



## Chapter 2

# TOP LEVEL VIEW OF COMPUTER FUNCTION



## Contents

- Understand the basic elements of an instruction cycle and the role of interrupts.
- Describe the concept of interconnection within a computer system (buses)
- Describe the concept of interconnection within a computer system.
- Understand the difference between synchronous and asynchronous bus timing.
- Explain the need for multiple buses arranged in a hierarchy.
- Assess the relative advantages of point-to-point interconnection compared to bus interconnection.
- Present an overview of QPI, PCIe



- Present an overview of the main characteristics of computer memory systems and the use of a memory hierarchy.
- Describe the basic concepts and intent of cache memory, key elements of cache design.
- Distinguish between direct mapping and associative mapping.
- Present an overview of the types of semiconductor main memory, internal and external memory.



## Computer Components

- Contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton
- Referred to as the *von Neumann architecture* and is based on three key concepts:
  - 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
- Hardwired program
  - The result of the process of connecting the various components in the desired configuration

## Hardware and Software Approaches

- Computer consist of 2 part : hardware and 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

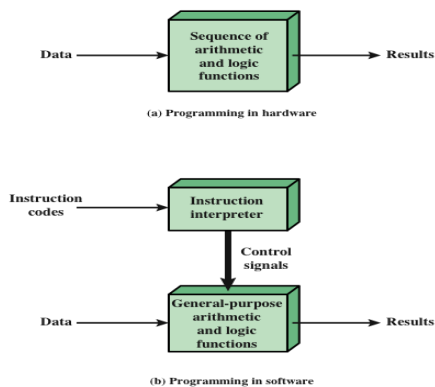


Figure 3.1 Hardware and Software Approaches

- 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
  - Also need temp storage (memory) and way to get input and output



## ■ Software

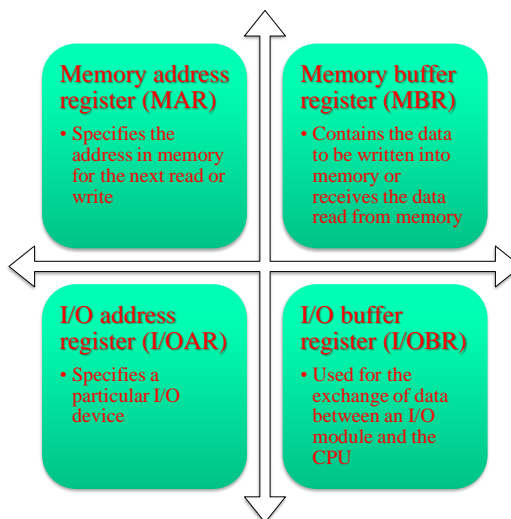
- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

## ■ Major components:

- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module : Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module : Means of reporting results

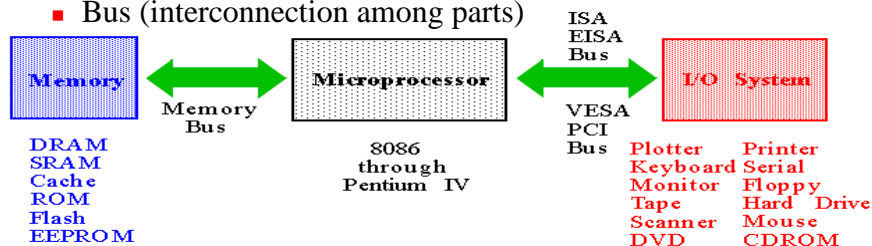


## ■ Memory



## Computer system

- At the most basic level, a computer is a device consisting of four parts:
  - A processor to interpret and execute programs
  - A memory to store both data and programs
  - A mechanism for transferring data to and from the outside world.
  - Bus (interconnection among parts)



## Computer Components: Top Level View

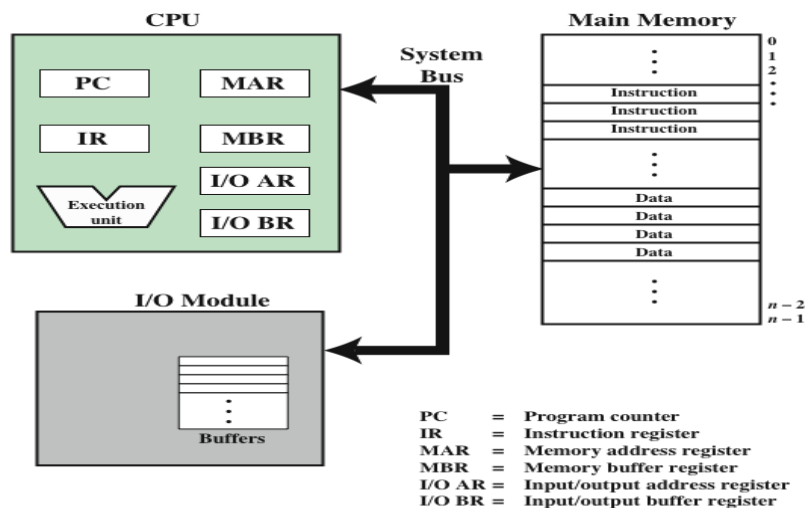


Figure 3.2 Computer Components: Top-Level View




## Function of some registers


- Program Counter (PC) : contain address of next instruction
- Instruction Register (IR) : contain instruction code
- Memory Address Register (MAR) - usually connected directly to address lines of bus
- Memory Buffer Register (MBR) - usually connected directly to data lines of bus



- Program Status Word (PSW) - also essential, common fields or flags contained include:
  - Sign - sign bit of last arithmetic operation
  - Zero - set when result of last arithmetic operation is 0
  - Carry - set if last op resulted in a carry into or borrow out of a high-order bit
  - Equal - set if a logical compare result is equality
  - Overflow - set when last arithmetic operation caused overflow
  - Interrupt Enable/Disable - used to enable or disable interrupts
  - Supervisor - indicates if privileged ops can be used

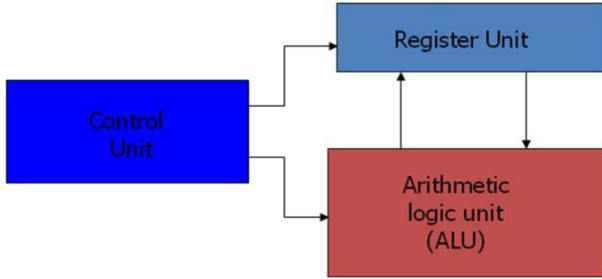


- Other optional registers
  - Pointer to a block of memory containing additional status info (like process control blocks)
  - An interrupt vector
  - A system stack pointer
  - A page table pointer
  - I/O registers



### **CPU (Central Processing Unit)**

- Performs data processing operations
- Consist of 4 main part :
  - Control unit
  - Register file
  - ALU (Arithmetic and logic unit)
  - Internal Bus



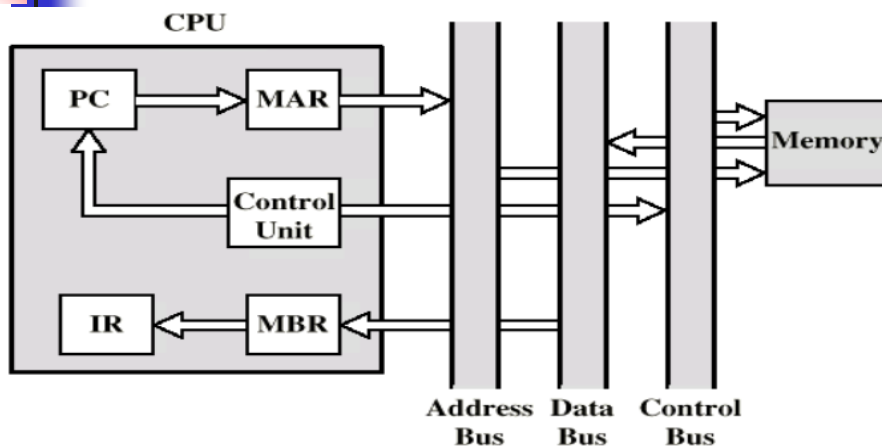
```
graph LR; CU[Control Unit] --- RU[Register Unit]; CU --- ALU[Arithmetic logic unit (ALU)]; RU <--> ALU
```



