Digital Design and Computer Organization

Module 3:

Basic Structure of Computers: Basic Operational Concepts**,** Numbers, Arithmetic Operations and Characters. Memory Location and Addresses**.**

Input/Output Organization: Accessing I/O Devices, Interrupts, Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device Requests, Exceptions, Direct Memory Access, Buses, Standard I/O Interfaces – PCI Bus, USB.

Basic Structure of Computers:

Computer: Computer is a fast electronic calculating machine that accepts digitized input information, processing it according to a list of internally stored instructions and produces the resulting output information. The list of instructions is called as a Computer program and the internal storage is called as Computer memory.

Types of Languages: Just as humans use language to communicate, and different regions have different languages, computers also have their own languages that are specific to them.

- Different kinds of languages have been developed to perform different types of work on the computer.
- Basically, languages can be divided into two categories according to how the computer understands them.

i. **Low-Level Languages:** A language that corresponds directly to a specific machine. Low-level computer languages are either machine codes or are very close them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary.

- There are two types of low-level languages:
  - **Machine Language:** a language that is directly interpreted into the hardware. Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex. It is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.
    - **Advantages:**
      - ➢ Machine language makes fast and efficient use of the computer.
      - ➢ It requires no translator to translate the code. It is directly understood by the computer.
    - **Disadvantages:**
      - ➢ All operation codes have to be remembered
      - ➢ All memory addresses have to be remembered.
      - ➢ It is hard to amend or find errors in a program written in the machine language.

  - **Assembly Language:** A slightly more user-friendly language that directly corresponds to machine language. Assembly language was developed to overcome some of the many inconveniences of machine language. This is another low-level but very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and l's. These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.'

- **Advantages:**
  - ➢ Assembly language is easier to understand and use as compared to machine language.
  - ➢ It is easy to locate and correct errors.
  - ➢ It is easily modified.
- **Disadvantages:**
  - ➢ Like machine language, it is also machine dependent/specific.
  - ➢ Since it is machine dependent, the programmer also needs to understand the hardware.

ii. **High-Level Languages:** Any language that is independent of the machine. High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily, in their own native language environment (English).

- High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high-level language is translated into many machine language instructions that the computer can understand.

- **Advantages:**
  - ➢High-level languages are user-friendly
  - ➢ They are easier to learn.
  - ➢ They are easier to maintain
  - ➢ A program written in a high-level language can be translated into many machine languages and can run on any computer o programs developed in a high-level language can be run on any computer text
- **Disadvantages:**
  - ➢ A high-level language has to be translated into the machine language by a translator, which takes up time.

TYPES OF COMPUTERS

- **Desktop Computers**
  - These are most commonly used computers in home, schools and offices.
  - This has
    - → processing- & storage-units
    - → video & audio output-units
    - → Keyboard & mouse input-units

➢ **Notebook Computers (Laptops)**
- This is a compact version of a personal-computer (PC) made as a portable-unit.

➢ **Workstations**
- These have more computational-power than PC.

➢ **Enterprise Systems (Mainframes)**
- These are used for business data-processing.
- These have large computational-power and larger storage-capacity than workstations.
- These are referred to as
  - → server at low-end and
  - → Super-computers at high end.

➢ **Servers**
- These have large database storage-units and can also execute requests from other computers.
- These are used in banks & educational institutions.

➢ **Super Computers**
- These are used for very complex numerical-calculations.
- These are used in weather forecasting, aircraft design and military applications.

Basic Operational Concepts

- **BASIC CONCEPTS**

- **Computer Architecture (CA)** is concerned with the structure and behaviour of the computer.
- CA includes the information formats, the instruction set and techniques for addressing memory.
- In general covers, CA covers 3 aspects of computer-design namely: 1) Computer Hardware, 2) Instruction set Architecture and 3) Computer Organization.

1. Computer Hardware

➢ It consists of electronic circuits, displays, magnetic and optical storage media and communication facilities.

2. Instruction Set Architecture

➢ It is programmer visible machine interface such as instruction set, registers, memory organization and exception handling.

➢ Two main approaches are 1) CISC and 2) RISC.

(CISC-> Complex Instruction Set Computer, RISC-> Reduced Instruction Set Computer)

3. Computer Organization

➢ It includes the high-level aspects of a design, such as

→ memory-system
→ bus-structure &
→ design of the internal CPU.

➢ It refers to the operational units and their interconnections that realize the architectural specifications.

➢ It describes the function of and design of the various units of digital computer that store and process information.

FUNCTIONAL UNITS

- A computer consists of 5 functionally independent main parts:

1) Input
2) Memory
3) ALU
4) Output &
5) Control units.

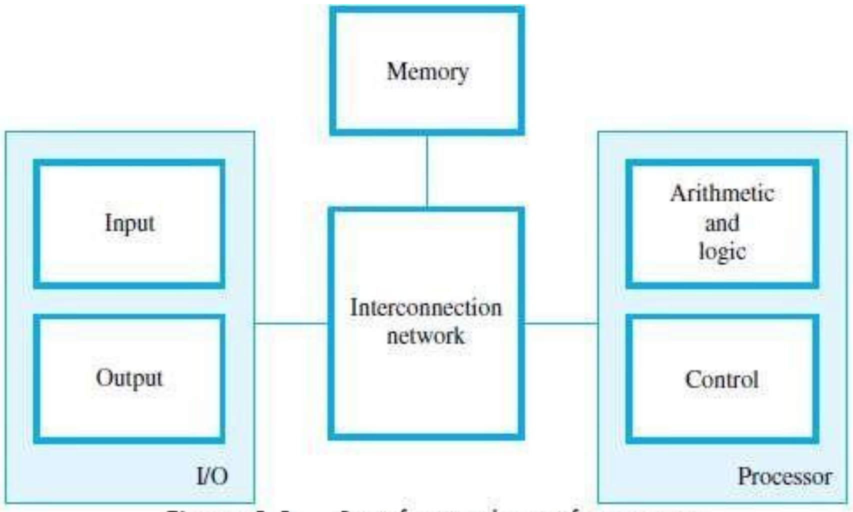


Figure 1.1 Basic functional units of a computer.

BASIC OPERATIONAL CONCEPTS

- An Instruction consists of 2 parts, 1) Operation code (Opcode) and 2) Operands.

OPCODE	OPERANDS

- The data/operands are stored in memory.
- The individual instruction are brought from the memory to the processor.
- Then, the processor performs the specified operation.
- Let us see a typical instruction

ADD LOCA, R0

- This instruction is an addition operation. The following are the steps to execute the instruction:

Step 1: Fetch the instruction from main-memory into the processor.
Step 2: Fetch the operand at location LOCA from main-memory into the processor.
Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0.
Step 4: Store the result (sum) in R0.

- The same instruction can be realized using 2 instructions as:
  Load LOCA, R1 Add R1, R0

- The following are the steps to execute the instruction:
  Step 1: Fetch the instruction from main-memory into the processor.
  Step 2: Fetch the operand at location LOCA from main-memory into the register R1.
  Step 3: Add the content of Register R1 and the contents of register R0.
  Step 4: Store the result (sum) in R0.

MAIN PARTS OF PROCESSOR

- The processor contains ALU, control-circuitry and many registers.
- The processor contains „n‟ general-purpose registers R0 through Rn-1.
- The IR holds the instruction that is currently being executed.
- The control-unit generates the timing-signals that determine when a given action is to take place.
- The PC contains the memory-address of the next-instruction to be fetched & executed.
- During the execution of an instruction, the contents of PC are updated to point to next instruction.
- The MAR holds the address of the memory-location to be accessed.
- The MDR contains the data to be written into or read out of the addressed location.
- MAR and MDR facilitates the communication with memory. (IR à Instruction-Register, PC à Program Counter)
  (MAR à Memory Address Register, MDRà Memory Data Register)

STEPS TO EXECUTE AN INSTRUCTION

1) The address of first instruction (to be executed) gets loaded into PC.
2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.
3) After certain amount of elapsed time, the first instruction is read out of memory and placed into MDR.
4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.
5) To fetch an operand, it's address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.
6) Likewise required number of operands is fetched into processor.
7) Finally, ALU performs the desired operation.
8) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.
9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.
10) At some point during execution, contents of PC are incremented to point to next instruction in the program.

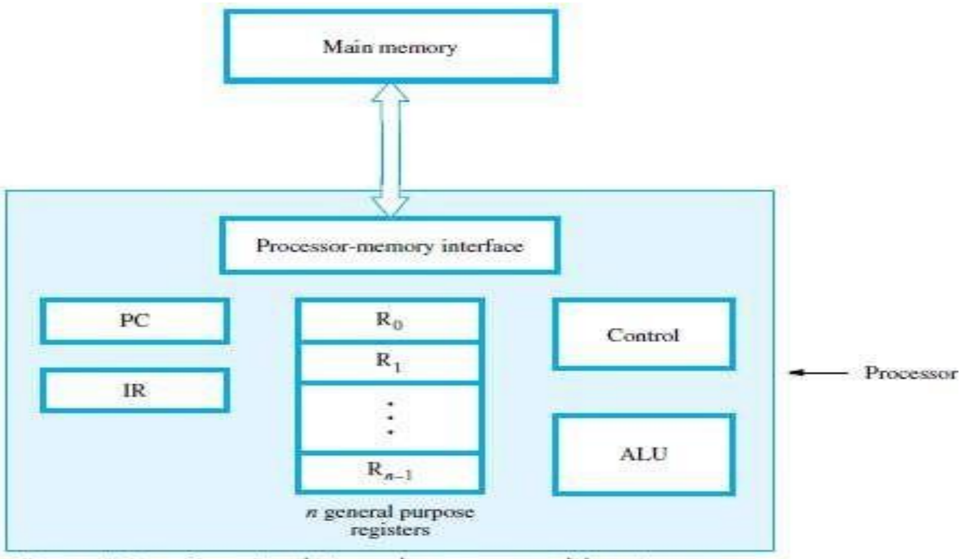


Figure 1.2 Connection between the processor and the main memory.

BUS STRUCTURE

- A bus is a group of lines that serves as a connecting path for several devices.
- A bus may be lines or wires.
- The lines carry data or address or control signal.
- There are 2 types of Bus structures: 1) Single Bus Structure and 2) Multiple Bus Structure.

1) Single Bus Structure

- ➢ Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time.
- ➢ Bus control lines are used to arbitrate multiple requests for use of the bus.
- ➢ **Advantages:**
  1) Low cost &
  2) Flexibility for attaching peripheral devices.

2) Multiple Bus Structure

- ➢ Systems that contain multiple buses achieve more concurrency in operations.
- ➢ Two or more transfers can be carried out at the same time.
- ➢ **Advantage:** Better performance.
- ➢ **Disadvantage:** Increased cost.

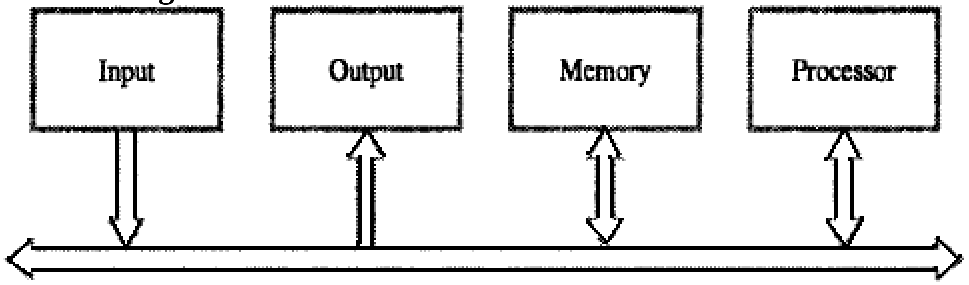


Figure 1.3 Single-bus structure.

