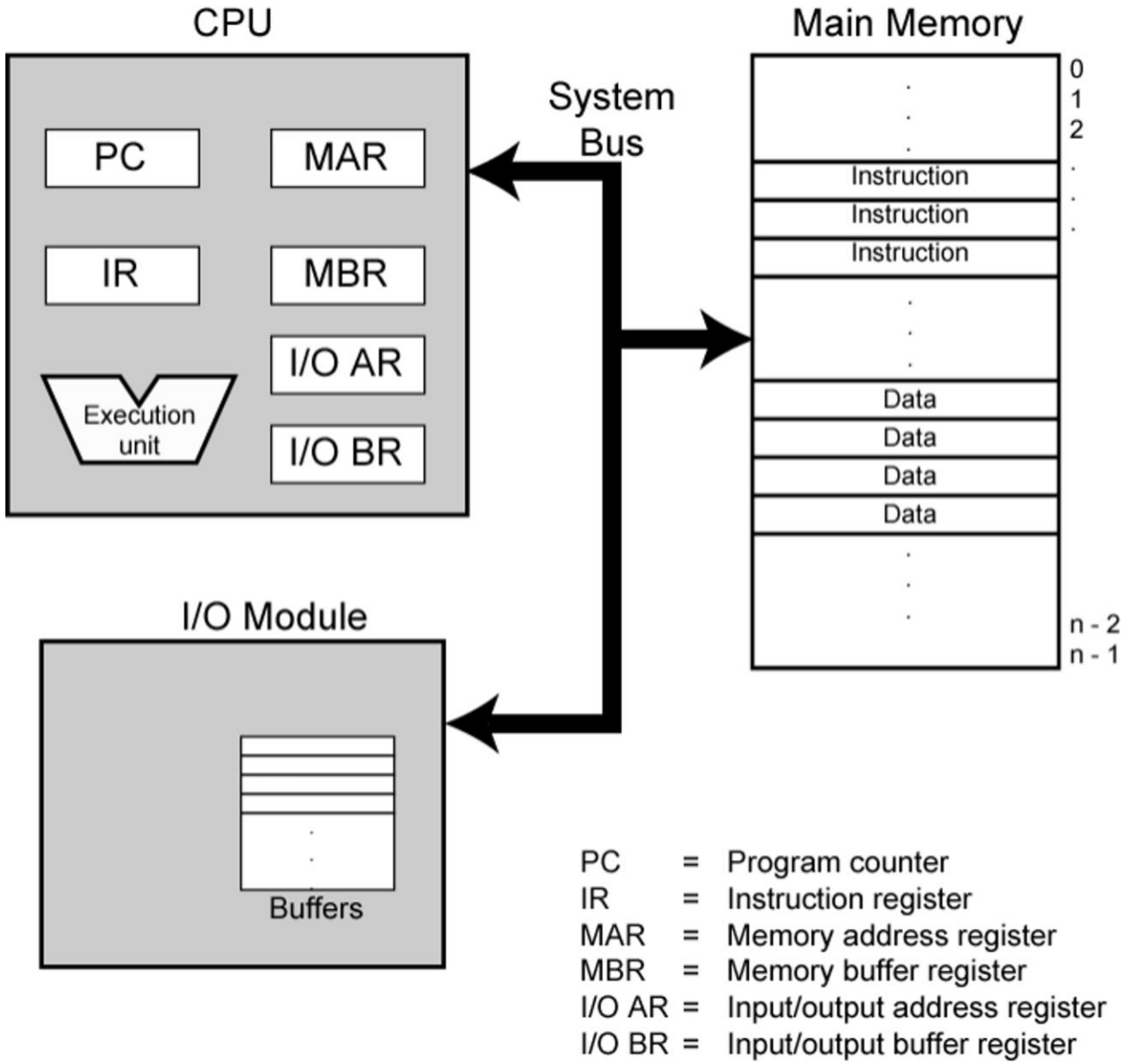


Figure 2

Components

- The Control Unit and the Arithmetic and Logic Unit constitute the Central Processing Unit.
- Data and instructions need to get into the system and results out
  - Input/output
- Temporary storage of code and results is needed
  - Main memory

I/O Components

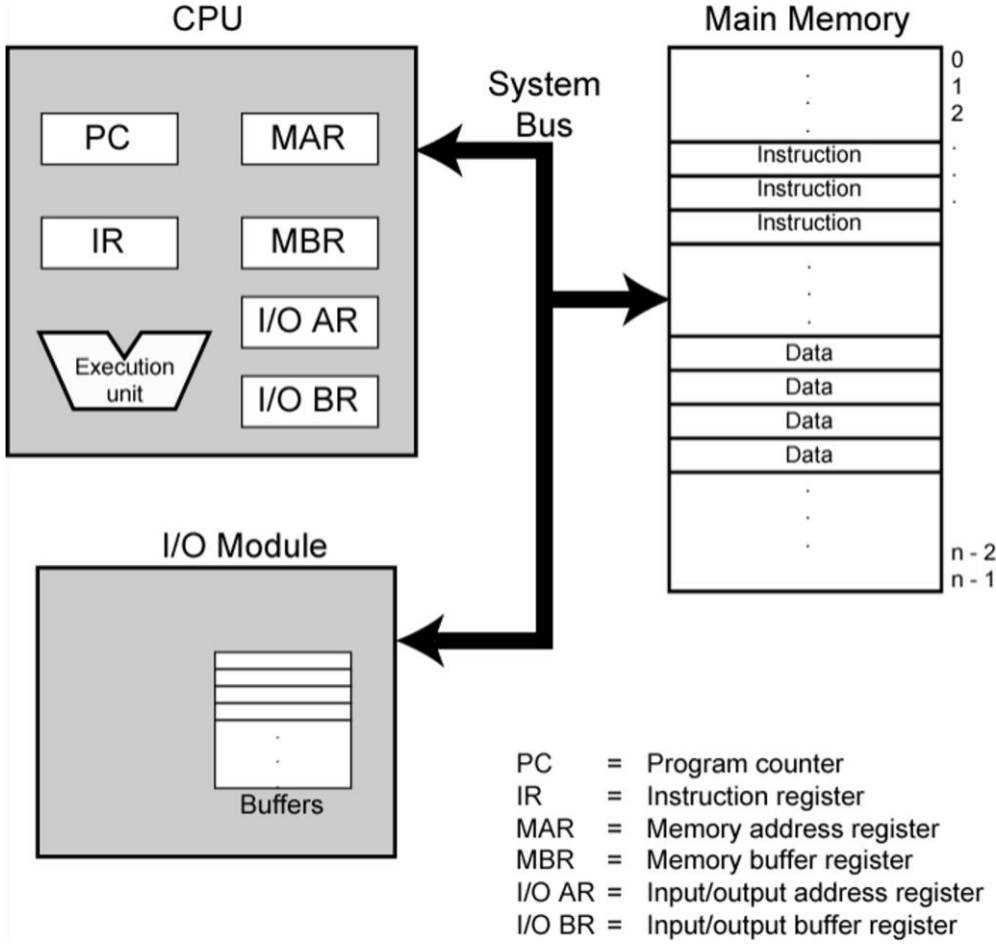
- Data and instructions must be put into the system. For this we need some sort of input module.
- This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system.
- A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as *I/O components*.

Main Memory (Temporary Storage)

- An input device will bring instructions and data in sequentially. But a program is not invariably executed sequentially; it may jump around (e.g., the IAS jump instruction).
- Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to store temporarily both instructions and data.
- That module is called *memory*, or *main memory*, to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.

Memory Locations and I/O

- A memory module consists of a set of locations, defined by sequentially numbered addresses.
- Each location contains a binary number that can be interpreted as either an instruction or data.
- An I/O module transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.



Computer Function

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program.
- In its simplest form, instruction processing consists of two steps:
  - The processor reads (*fetches*) instructions from memory one at a time.
  - Executes each instruction.

Program Execution

- Program execution consists of repeating the process of instruction fetch and instruction execution.
- The instruction execution may involve several operations and depends on the nature of the instruction. The processing required for a single instruction is called an **instruction cycle**.
- The instruction cycle involves two steps which are referred to as the **fetch cycle** and the **execute cycle**.

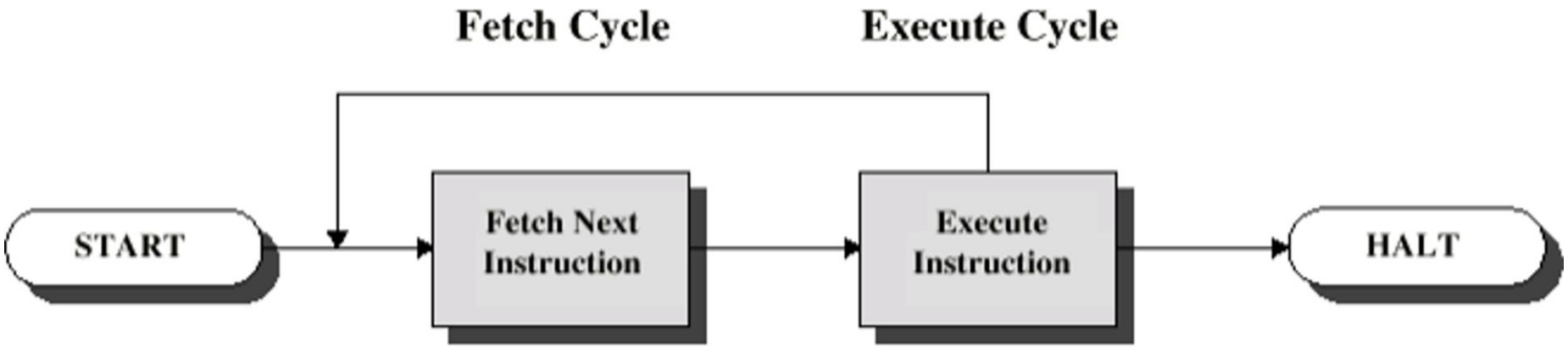
Program Execution Stop

Program execution halts only:

- if the machine is turned off,
- some sort of unrecoverable error occurs,
- or a program instruction that halts the computer is encountered.