- 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



### Example of registers in CPU



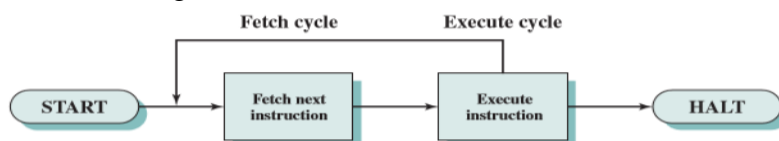
MBR = Memory buffer register  
 MAR = Memory address register  
 IR = Instruction register  
 PC = Program counter



- PC contains address of next instruction to be fetched
- This address is moved to MAR and placed on address bus
- Control unit requests a memory read
- Result is
  - placed on data bus
  - result copied to MBR
  - then moved to IR
- Meanwhile, PC is incremented

## Instruction Cycle

- INSTRUCTION FETCH & EXECUTE
  - The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
  - 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 fetched instruction is loaded into a register in the instruction register (IR).





## Fetch Cycle

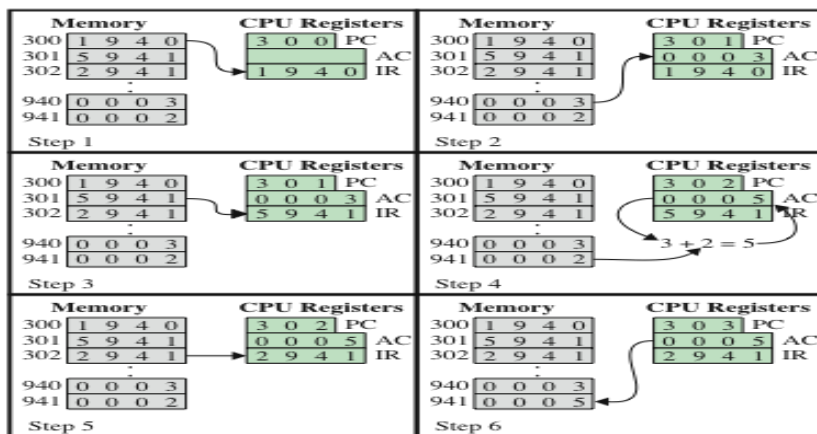
- At the beginning of each instruction cycle the processor fetches an instruction from memory
- The program counter (PC) holds the address of the instruction to be fetched next
- Processor fetches instruction from memory location pointed to by PC
- The processor increments the PC so that it will fetch the next instruction in sequence
- The fetched instruction is loaded into the instruction register (IR)
- The processor interprets the instruction and performs the required action



## Execute Cycle

- Four categories of actions
  - 1. Processor-memory : data transfer between CPU and main memory
  - 2. Processor I/O: Data transfer between CPU and I/O module
  - 3. Data processing: Some arithmetic or logical operation on data
  - 4. Control: Alteration of sequence of operations e.g. jump
- Instruction execution may involve a combination of these

## Example of program execution



**Figure 3.5 Example of Program Execution**  
(contents of memory and registers in hexadecimal)

## Explain

- The PC contains 300, the address of the first instruction. The instruction (the value 1940 in hex) is loaded into IR and PC is incremented. This process involves the use of MAR and MBR.
- The first hexadecimal digit in IR indicates that the AC is to be loaded. The remaining three hexadecimal digits specify the address (940) from which data are to be loaded.
- The next instruction (5941) is fetched from location 301 and PC is incremented.



- The old contents of AC and the contents of location 941 are added and the result is stored in the AC.
- The next instruction (2941) is fetched from location 302 and the PC is incremented
- The contents of the AC are stored in location 941.



## Instruction Cycle State Diagram

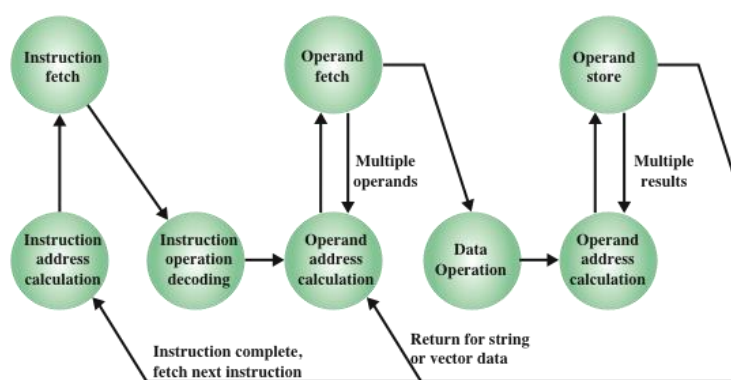


Figure 3.6 Instruction Cycle State Diagram



## Interrupt

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



## Classes of Interrupts

<b>Program</b>	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
<b>Timer</b>	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
<b>I/O</b>	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
<b>Hardware failure</b>	Generated by a failure such as power failure or memory parity error.



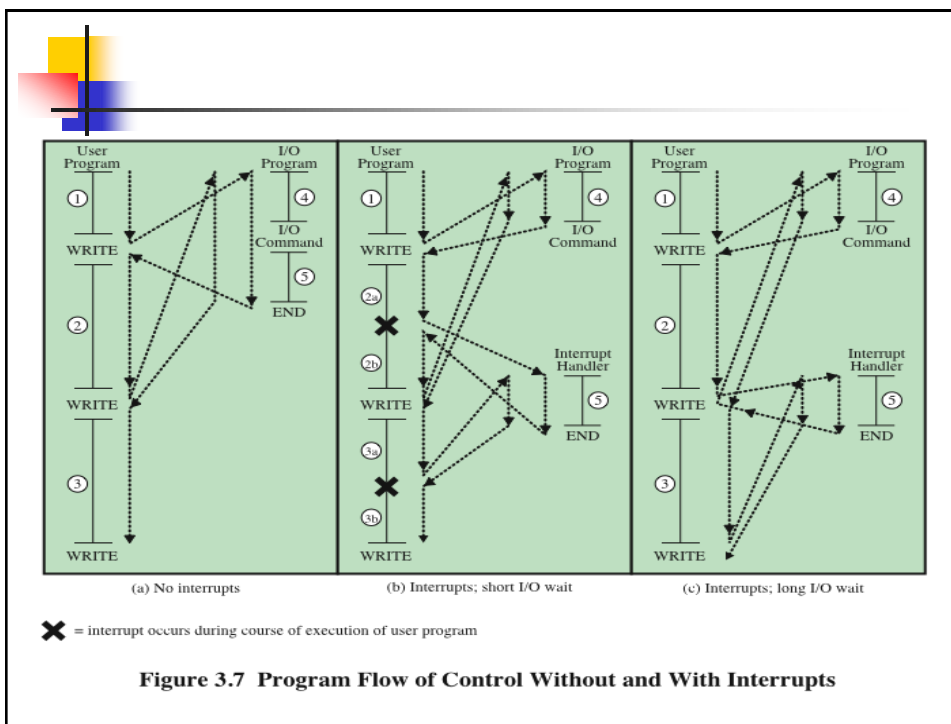
## Software Interrupts

- Some processors support “Software Interrupts”
  - In particular, both the Intel x86 family that we will use for assembler and the ARM family use them extensively
- Software interrupts are not really interrupts at all.
- A software interrupt is a machine instruction that causes a transfer of control through the same mechanism used by true interrupts
- Typically used for low-level calls to the operating system or components such as device drivers



