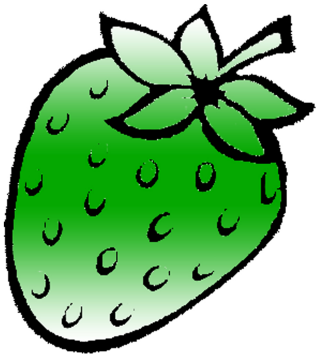
STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com

UNIT II: SYSTEM BUSES

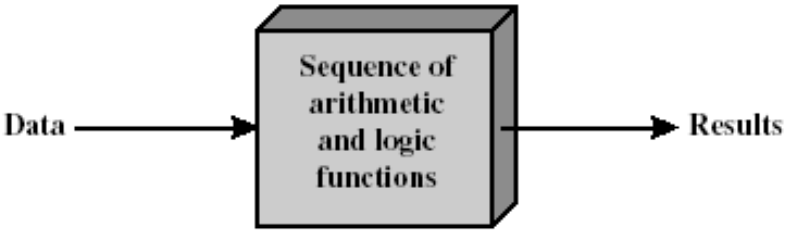
Agenda

- Computer Components
- Computer functions and flow control
  - Instruction Cycle: Fetch & Execute
  - Interrupts
- Interconnection Structures
- Bus Interconnection
  - Bus Structure
  - Multiple Bus Hierarchies
  - Elements of Bus Design

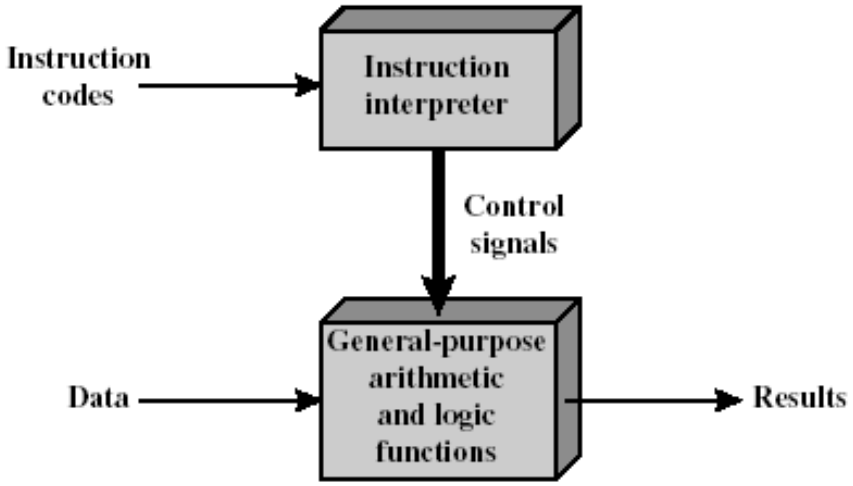
Computer Components

- In **Hardwired systems**, there is a small set of basic logic components that can be combined in various ways to store binary data & to perform arithmetical & logical operations on that data.
- Hardwired systems are inflexible.
- **General purpose hardware** can do different tasks, given correct control signals.
- Instead of re-wiring the hardware for each new program, the programmer merely needs to supply a new set of control signals

Computer Components



(a) Programming in hardware



(b) Programming in software

Hardware & Software Approaches

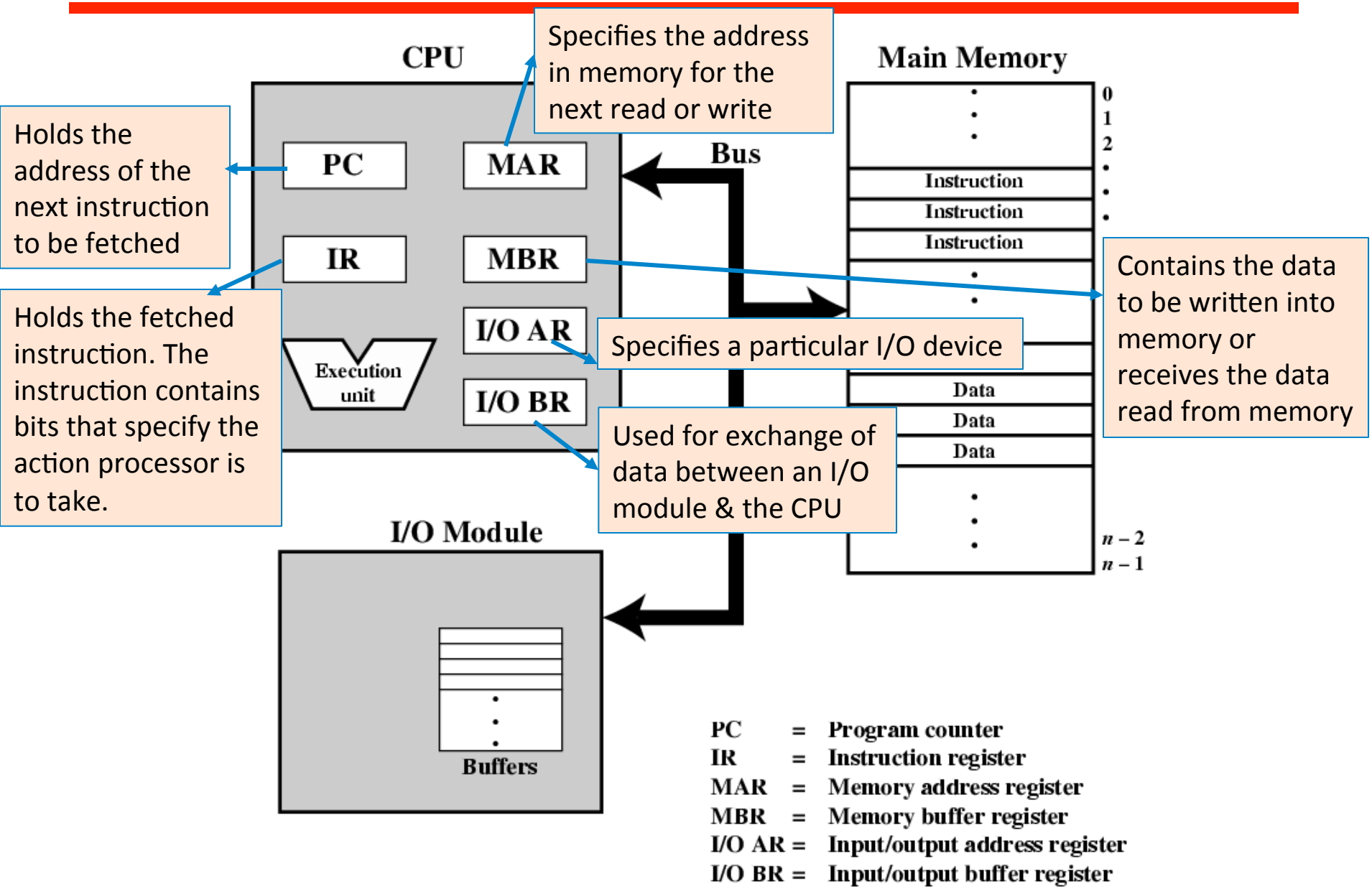
Program

- 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

Computer Components

- The Control Unit (CU) and the Arithmetic and Logic Unit (ALU) constitute the Central Processing Unit (CPU)
- Data and instructions need to get into the system and results need to get out
  - Input/output (I/O module)
- Temporary storage of code and results is needed
  - Main memory (RAM)

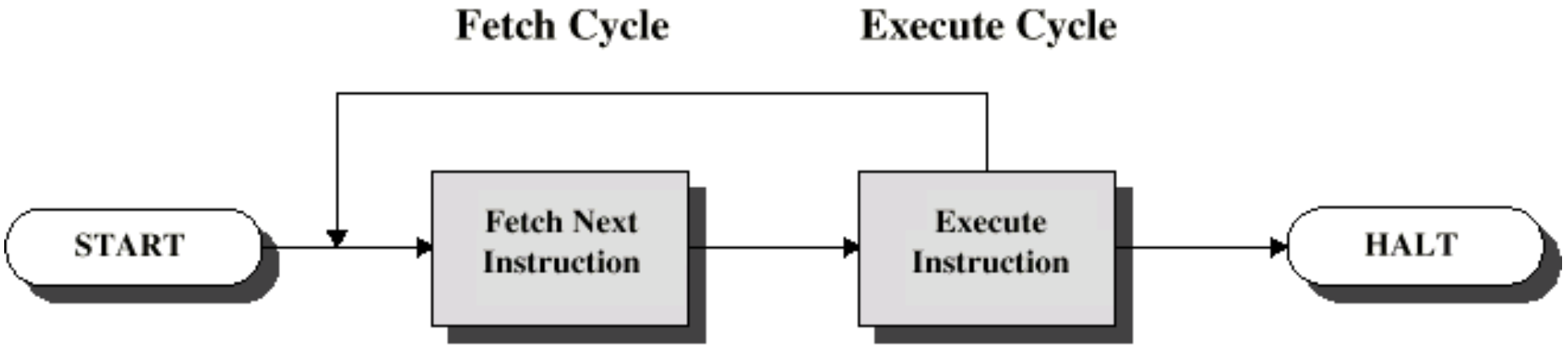
Computer Components: Top Level View



Computer Function

- The basic function performed by a computer is execution of a program, which consists of set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program.

Instruction Cycle



Instruction Cycle: 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) for execution. The instruction contains bits that specify the action processor is to take.

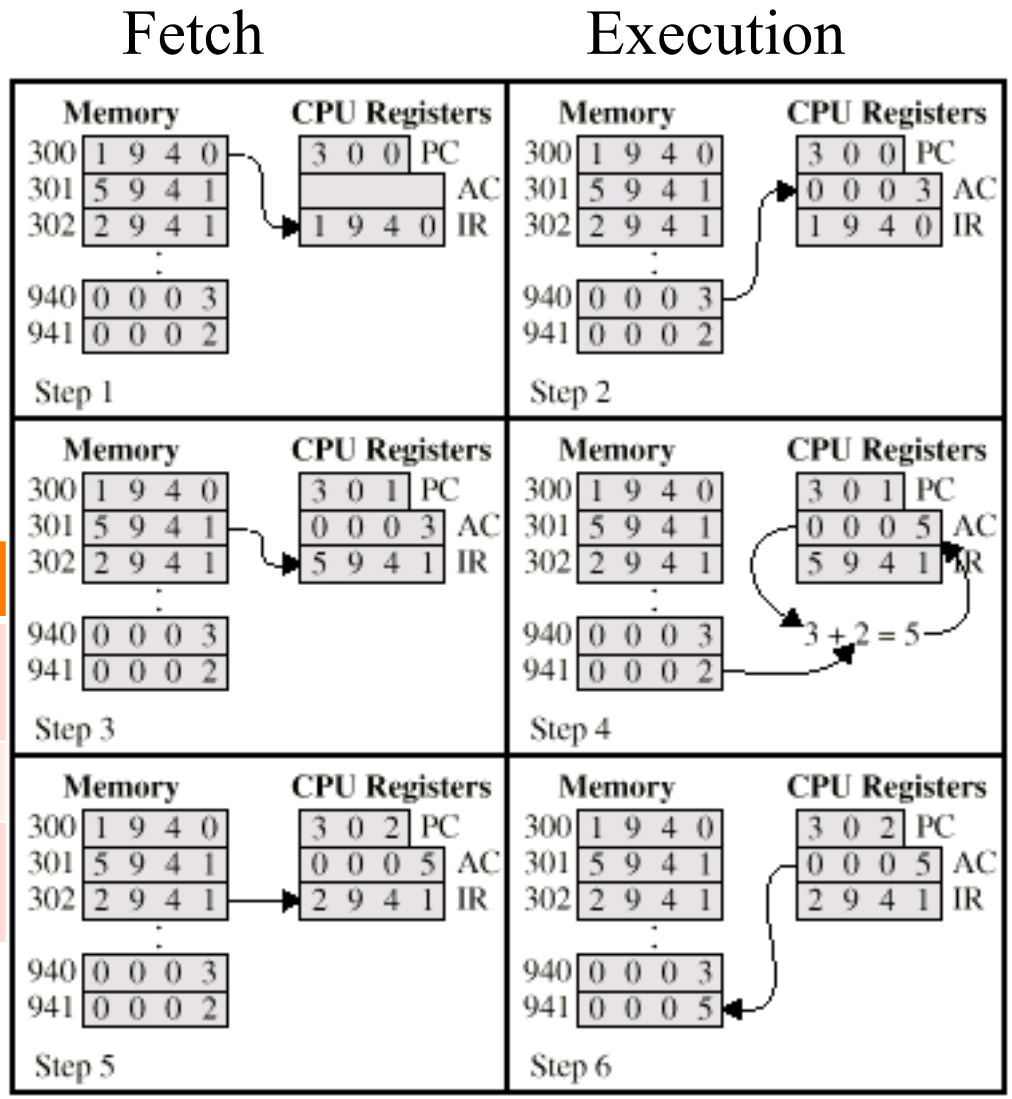
Instruction Cycle: Execute Cycle

- CPU executes the instruction
- Processor interprets instruction and performs required actions, such as:
  - 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