- The devices connected to a bus vary widely in their speed of operation.
- To synchronize their operational-speed, buffer-registers can be used.
- **Buffer Registers**
  → are included with the devices to hold the information during transfers.
  → prevent a high-speed processor from being locked to a slow I/O device during data transfers.

PERFORMANCE

- The most important measure of performance of a computer is how quickly it can execute programs.
- The speed of a computer is affected by the design of
  1) Instruction-set.
  2) Hardware & the technology in which the hardware is implemented.
  3) Software including the operating system.
- Because programs are usually written in a HLL, performance is also affected by the compiler that translates programs into machine language. (HLLà High Level Language).
- For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

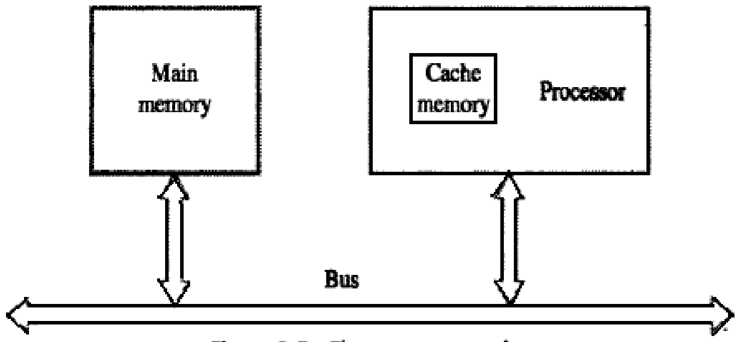


Figure 1.5 The processor cache.

examine the flow of program instructions and data between the memory & the processor.

- At the start of execution, all program instructions are stored in the main-memory.
- As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.
- Later, if the same instruction is needed a second time, it is read directly from the cache.
- A program will be executed faster

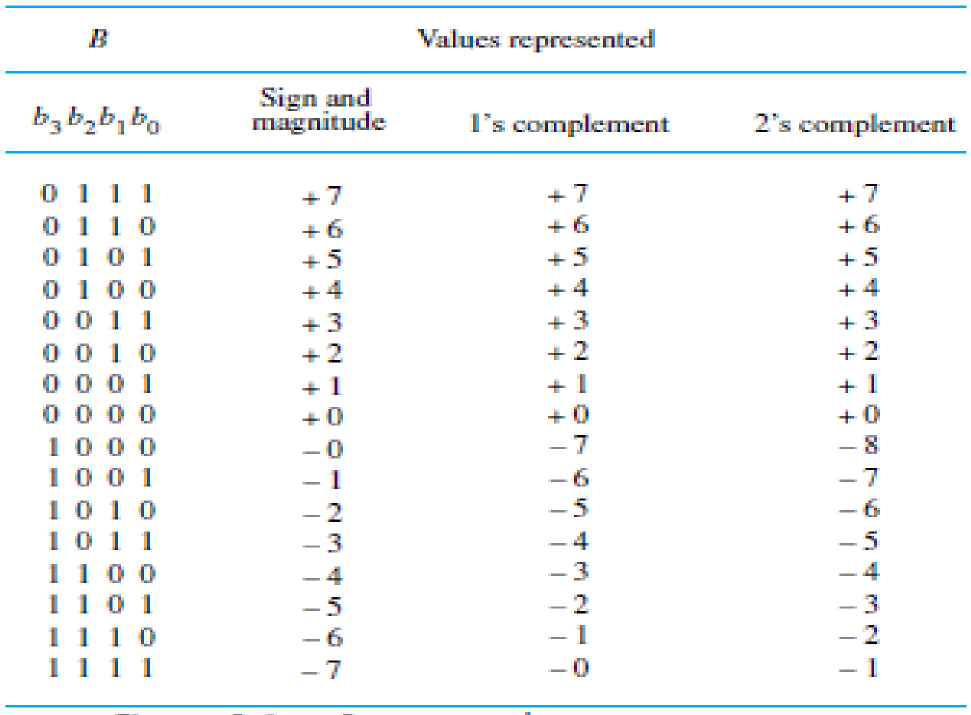
if movement of instruction/data between the main-memory and the processor is minimized which is achieved by using the cache.

NUMBER REPRESENTATION

- Numbers can be represented in 3 formats:
  1) Sign and magnitude
  2) 1's complement
  3) 2's complement
- In all three formats, MSB=0 for +ve numbers & MSB=1 for -ve numbers.
- **In sign-and-magnitude system,**
  negative value is obtained by changing the MSB from 0 to 1 of the corresponding positive value.
  For ex, +5 is represented by 0101 &
  -5 is represented by 1101.
- **In 1's complement system,**
  negative values are obtained by complementing each bit of the corresponding positive number.
  For ex, -5 is obtained by complementing each bit in 0101 to yield 1010.

(In other words, the operation of forming the 1's complement of a given number is equivalent to subtracting that number from 2^n-1).

- **In 2's complement system,**
  forming the 2's complement of a number is done by subtracting that number from 2^n.
  For ex, -5 is obtained by complementing each bit in 0101 & then adding 1 to yield 1011.(In other words, the 2's complement of a number is obtained by adding 1 to the 1's complement of that number).
- 2's complement system yields the most efficient way to carry out addition/subtraction operations



B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Figure 1.3 Binary, signed-integer representations.

ADDITION OF POSITIVE NUMBERS

- Consider adding two 1-bit numbers.
- The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2. We say that sum is 0 and thecarry-out is 1.

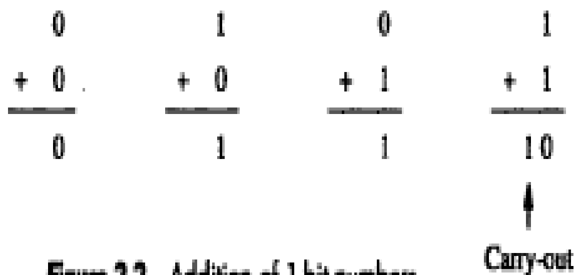


Figure 2.2 Addition of 1-bit numbers.

ADDITION & SUBTRACTION OF SIGNED NUMBERS

- Following are the two rules for addition and subtraction of n-bit signed numbers using the 2'scomplement representation system (Figure 1.6).

Rule 1:

- ➢ **To Add** two numbers, add their n-bits and ignore the carry-out signal from the MSB position.
- ➢ Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

Rule 2:

- ➢ **To Subtract** two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1.
- ➢ Result will be algebraically correct, if it lies in the range (2^{n-1}) to $+(2^{n-1}-1)$.

- When the result of an arithmetic operation is outside the representable-range, an arithmetic overflowis said to occur.
- To represent a signed in 2's complement form using a larger number of bits, repeat the sign bit asmany times as needed to the left. This operation is called **sign extension**.
- In 1's complement representation, the result obtained after an addition operation is not always correct. The carry-out(c_n) cannot be ignored. If c_n=0, the result obtained is correct. If c_n=1, then a 1 must be added to the result to make it correct.

MEMORY-LOCATIONS & ADDRESSES

- **Memory** consists of many millions of storage cells (flip-flops).
- Each cell can store a bit of information i.e. 0 or 1 (Figure 2.1).
- Each group of n bits is referred to as a **word** of information, and n is called the **word length**.
- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a **byte**.
- Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through 2^k-1 as the addresses of successive-locations in the memory).
- If 2^k = no. of addressable locations;
  then 2^k addresses constitute the address-space of the computer.

  For example, a 24-bit address generates an address-space of 2^{24} locations (16 MB).

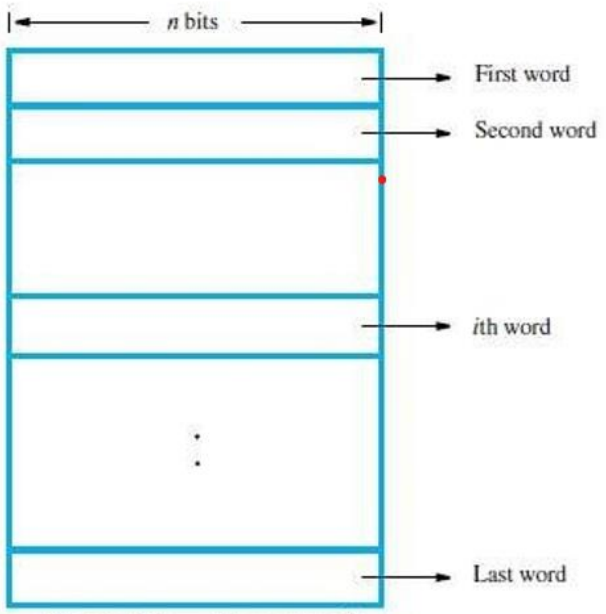


Figure 2.1 Memory words.

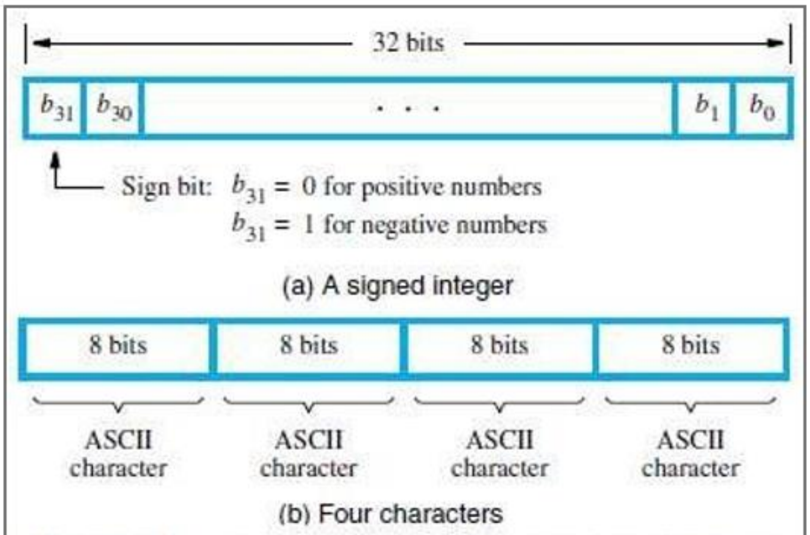


Figure 2.2 Examples of encoded information in a 32-bit word.

BYTE-ADDRESSABILITY

- In byte-addressable memory, successive addresses refer to successive byte locations in the memory.
- Byte locations have addresses 0, 1, 2. . . . .
- If the word-length is 32 bits, successive words are located at addresses 0, 4, 8. . with each word having 4 bytes.

BIG-ENDIAN & LITTLE-ENDIAN ASSIGNMENTS

- There are two ways in which byte-addresses are arranged (Figure 2.3).
  1) **Big-Endian:** Lower byte-addresses are used for the more significant bytes of the word.
  2) **Little-Endian:** Lower byte-addresses are used for the less significant bytes of the word

In both cases, byte-addresses 0, 4, 8. are taken as the addresses of successive words in the memory.