Instruction Cycle: Fetch and Execute

- Two steps:
  - Fetch
  - Execute



Fetch Cycle

- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

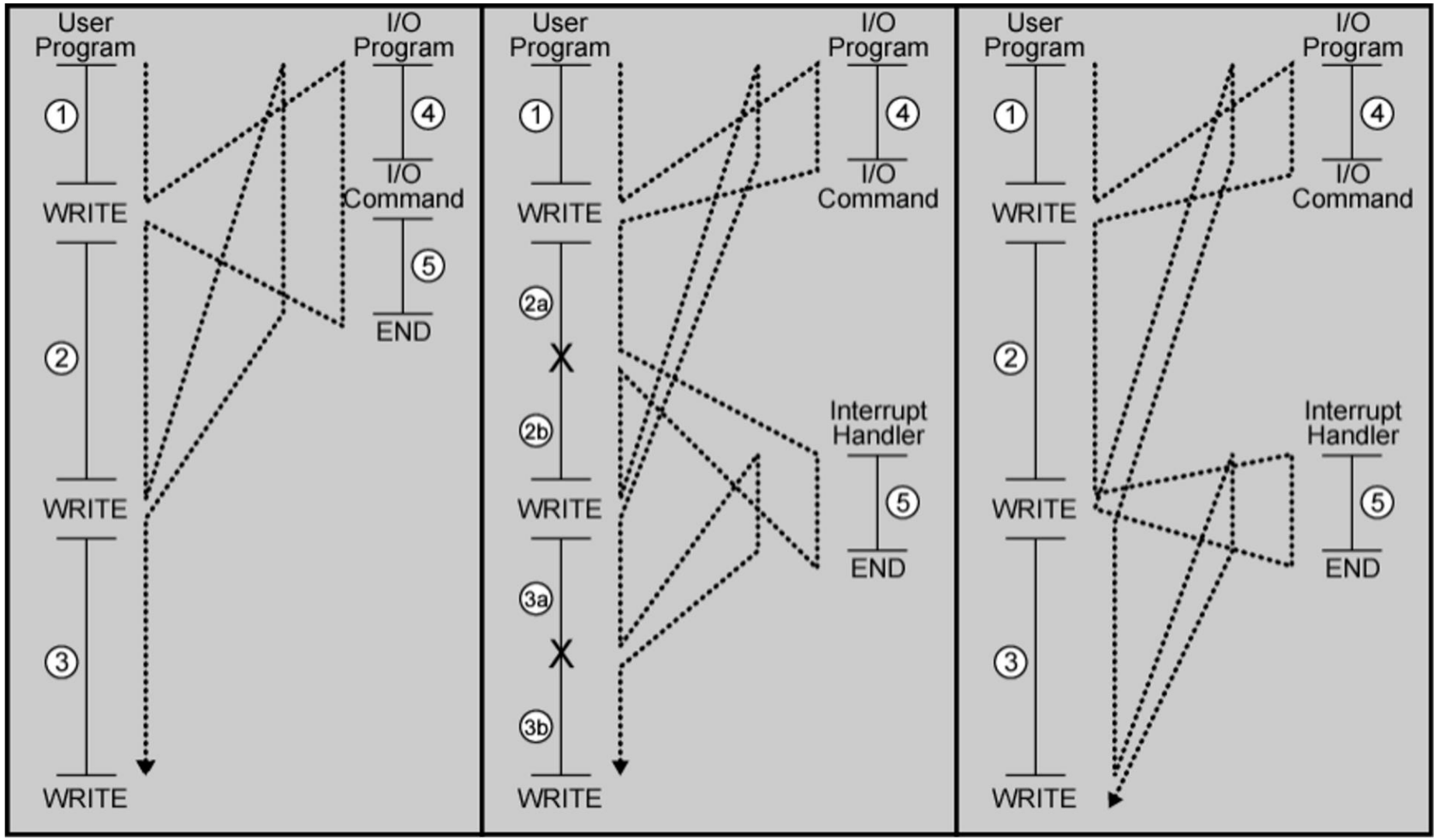
Execute Cycle

- Processor-memory
  - data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above

Interrupts

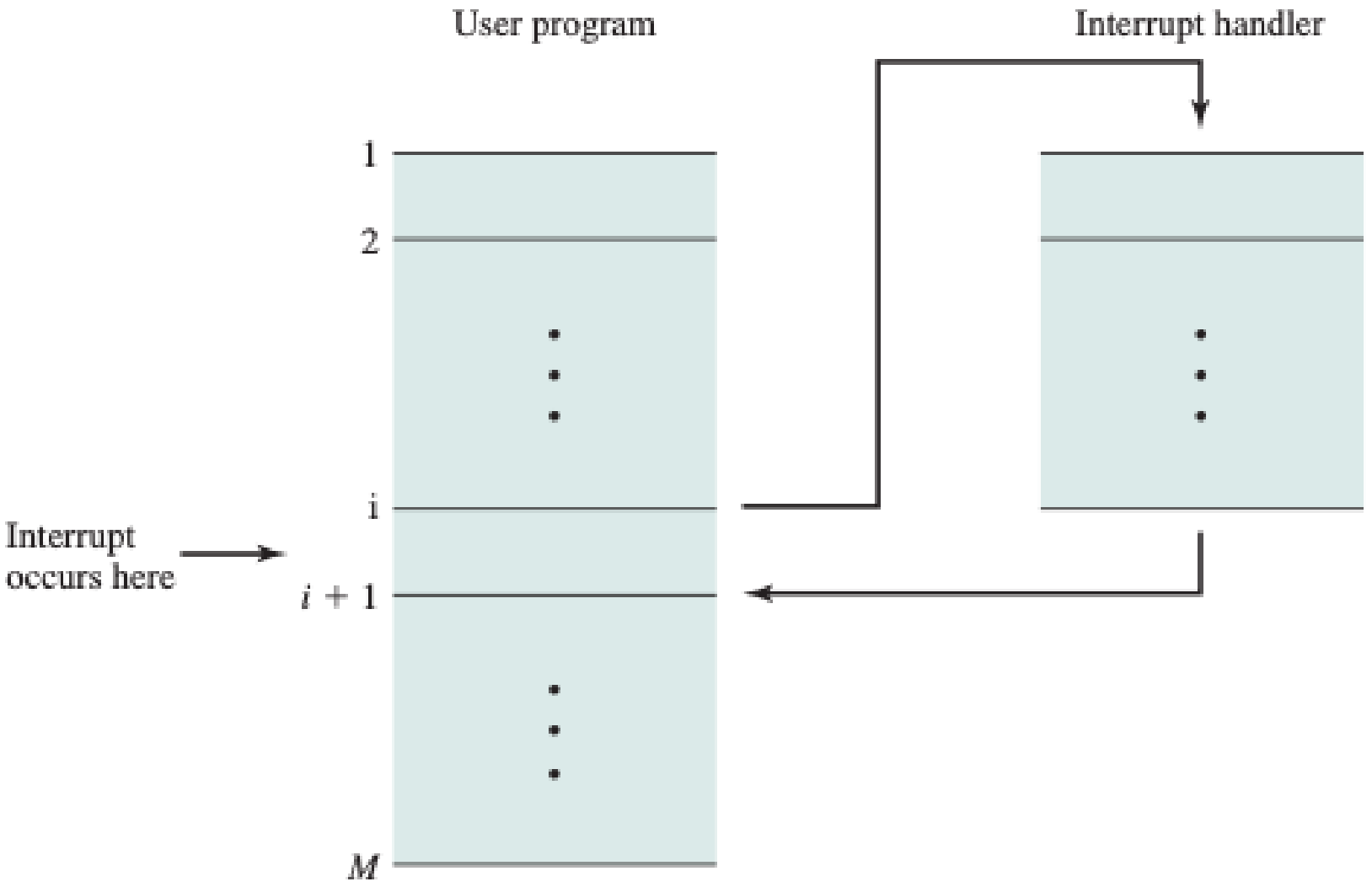
- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Program
  - e.g. overflow, division by zero
- Timer
  - Generated by internal processor timer
  - Used in pre-emptive multitasking
- I/O
  - from I/O controller
- Hardware Failure
  - e.g. power failure