Example of Program Execution

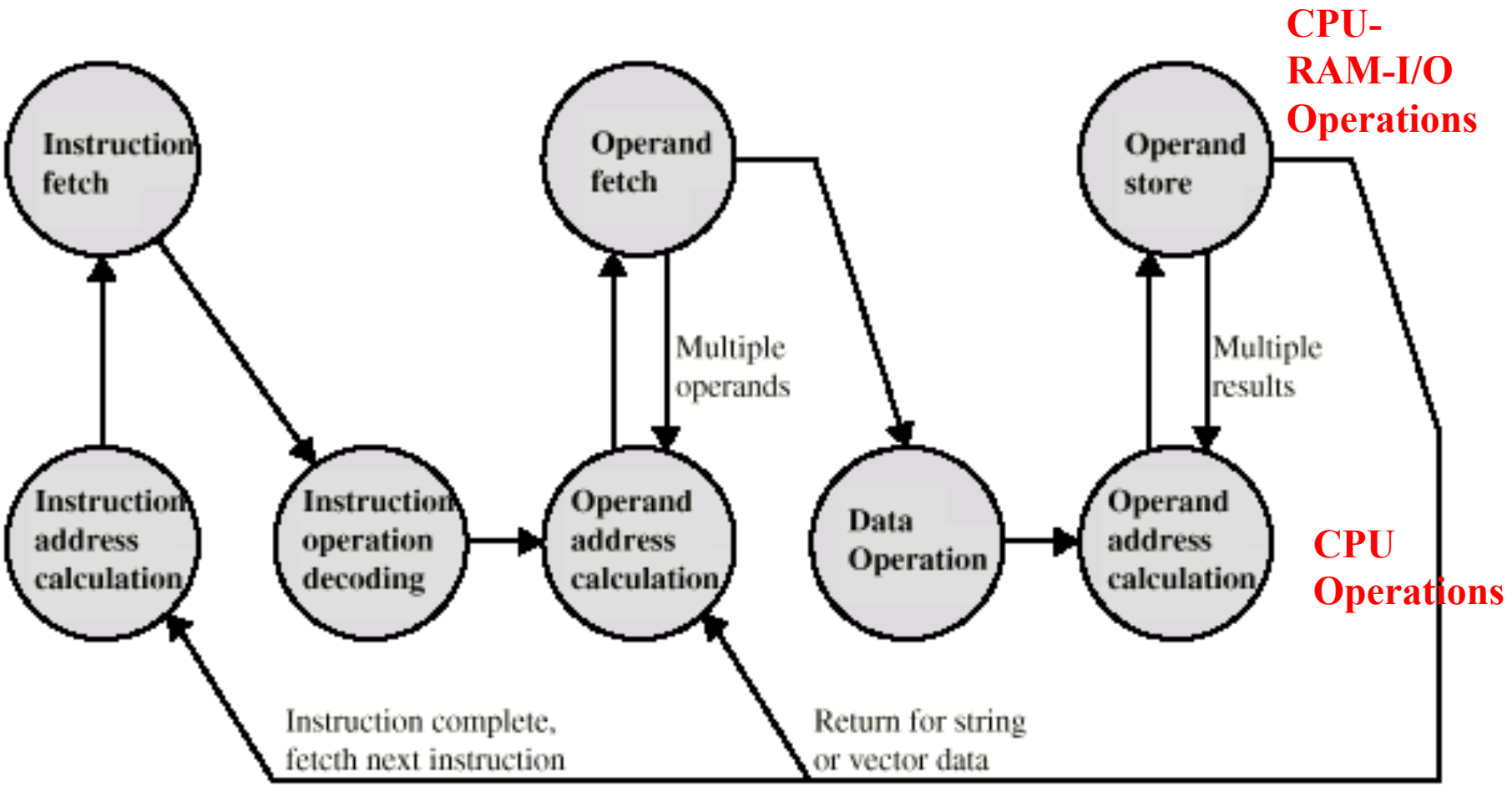
- Internal CPU Registers:
  - **Program Counter(PC)** = Address of instruction
  - **Instruction Register(IR)** = Instruction being executed. The instruction contains bits that specify the action processor is to take.
  - **Accumulator(AC)** = Temporary storage
- Partial list of opcodes:

Binary	Hexadecimal	Instruction
0001	1	Load AC from memory
0010	2	Store AC to memory
0101	5	Add to AC from memory



Note use of hexadecimal

Instruction Cycle – State Diagram



Instruction Cycle – State Diagram

- **Instruction Address Calculation (iac):** Determine the address of the next instruction
- **Instruction Fetch (if):** Read instruction from its memory location into processor.
- **Instruction Operation Decoding (iod):** analyze instruction to determine operation type to be performed and operands to be used.
- **Operand Address Calculation (oac):** If the operation involves reference to an operand in memory then determine the address of the operand.
- **Operand Fetch (of):** Fetch the operand from memory or read it from I/O
- **Data Operation (do):** Perform indicated operation in the instruction
- **Operand Store (os):** Write result into memory or out to I/O

Interrupts

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Interrupts are provided primarily as a way to improve processing efficiency.
- **Classes of Interrupts:**
  - **Program:** Generated by some condition that occurs as a result of an instruction execution. e.g. overflow, division by zero
  - **Timer:** Generated by a timer within the processor. This allows the operating system to perform certain functions on regular basis.
  - **I/O:** Generated by an I/O controller to signal normal completion of an operation or to signal variety of error conditions.
  - **Hardware failure:** Generated by a failure such as power failure.

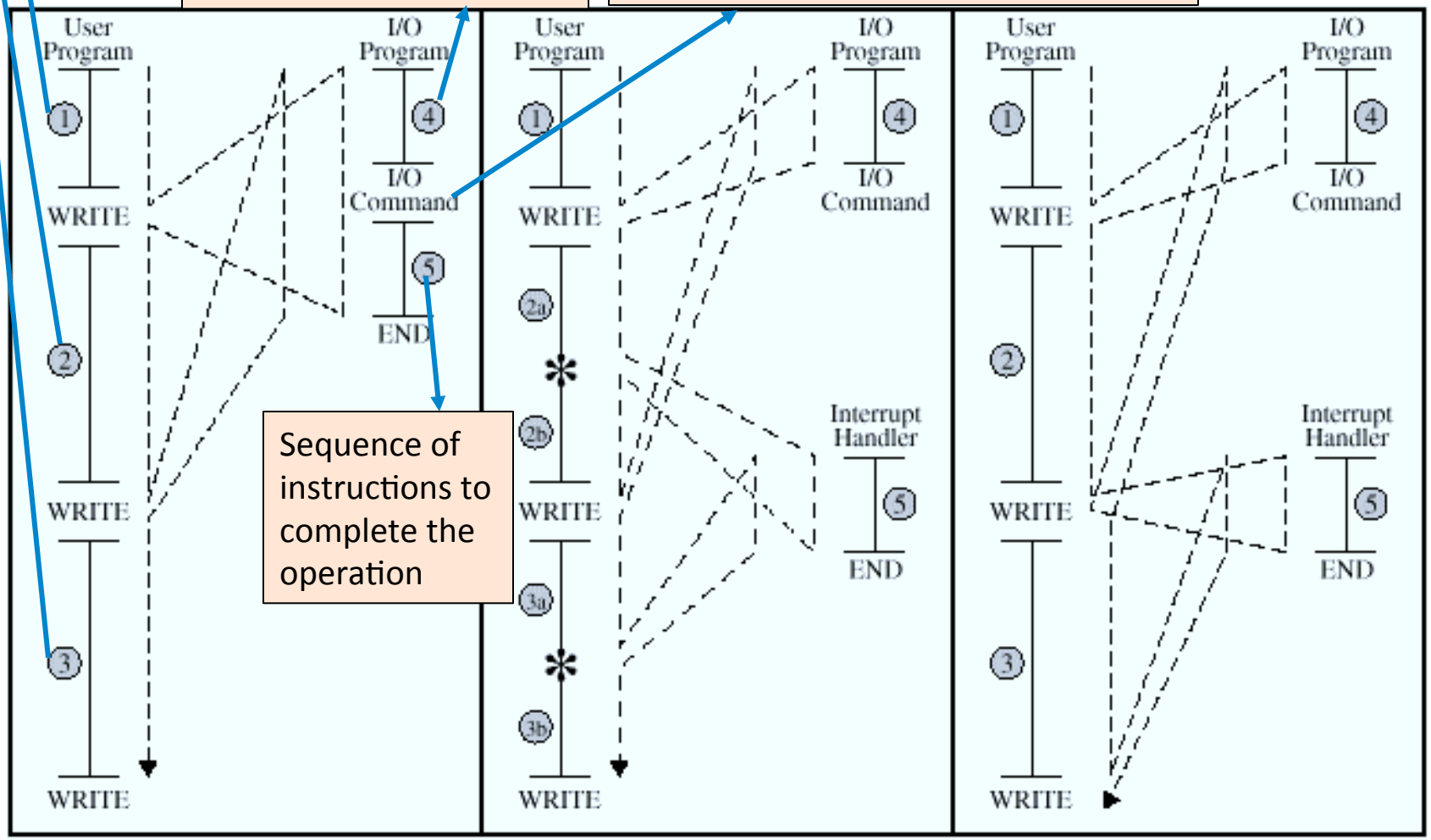
USER PROGRAM PERFORMS A SERIES OF WRITE CALLS INTERLEAVED WITH PROCESSING

Sequence of instructions that do not involve I/O

Sequence of instructions to prepare for the actual I/O operation

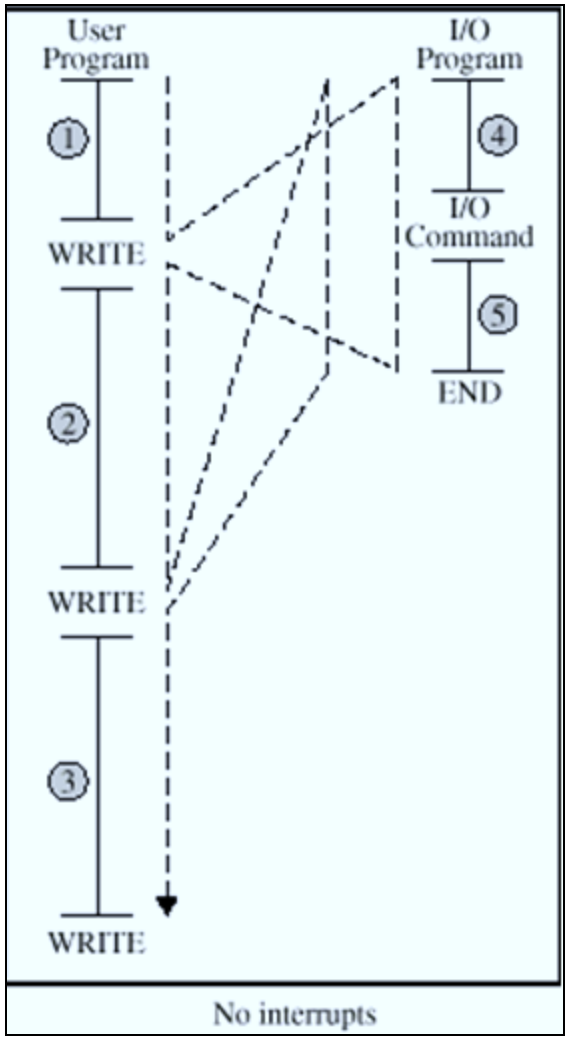
Once this command is issued, the program must wait for the I/O device to perform the requested function.