## Why use interrupts?

- I/O Interrupts are used to improve CPU utilization
- Most I/O devices are relatively slow compared to the CPU
- Human interface devices and printers are especially slow
  - Keyboard: at best still fewer than 10 keystrokes per second
  - Printer: sending a single byte with the value 12 decimal causes a page eject (several seconds)



## Processing without interrupts

- Fig above a has three code segments (1,2,3) that do not perform I/O
- WRITE calls the OS to perform an I/O Write
  - Code sequence 4 prepares for the I/O transfer (check device status, copy data to buffer, etc.)
  - OS issues I/O command after seq 4.
  - OS then has to wait and poll device status until I/O completes
  - Code seq 5 is post I/O processing; e.g., set status flag
- The user's program is suspended until I/O completes



## Processing with interrupts

- Fig 3.7b shows processing with interrupts
- The WRITE call again transfers control to OS
- After write preparation in block 4, control returns to user program
  - I/O proceeds concurrently with user program
- When I/O completes, device issues an interrupt request
- OS interrupts user program (marked with \*) and executes post I/O code in block 5



## Interrupt processing

- When the external device needs to be serviced—the I/O module for that 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 an **interrupt handler**, and resuming the original execution after the device is serviced.





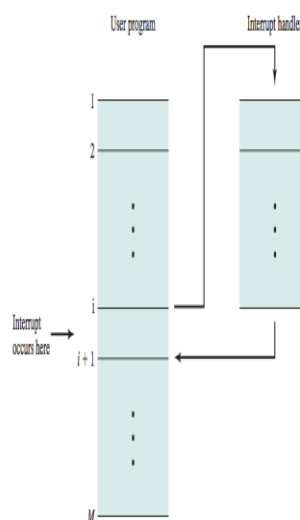
## Interrupt Cycle

- 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



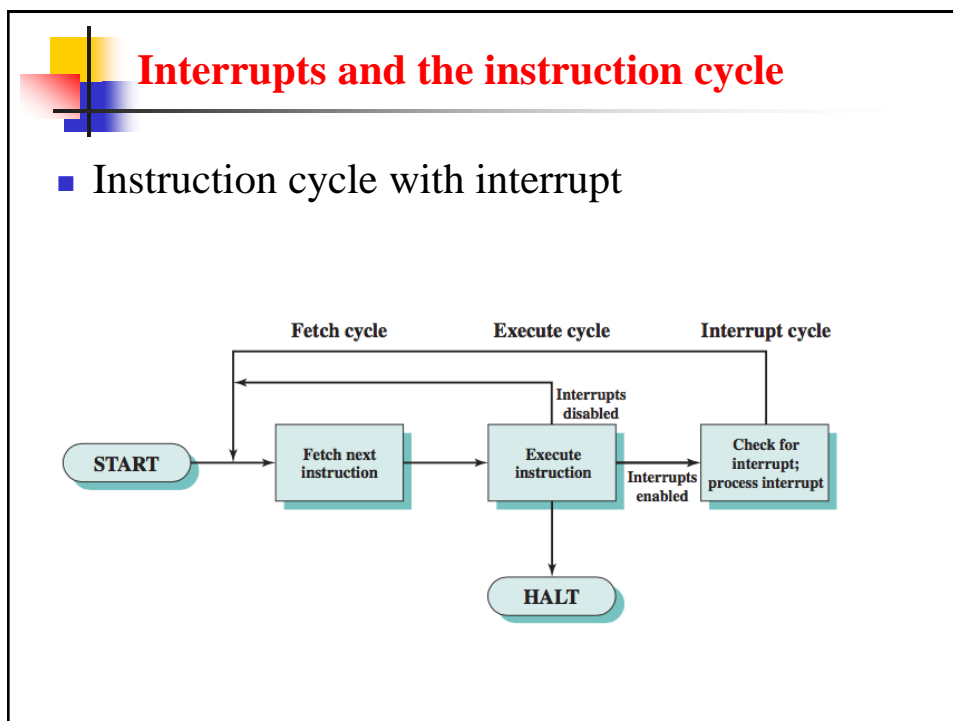
## Transfer of Control via Interrupts

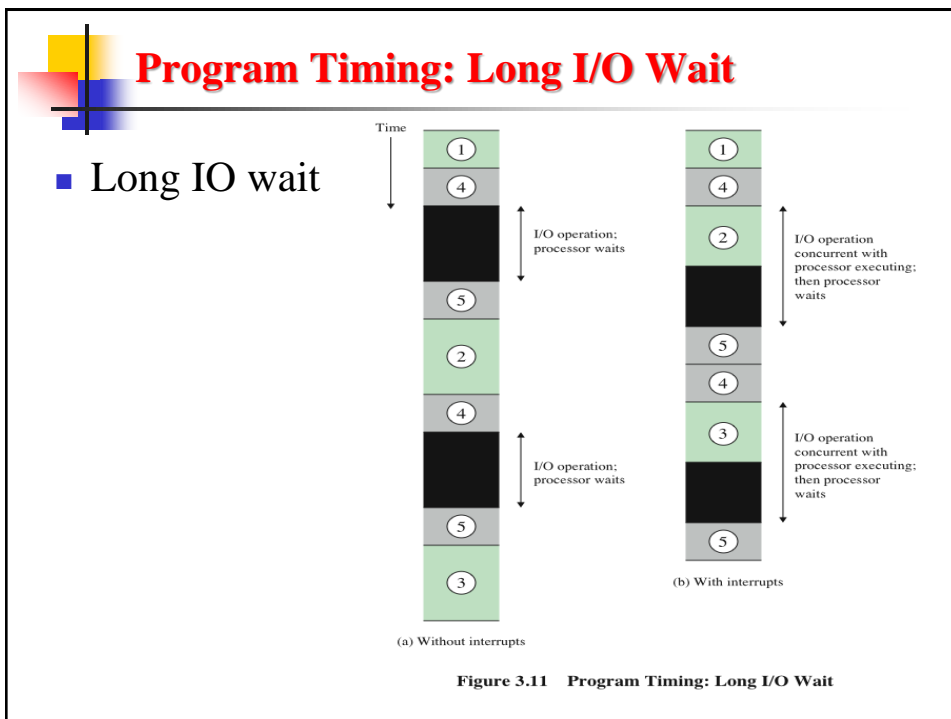
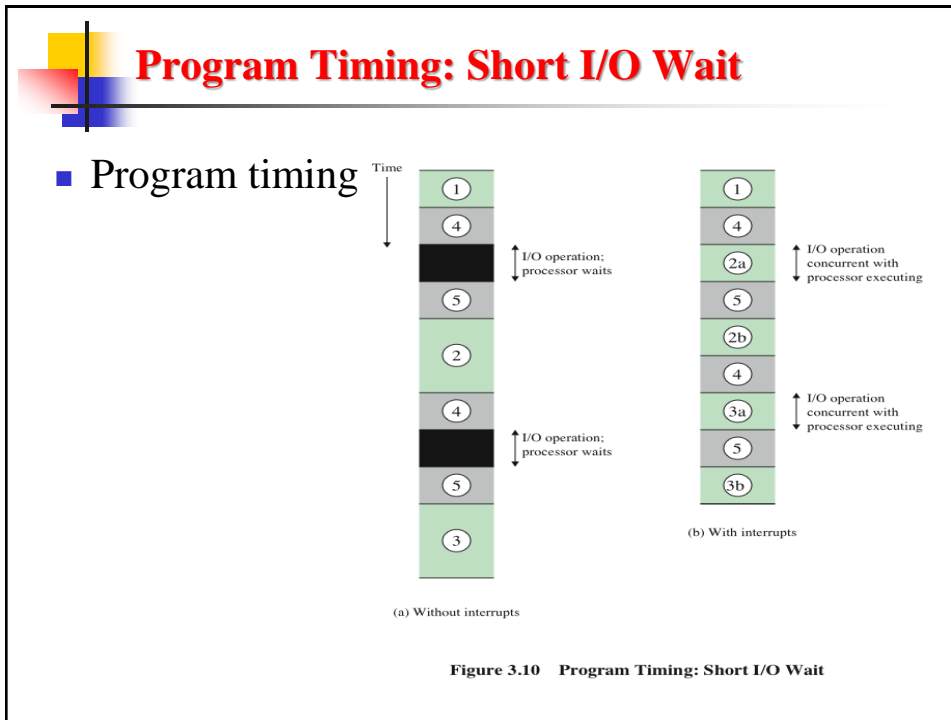
- From the point of view of the user program, an interrupt is just an interruption of the normal sequence of execution.
- When the interrupt processing is completed, execution resumes (Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the operating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