Program Flow Control



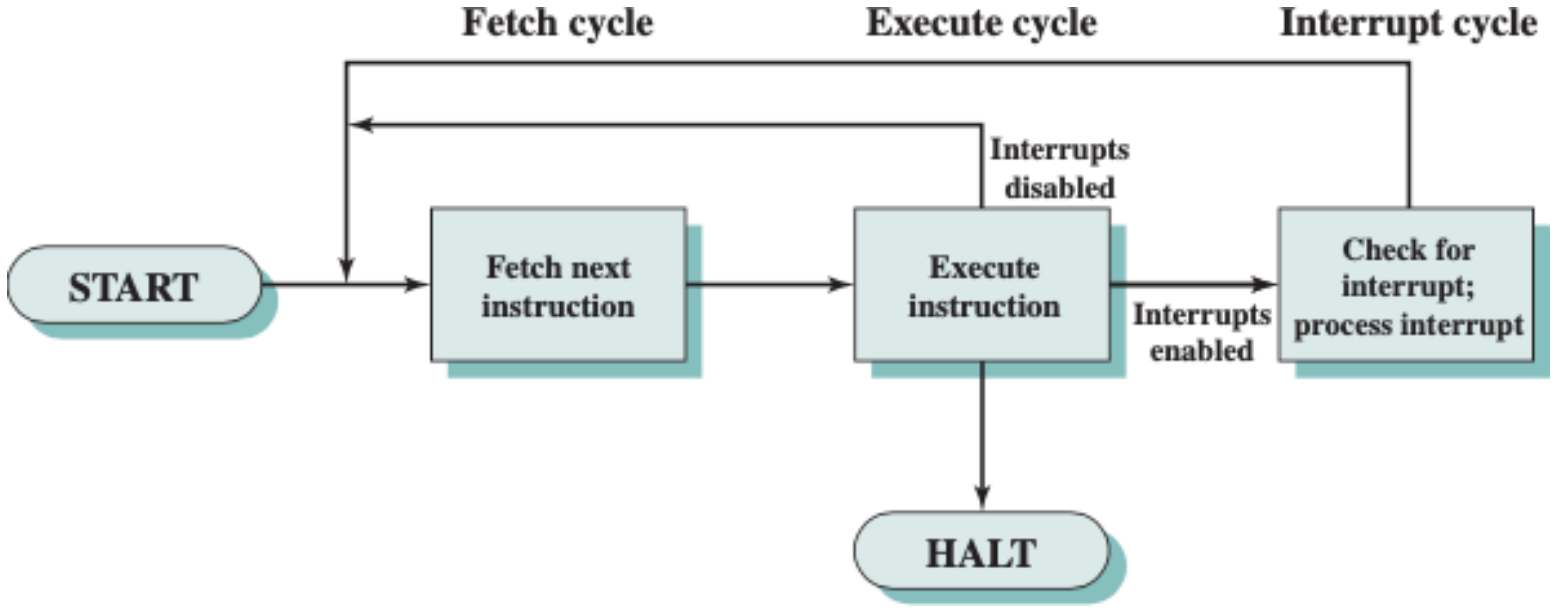
(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

Transfer of Control via Interrupts

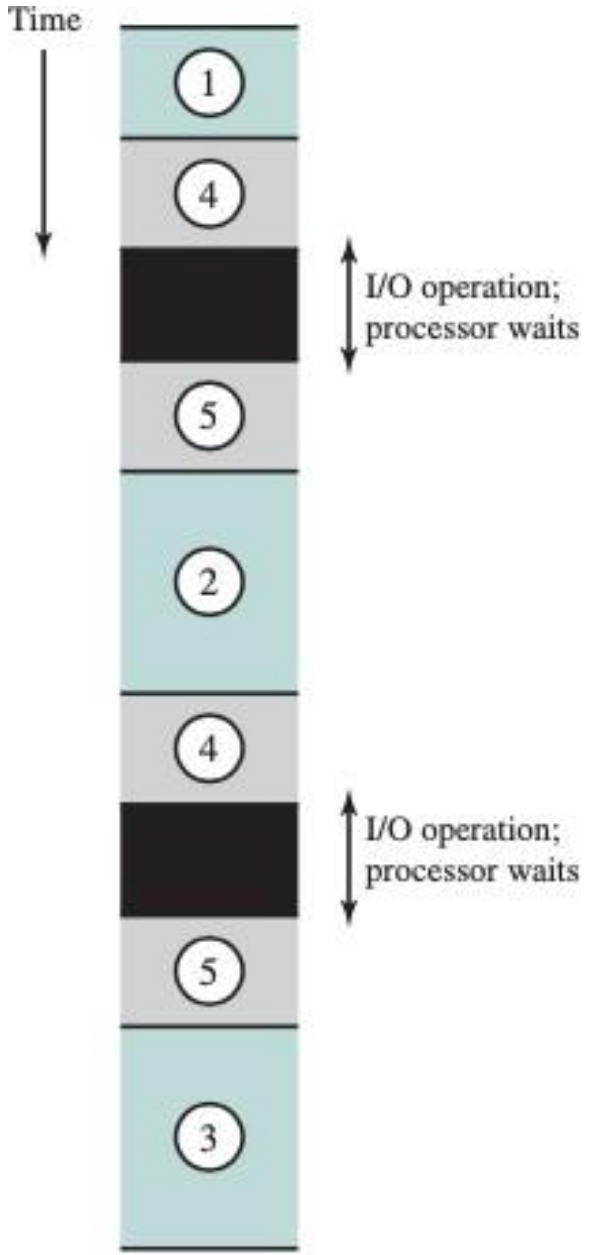


Interrupt Cycle

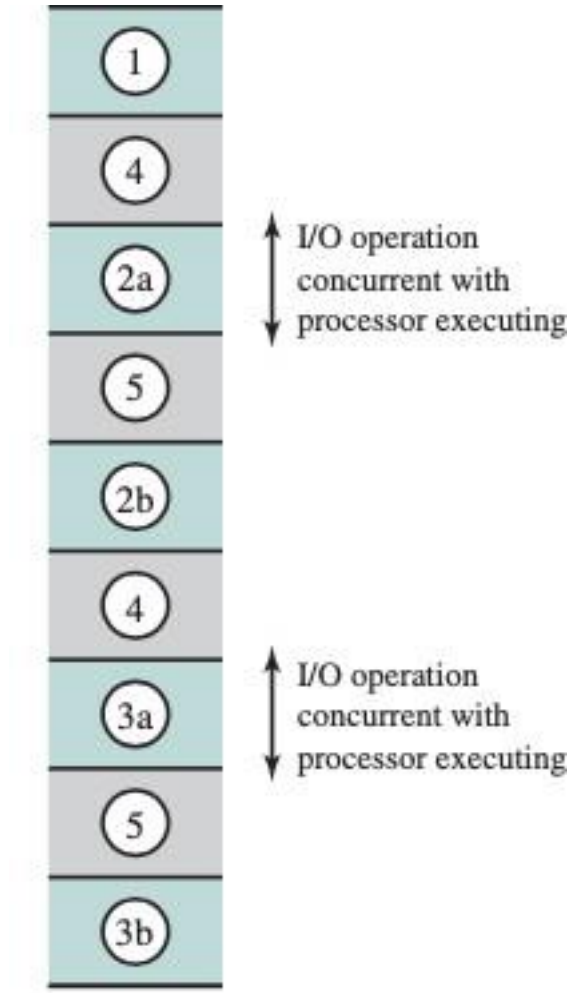
- Added to instruction cycle
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program