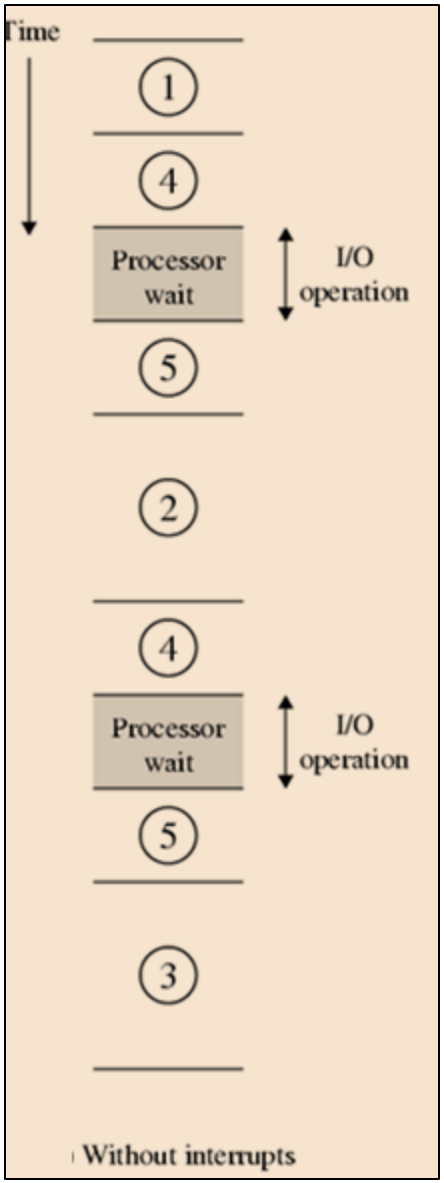
Sequence of instructions to complete the operation



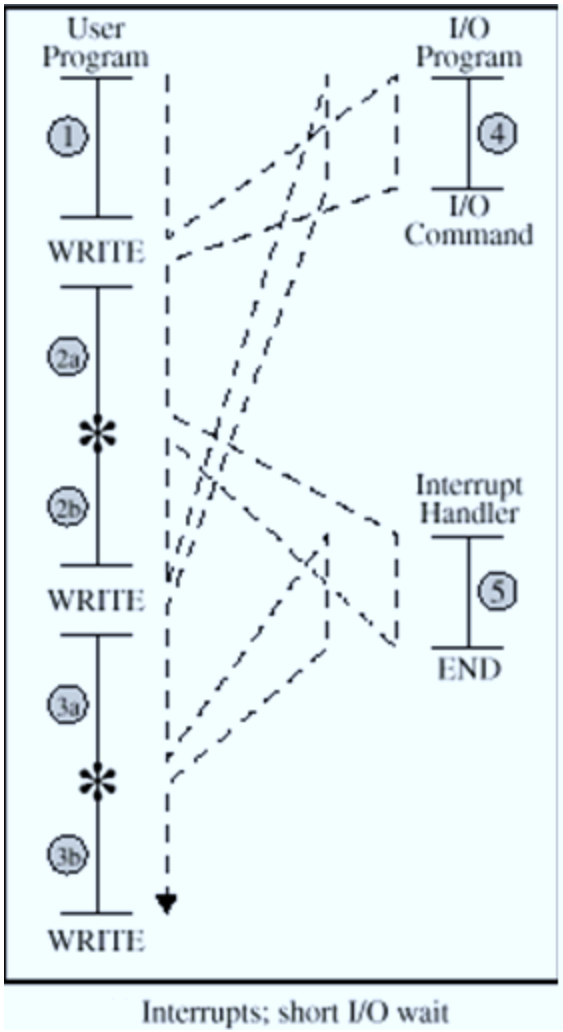
(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

No Interrupt: Timing Diagram

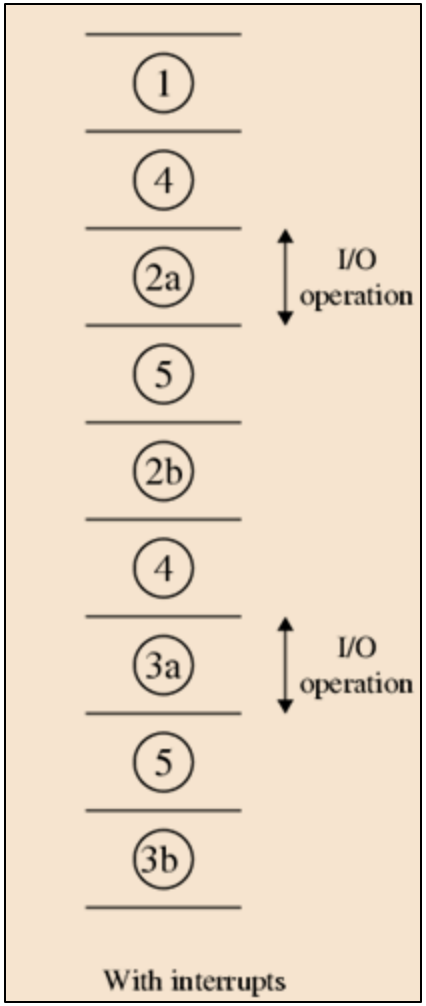




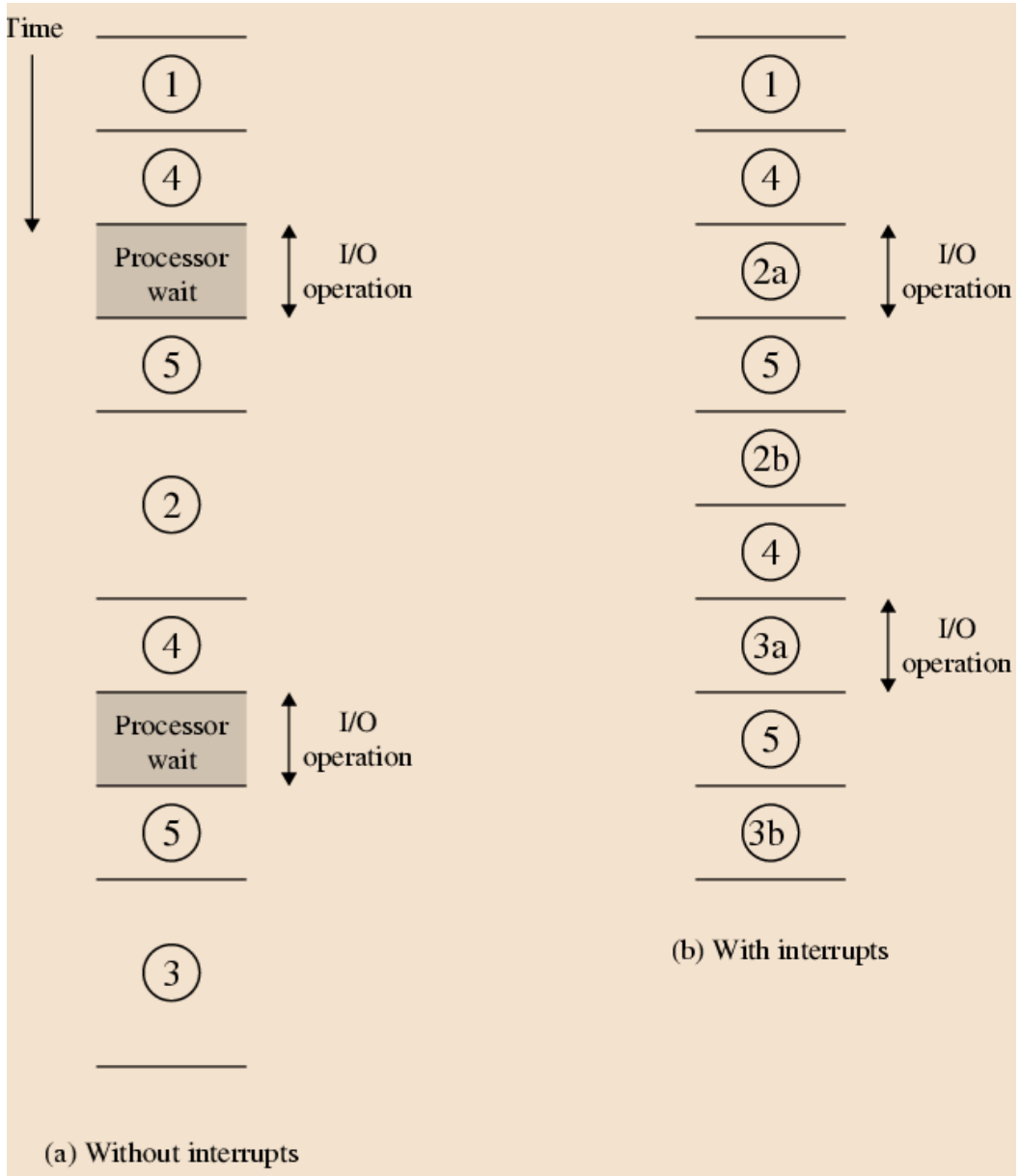
Short I/O Wait: Timing Diagram



Interrupts; short I/O wait

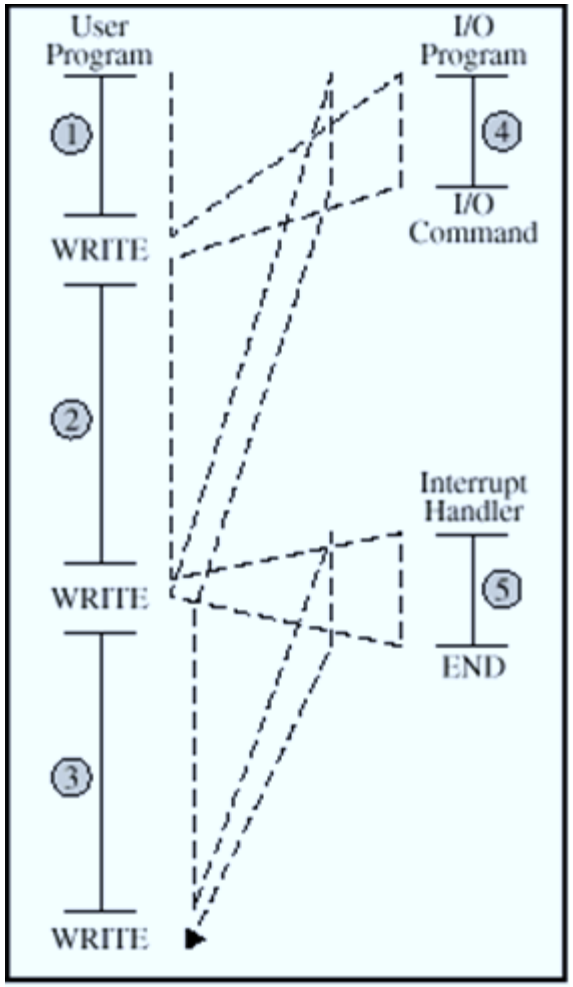


Comparison of Timing Diagrams

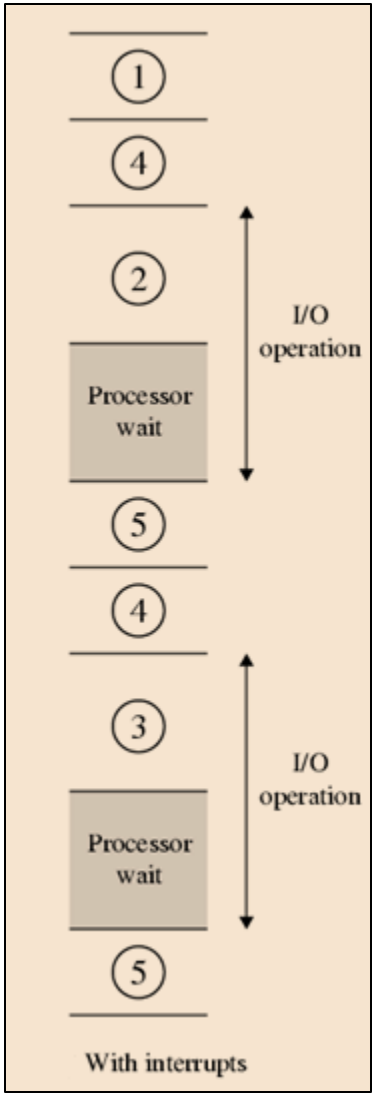


There is a gain in efficiency using interrupts as processor doesn't have to wait for an I/O operation.