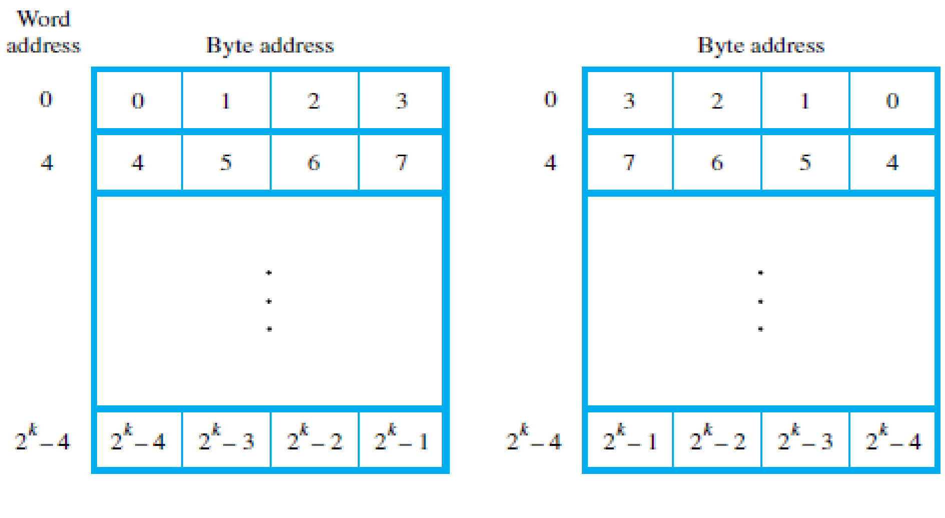


Figure 2.3 Byte and word addressing.

Consider a 32-bit integer (in hex): 0**x**12345678 which consists of 4 bytes: 12, 34, 56, and 78.

- ➢ Hence this integer will occupy 4 bytes in memory.
- ➢ Assume, we store it at memory address starting 1000.
- ➢ On little-endian, memory will look like

Address	Value
1000	78
1001	56
1002	34
1003	12

- On big-endian, memory will look like

Address	Value
1000	12
1001	34
1002	56
1003	78

WORD ALIGNMENT

- Words are said to be **Aligned** in memory if they begin at a byte-address that is a multiple of the number of bytes in a word.
- For example,
  - If the word length is 16(2 bytes), aligned words begin at byte-addresses 0, 2, 4 . . ….
  - If the word length is 64(2 bytes), aligned words begin at byte-addresses 0, 8, 16 . . ….
- Words are said to have **Unaligned Addresses**, if they begin at an arbitrary byte-address.

ACCESSING NUMBERS, CHARACTERS & CHARACTERS STRINGS

- A number usually occupies one word. It can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte-address.
- There are two ways to indicate the length of the string:
  1) A special control character with the meaning "end of string" can be used as the last character in the string.
  2) A separate memory word location or register can contain a number indicating the length of the string in bytes.

MEMORY OPERATIONS

- Two memory operations are:

1) Load (Read/Fetch) &
2) Store (Write).

- The **Load** operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged.
- Steps for Load operation:
  1) Processor sends the address of the desired location to the memory.
  2) Processor issues „read‟ signal to memory to fetch the data.
  3) Memory reads the data stored at that address.
  4) Memory sends the read data to the processor.
- The **Store** operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location.
- Steps for Store operation are:
  1) Processor sends the address of the memory-location where it wants to store data.
  2) Processor issues „write‟ signal to memory to store the data.
  3) Content of register(MDR) is written into the specified memory-location.

INSTRUCTIONS & INSTRUCTION SEQUENCING

- A computer must have instructions capable of performing 4 types of operations:
  1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
  2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
  3) Program sequencing and control (CALL.RET, LOOP, INT).
  4) I/0 transfers (IN, OUT).

Input/Output Organization

ACCESSING I/O-DEVICES

- A **single bus-structure** can be used for connecting I/O-devices to a computer (Figure 7.1).
- Each I/O device is assigned a unique set of address.
- Bus consists of 3 sets of lines to carry address, data & control signals.
- When processor places an address on address-lines, the intended-device responds to the command.
- The processor requests either a read or write-operation.
- The requested-data are transferred over the data-lines.

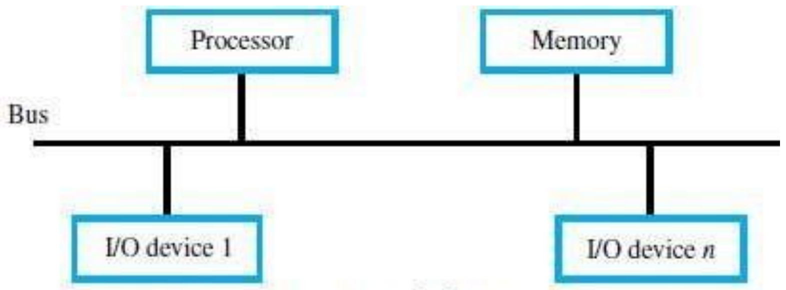


Figure 7.1 A single-bus structure.

- There are 2 ways to deal with I/O-devices: 1) Memory-mapped I/O & 2) I/O-mapped I/O.

1) Memory-Mapped I/O

- ➢ Memory and I/O-devices share a common address-space.
- ➢ Any data-transfer instruction (like Move, Load) can be used to exchange information.
- ➢ For example,

Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0.

Here, DATAIN address of the input-buffer of the keyboard.

2) I/O-Mapped I/O

- ➢ Memory and I/0 address-spaces are different.
- ➢ A special instructions named **IN** and **OUT** are used for data-transfer.
- ➢ Advantage of separate I/O space: I/O-devices deal with fewer address-lines.

I/O Interface for an Input Device

1) **Address Decoder:** enables the device to recognize its address when this addressappears on the address-lines (Figure 7.2).
2) **Status Register:** contains information relevant to operation of I/O-device.
3) **Data Register:** holds data being transferred to or from processor. There are 2 types:
   i) DATAIN Input-buffer associated with keyboard.
   ii) DATAOUT Output data buffer of a display/printer.

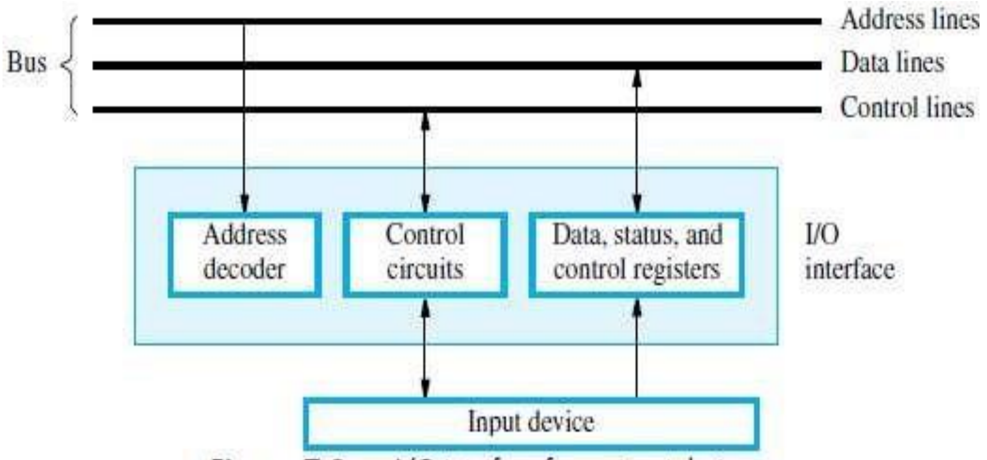


Figure 7.2 I/O interface for an input device.

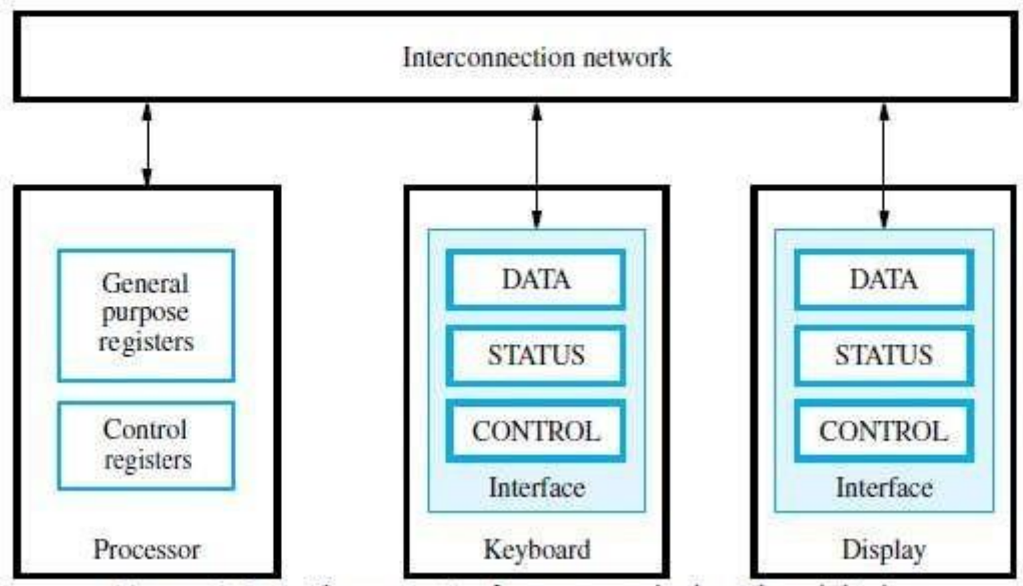


Figure 3.2 The connection for processor, keyboard, and display.

MECHANISMS USED FOR INTERFACING I/O-DEVICES

1) Program Controlled I/O

- Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/Odevice. (We say that the processor polls the device).
- Main drawback:
  The processor wastes time in checking status of device before actual data-transfer takes place.

2) Interrupt I/O

- I/O-device initiates the action instead of the processor.
- I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation.
- Like this, required synchronization is done between processor & I/O device.

3) Direct Memory Access (DMA)

- Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor.
- DMA is a technique used for high speed I/O-device.

Interrupts

- There are many situations where other tasks can be performed while waiting for an I/O device tobecome ready.
- A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready.
- Interrupt-signal is sent on the interrupt-request line.
- The processor can be performing its own task without the need to continuously check the I/O-device.
- The routine executed in response to an interrupt-request is called ISR.
- The processor must inform the device that its request has been recognized by sending INTA signal.
  (INTR Interrupt Request, INTA Interrupt Acknowledge, ISR Interrupt Service Routine)
- For example, consider COMPUTE and PRINT routines (Figure 3.6).