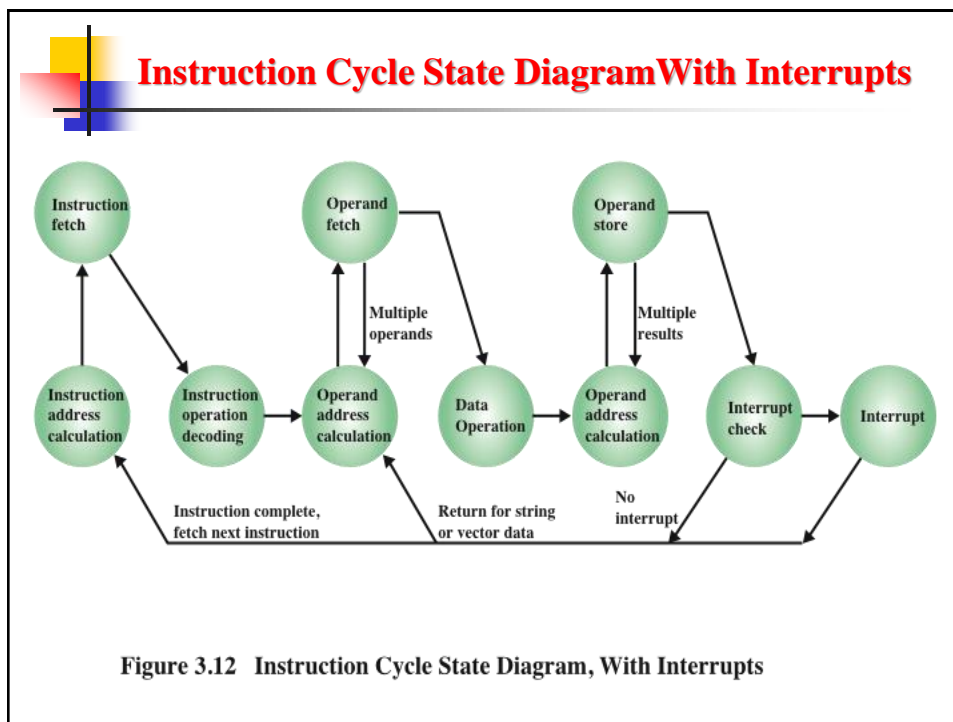


## Interrupts and the instruction cycle

- An *interrupt cycle* is added to the instruction cycle in which 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.
- If an interrupt is pending, the processor:
  - suspends execution of the current program being executed and saves the address of the next instruction to be executed and any other data relevant to the processor's current activity.
  - Sets the program counter to the starting address of an *interrupt handler* routine.

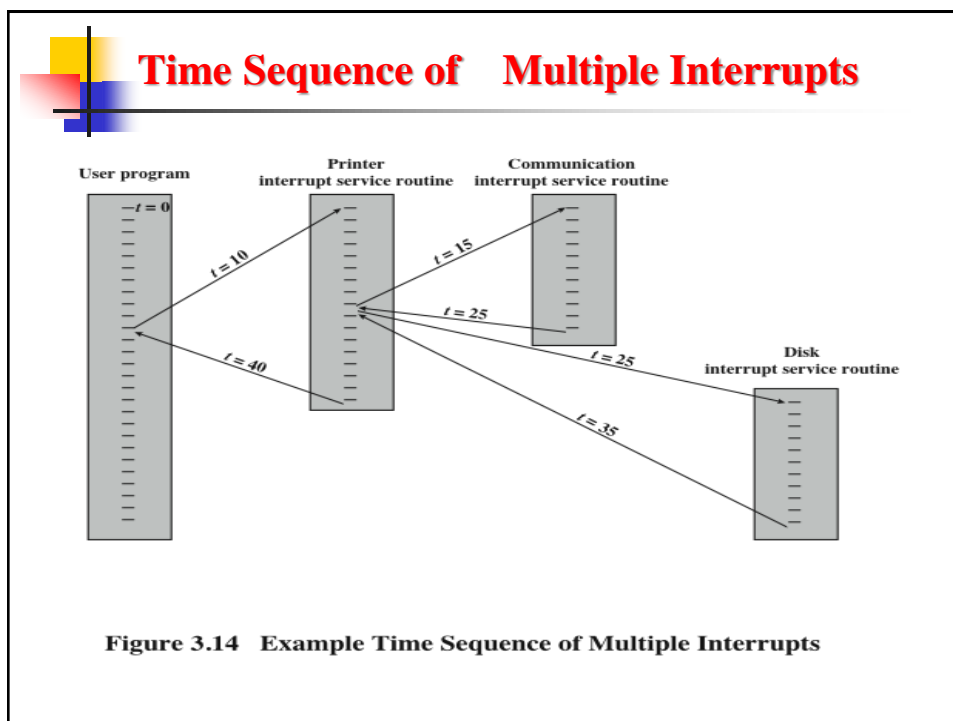
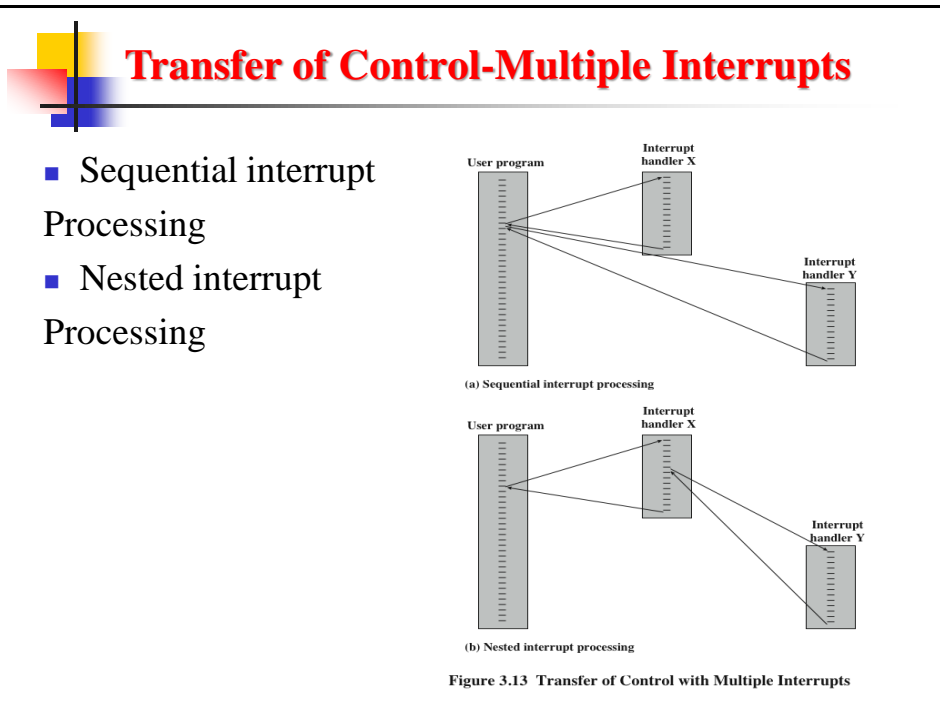