Long I/O Wait: Timing Diagram

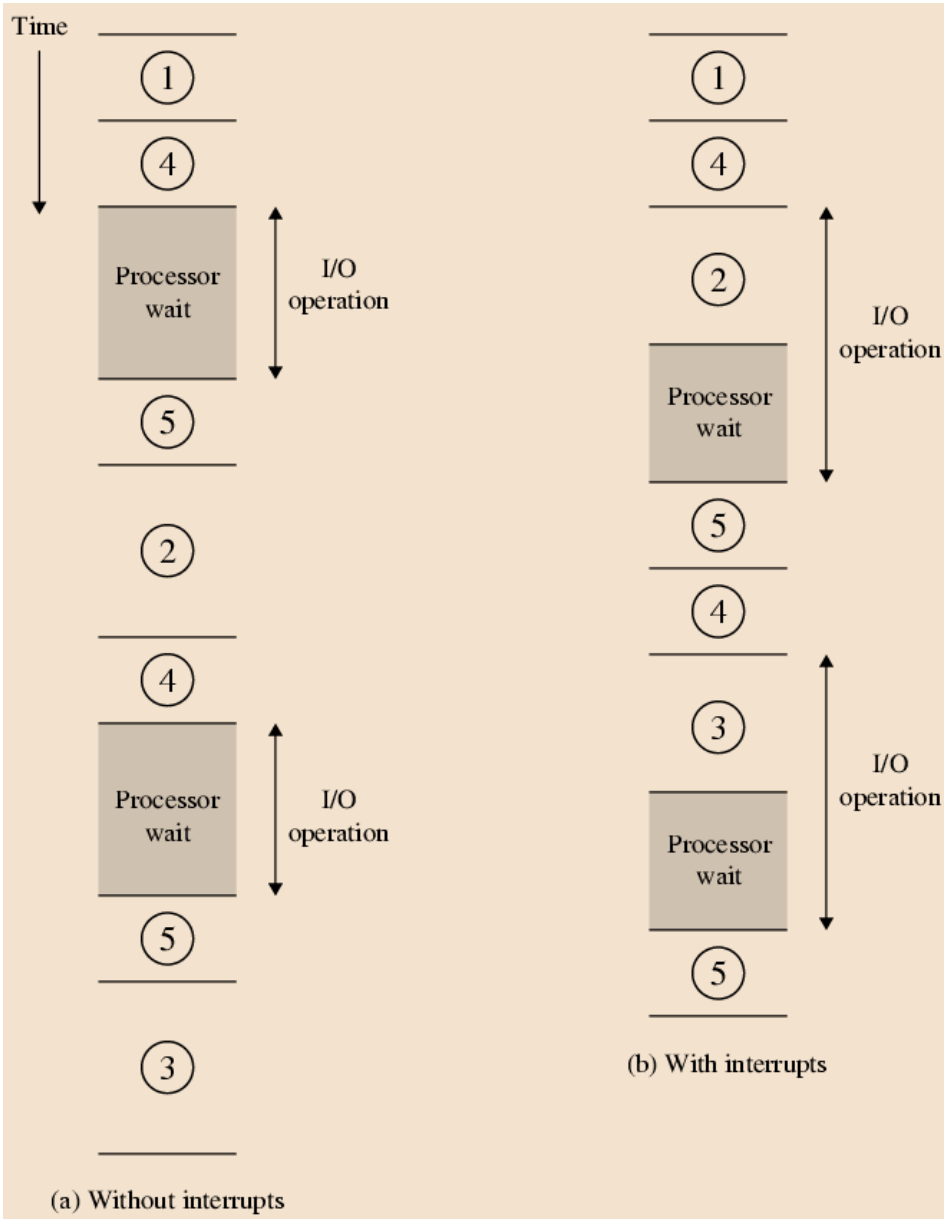


Interrupts; long I/O wait



With interrupts

Comparison

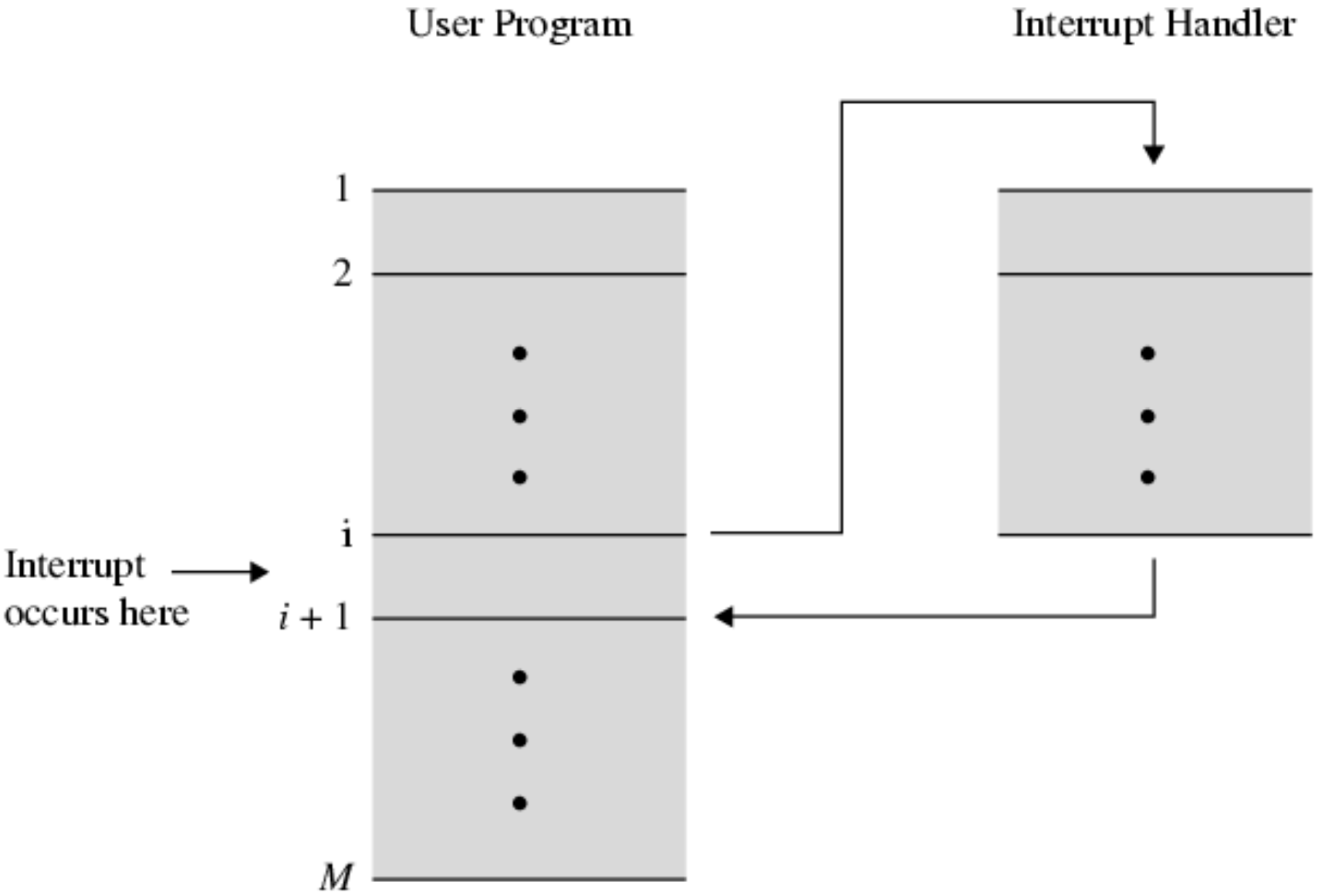


There is a gain in efficiency using interrupts because part of the time during which the I/O operation is underway overlaps with the execution of user instructions.