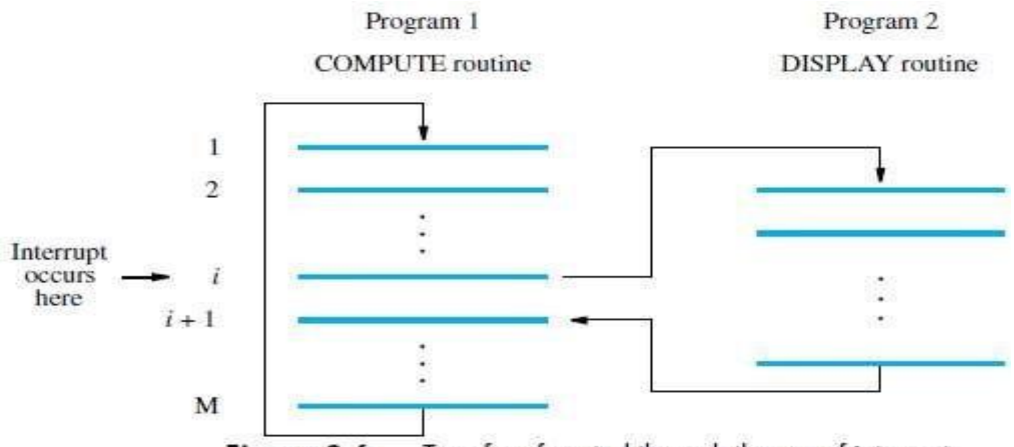


Figure 3.6 Transfer of control through the use of interrupts.

- The processor first completes the execution of instruction i.
- Then, processor loads the PC with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i+1.
- Therefore, when an interrupt occurs, the current content of PC is put in temporary storage location.
- A return at the end of ISR reloads the PC from that temporary storage location.
- This causes the execution to resume at instruction i+1.
- When processor is handling interrupts, it must inform device that its request has been recognized.
- This may be accomplished by INTA signal.
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of **PC & Status register.**
- Saving registers also increases the Interrupt Latency.
- **Interrupt Latency** is a delay between

  → time an interrupt-request is received and

  → start of the execution of the ISR.

Generally, the long interrupt latency in unacceptable

INTERRUPT HARDWARE

- Most computers have several I/O devices that can request an interrupt.
- A single interrupt-request (IR) line may be used to serve n devices (Figure 4.6).
- All devices are connected to IR line via switches to ground.
- To request an interrupt, a device closes its associated switch.
- Thus, if all IR signals are inactive, the voltage on the IR line will be equal to Vdd.
- When a device requests an interrupt, the voltage on the line drops to 0.
- This causes the INTR received by the processor to go to 1.
- The value of INTR is the logical OR of the requests from individual devices.

  INTR=INTR1+ INTR2+ .... +INTRn

- A special gate known as open-collector or open-drain are used to drive the INTR line.
- The Output of the open collector control is equal to a switch to the ground that is

  → open when gates input is in ”0‟ state and

  → closed when the gates input is in “1‟ state.
- Resistor R is called a **Pull-up Resistor** because it pulls the line voltage up to the high-voltage state when the switches are open

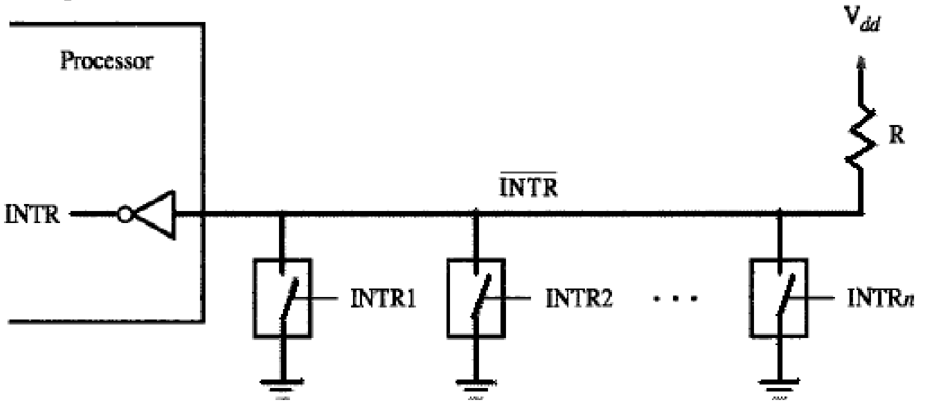


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

ENABLING & DISABLING INTERRUPTS

- All computers fundamentally should be able to enable and disable interruptions as desired.
- The problem of infinite loop occurs due to successive interruptions of active INTR signals.
- There are 3 mechanisms to solve problem of infinite loop:
  1) Processor should ignore the interrupts until execution of first instruction of the ISR.
  2) Processor should automatically disable interrupts before starting the execution of the ISR.
  3) Processor has a special INTR line for which the interrupt-handling circuit.

     Interrupt-circuit responds only to leading edge of signal. Such line is called edge-triggered.

- Sequence of events involved in handling an interrupt-request:
  1) The device raises an interrupt-request.
  2) The processor interrupts the program currently being executed.

3) Interrupts are disabled by changing the control bits in the processor status register (PS).
4) The device is informed that its request has been recognized.
In response, the device deactivates the interrupt-request signal.
5) The action requested by the interrupt is performed by the interrupt-service routine.
6) Interrupts are enabled and execution of the interrupted program is resumed.

HANDLING MULTIPLE DEVICES

- While handling multiple devices, the issues concerned are:
  1) How can the processor recognize the device requesting an interrupt?
  2) How can the processor obtain the starting address of the appropriate ISR?
  3) Should a device be allowed to interrupt the processor while another interrupt is beingserviced?
  4) How should 2 or more simultaneous interrupt-requests be handled?

POLLING

- Information needed to determine whether device is requesting interrupt is available in status-register
- Following condition-codes are used:
  - DIRQ Interrupt-request for display.
  - KIRQ Interrupt-request for keyboard.
  - KEN keyboard enable.
  - DEN Display Enable.
  - SIN, SOUT status flags.
- For an input device, SIN status flag in used.
  SIN = 1 when a character is entered at the keyboard.
  SIN = 0 when the character is read by processor.
  IRQ=1 when a device raises an interrupt-requests (Figure 4.3).
- Simplest way to identify interrupting-device is to have ISR poll all devices connected to bus.
- The first device encountered with its IRQ bit set is serviced.
- After servicing first device, next requests may be serviced.
- **Advantage:** Simple & easy to implement.
  Disadvantage: More time spent polling IRQ bits of all devices.

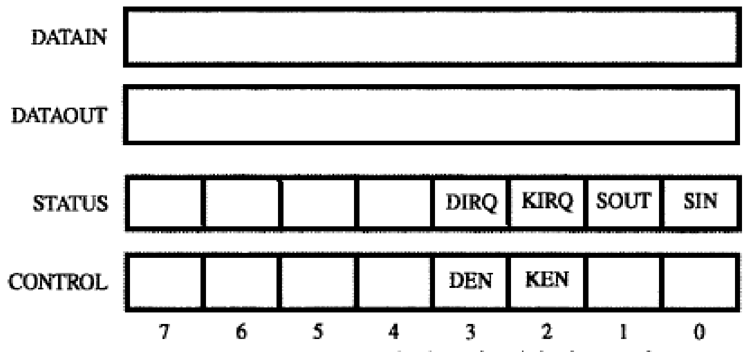


Figure 4.3 Registers in keyboard and display interfaces.

```
        Move       #LINE,R0         Initialize memory pointer.
WAITK   TestBit    #0,STATUS        Test SIN.
        Branch=0   WAITK            Wait for character to be entered.
        Move       DATAIN,R1        Read character.
WAITD   TestBit    #1,STATUS        Test SOUT.
        Branch=0   WAITD            Wait for display to become ready.
        Move       R1,DATAOUT       Send character to display.
        Move       R1,(R0)+         Store charater and advance pointer.
        Compare    #$0D,R1          Check if Carriage Return.
        Branch≠0   WAITK            If not, get another character.
        Move       #$0A,DATAOUT     Otherwise, send Line Feed.
        Call       PROCESS          Call a subroutine to process the
                                      the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

VECTORED INTERRUPTS

- A device requesting an interrupt identifies itself by sending a special-code to processor over bus.
- Then, the processor starts executing the ISR.

- The special-code indicates starting-address of ISR.
- The special-code length ranges from 4 to 8 bits.
- The location pointed to by the interrupting-device is used to store the staring address to ISR.
- The staring address to ISR is called the **interrupt vector**.
- Processor
  → loads interrupt-vector into PC &
  → executes appropriate ISR.
- When processor is ready to receive interrupt-vector code, it activates INTA line.
- Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal.
- The interrupt vector also includes a new value for the Processor Status Register.

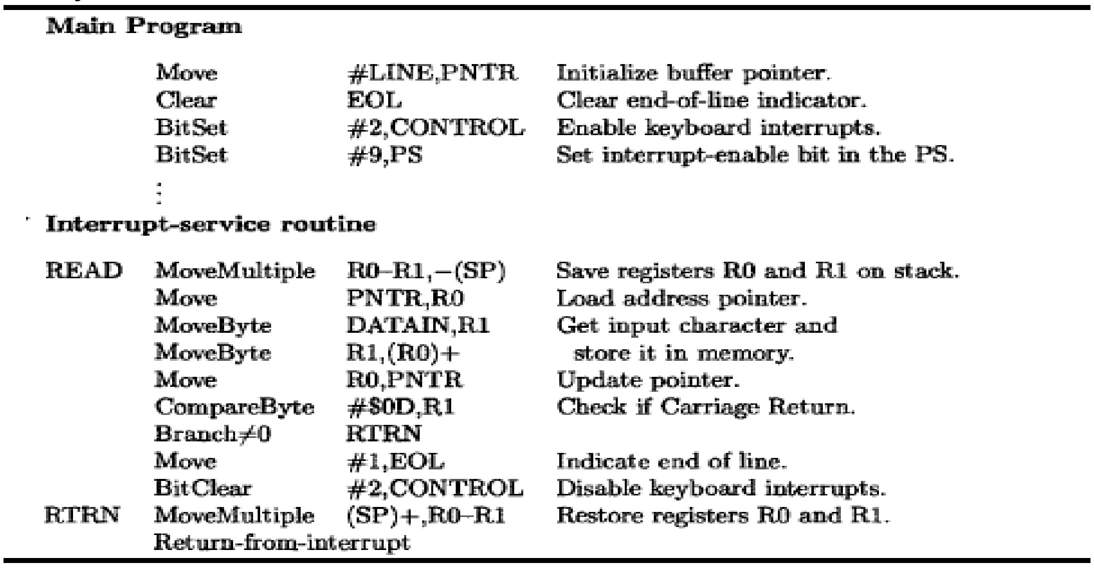
CONTROLLING DEVICE REQUESTS

- Following condition-codes are used:
  - ➢ KEN Keyboard Interrupt Enable.
  - ➢ DEN Display Interrupt Enable.
  - ➢ KIRQ/DIRQ Keyboard/Display unit requesting an interrupt.
- There are 2 independent methods for controlling interrupt-requests. (IE interrupt-enable).

  1) At Device-end

  IE bit in a control-register determines whether device is allowed to generate an interrupt-request.

  2) At Processor-end, interrupt-request is determined by
  → IE bit in the PS register or
  → Priority structure



```
Main Program

        Move            #LINE,PNTR      Initialize buffer pointer.
        Clear           EOL             Clear end-of-line indicator.
        BitSet          #2,CONTROL      Enable keyboard interrupts.
        BitSet          #9,PS           Set interrupt-enable bit in the PS.
          :