### Multiple Interrupts

- Two strategies for handling multiple interrupts:
  - 1. Disable interrupts
    - Processor will ignore further interrupts while processing one interrupt
    - Interrupts remain pending and are checked after first interrupt has been processed
    - Interrupts handled in sequence as they occur
  - 2. Define priorities
    - Low priority interrupts can be interrupted by higher priority interrupts
    - When higher priority interrupt has been processed, processor returns to previous interrupt





## Interrupt vector table

- INT and INT3 behave in a similar way.
  - INT n:
    - Calls ISR located at vector n ( $n*4$ ).
    - The INT instruction requires two bytes of memory, opcode plus n.
- Is a table contain 256 address pointer



## Example of interrupt vector table

080H	32-255 User defined	
	14-31 Reserved	
040H	Coprocessor error	16
03CH	Unassigned	15
038H	Page fault	14
034H	General protection	13
030H	Stack seg overrun	12
02CH	Segment not present	11
028H	Invalid task state seg	10
024H	Coproc seg overrun	9
020H	Double fault	8
01CH	Coprocessor not avail	7
018H	Undefined Opcode	6
014H	Bound	5
010H	Overflow (INTO)	4
00CH	1-byte breakpoint	3
008H	NMI pin	2
004H	Single-step	1
000H	Divide error	0

The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H through 0003FFH.

There are 256 4-byte entries (segment and offset in real mode).

Seg high	Seg low	Offset high	Offset low
Byte 3	Byte 2	Byte 1	Byte 0

## Exception Table

- A jump table for exceptions (also called *Interrupt Vector Table*)
  - Each type of event has a unique exception number  $k$
  - $k$  = index into exception table (a.k.a interrupt vector)
  - Handler  $k$  is called each time exception  $k$  occurs

The diagram illustrates the structure of an Exception Table. It is a vertical array of boxes, each representing an entry in the table. The entries are indexed from 0 to  $n-1$ . Each entry contains a dot, representing a pointer to the corresponding exception handler code. Arrows point from each entry to a yellow box labeled 'code for exception handler  $k$ ', where  $k$  is the index. For example, entry 0 points to 'code for exception handler 0', entry 1 points to 'code for exception handler 1', entry 2 points to 'code for exception handler 2', and entry  $n-1$  points to 'code for exception handler  $n-1$ '. The label 'Exception Table' is placed above the array, and 'Exception numbers' is placed below the indices.

## Exception Table (Excerpt)

- Example of interrupt vector table

Exception Number	Description	Exception Class
0	Divide error	Fault
13	General protection fault	Fault
14	Page fault	Fault
18	Machine check	Abort
32-255	OS-defined	Interrupt or trap



## I/O Function

- An I/O device (e.g., disk controller) can exchange data directly with the processor
- Just as the processor can read data from memory and write data to memory, it can also read data from I/O devices and write data to I/O devices



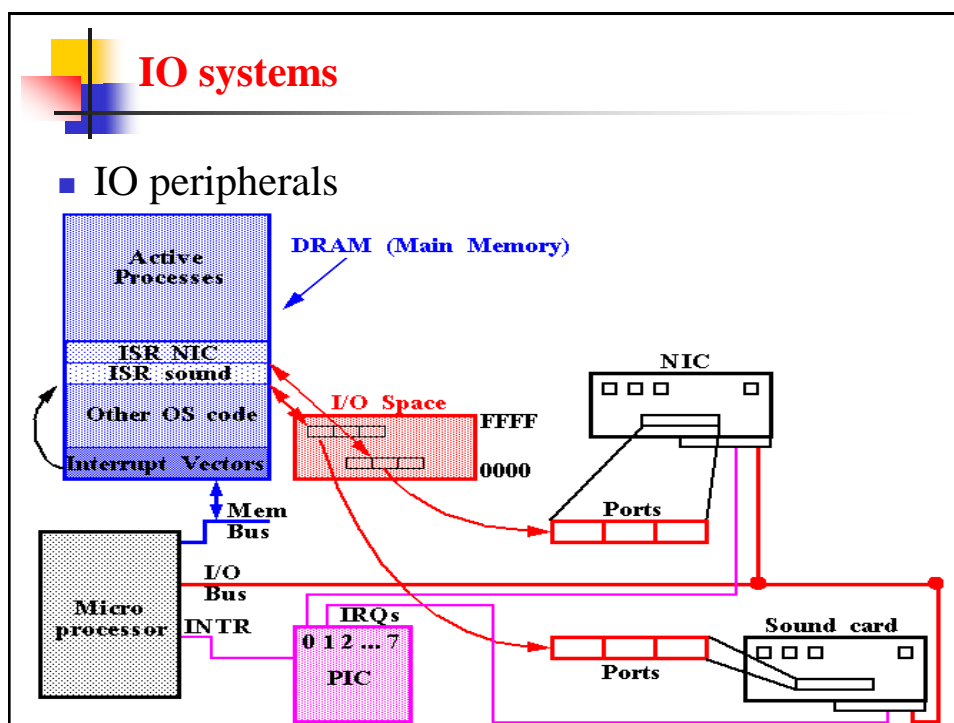
## Direct Memory Access (DMA)


- In some cases it may be desirable to allow I/O devices to exchange data directly with memory
- The processor will “grant permission” for this exchange to take place
- Processor can then proceed to other work (provided that it does not use the bus granted to the I/O device)
- This operation is called Direct Memory Access (DMA)




## I/O Module

- I/O is functionally similar to memory, but usually much slower
- Like memory can read and write, but a single I/O module may handle more than one device
- Each interface of an I/O device is referred to as a *port* and given a unique address
- I/O devices also have external connections
- Ports numbered 0 to M-1 for M ports
- Think of port as an address in I/O space
- I/O devices can also generate interrupts



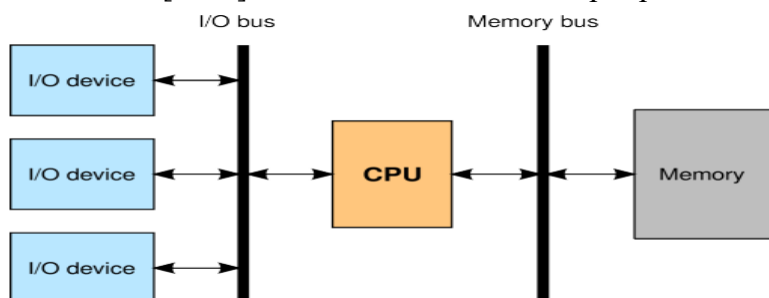


	Controlled by	Direction	Data traffic
Keyboard	Human	Input	ca. 100 byte/s
Mouse	Human	Input	ca. 200 byte/s
Sound device	Human	Output	ca. 96 kB/s
Printer	Human	Output	ca. 200 kB/s
Graphics card	Human	Output	ca. 500 MB/s
Modem	Machine	In/Out	2-8 kB/s
Ethernet network interface	Machine	In/Out	ca. 12.5 MB/s
Disk (HDD)	Machine	In/Out	ca. 50 MB/s
GPS	Machine	Input	ca. 100 byte/s