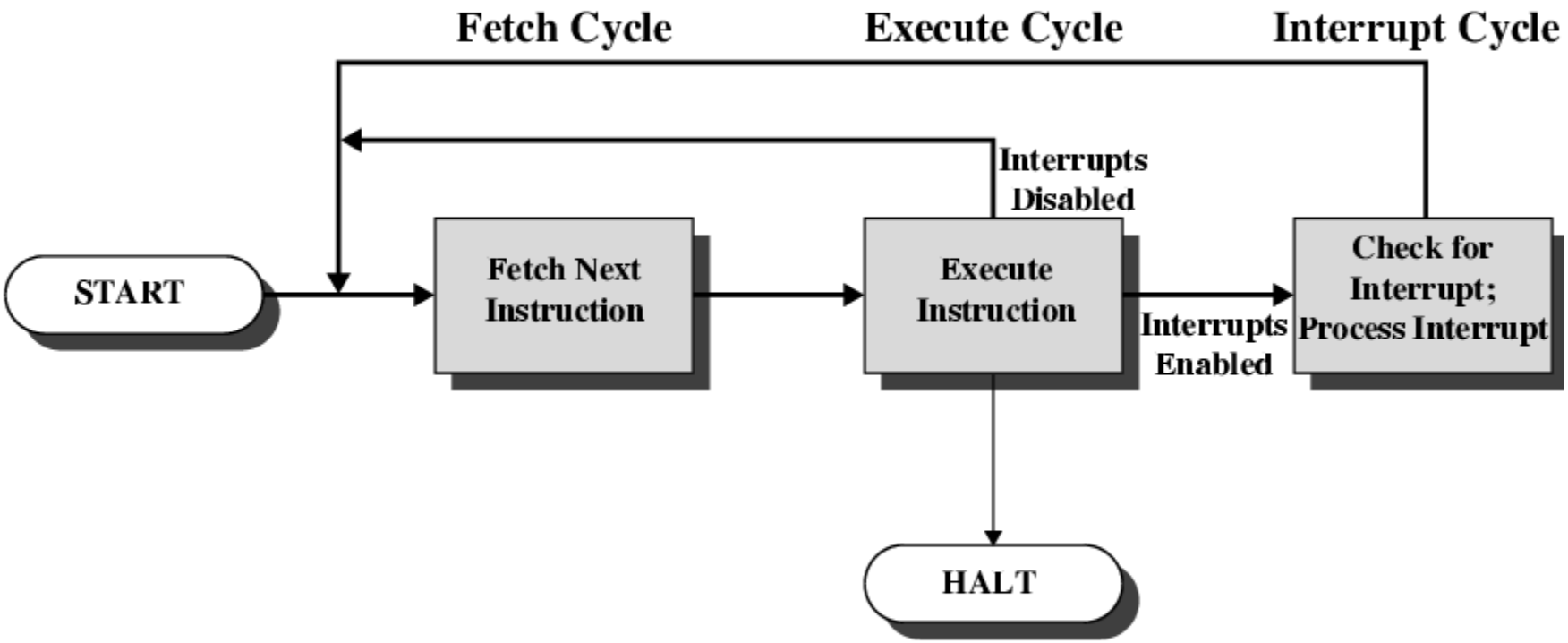
Interrupt Cycle

- 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

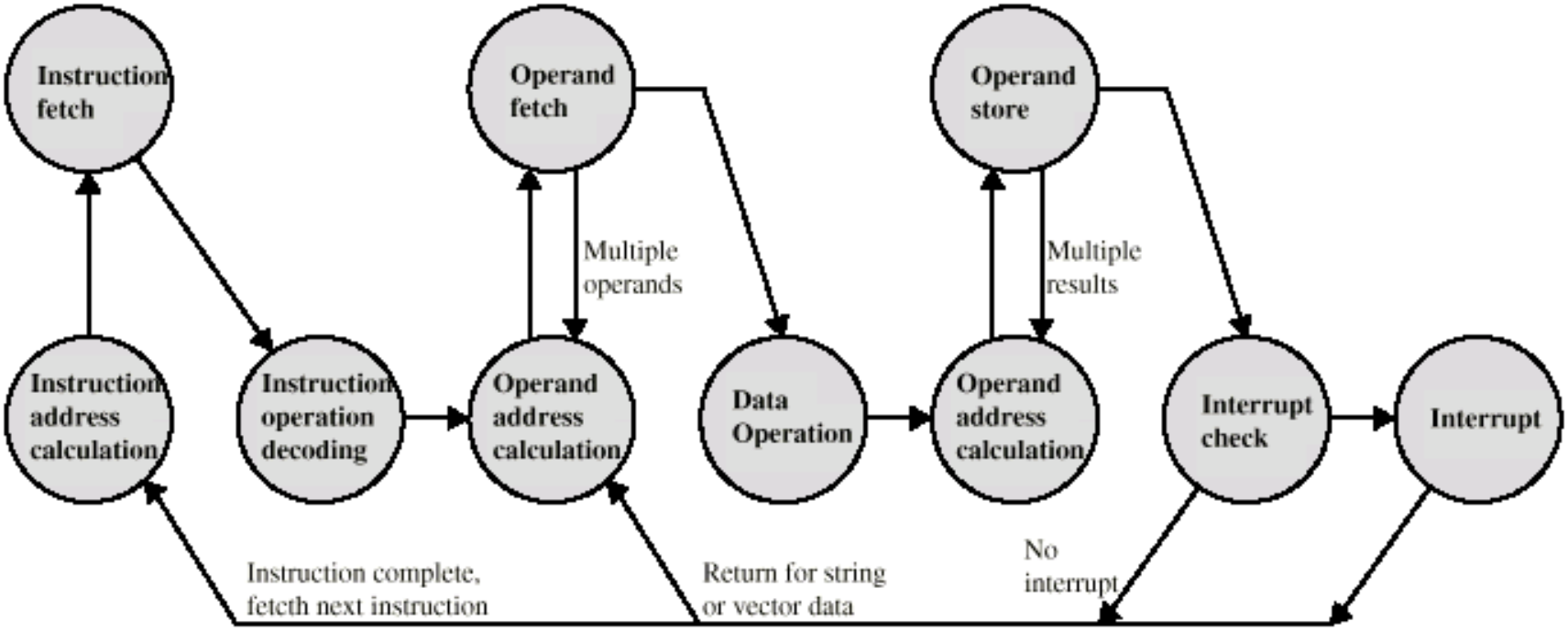
Transfer of Control via Interrupts



Instruction Cycle with Interrupts



Instruction Cycle (with Interrupts) - State Diagram

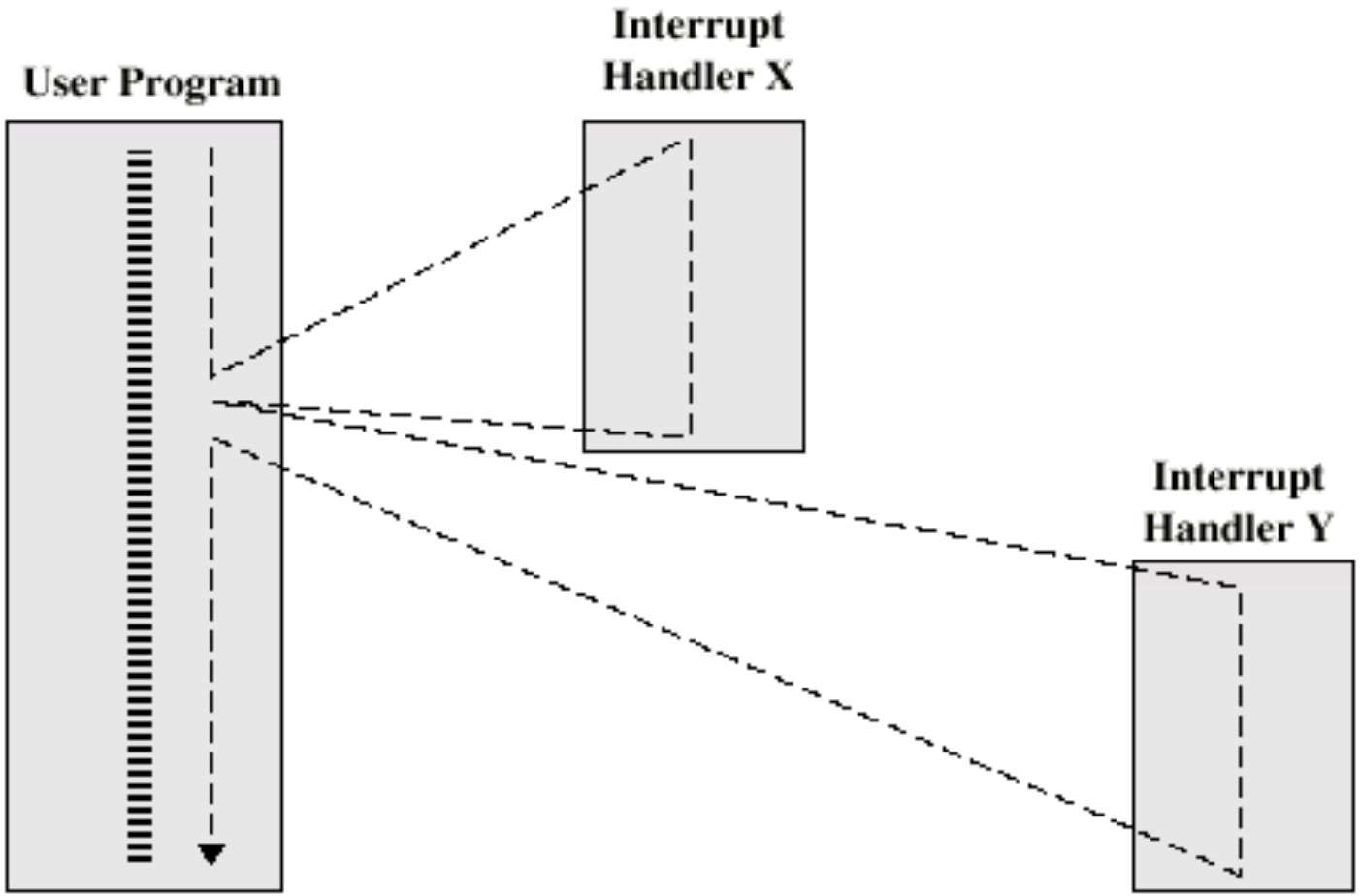


Multiple Interrupts

- Disable interrupts (approach #1)
  - 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 (approach #2)
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

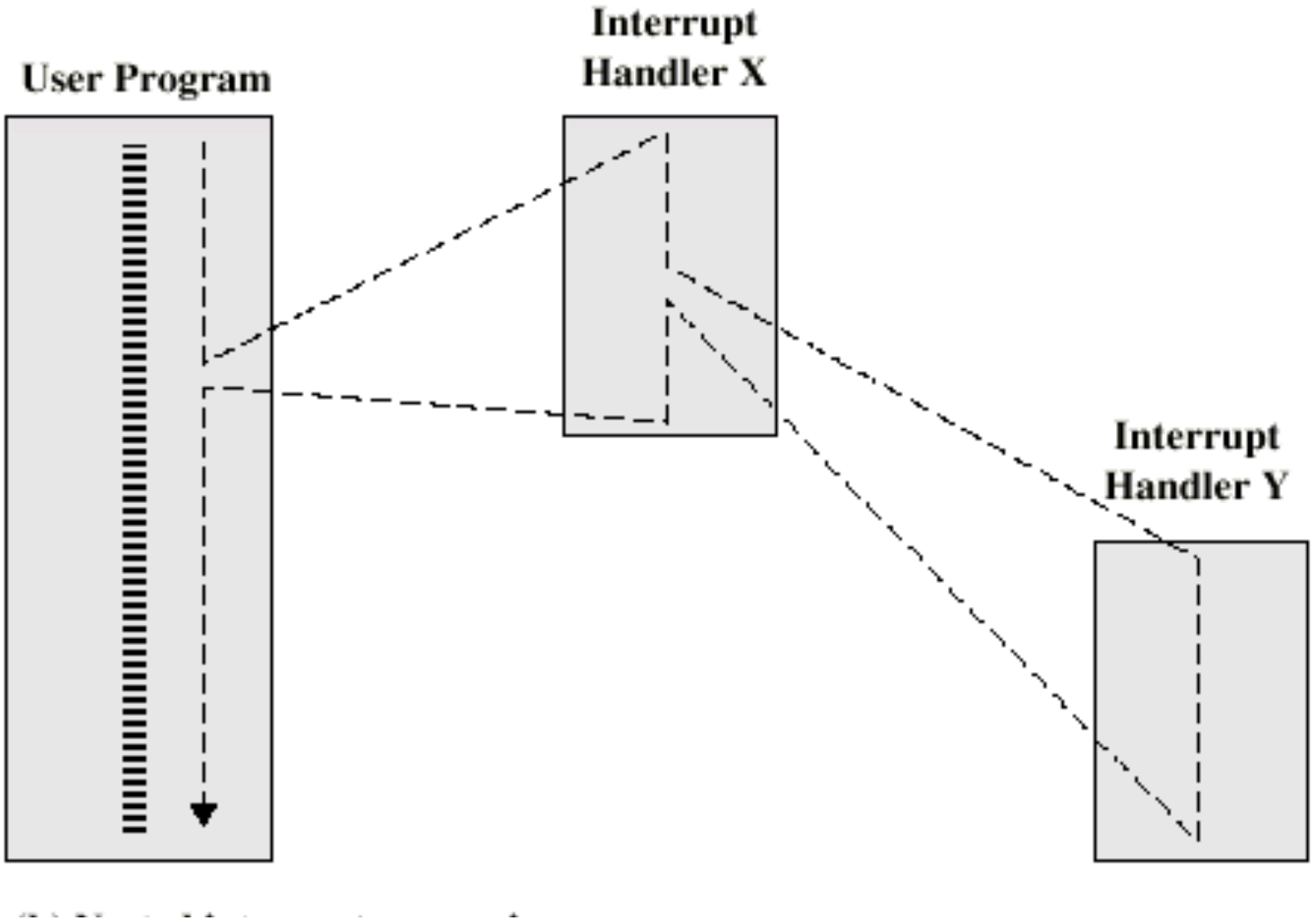
Multiple Interrupts

- **SEQUENTIAL INTERRUPT PROCESSING:**



Multiple Interrupts

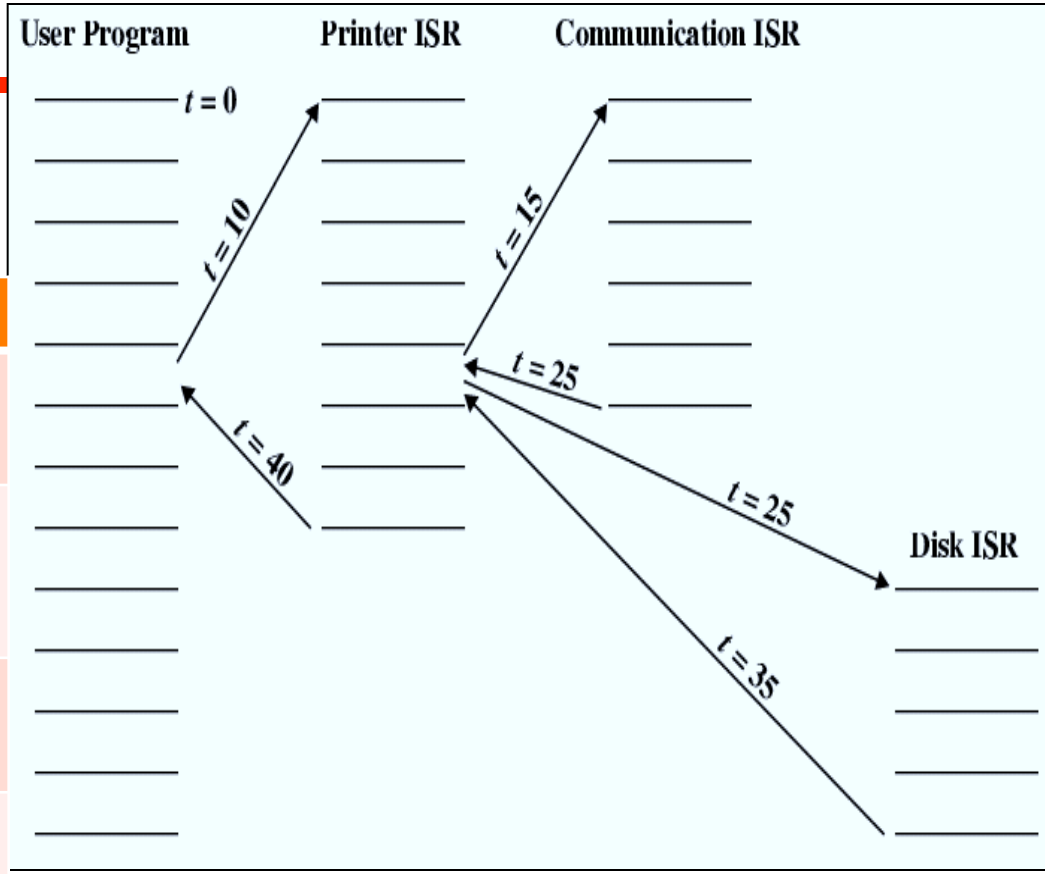
- **NESTED INTERRUPT PROCESSING:**



Time Sequence of Multiple Interrupts

I/O DEVICES	PRIORITY
Printer	2
Disk	4
Communication Line	5

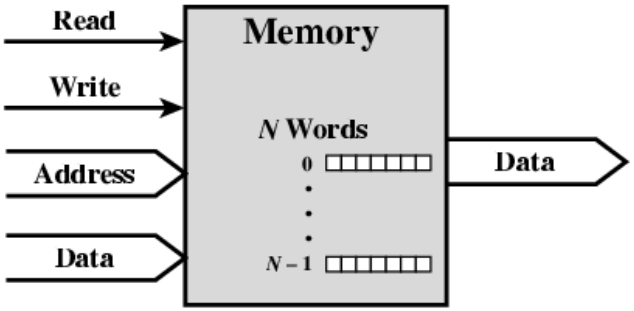
Time	Interrupt	Access	State
At t=10	Printer interrupt occurs	Granted	Printer ISR Execute
At t=15	Communication line interrupt occurs	Granted (high priority)	Communication ISR Execute & Printer ISR wait
At t=20	Disk Interrupt occurs	Rejected (low priority)	Wait
At t=25			Communication ISR completes
At t=25			Disk execute
At t=35			Disk ISR completes
At t=35			Printer ISR resumes
At t=40			Printer ISR completes