Interrupt-service routine

READ    MoveMultiple    R0–R1,–(SP)     Save registers R0 and R1 on stack.
        Move            PNTR,R0         Load address pointer.
        MoveByte        DATAIN,R1       Get input character and
        MoveByte        R1,(R0)+          store it in memory.
        Move            R0,PNTR         Update pointer.
        CompareByte     #$0D,R1         Check if Carriage Return.
        Branch≠0        RTRN
        Move            #1,EOL          Indicate end of line.
        BitClear        #2,CONTROL      Disable keyboard interrupts.
RTRN    MoveMultiple    (SP)+,R0–R1     Restore registers R0 and R1.
        Return-from-interrupt
```

Figure 4.9 Using interrupts to read a line of characters from a keyboard via the registers in Figure 4.3.

INTERRUPT NESTING

- A multiple-priority scheme is implemented by using separate INTR & INTA lines for each device
- Each INTR line is assigned a different priority-level (Figure 4.7).
- Priority-level of processor is the priority of program that is currently being executed.
- Processor accepts interrupts only from devices that have higher-priority than its own.
- At the time of execution of ISR for some device, priority of processor is raised to that of the device.
- Thus, interrupts from devices at the same level of priority or lower are disabled.

Privileged Instruction

- Processor's priority is encoded in a few bits of PS word. (PS Processor-Status).
- Encoded-bits can be changed by **Privileged Instructions** that write into PS.
- Privileged-instructions can be executed only while processor is running in **Supervisor Mode**.
- Processor is in supervisor-mode only when executing operating-system routines.

Privileged Exception

- User program cannot
  → accidently or intentionally change the priority of the processor &
  → disrupt the system-operation.

• An attempt to execute a privileged-instruction while in user-mode leads to a **Privileged Exception**.

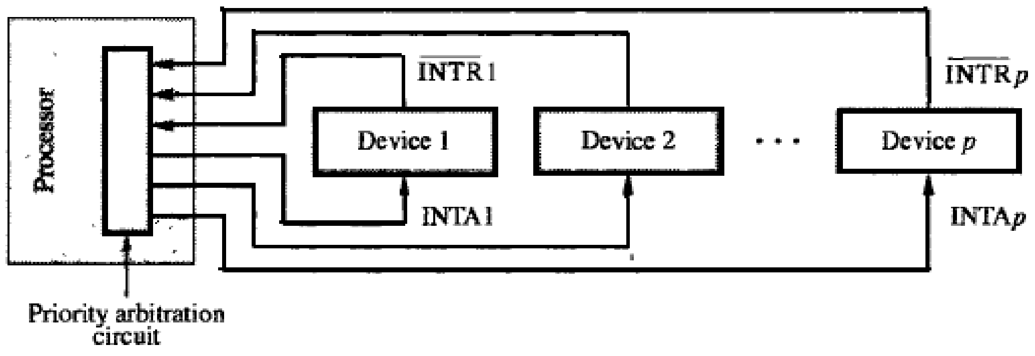

Processor
$\overline{\text{INTR}}1$
$\overline{\text{INTR}}p$
Device 1
Device 2
Device p
INTA1
INTAp
Priority arbitration circuit

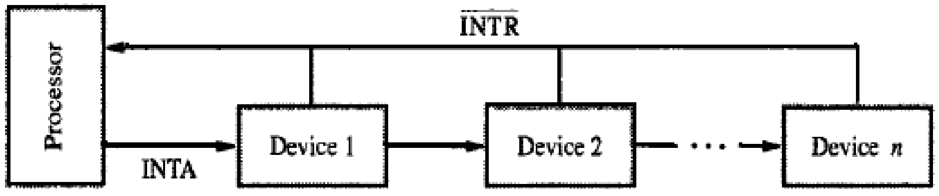
Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

SIMULTANEOUS REQUESTS

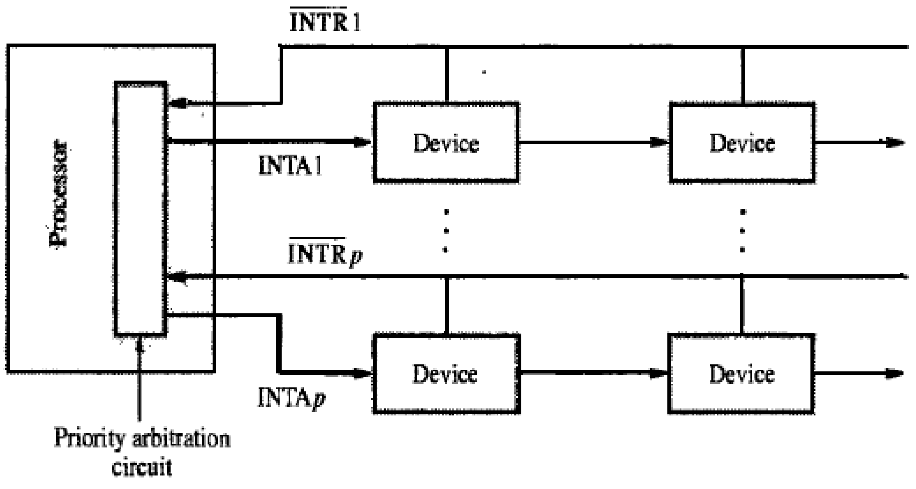
• The processor must have some mechanisms to decide which request to service when simultaneousrequests arrive.
• INTR line is common to all devices (Figure 4.8a).
• INTA line is connected in a daisy-chain fashion.
• INTA signal propagates serially through devices.
• When several devices raise an interrupt-request, INTR line is activated.
• Processor responds by setting INTA line to 1. This signal is received by device 1.
• Device-1 passes signal on to device 2 only if it does not require any service.
• If device-1 has a pending-request for interrupt, the device-1
  → blocks INTA signal &
  → proceeds to put its identifying-code on data-lines.
• Device that is electrically closest to processor has highest priority.
• **Advantage:** It requires fewer wires than the individual connections.

Arrangement of Priority Groups

• Here, the devices are organized in groups & each group is connected at a different priority level.
• Within a group, devices are connected in a daisy chain. (Figure 4.8b).



(a) Daisy chain


$\overline{\text{INTR}}1$
Processor
INTA1
Device
Device
$\overline{\text{INTR}}p$
INTAp
Device
Device
Priority arbitration circuit

(b) Arrangement of priority groups

Figure 4.8 Interrupt priority schemes.

EXCEPTIONS

- An **interrupt** is an event that causes
  → execution of one program to be suspended &
  → execution of another program to begin.
- **Exception** refers to any event that causes an interruption. For ex: I/O interrupts.

1. Recovery from Errors

- These are techniques to ensure that all hardware components are operating properly.
- For ex: Many computers include an ECC in memory which allows detection of errors in stored-data. (ECC Error Checking Code, ESR Exception Service Routine).
- If an error occurs, control-hardware
  → detects the errors &
  → informs processor by raising an interrupt.
- When exception processing is initiated (as a result of errors), processor.
  → suspends program being executed &
  → starts an ESR. This routine takes appropriate action to recover from the error.

2. Debugging

- Debugger
  → is used to find errors in a program and
  → uses exceptions to provide 2 important facilities: i) Trace & ii) Breakpoints

i)Trace

- When a processor is operating in trace-mode, an exception occurs after execution of every instruction(using debugging-program as ESR).
- Debugging-program enables user to examine contents of registers, memory-locations and so on.
- On return from debugging-program,
  next instruction in program being debugged is executed,then debugging-program is activated again.
- The trace exception is disabled during the execution of the debugging-program.

ii) Breakpoints

- Here, the program being debugged is interrupted only at specific points selected by user.
- An instruction called Trap (or Software interrupt) is usually provided for this purpose.
- When program is executed & reaches breakpoint, the user can examine memory & register contents.

3. Privilege Exception

- To protect OS from being corrupted by user-programs, **Privileged Instructions** are executed onlywhile processor is in supervisor-mode.
- For e.g.
  When processor runs in user-mode, it will not execute instruction that change priority of processor.
- An attempt to execute privileged-instruction will produce a **Privilege Exception**.
- As a result, processor switches to supervisor-mode & begins to execute an appropriate routine in OS.

DIRECT MEMORY ACCESS (DMA)

- The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA.
- DMA controller
  → is a control circuit that performs DMA transfers (Figure 8.13).
  → is a part of the I/O device interface.
  → performs the functions that would normally be carried out by processor.
- While a DMA transfer is taking place, the processor can be used to execute another program.

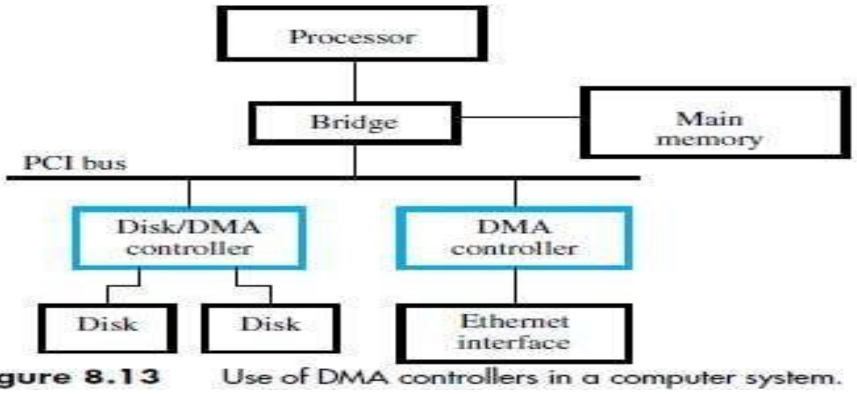


Figure 8.13 Use of DMA controllers in a computer system.