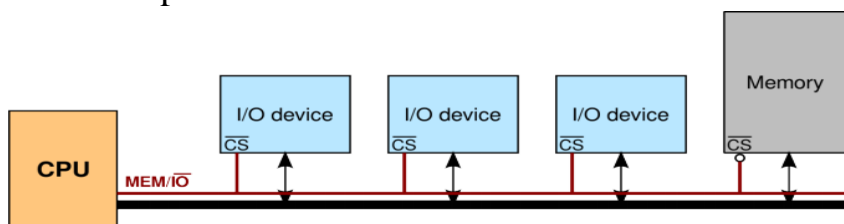
- 
- Questions to investigate:
    - **How does the CPU communicate with I/O devices?**
    - How do I/O devices communicate with the CPU?
    - How to transmit data efficiently, without errors?
    - How to connect the I/O devices to the CPU?
  - IO addressing
    - According to address separation there are two possibilities:
      - **Separate** I/O and Memory address space
      - **Shared** I/O and Memory address space

## Separate memory and IO addresses

- Two separate address spaces:
  - x86: memory: 0 – 4GB, I/O: 0 – 64kB
- Separate instructions for I/O and memory operations
  - $R0 \leftarrow \text{MEM}[0x60]$ : data movement from memory
  - $R0 \leftarrow \text{IO}[0x60]$ : data movement from I/O peripheral

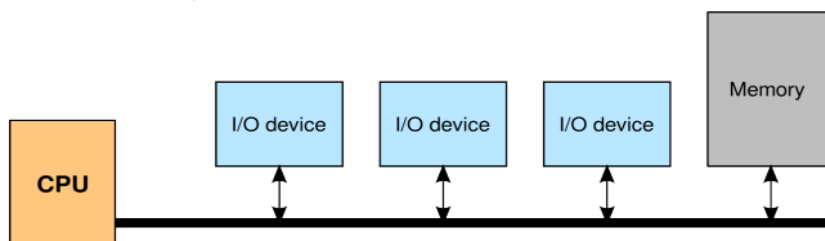


- Alternative implementation
  - The CPU has a shared bus for the I/O and the memory
  - A selector signal determines the target of the communication
  - More cost effective (less wires)
  - Example: x86

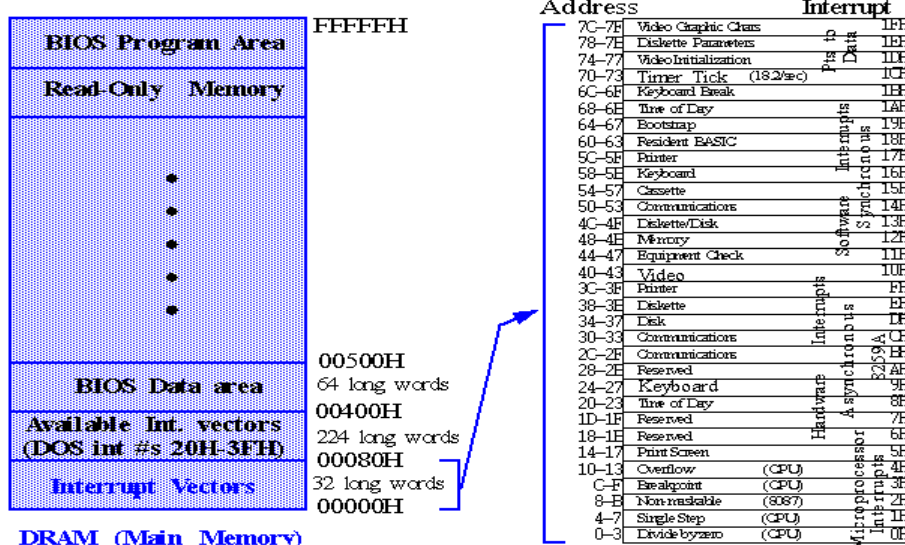


## Memory mapped IO

- The CPU has a single address space
- There are special memory addresses reserved for I/O communication
- Memory read/write requests from/to these addresses are answered by I/O devices



## IO Address space on x86

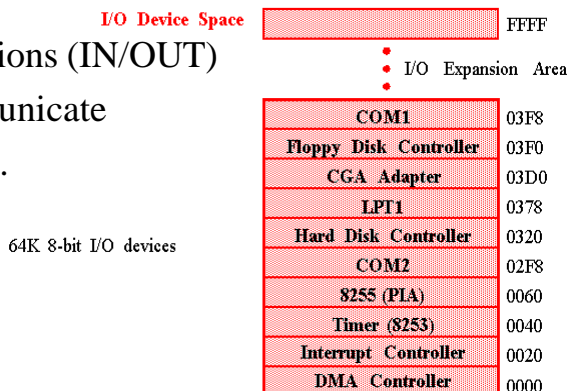


## Example IO address space on 8086

### I/O Space

- It is important to notice that these I/O addresses are *NOT* memory-mapped addresses on the 80x86 machines.

- Special instructions (IN/OUT) are used to communicate to the I/O devices.



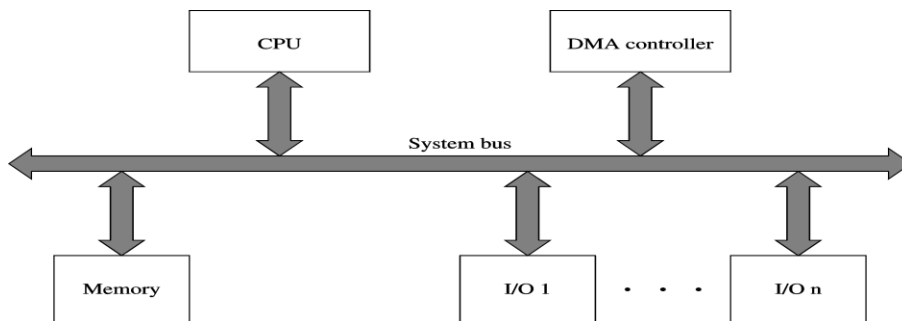
## Memory

- Memory consists of  $n$  words of equal length numbered from 0 to  $n-1$
- A word of data can be read or written
- Control signals specify R/W operation at location specified by address
- Needs three sets of signal lines:
  - Data
  - Address
  - Control (R/W and timing)

## Interconnection Structures

### ■ Computer modules

- Computer is a network of basic modules.
- There must be paths for connecting the modules.
- The collection of paths connecting the various modules is called the interconnection structure. The design of this structure will depend on the exchanges that must be made among modules.



## Computer Modules

### ■ Memory modul :

- A memory module will consist of  $N$  words of equal length.
- Each word is assigned a unique numerical address ( $0, 1, \dots, N - 1$ )
- A word of data can be **read** from or **written** into the memory.

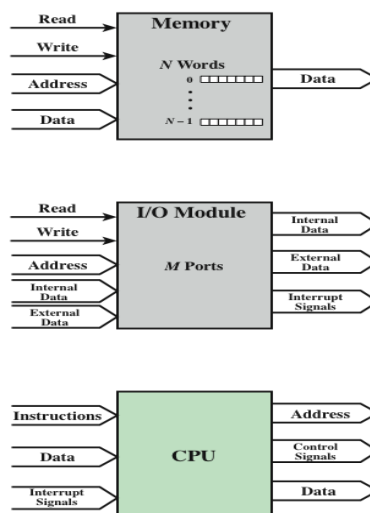




Figure 3.15 Computer Modules



- IO modul :
  - Function similar to memory.
  - An I/O module may send interrupt signals to the processor.
- Processor
  - The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system.
  - The processor also receives interrupt signals.



## Types of transfers

- The interconnection structure must support the following types of transfers
  - Memory  $\leftrightarrow$  processor: The processor reads/writes an instruction or a unit of data from/to memory.
  - I/O  $\leftrightarrow$  processor: The processor reads/sends data from/to an I/O device via an I/O module.
  - I/O to or from memory: an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access.