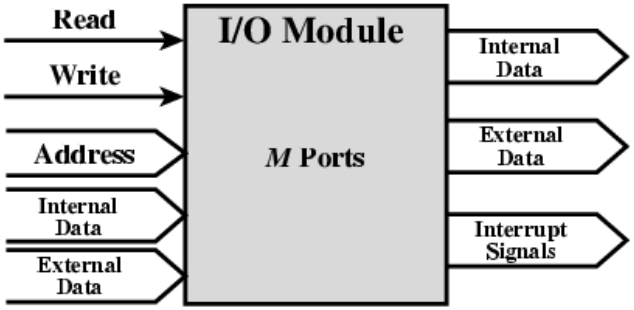


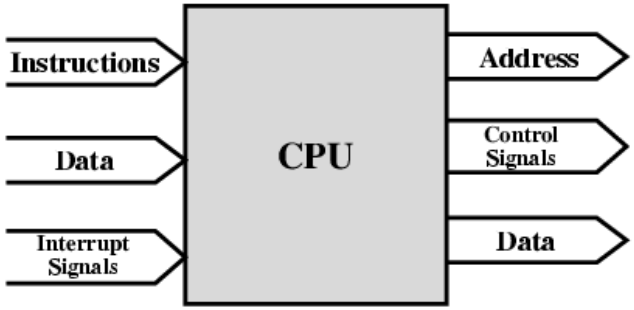
Interconnection Structures

- A computer consists of a set of components or modules of three basic types – *Processor, Memory & I/O* that communicate with each other.
- The collection of paths connecting the various modules is called as *Interconnection Structure.*

Interconnection Structures – Computer Modules

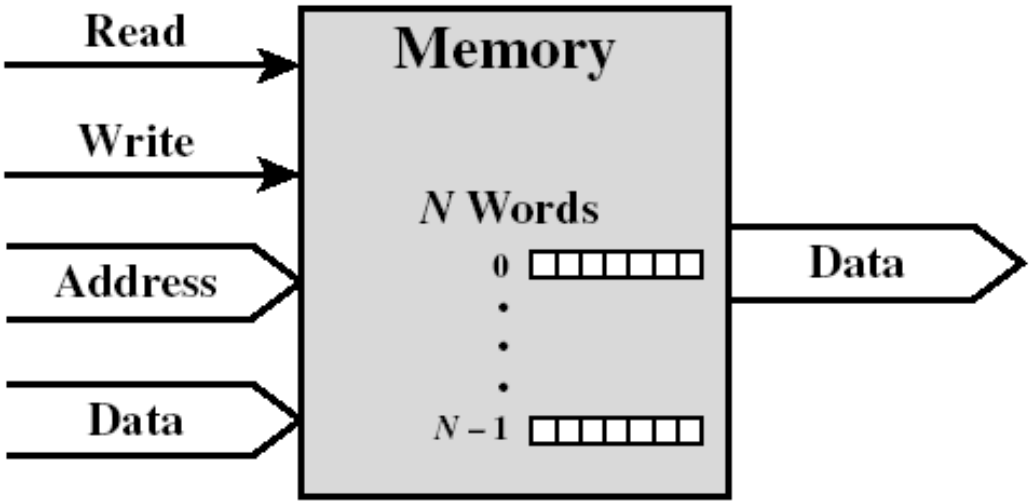






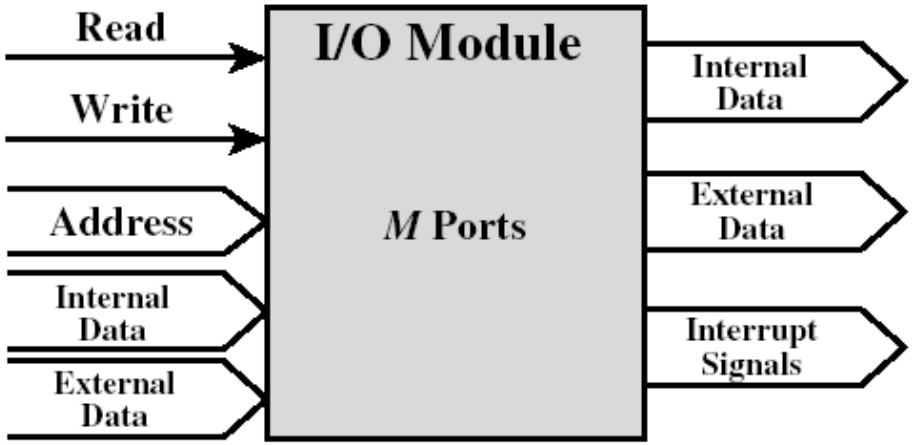
Memory

- It consist of N words of equal length. Each word is assigned a unique numerical address (0,1,……N-1).
- A word can be read from or written into the memory.
- Receives and sends data
- Receives addresses (of locations)
- Receives control signals
  - Read
  - Write
  - Timing



Input/Output

- Two operations Read & Write.
- I/O Module Can Control More Than One External Device. Each interface to an external device is referred to as port & has unique address (0,1, ….M-1)
- Output
  - Receive data from computer
  - Send data to peripheral
- Input
  - Receive data from peripheral
  - Send data to computer

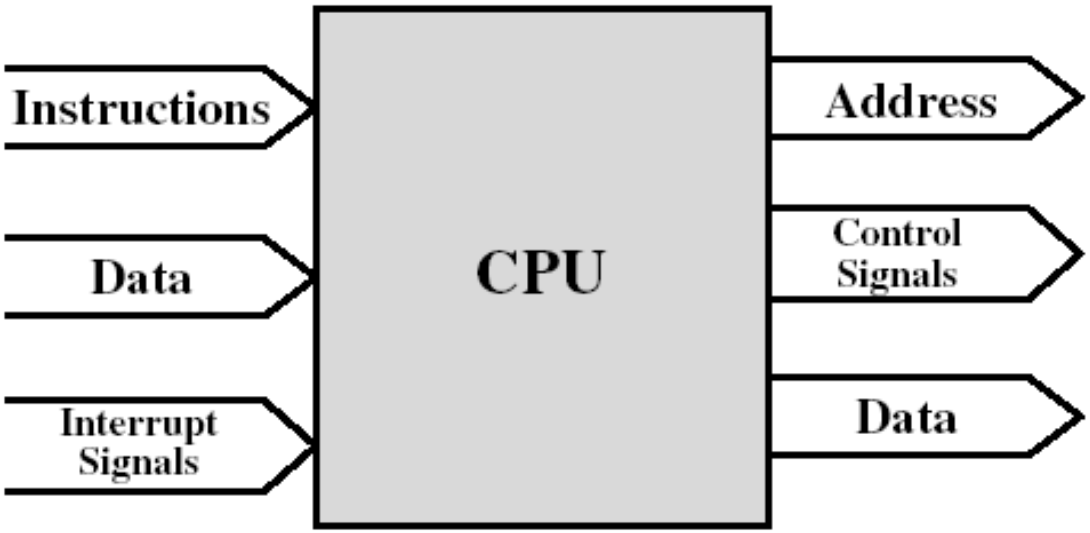


Input/Output

- 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)

Processor

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units to control the overall operation of the system.
- Receives interrupts signals



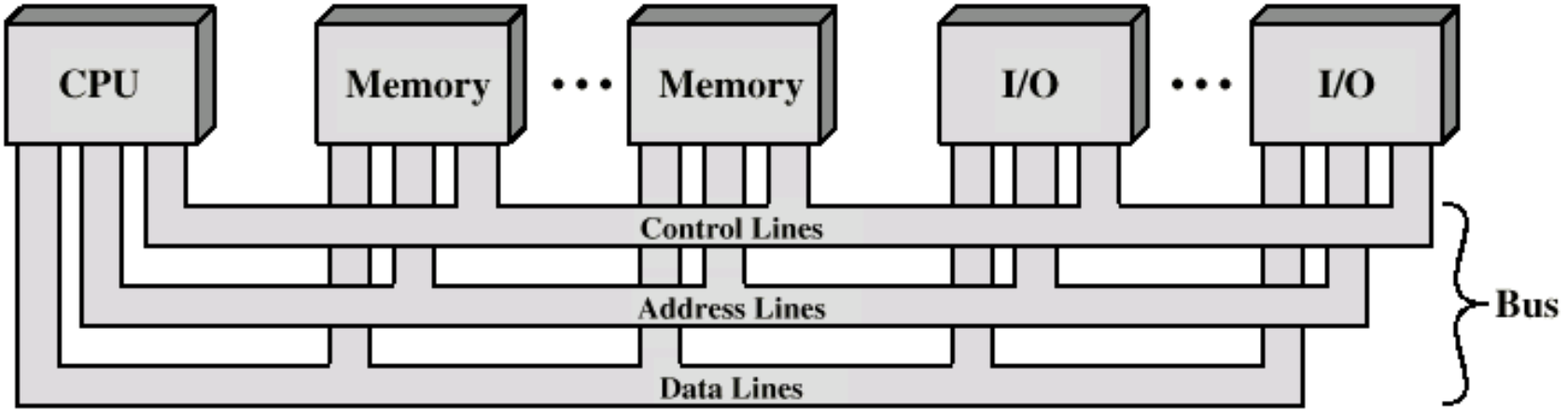
Interconnection Structures

- Interconnection Structures must support the following types of transfers:
  - Memory to processor: Processor reads an instruction or a unit of data from memory.
  - Processor to Memory: Processor writes a unit of data to memory.
  - I/O to Processor: Processor reads data from an I/O device via an I/O module.
  - Processor to I/O: Processor sends data to the I/O device.
  - I/O to or from Memory: I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

Buses

- A communication pathway connecting two or more devices.
- A key characteristic of a bus is that it 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 become garbled. Thus, only one device at a time can successfully transmit.
- Each line is capable of transmitting signals representing binary 1 and binary 0. For example, an 8-bit unit of data can be transmitted over eight bus lines.
- A bus that connects major computer components (processor, memory, I/O) is called a system bus

Bus Interconnection Scheme



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, the number of lines being referred to as the width of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time.
- 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.

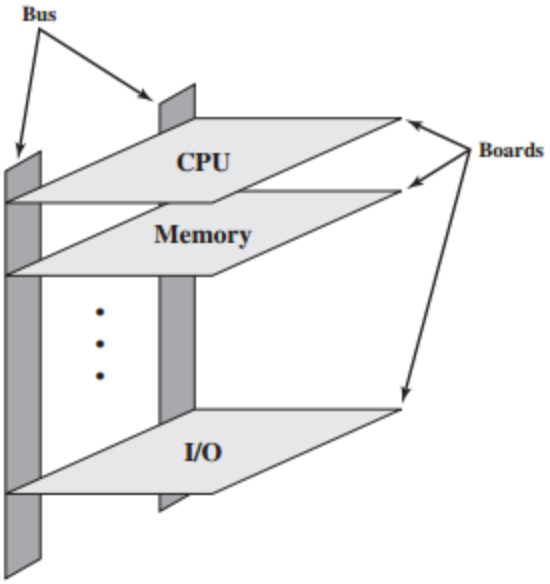
Address bus

- The address lines are used to designate the source or destination of the data on the data bus.
- For example, 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.
- Furthermore, the address lines are generally also used to address I/O ports.
- Typically, 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 within the module.