Program Timing Short I/O Wait

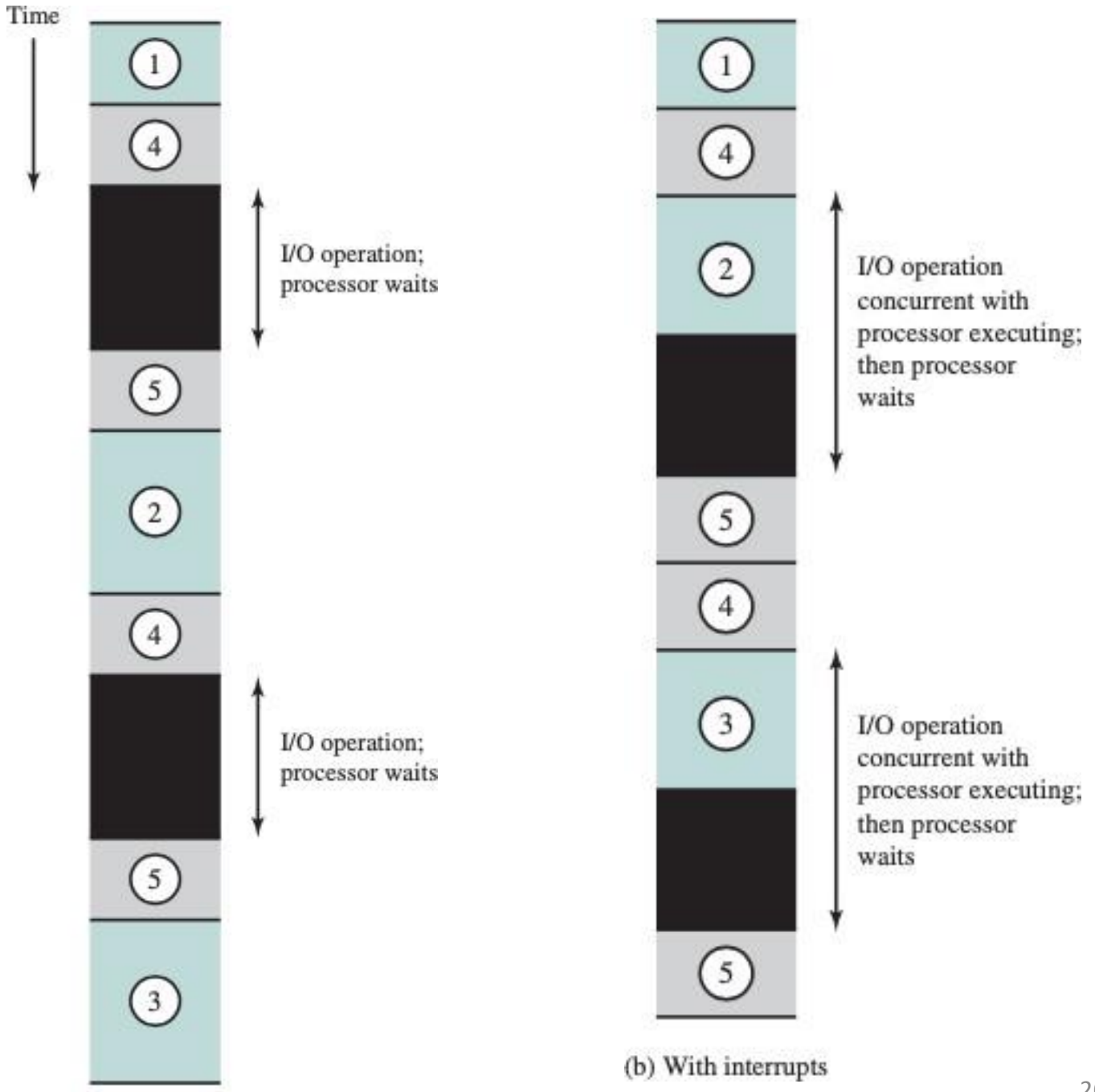


(a) Without interrupts



(b) With interrupts

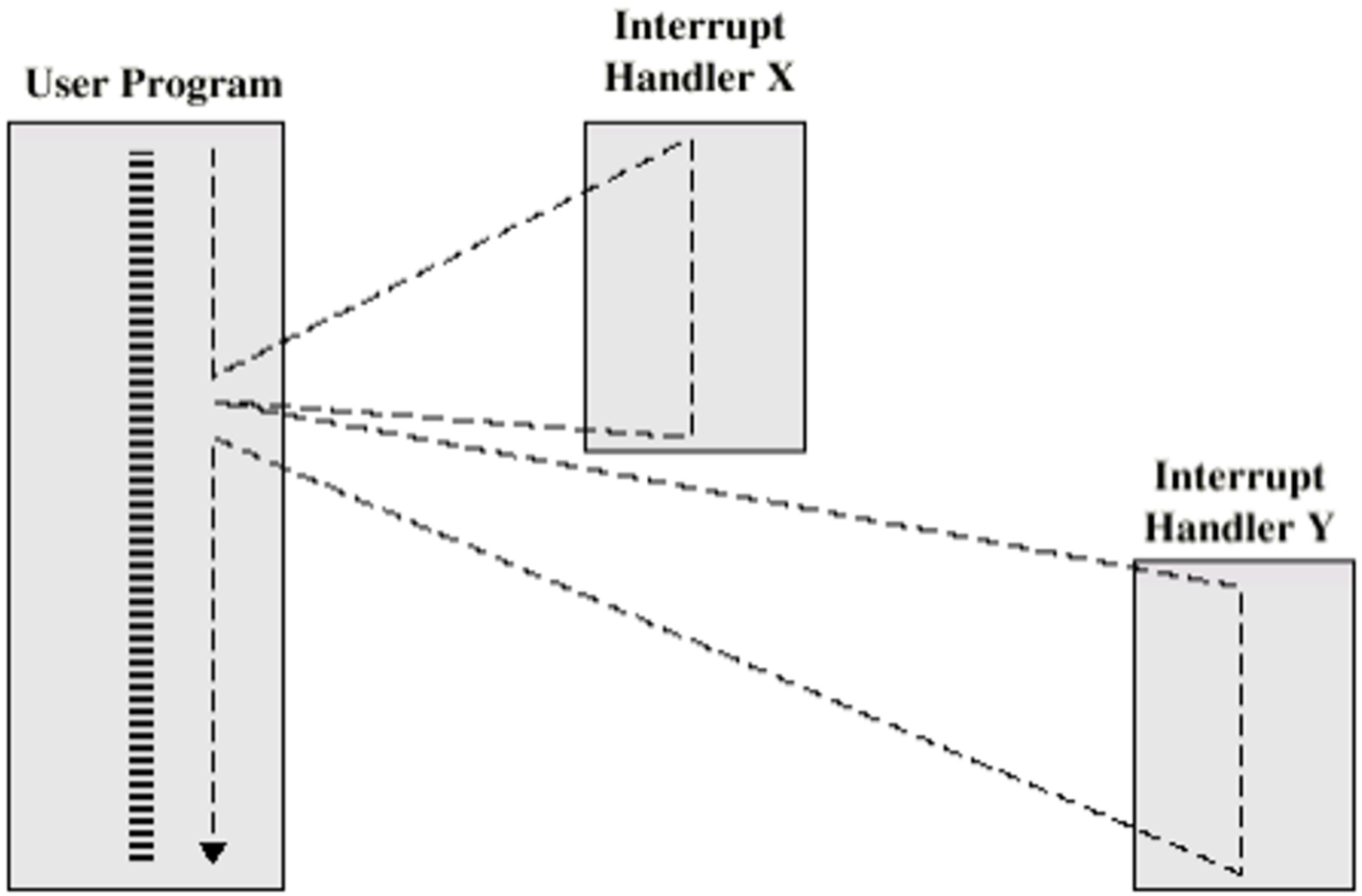
Program Timing
Long I/O Wait



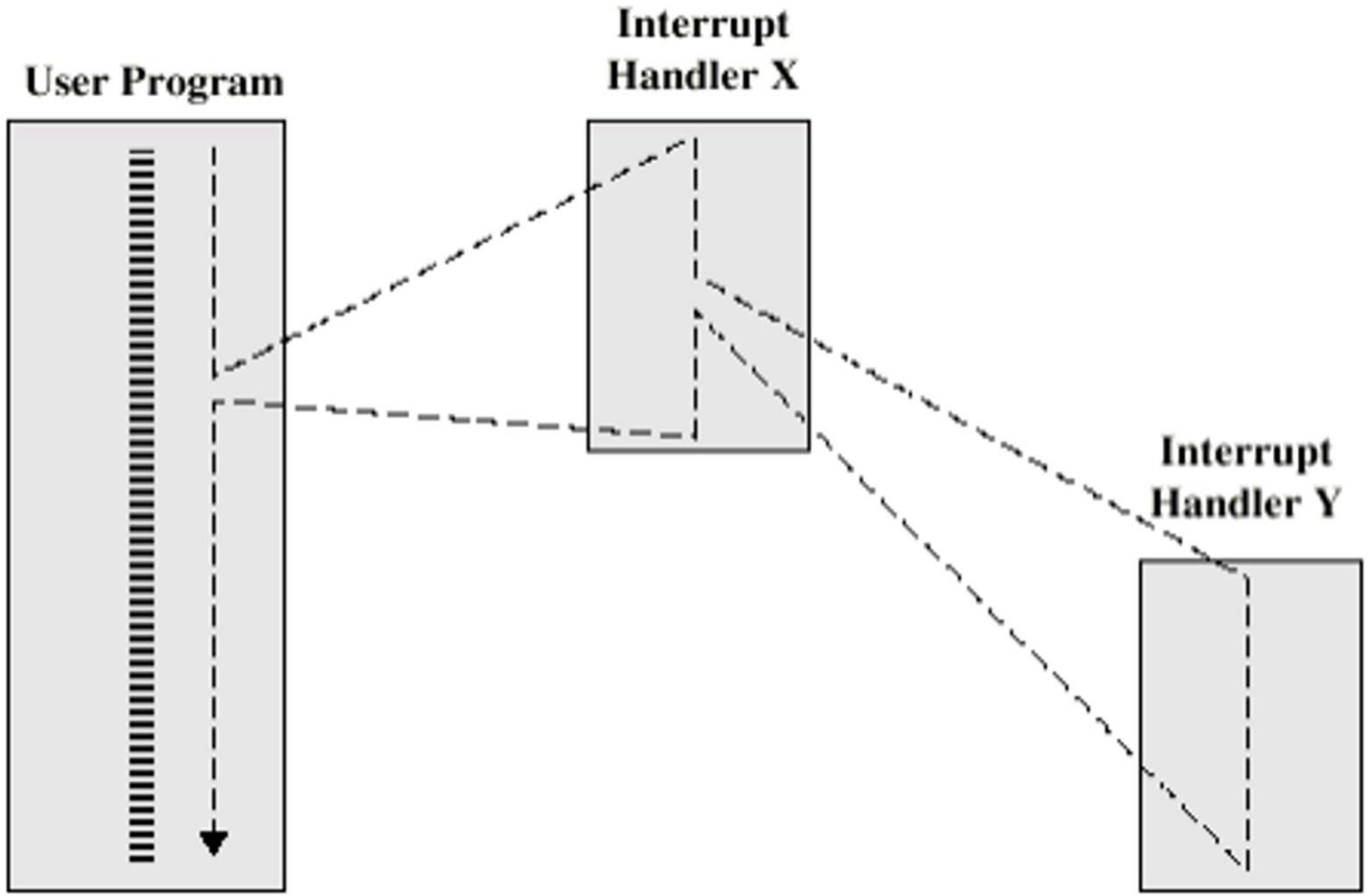
Multiple Interrupts

- Disable interrupts
  - Processor will ignore further interrupts whilst processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur
- Define priorities
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