- DMA interface has three registers (Figure 8.12):

1) First register is used for storing starting-address.
2) Second register is used for storing word-count.
3) Third register contains status- & control-flags.

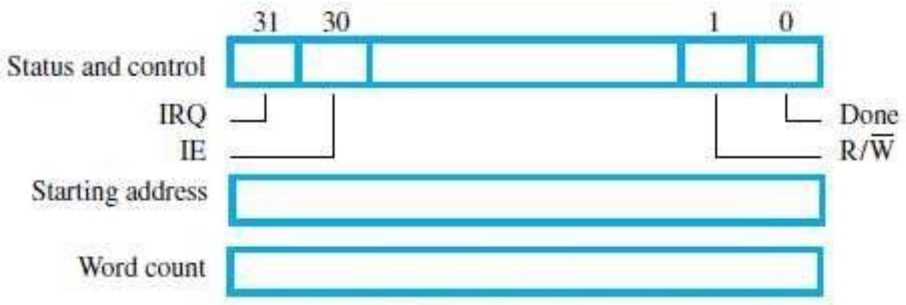


Figure 8.12 Typical registers in a DMA controller.

- The R/W bit determines direction of transfer.
  If R/W=1, controller performs a read-operation (i.e. it transfers data from memory to I/O), Otherwise, controller performs a write-operation (i.e. it transfers data from I/O to memory).
- If Done=1, the controller
  → has completed transferring a block of data and
  → is ready to receive another command. (IE Interrupt Enable).
- If IE=1, controller raises an interrupt after it has completed transferring a block of data.
- If IRQ=1, controller requests an interrupt.
- Requests by DMA devices for using the bus are always given higher priority than processor requests.
- There are 2 ways in which the DMA operation can be carried out:
  1) Processor originates most memory-access cycles.
  ➢ DMA controller is said to "steal" memory cycles from processor.
  ➢ Hence, this technique is usually called **Cycle Stealing**.
  2) DMA controller is given exclusive access to main-memory to transfer a block of data withoutany interruption. This is known as **Block Mode** (or burst mode).

BUS

- Bus → is used to inter-connect main-memory, processor & I/O-devices
  → includes lines needed to support interrupts & arbitration.

- Primary function: To provide a communication-path for transfer of data.
- **Bus protocol** is set of rules that govern the behavior of various devices connected to the buses.
- Bus-protocol specifies parameters such as:
  → asserting control-signals
  → timing of placing information on bus
  → rate of data-transfer.
- A typical bus consists of 3 sets of lines:
  1) Address,
  2) Data &
  3) Control lines.
- Control-signals
  → specify whether a read or a write-operation is to be performed.
  → carry timing information i.e. they specify time at which I/O-devices place data on the bus.
- R/W line specifies
  → read-operation when R/W=1.
  → write-operation when R/W=0.
- During data-transfer operation,
  ➢ One device plays the role of a bus-master.
  ➢ Master-device initiates the data-transfer by issuing read/write command on the bus.
  ➢ The device addressed by the master is called as Slave.
- Two types of Buses: 1) Synchronous and 2) Asynchronous.

SYNCHRONOUS BUS

- All devices derive timing-information from a common clock-line.

- Equally spaced pulses on this line define equal time intervals.
- During a ”bus cycle‟, one data-transfer can take place.

A sequence of events during a read-operation

- At time t0, the master (processor)
  → places the device-address on address-lines &
  → sends an appropriate command on control-lines (Figure 7.3).
- The command will
  → indicate an input operation &
  → specify the length of the operand to be read.
- Information travels over bus at a speed determined by physical & electrical characteristics.
- Clock pulse width(t1-t0) must be longer than max. propagation-delay b/w devices connected to bus.
- The clock pulse width should be long to allow the devices to decode the address & control signals.
- The slaves take no action or place any data on the bus before t1.
- Information on bus is unreliable during the period t0 to t1 because signals are changing state.
- Slave places requested input-data on data-lines at time t1.
- At end of clock cycle (at time t2), master strobes (captures) data on data-lines into its input-buffer
- For data to be loaded correctly into a storage device, data must be available at input of that device for a period greater than setup-time of device.

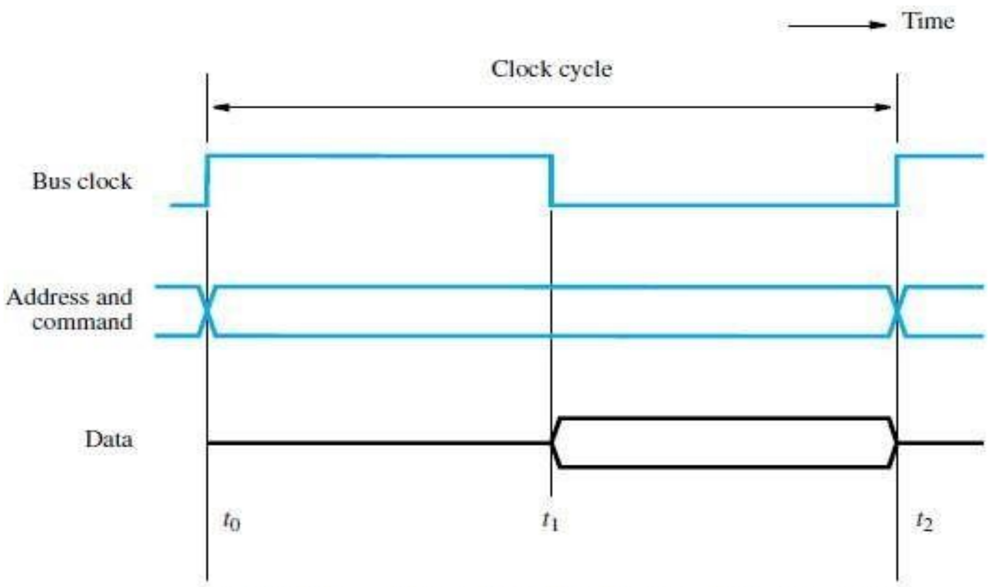


Figure 7.3 Timing of an input transfer on a synchronous bus.

A Detailed Timing Diagram for the Read-operation

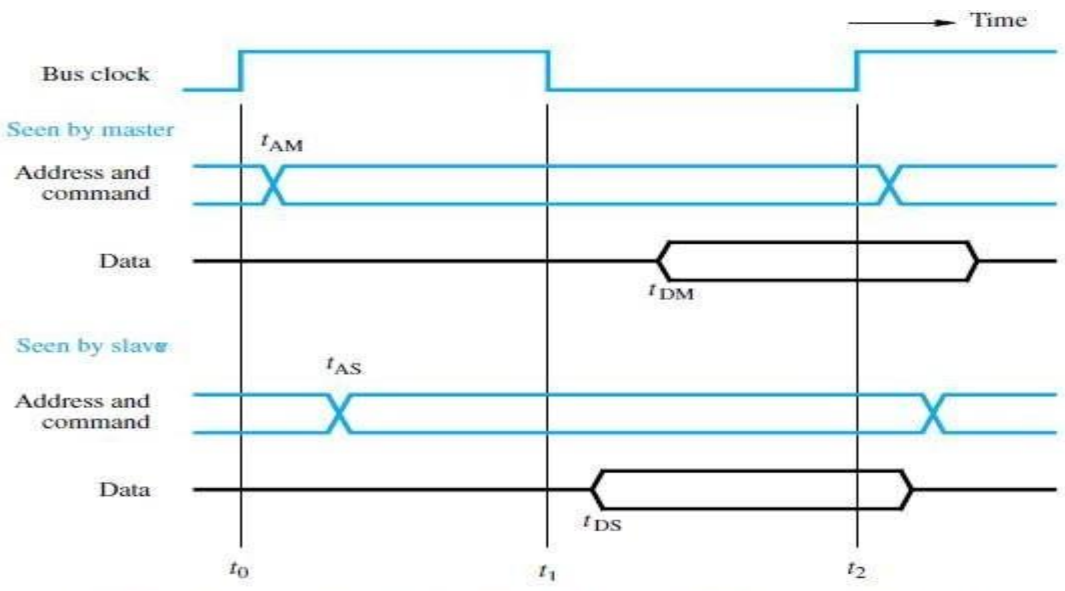


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

- The picture shows two views of the signal except the clock (Figure 7.4).
- One view shows the signal seen by the master & the other is seen by the salve.
- Master sends the address & command signals on the rising edge at the beginning of clock period (t0).
- These signals do not actually appear on the bus until tam.
- Sometimes later, at tAS the signals reach the slave.
- The slave decodes the address.
- At t1, the slave sends the requested-data.
- At t2, the master loads the data into its input-buffer.
- Hence the period t2, tDM is the setup time for the master‟s input-buffer.
- The data must be continued to be valid after t2, for a period equal to the hold time of that buffers.

Disadvantages

- The device does not respond.
- The error will not be detected.

Multiple Cycle Transfer for Read-operation

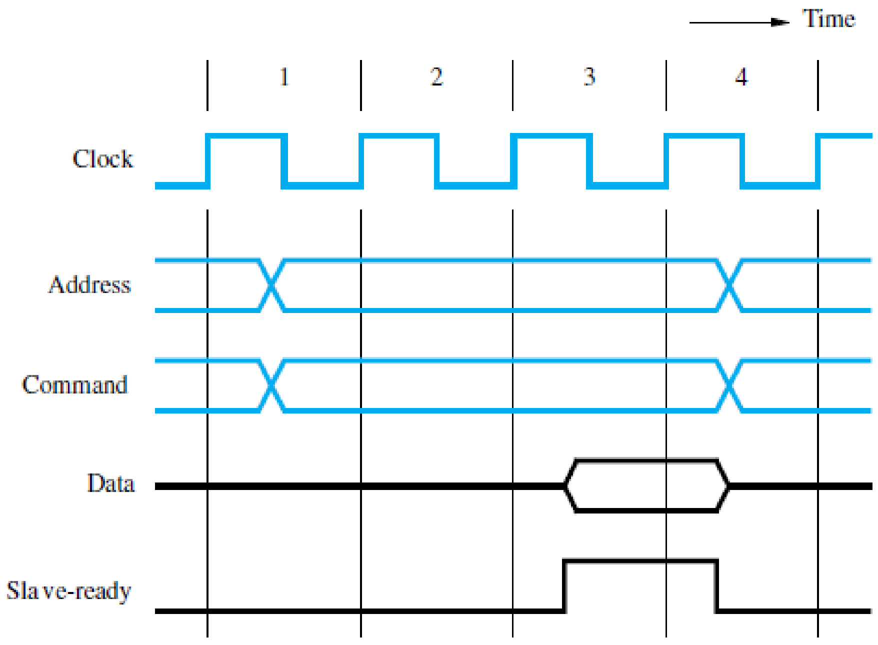


Figure 7.5 An input transfer using multiple clock cycles.

- During, clock cycle-1, master sends address/command info the bus requesting a “read‟ operation.
- The slave receives & decodes address/command information (Figure 7.5).
- At the active edge of the clock i.e. the beginning of clock cycle-2, it makes accession to respond immediately.
- The data become ready & are placed in the bus at clock cycle-3.
- At the same times, the slave asserts a control signal called **slave-ready**.
- The master strobes the data to its input-buffer at the end of clock cycle-3.
- The bus transfer operation is now complete.
- And the master sends a new address to start a new transfer in clock cycle4.
- The slave-ready signal is an acknowledgement from the slave to the master.

ASYNCHRONOUS BUS

- This method uses handshake-signals between master and slave for coordinating data-transfers.

• There are 2 control-lines:

1) **Master-Ready (MR)** is used to indicate that master is ready for a transaction.

2) **Slave-Ready (SR)** is used to indicate that slave is ready for a transaction.

The Read Operation proceeds as follows:

• At t0, master places address/command information on bus.

• At t1, master sets MR-signal to 1 to inform all devices that the address/command-info is ready.

 ➢ MR-signal =1 causes all devices on the bus to decode the address.