Control Bus

- The control lines are used to control the access to and the use of the data and address lines.
- Typical control lines include:
  - 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
- Timing signals indicate the validity of data and address information.
- Command signals specify operations to be performed.

Bus



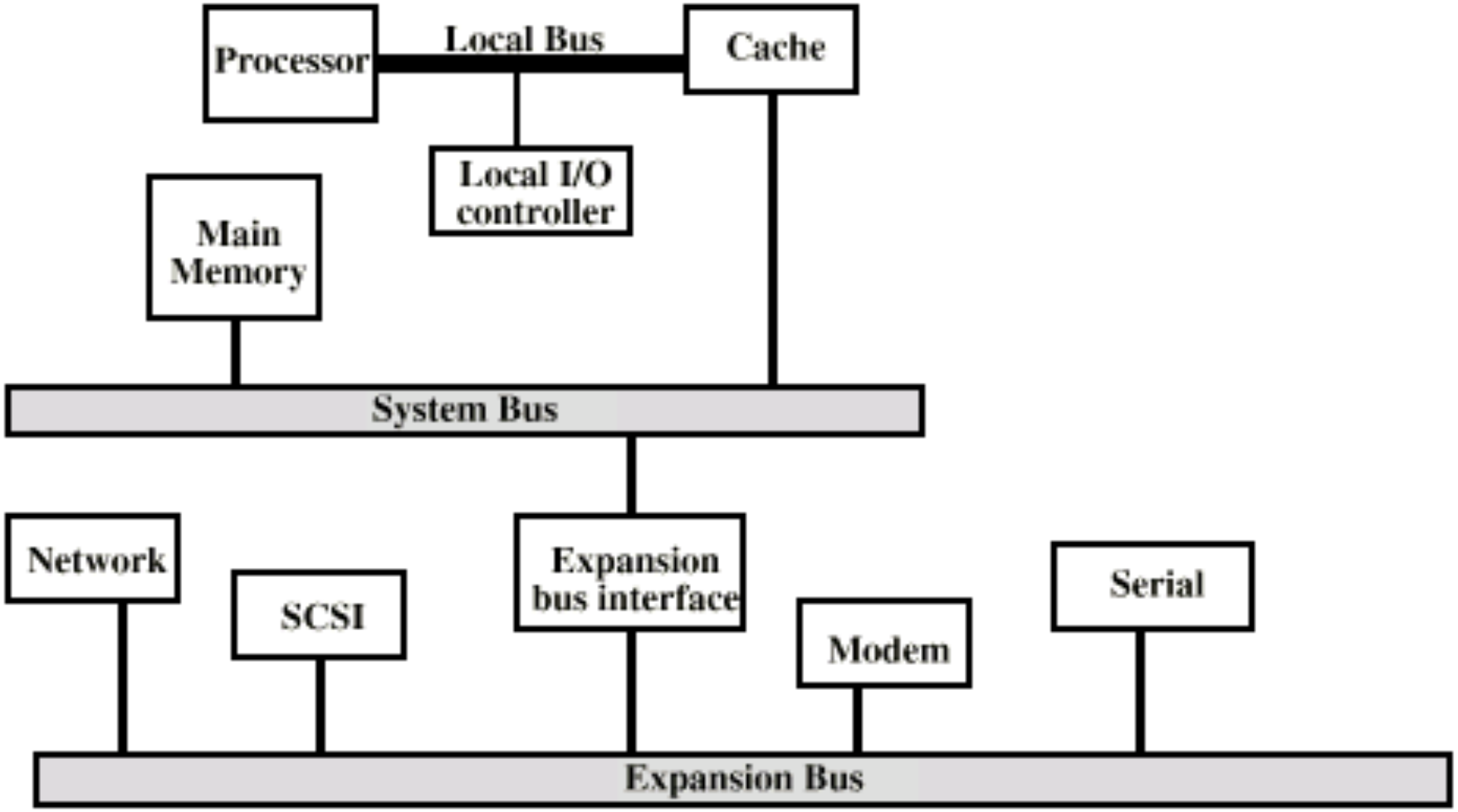
Typical Physical Realization of a Bus Architecture

- This arrangement is most convenient. A small computer system may be acquired and then expanded later (more memory, more I/O) by adding more boards. If a component on a board fails, that board can easily be removed and replaced.

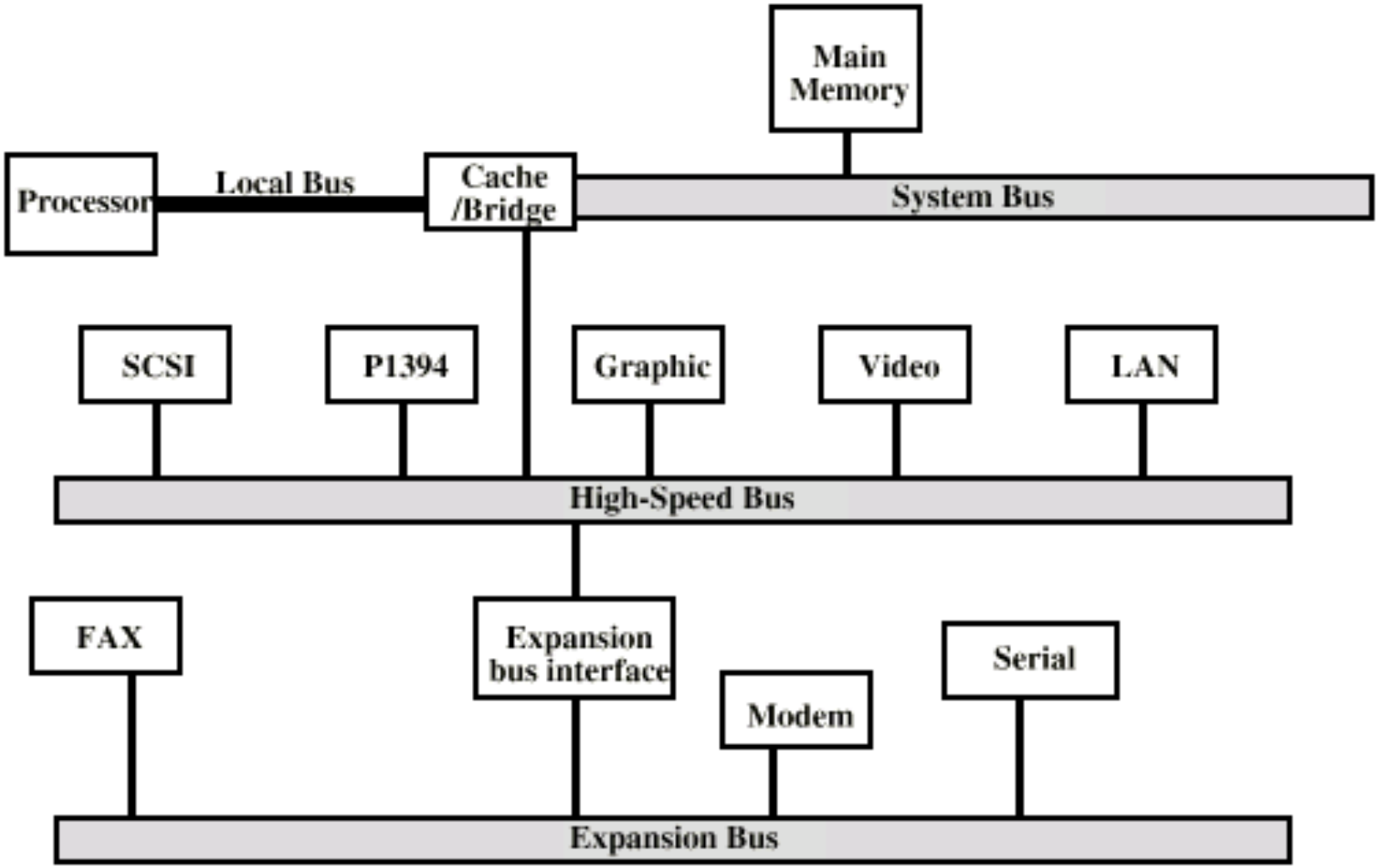
Multiple-Bus Hierarchies

- If a great number of devices are connected to the bus, performance will suffer. There are two main causes:
  - In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.
  - The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.
- Most systems use **Multiple Buses** to overcome these problems.

Traditional (ISA) (with cache)



High Performance Bus



Elements of Bus Design

Table 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

Bus Types

- Dedicated
  - Separate data & address lines
- Multiplexed
  - Shared lines
  - Address valid or data valid control line
  - Advantage - fewer lines which saves space and cost.
  - Disadvantages
    - More complex circuitry is needed within each module.
    - Also, there is a potential reduction in performance because certain events that share the same lines cannot take place in parallel.

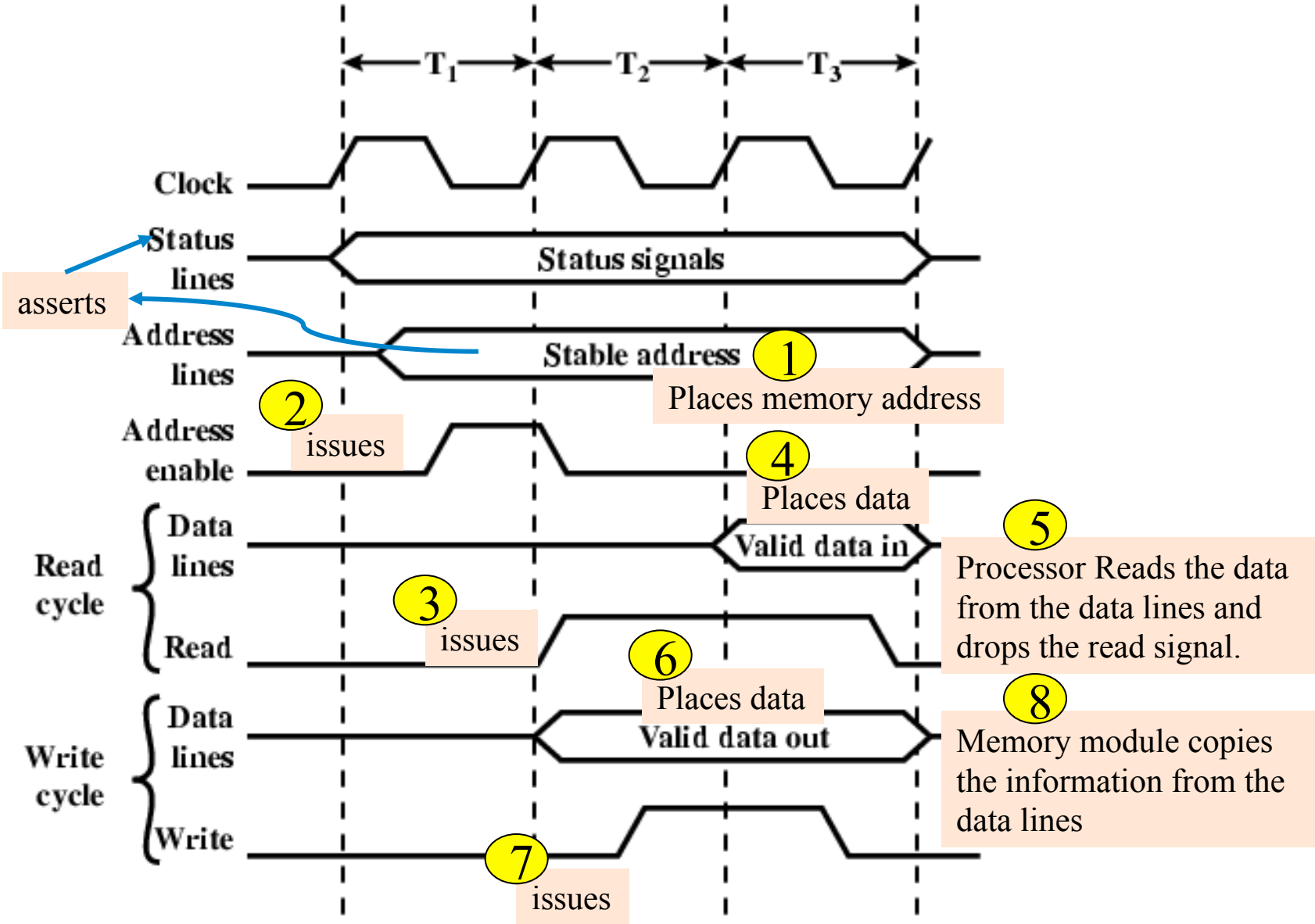
Bus Arbitration

- Only one unit at a time can successfully transmit over the bus, some method of arbitration is needed.
- Arbitration may be centralised or distributed
- **Centralised Arbitration:** A single hardware device, referred to as a bus controller or arbiter, is responsible for allocating time on the bus. The device may be a separate module or part of the processor.
- **Distributed Arbitration:** There is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.
- With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master may then initiate a data transfer (e.g., read or write) with some other device, which acts as slave for this particular exchange.