## Bus interconnection

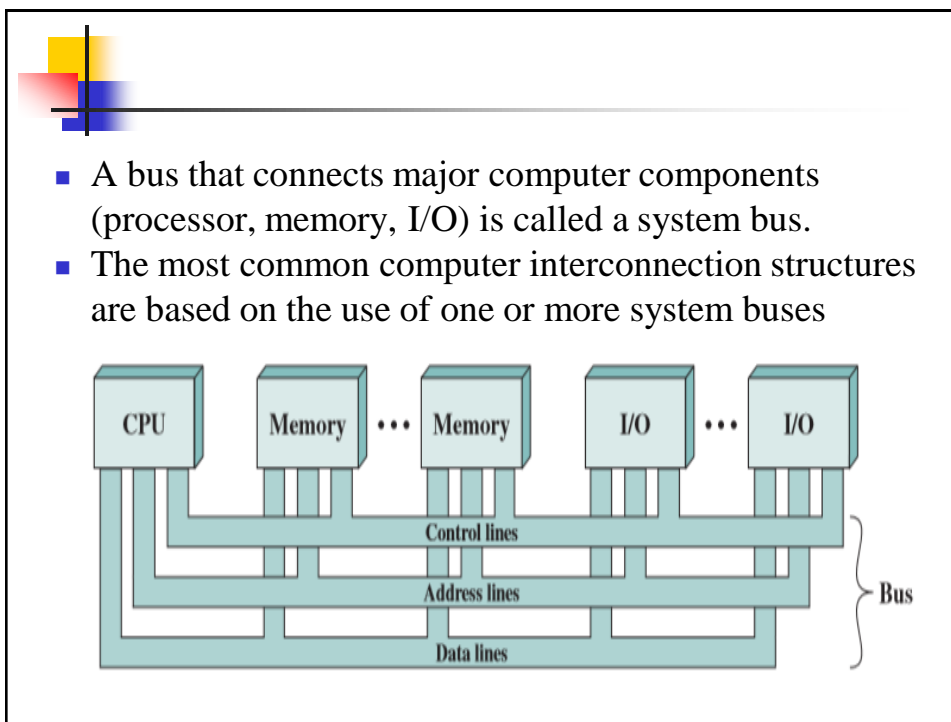
---

- A bus is a communication pathway connecting two or more devices.
- Key characteristic of a bus: 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**



- A bus consists of multiple lines. Each line is capable of transmitting signals representing binary 1 and binary 0. Several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.
- Computer systems contain a number of **different buses** that provide pathways between components.





## System Bus

- Computers normally contain several buses
- The bus that interconnects major components (processor, memory, I/O devices) is called the system bus
- A system bus typically contains from 50 to several hundred lines
- Lines are grouped
  - Major groupings are data, address and control signals
  - Power lines may not be shown in bus diagrams



- A communication pathway connecting two or more devices
  - Key characteristic is that it is a shared transmission medium
- Signals transmitted by any one device are 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
- Typically consists of multiple communication lines
  - Each line is capable of transmitting signals representing binary 1 and binary 0



- Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy
- *System bus*
  - A bus that connects major computer components (processor, memory, I/O)
- The most common computer interconnection structures are based on the use of one or more system buses



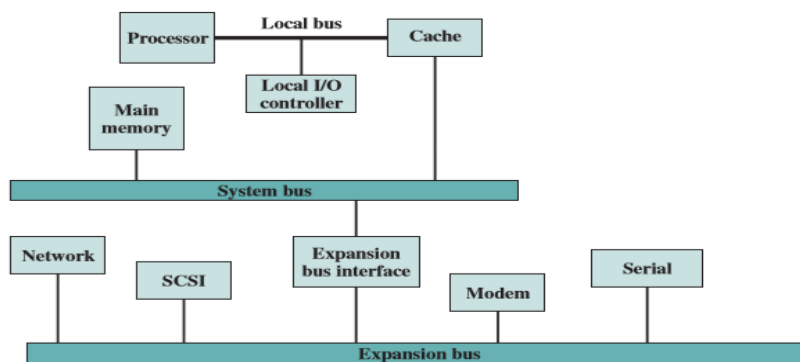
## Bus Structure

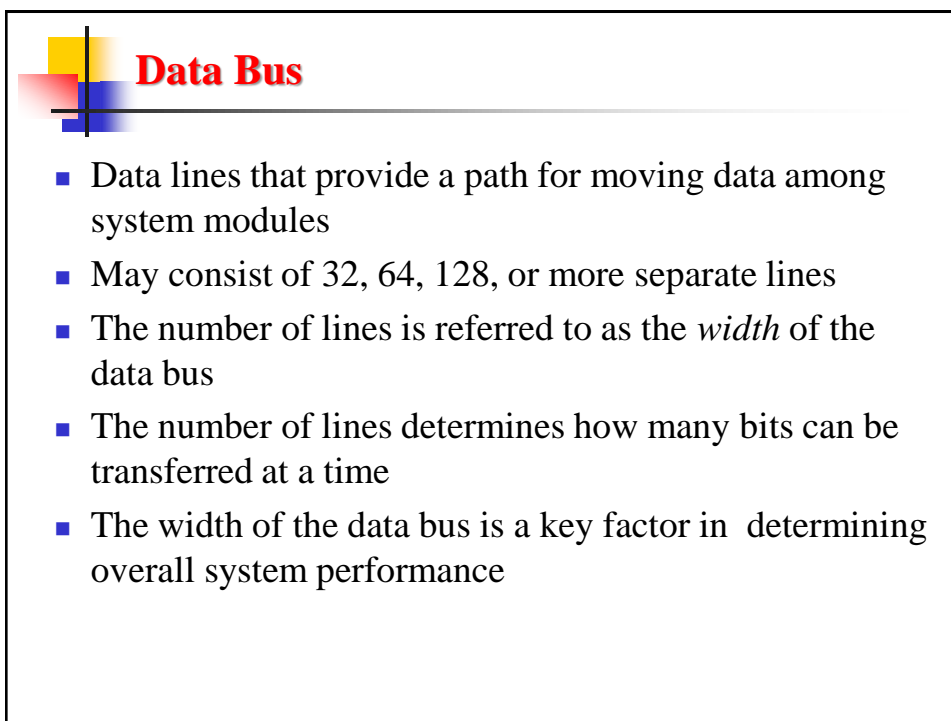
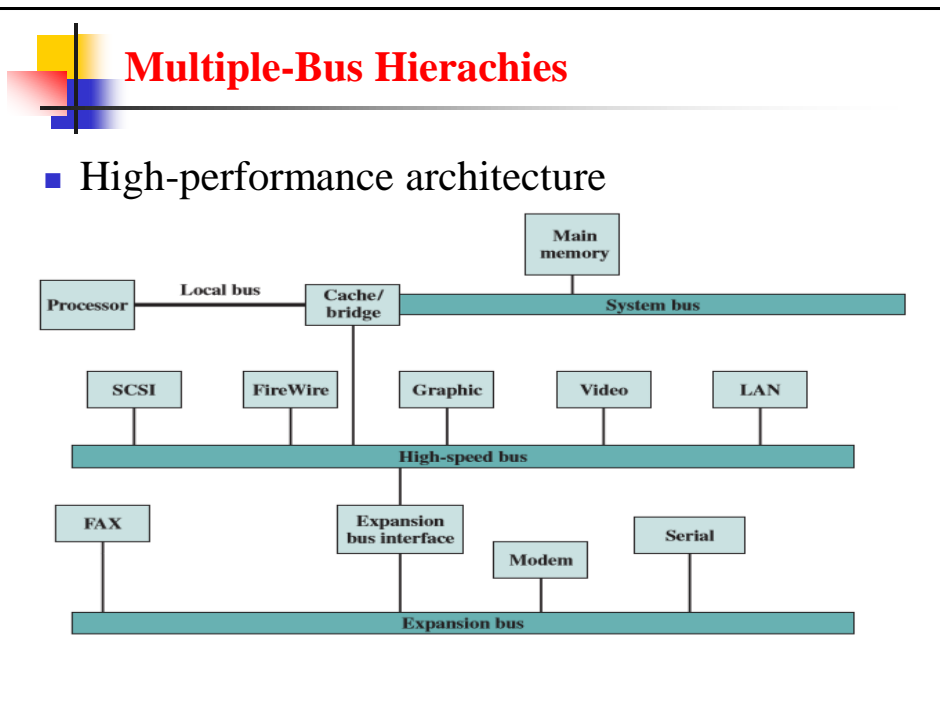
- A system bus consists of from about fifty to hundreds of separate lines and can be classified into three functional groups: data, address, and control lines:
  - Data bus: 32, 64, 128 lines (width → system performance)
  - Address bus: 8, 16, 32 lines (width → max memory capacity)
  - Control bus:
    - Memory read/write
    - I/O read/write
    - Transfer ACK
    - Bus request/grant
    - Interrupt request/ACK
    - Clock
    - Reset