 ➢ The delay t1 – t0 is intended to allow for any skew that may occurs on the bus.

 ➢ Skew occurs when 2 signals transmitted from 1 source arrive at destination at different time

 ➢ Therefore, the delay t1 – t0 should be larger than the maximum possible bus skew.

• At t2, slave

 → performs required input-operation &

 → sets SR signal to 1 to inform all devices that it is ready (Figure 7.6).

• At t3, SR signal arrives at master indicating that the input-data are available on bus.

• At t4, master removes address/command information from bus.

• At t5, when the device-interface receives the 1-to-0 transition of MR signal, it removes data and SR signal from the bus. This completes the input transfer.

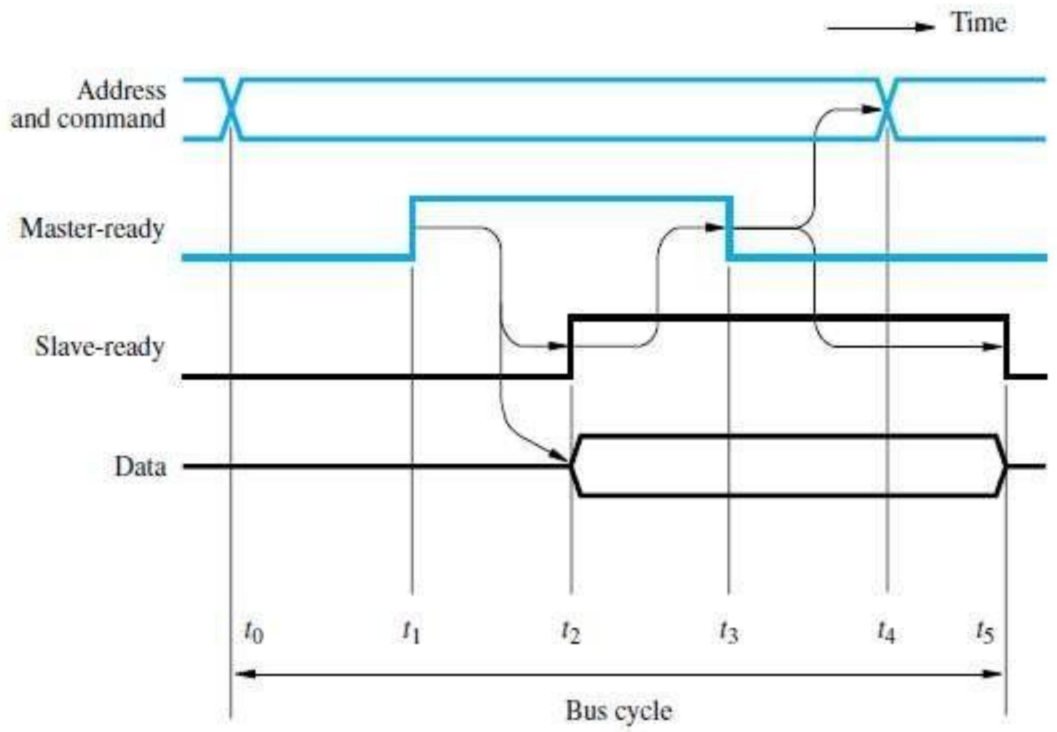


Figure 7.6 Handshake control of data transfer during an input operation.

• A change of state is one signal is followed by a change is the other signal. Hence this scheme iscalled as **Full Handshake**.

• **Advantage:** It provides the higher degree of flexibility and reliability.

STANDARD I/O INTERFACE

- Consider a computer system using different interface standards.
- Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus (Figure 4.38).
- These two buses are interconnected by a circuit called **Bridge**.
- The bridge translates the signals and protocols of one bus into another.
- The bridge-circuit introduces a small delay in data transfer between processor and the devices.

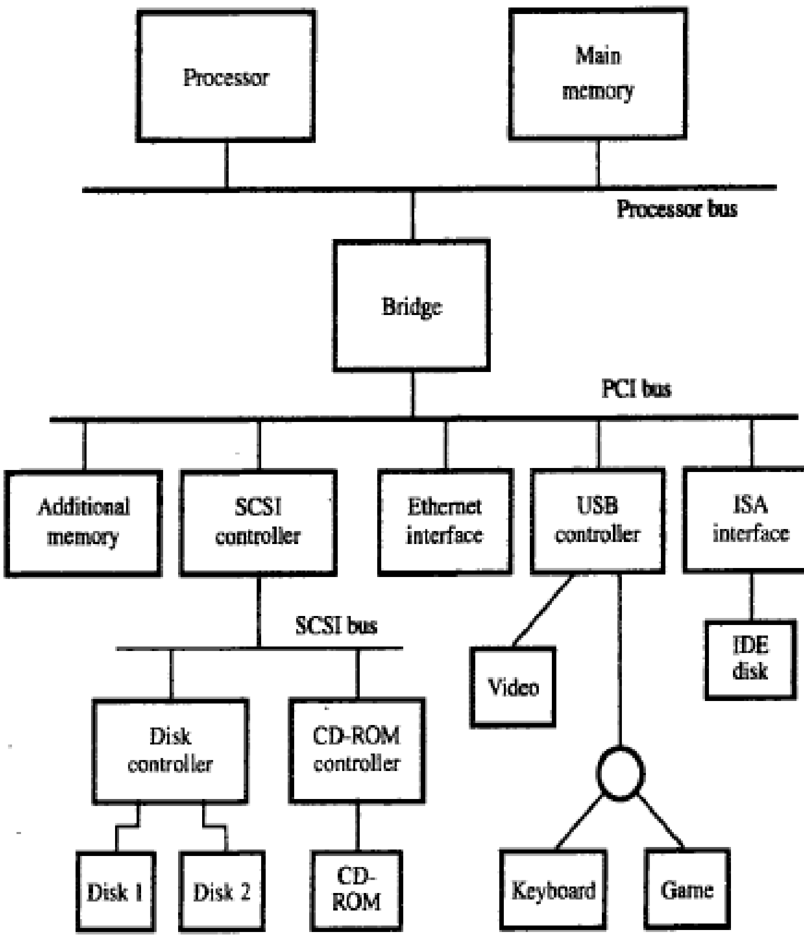


Figure 4.38 An example of a computer system using different interface standards.

- The 3 major standard I/O interfaces are:
  1) PCI (Peripheral Component Interconnect)
  2) SCSI (Small Computer System Interface)
  3) USB (Universal Serial Bus)
- PCI defines an expansion bus on the motherboard.
- SCSI and USB are used for connecting additional devices both inside and outside the computer-box.
- SCSI bus is a high-speed parallel bus intended for devices such as disk and video display.
- USB uses a serial transmission to suit the needs of equipment ranging from keyboard to gamecontrol to internal connection.
- IDE (Integrated Device Electronics) disk is compatible with ISA which shows the connection to an Ethernet.

PCI

- PCI is developed as a low cost bus that is truly processor independent.
- PCI supports high speed disk, graphics and video devices.
- PCI has plug and play capability for connecting I/O devices.
- To connect new devices, the user simply connects the device interface board to the bus.

DATA TRANSFER IN PCI

- The data are transferred between cache and main-memory.
- The data is a sequence of words which are stored in successive memory-locations.
- During **read-operation**,
  - ➢ When the processor specifies an address, the memory responds by sending a sequence ofdata-words from successive memory-locations.
- During **write-operation**,
  - ➢ When the processor sends an address, a sequence of data-words is written into successive memory-locations.
- PCI supports read and write-operation.
- A read/write-operation involving a single word is treated as a burst of length one.
- PCI has 3 address-spaces. They are
  1) Memory address-space
  2) I/O address-space &
  3) Configuration address-space.
- I/O Address-space Intended for use with processor.
- Configuration space Intended to give PCI, its plug and play capability.
- **PCI Bridge** provides a separate physical connection to main-memory.
- The master maintains the address information on the bus until data-transfer is completed.
- At any time, only one device acts as **Bus-Master**.
- A master is called "initiator" which is either processor or DMA.
- The addressed-device that responds to read and write commands is called a **Target.**
- A complete transfer operation on the bus, involving an address and burst of data is called a **Transaction.**

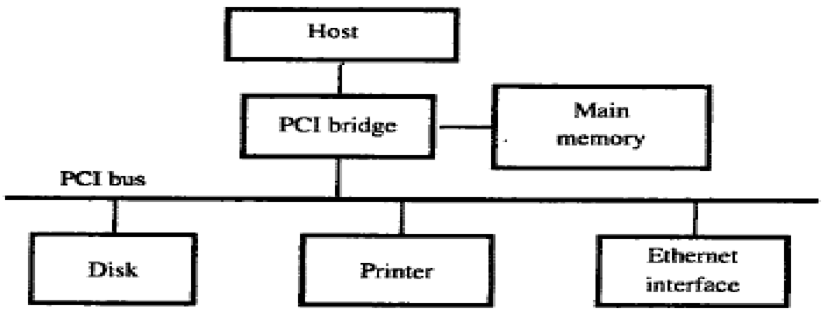


Figure 4.39 Use of a PCI bus in a computer system.

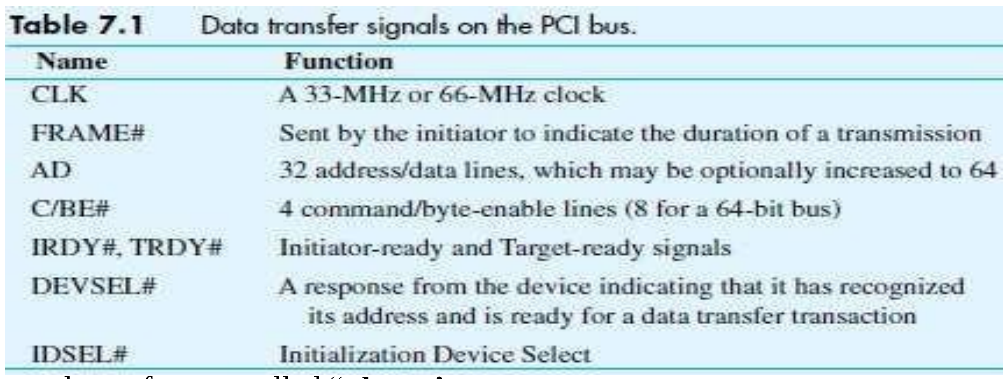


Table 7.1 Data transfer signals on the PCI bus.

Name	Function
CLK	A 33-MHz or 66-MHz clock
FRAME#	Sent by the initiator to indicate the duration of a transmission
AD	32 address/data lines, which may be optionally increased to 64
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus)
IRDY#, TRDY#	Initiator-ready and Target-ready signals
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction
IDSEL#	Initialization Device Select

- Individual word transfers are called "**phases'**.

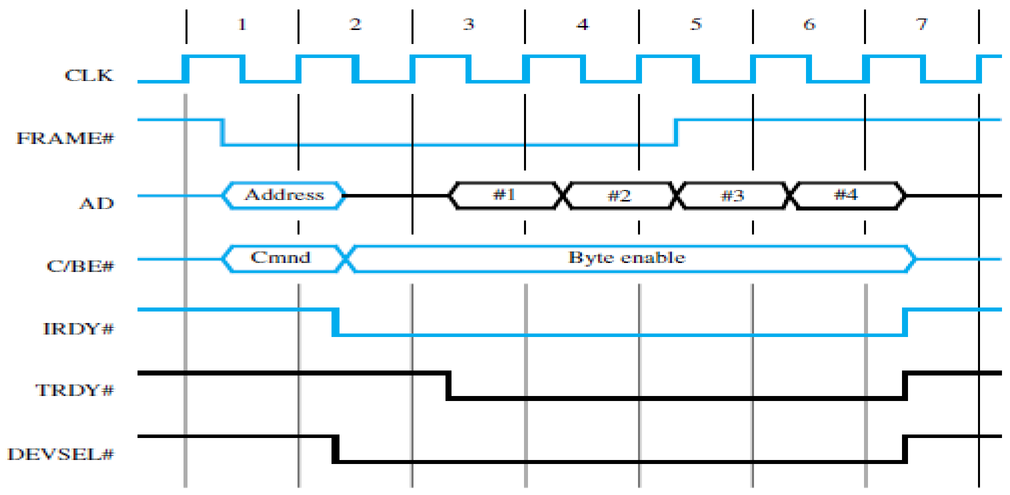


Figure 7.19 A Read operation on the PCI bus.

- During Clock cycle-1,
  - ➢ The processor a
    - → asserts FRAME# to indicate the beginning of a transaction;
    - → sends the address on AD lines and command on C/BE# Lines.
- During Clock cycle-2,
  - ➢ The processor removes the address and disconnects its drives from AD lines.
  - ➢ Selected target
    - → enables its drivers on AD lines and
    - → fetches the requested-data to be placed on bus.
  - ➢ Selected target
    - → asserts DEVSEL# and
    - → maintains it in asserted state until the end of the transaction.
  - ➢ C/BE# is
    - → used to send a bus command and it is
    - → used for different purpose during the rest of the transaction.
- During Clock cycle-3,
  - ➢ The initiator asserts IRDY# to indicate that it is ready to receive data.
  - ➢ If the target has data ready to send then it asserts TRDY#. In our eg, the target sends 3more words of data in clock cycle 4 to 6.
- During Clock cycle-5
  - ➢ The indicator uses FRAME# to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME# during clock cycle 5.
- During Clock cycle-7,
  - ➢ After sending 4th word, the target
    - → disconnects its drivers and
    - → negates DEVSEL# during clock cycle 7.

DEVICE CONFIGURATION OF PCI

- The PCI has a configuration ROM that stores information about that device.
- The configuration ROM's of all devices are accessible in the configuration address-space.
- The initialization software read these ROM's whenever the system is powered up or reset.
- In each case, it determines whether the device is a printer, keyboard or disk controller.
- Devices are assigned address during initialization process.
- Each device has an input signal called IDSEL# (Initialization device select) which has 21 address-lines (AD11 to AD31).