Timing

- Co-ordination of events on bus.
- **Synchronous Timing:**
  - 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
- **Asynchronous Timing:** The occurrence of one event on a bus follows and depends on the occurrence of a previous event.

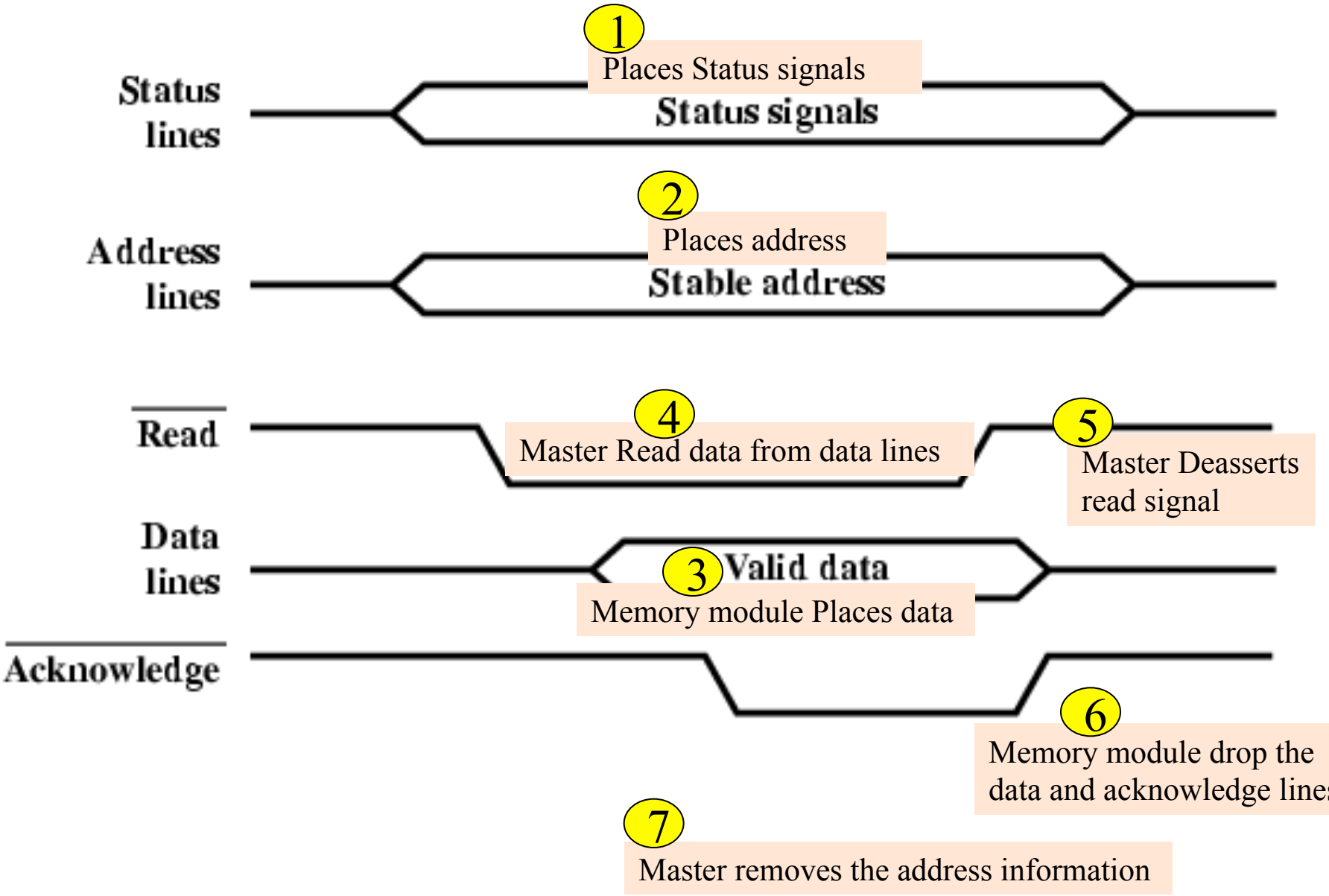
Synchronous Timing Diagram



Synchronous Timing

- The processor places a memory address on the address lines during the first clock cycle and may assert various status lines.
- Once the address lines have stabilized, the processor issues an address enable signal.
- For a 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 after the data lines have stabilized.
- The memory module copies the information from the data lines during the third clock cycle.

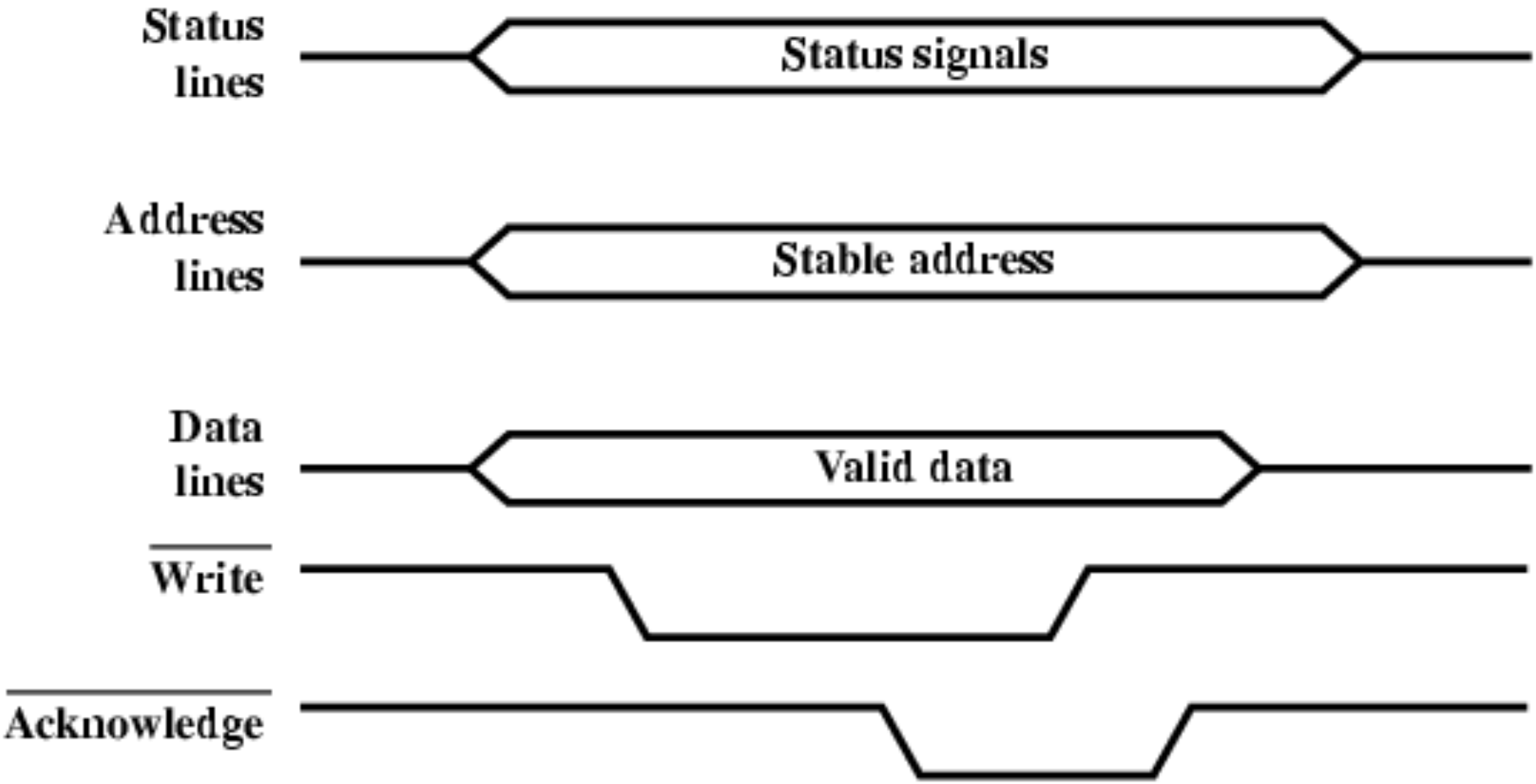
Asynchronous Timing – Read Diagram



Asynchronous Timing – Read Diagram

- The processor places address and status signals on the bus.
- After pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals.
- The appropriate memory decodes the address and responds by placing the data on the data line.
- Once the data lines have stabilized, the memory module asserts the acknowledged line to signal the processor that the data are available.
- Once the master has read the data from the data lines, it deasserts the read signal.
- This causes the memory module to drop the data and acknowledge lines.
- Finally, once the acknowledge line is dropped, the master removes the address information.

Asynchronous Timing – Write Diagram



Asynchronous Timing – Write Diagram

- The master places the data on the data line at the same time that is puts signals on the status and address lines.
- The memory module responds to the write command by copying the data from the data lines and then asserting the acknowledge line.
- The master then drops the write signal and the memory module drops the acknowledge signal.

Comparison

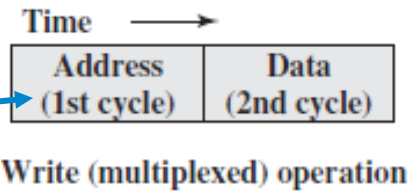
- Synchronous timing is simpler to implement and test. However, it is less flexible than asynchronous timing. Because all devices on a synchronous bus are tied to a fixed clock rate, the system cannot take advantage of advances in device performance.
- With asynchronous timing, a mixture of slow and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

BUS WIDTH

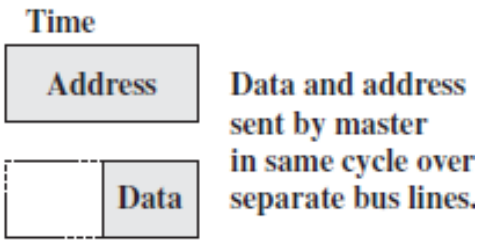
- The width of the data bus has an impact on system performance: The wider the data bus, the greater the number of bits transferred at one time.
- The width of the address bus has an impact on system capacity: the wider the address bus, the greater the range of locations that can be referenced.

Data Transfer Types

Bus is used for specifying the address and then for transferring the data



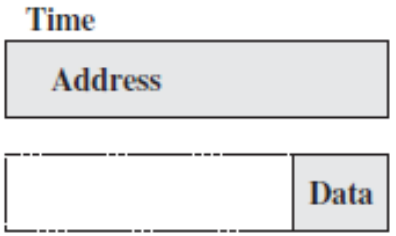
Write (multiplexed) operation



Write (non-multiplexed) operation

For read operation, there is a wait while the data are being fetched

Read (multiplexed) operation

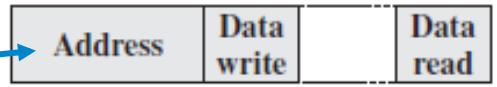


Read (non-multiplexed) operation

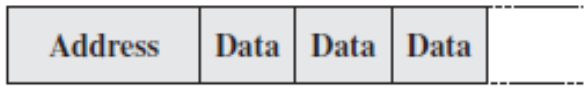
Simply a read followed immediately by a write to the same address.

Read-modify-write operation

Write followed immediately by a read from the same address.



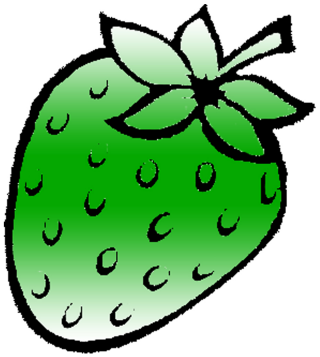
Read-after-write operation



Block data transfer

Figure Bus Data Transfer Types

STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com