## Multiple-Bus Hierachies

- A great number of devices connected to the bus will suffer system performance (bottleneck).








## Address Bus

- Used to designate the source or destination of the data on the data bus
  - If the processor wishes to read a word of data from memory it puts the address of the desired word on the address lines
- Width determines the maximum possible memory capacity of the system
- 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 within the module




## Control Bus

- Used to control the access and the use of the data and address lines
- Because the 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
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed




## Typical Control Signals

- Memory read/write signals
- I/O read/write signals
- Bus request/grant
- Transfer ACK (acknowledgement)
  - Indicates that have been accepted from or placed on bus
- Interrupt Request/ACK
- Clock signals synchronize operations
- Reset: initializes all modules




## Basic Bus Operation

- Module that wishes to send data must
  - Obtain use of the bus
  - Then transfer data
- Module that requests data from another module must
  - Obtain use of the bus
  - Transfer request to other module over bus
  - Wait for data to be written to the bus




- What do buses look like?
  - Parallel lines on circuit boards
  - Ribbon cables
  - Strip connectors on mother boards : e.g. PCI
  - Sets of wires
- With VLSI, many components (such as L1 cache) are on the same chip as the processor
- An on-chip bus connects these components



### Single Bus Problems


- Lots of devices on one bus leads to:
  - Propagation delays
    - Long data paths mean that co-ordination of bus use can adversely affect performance
  - Bottlenecks when aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems
  - Hierarchical structure
  - High-speed limited access buses close to the processor
  - Slower-speed general access buses farther away from the processor



## Bus can be a bottleneck

---

- Can increase data rates and bus width, but peripheral data rates are increasing rapidly
  - Video and graphics controllers
  - Network interfaces (1GB ethernet)
  - High speed storage devices




## Basic Elements of Bus Design

---

- These key elements serve to classify and differentiate buses

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






## Bus Types

---

- Dedicated (functional)
  - Separate data & address lines
- Multiplexed (Time multiplexing)
  - Shared lines
  - Address valid or data valid control line
  - Advantage - fewer lines
  - Disadvantages
  - More complex control
  - Performance – cannot have address and data simultaneously on bus
- Dedicated (physical)
  - Bus connects subset of modules
  - Example: all I/O devices on a slow bus
  - Provides high throughput, but cost and complexity increase



## Bus Arbitration

---

- Because only one unit at a time can successfully transmit over the bus, some method of **arbitration** is needed.
- Two types of arbitration: **centralized** and **distributed**.
- Centralized scheme: a *bus controller* or *arbiter*, is responsible for allocating time on the bus.
- Distributed scheme: No central controller, each module contains access control logic and the modules act together to share the bus.
- The device which initiates data transfer is called the **master**, while the other device involved in the data exchange is called the **slave**.



## Timing

- Co-ordination 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

- Synchronous diagram

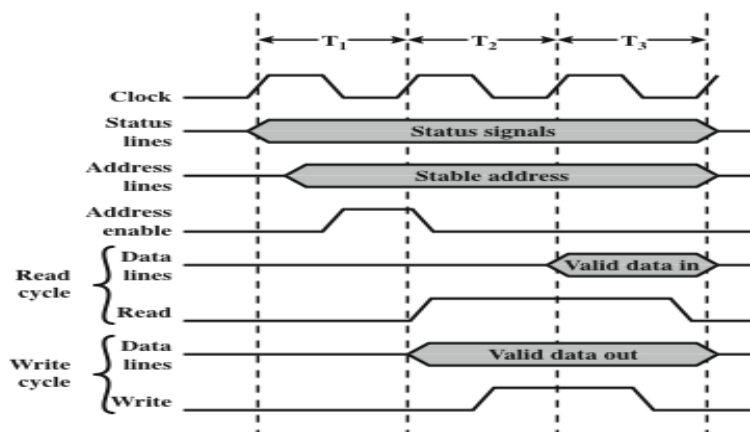


Figure 3.18 Timing of Synchronous Bus Operations



## Asynchronous Timing

- Occurrence of one event on bus follows and depends on a previous event
  - ACK signals are used to signal end of event
- Synchronous timing easier to implement and test
  - But all devices are limited to fixed clock rate
  - Cannot take advantage of newer, faster devices
- Asynchronous timing allows mixture of slow and fast devices to share bus comfortably



## Timing of Asynchronous Bus Operations

- Asynchronous diagram

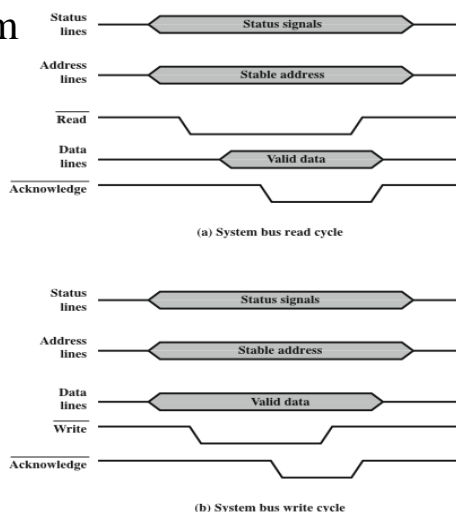


Figure 3.19 Timing of Asynchronous Bus Operations




## Bus Width

- Data width affects system performance
- Address width determines addressable memory
- Address: Width of address bus has an impact on system capacity i.e. wider bus means greater the range of locations that can be transferred.
- Data: width of data bus has an impact on system performance i.e. wider bus means number of bits transferred at one time.
- If bus has  $n$  bit width, CPU can manage  $2^n$  memory cells (location)




## Example of Bus

- *Address:*
  - *If I/O, a value between 0000H and FFFFH is issued.*
  - *If memory, it depends on the architecture:*
    - **20** -bits (8086/8088)
    - **24** -bits (80286/80386SX)
    - **25** -bits (80386SL/SLC/EX)
    - **32** -bits (80386DX/80486/Pentium)
    - **36** -bits (Pentium Pro/II/III)



---

- *Data:*
  - **8 -bits** (8088)
  - **16 -bits** (8086/80286/80386SX/SL/SLC/EX)
  - **32 -bits** (80386DX/80486/Pentium)
  - **64 -bits** (Pentium/Pro/II/III)
- *Control:*
  - *Most systems have at least 4 control bus connections (active low).*
  - **MRDC** (*Memory Read Control*), **MWRC** , **IORC** (*I/O Read Control*), **IOWC**



---