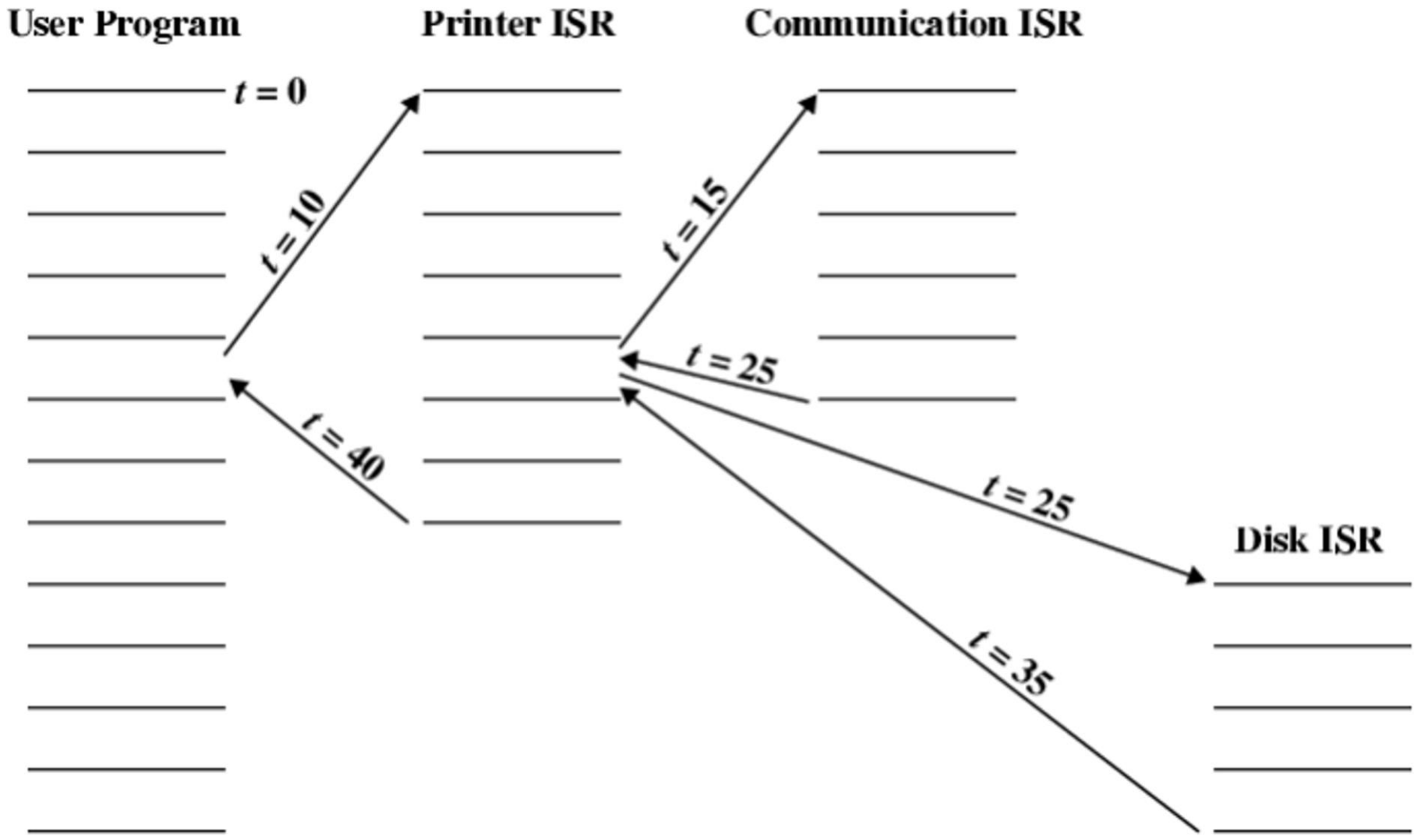
Multiple Interrupts - Sequential



Multiple Interrupts – Nested



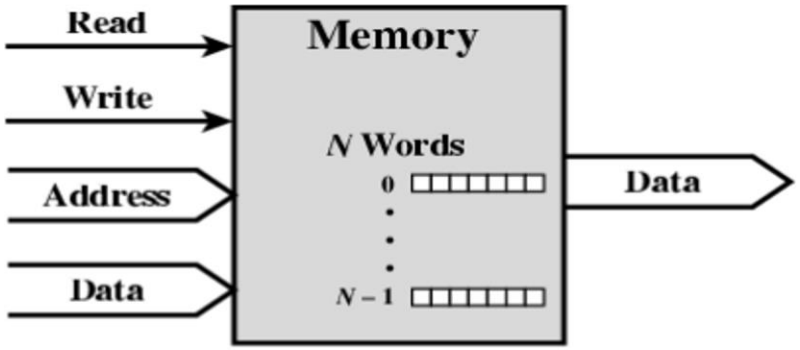
Time Sequence of Multiple Interrupts

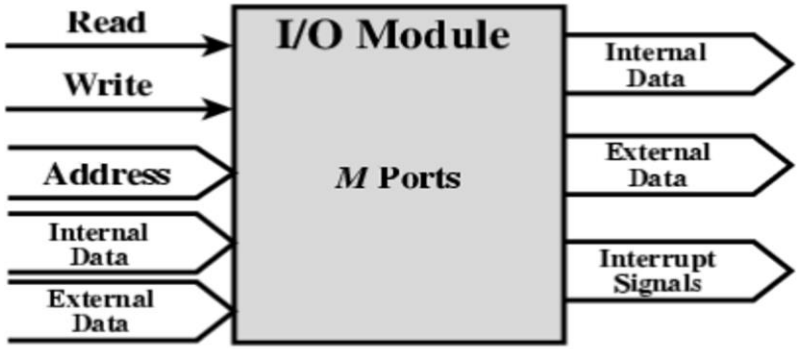


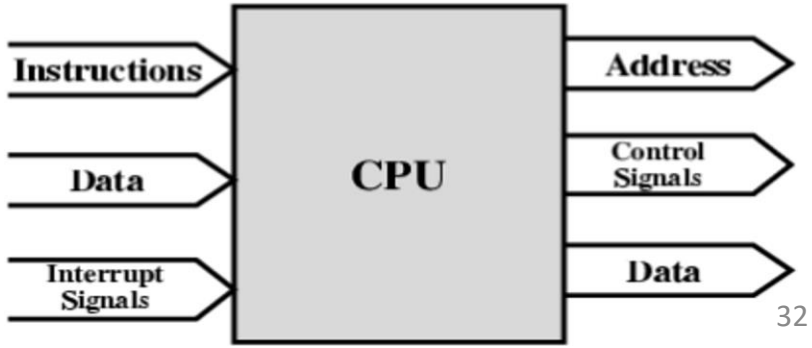
Connection

- All the units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

Computer Modules

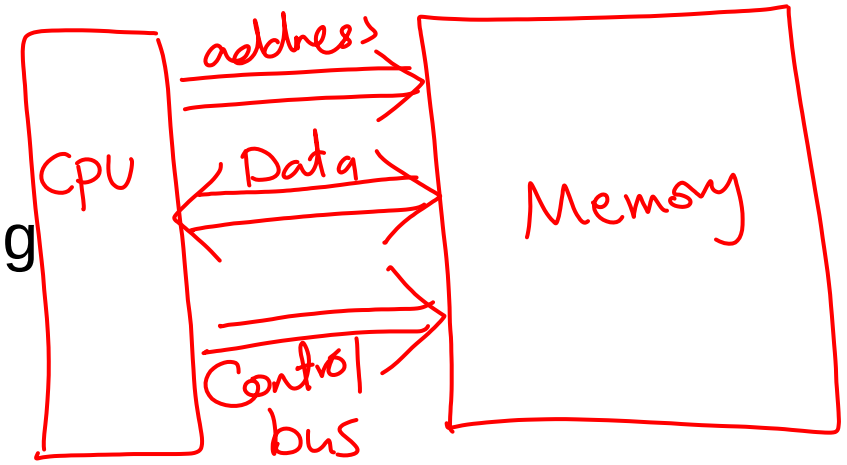


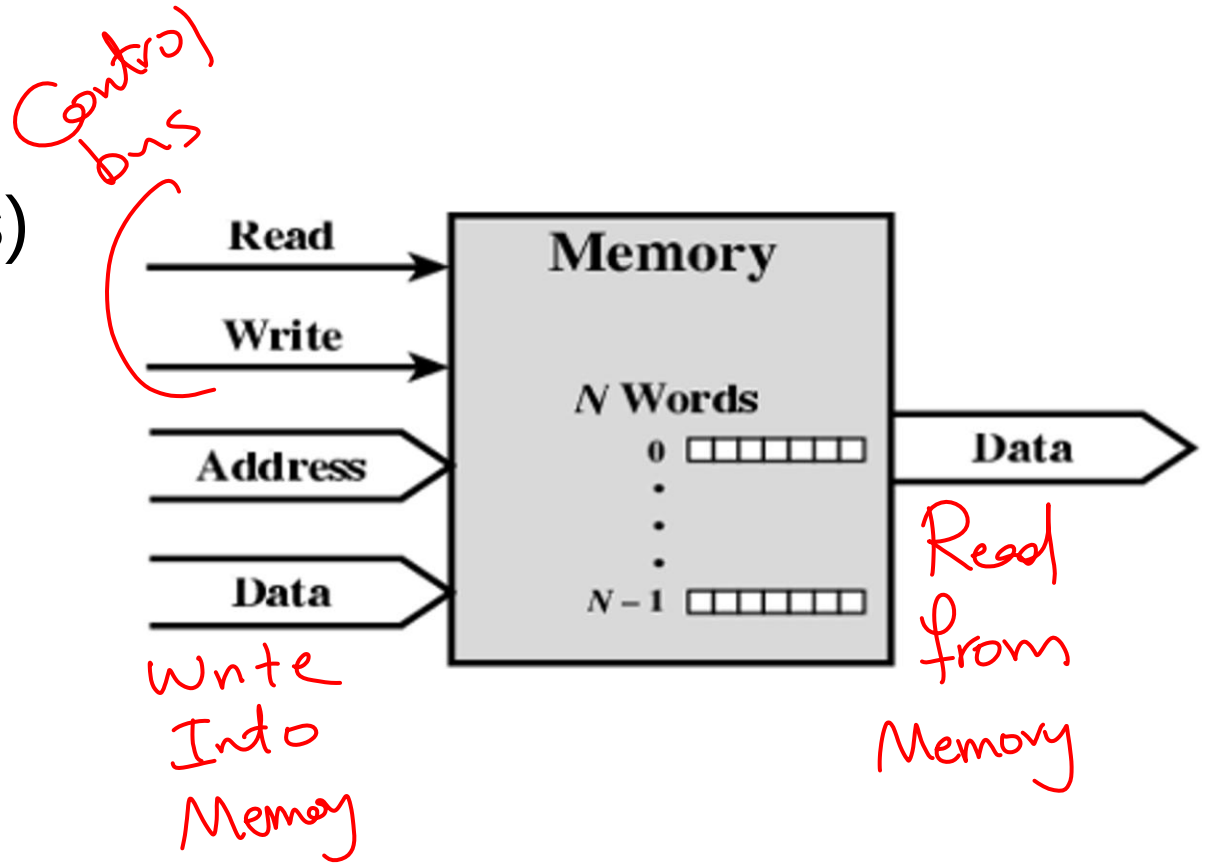




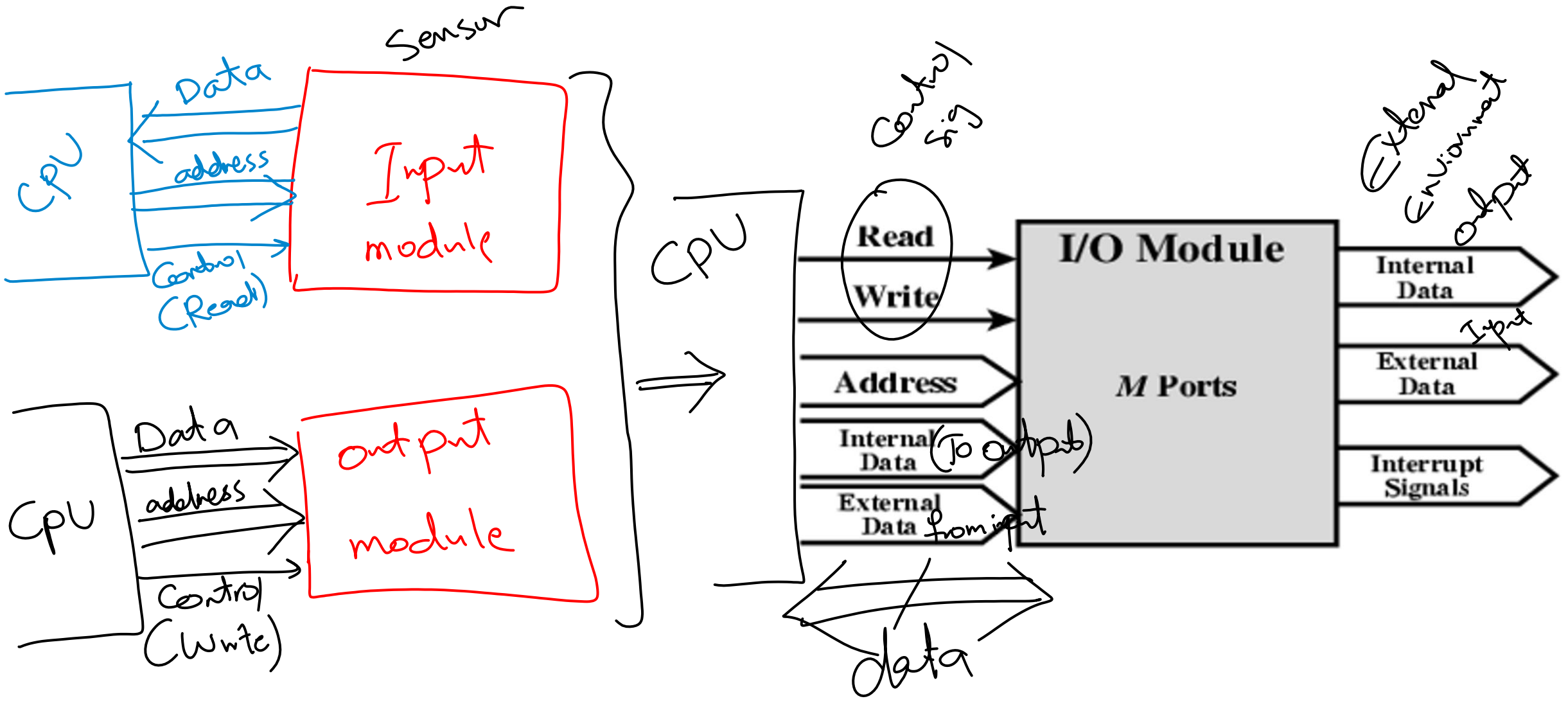
Memory Connection

- Receives and sends data
- Receives addresses (of locations)
- Receives control signals
  - Read
  - Write
  - Timing





Input/Output Connection(1)

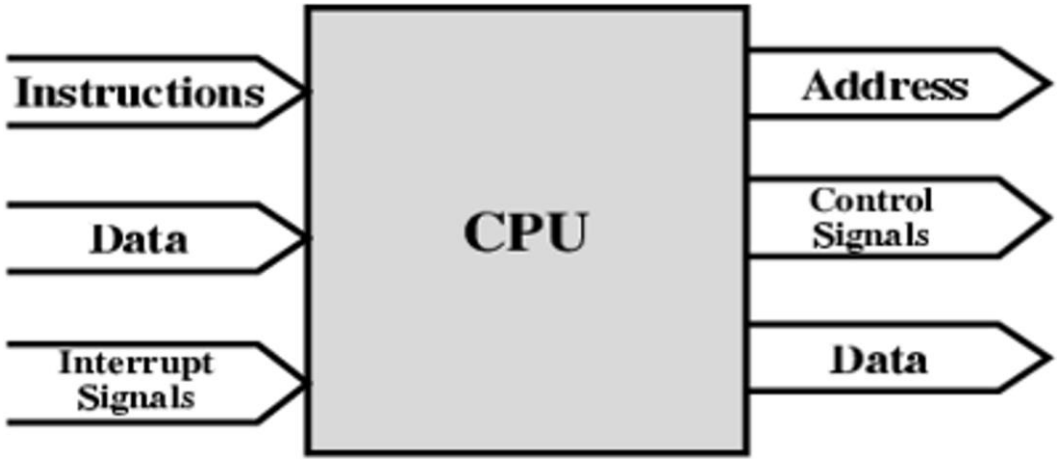


Input/Output Connection(2)

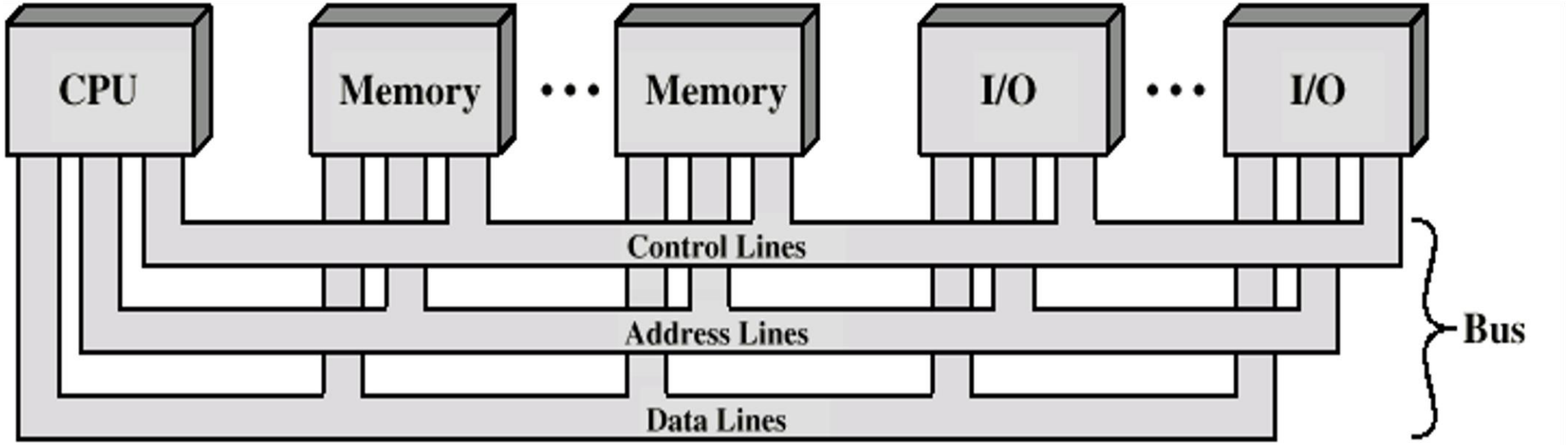
- Receive control signals from computer
- Send control signals to peripherals
  - e.g. spin disk
- Receive addresses from computer
  - e.g. port number to identify peripheral
- Send interrupt signals (control)

CPU Connection

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units
- Receives (& acts on) interrupts



Bus Interconnection Scheme



Buses

- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)

What is a Bus?

- A communication pathway connecting two or more devices
- Usually broadcast
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

Data Bus

- Carries data
  - Remember that there is no difference between “data” and “instruction” at this level
- Width is a key determinant of performance
  - 8, 16, 32, 64 bit

Address bus

- Identify the source or destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  - e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

- Control and timing information
  - Memory read/write signal
  - Interrupt request
  - Clock signals

Control Bus

- **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

Control Bus

- **Interrupt ACK:** acknowledges that the pending interrupt has been recognized
- **Clock:** is used to synchronize operations
- **Reset:** initializes all modules.