- During configuration operation,
  - ➢ The address is applied to AD input of the device and
  - ➢ The corresponding AD line is set to 1 and all other lines are set to 0.AD11 - AD31 **Upper** address-line

    A0 - A10 **Lower** address-line: Specify the type of the operation and to access thecontent of device configuration ROM.
- The configuration software scans all 21 locations. PCI bus has interrupt-request lines.
- Each device may requests an address in the I/O space or memory space.

USB

- USB stands for Universal Serial Bus.
- USB supports 3 speed of operation. They are,
  1) Low speed (1.5 Mbps)
  2) Full speed (12 mbps) &
  3) High speed (480 mbps).
- The USB has been designed to meet the key objectives. They are,
  1) Provide a simple, low-cost and easy to use interconnection system.
     This overcomes difficulties due to the limited number of I/O ports available on a computer.
  2) Accommodate a wide range of data transfer characteristics for I/O devices.For e.g. telephone and Internet connections
  3) Enhance user convenience through a "plug-and-play" mode of operation.
- **Advantage:** USB helps to add many devices to a computer system at any time without opening thecomputer-box.

Port Limitation

- ➢ Normally, the system has a few limited ports.
- ➢ To add new ports, the user must open the computer-box to gain access to the internalexpansion bus & install a new interface card.
- ➢ The user may also need to know to configure the device & the s/w.

Plug & Play

- ➢ The main objective: USB provides a plug & play capability.
- ➢ The plug & play feature enhances the connection of new device at any time, while the systemis operation.
- ➢ The system should
  - → Detect the existence of the new device automatically.
  - → Identify the appropriate device driver s/w.
  - → Establish the appropriate addresses.
  - → Establish the logical connection for communication.

DEVICE CHARACTERISTICS OF USB

- The kinds of devices that may be connected to a computer cover a wide range of functionality.
- The speed, volume & timing constrains associated with data transfer to & from devices variessignificantly.

Eg: 1 Keyboard

- ➢ Since the event of pressing a key is not synchronized to any other event in a computersystem, the data generated by keyboard are called asynchronous.
  The data generated from keyboard depends upon the speed of the human operator which isabout 100 bytes/sec.

Eg: 2 Microphone attached in a computer system internally/externally

- ➢ The sound picked up by the microphone produces an analog electric signal, which must beconverted into digital form before it can be handled by the computer.
- ➢ This is accomplished by sampling the analog signal periodically.
- ➢ The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a stream is called isochronous (i.e.) successive events are separated by equal period of time.
- ➢ If the sampling rate in „S" samples/sec then the maximum frequency captured by sampling process is s/2.
- ➢ A standard rate for digital sound is 44.1 KHz.

USB ARCHITECTURE

- To accommodate a large number of devices that can be added or removed at any time, the USB hasthe tree structure as shown in the figure 7.17.
- Each node of the tree has a device called a **Hub**.
- A hub acts as an intermediate control point between the host and the I/O devices.
- At the root of the tree, a **Root Hub** connects the entire tree to the host computer.
- The leaves of the tree are the I/O devices being served (for example, keyboard or speaker).
- A hub copies a message that it receives from its upstream connection to all its downstream ports.
- As a result, a message sent by the host computer is broadcast to all I/O devices, but only theaddressed-device will respond to that message.

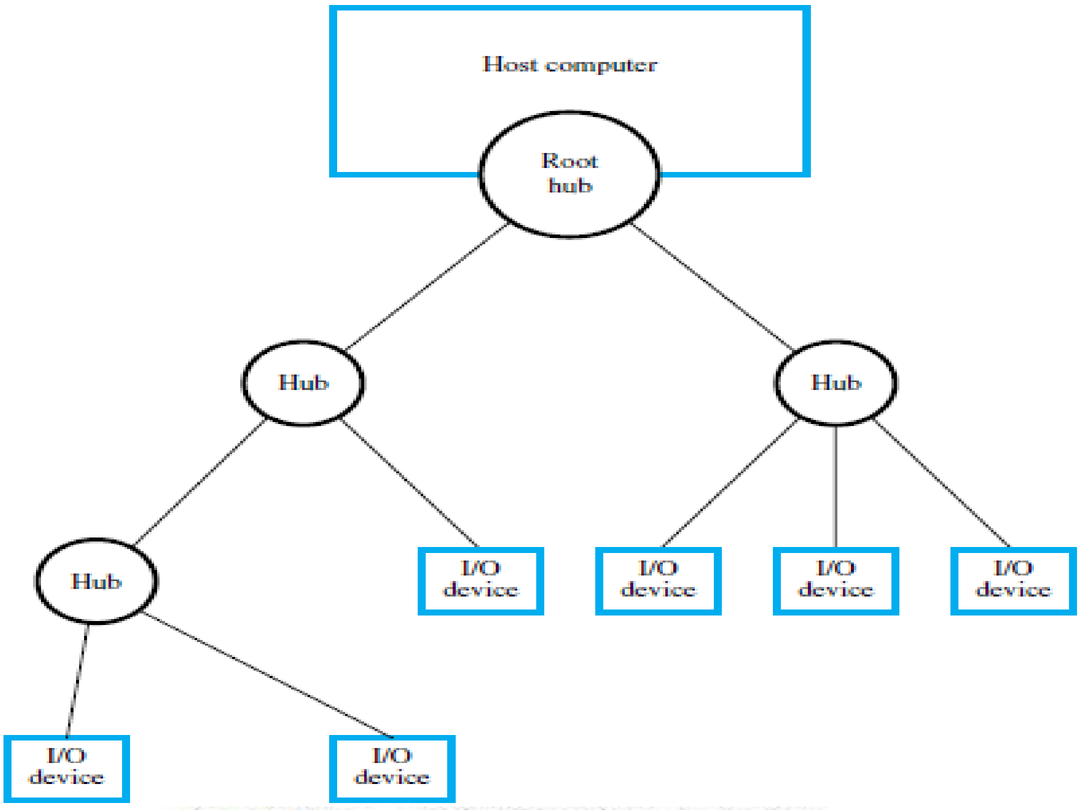


Figure 7.17 Universal Serial Bus tree structure.

USB ADDRESSING

- Each device may be a hub or an I/O device.
- Each device on the USB is assigned a 7-bit address.
- This address
  → is local to the USB tree and
  → is not related in any way to the addresses used on the processor-bus.
- A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily.
- When a device is first connected to a hub, or when it is powered-on, it has the address 0.
- The hardware of the hub detects the device that has been connected, and it records this fact as part of its own status information.
- Periodically, the host polls each hub to
  → collect status information and
  → learn about new devices that may have been added or disconnected.
- When the host is informed that a new device has been connected, it uses sequence of commands to
  → send a reset signal on the corresponding hub port.
  → read information from the device about its capabilities.
  → send configuration information to the device, and
  → assign the device a unique USB address.
- Once this sequence is completed, the device
  → begins normal operation and
  → responds only to the new address.

USB PROTOCOLS

- All information transferred over the USB is organized in packets.
- A packet consists of one or more bytes of information.
- There are many types of packets that perform a variety of control functions.
- The information transferred on USB is divided into 2 broad categories: 1) Control and 2) Data.
- Control packets perform tasks such as
  - → addressing a device to initiate data transfer.
  - → acknowledging that data have been received correctly or
  - → indicating an error.
- Data-packets carry information that is delivered to a device.
- A packet consists of one or more fields containing different kinds of information.
- The first field of any packet is called the **Packet Identifier (PID)** which identifies type of thatpacket.
- They are transmitted twice.
  1) The first time they are sent with their true values and
  2) The second time with each bit complemented.
- The four PID bits identify one of 16 different packet types.
- Some control packets, such as ACK (Acknowledge), consist only of the PID byte.

Control packets used for controlling data transfer operations are called **Token Packets.**

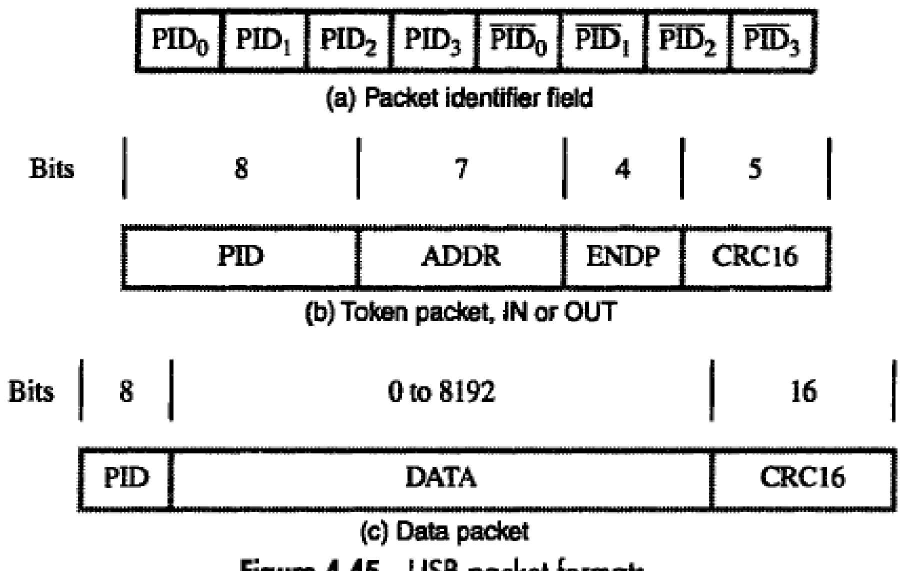


Figure 4.45 USB packet formats.