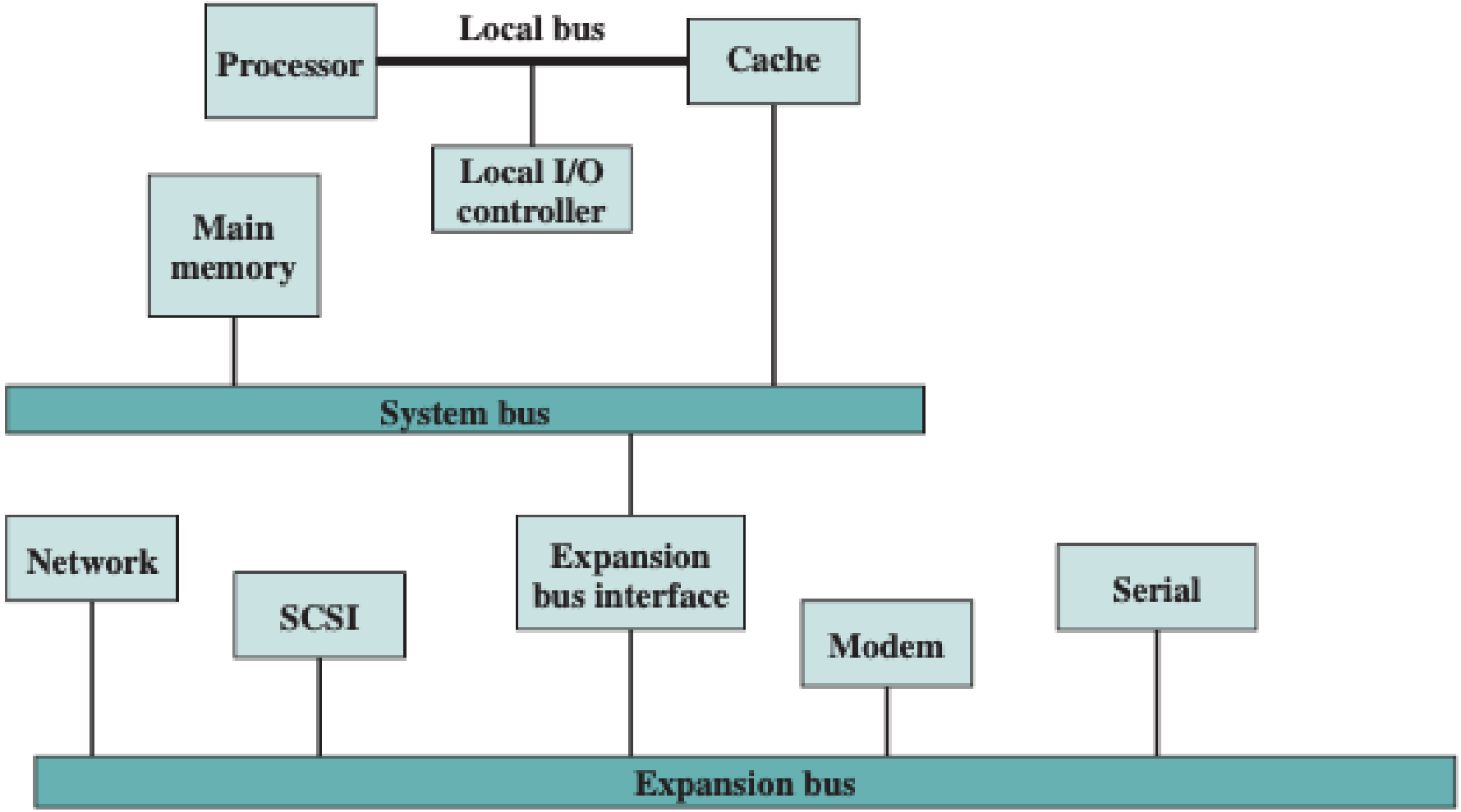
The operation of the bus is as follows:

- If one module wishes to send data to another, it must do two things: (1) obtain the use of the bus, and (2) transfer data via the bus.
- If one module wishes to request data from another module, it must (1) obtain the use of the bus, and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

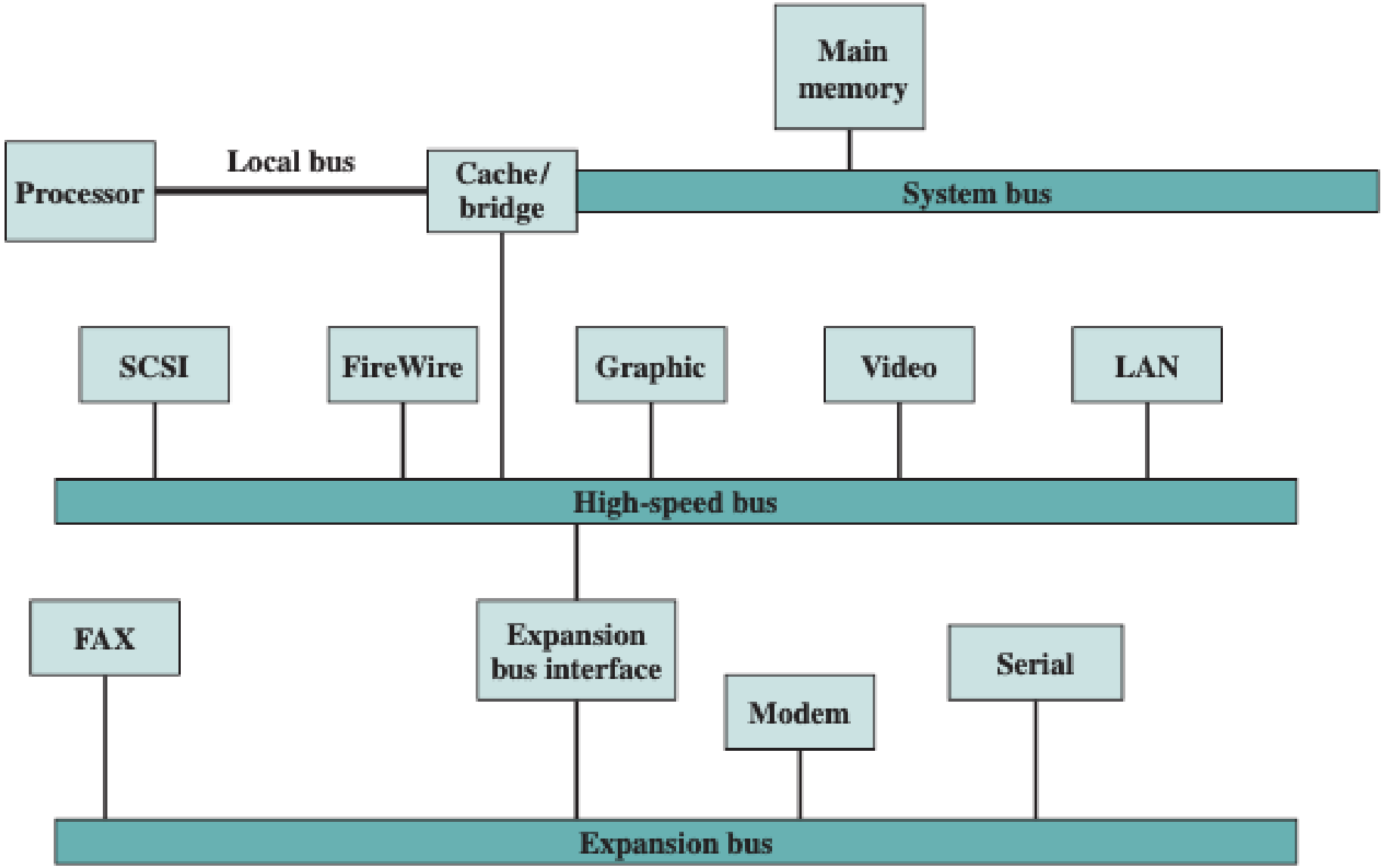
Single Bus and Multiple Bus

- Lots of devices on one bus leads to:
  - Propagation delays
    - Long data paths mean that coordination of bus use can adversely affect performance
    - If aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems

Traditional (ISA) - (with cache)



High Performance Bus



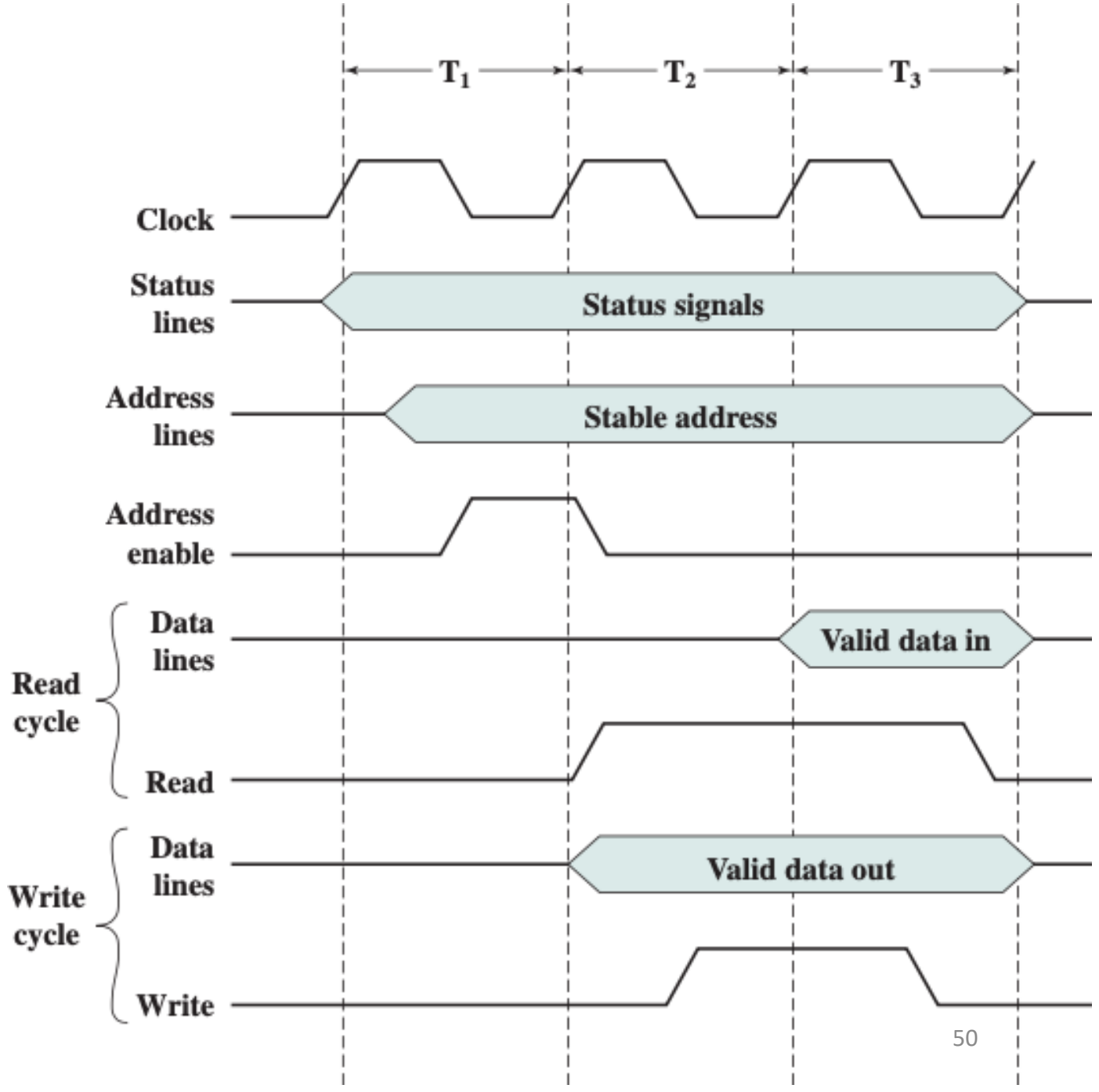
Bus Types

- Dedicated
  - Separate data & address lines
- Multiplexed
  - Shared lines
  - Address valid or data valid control line
  - Advantage - fewer lines
  - Disadvantages
    - More complex control
    - Ultimate performance

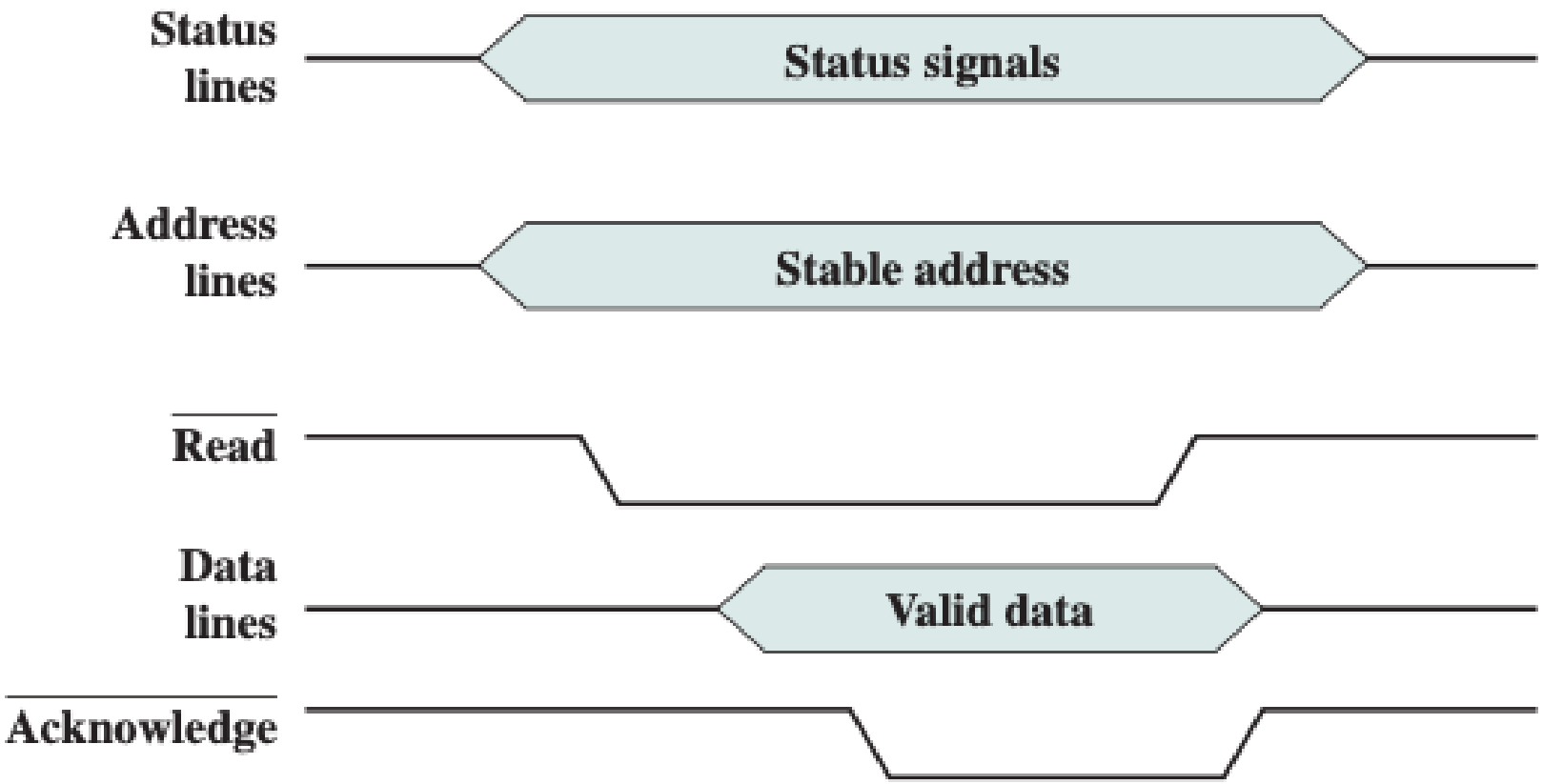
Timing

- Coordination of events on bus
- Synchronous
  - Events determined by clock signals
  - Control Bus includes clock line
  - A single 1-0 is a bus cycle
  - All devices can read clock line
  - Usually sync on leading edge
  - Usually a single cycle for an event

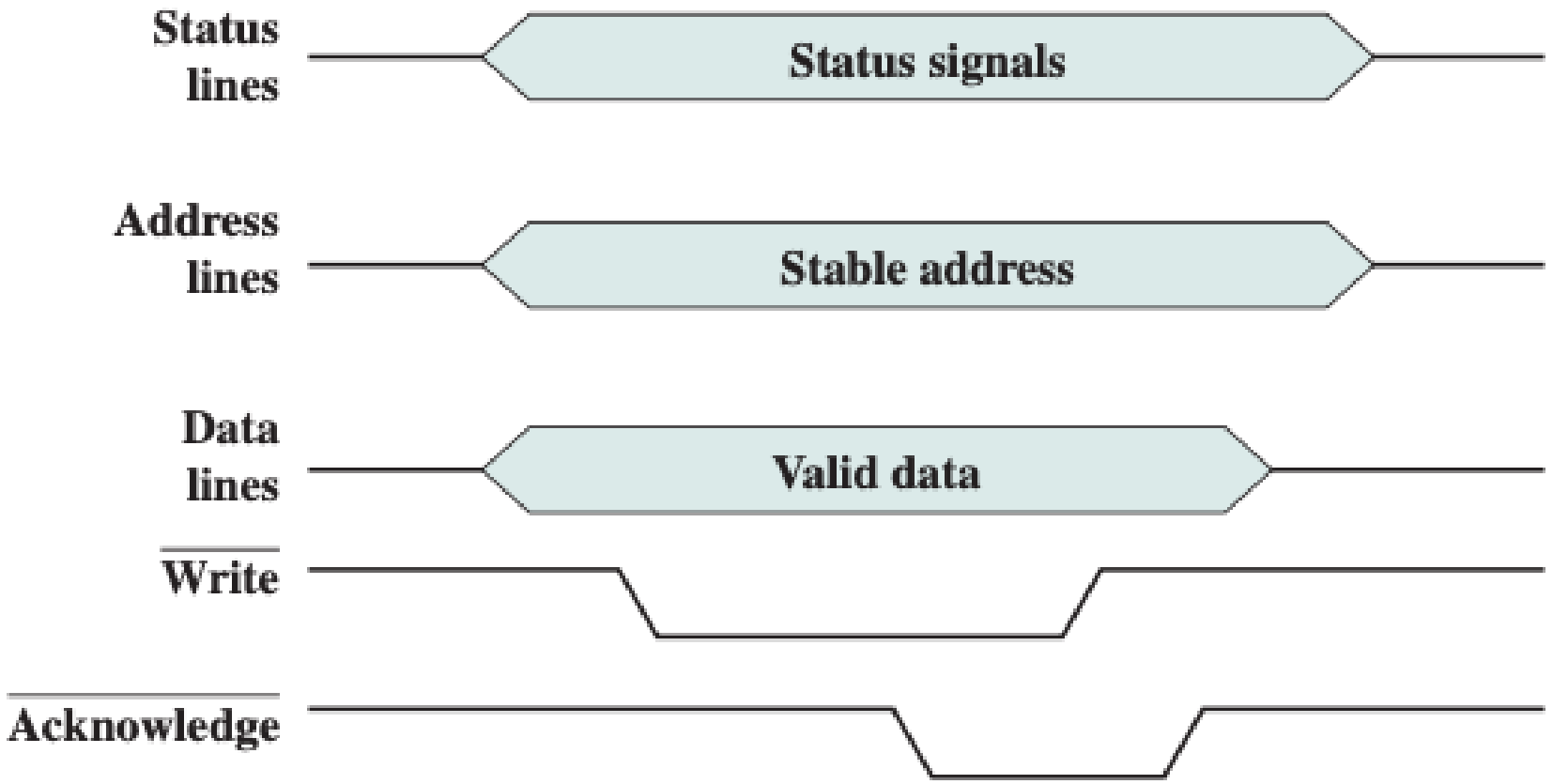
Synchronous Timing Diagram



Asynchronous Timing – Read Diagram



Asynchronous Timing – Write Diagram



Synchronous vs Asynchronous

- Synchronous timing is simpler to implement and test.
- Synchronous timing is less flexible than asynchronous timing.
  - Because all devices on a synchronous bus are tied to a fixed clock rate, the system cannot take advantage of advances in device performance.
  - With asynchronous timing, a mixture of slow and fast devices, using older and newer technology, can share a bus.