||Jai Sri Gurudev ||

Sri Adichunchanagiri Shikshana Trust®

SJB INSTITUTE OF TECHNOLOGY

Accredited by NBA & NAAC with 'A' Grade

No. 67, BGS Health & Education City, Dr. Vishnuvardhan Road

Kengeri, Bangalore – 560 060

Department of Electronics & Communication Engineering

Basic Structure of Computers & I/O Organization [21EC52]

MODULE - 1

Notes (as per VTU Syllabus)

V SEMESTER – B. E

Academic Year: 2023 – 2024 (ODD)

Course Coordinator :	**Dr. Supreeth H S G**
Designation :	**Associate Professor**

Syllabus

[As per Choice Based Credit System (CBCS) scheme]
SEMESTER – V

Subject Code: 21EC52 **IA Marks: 50**
Number of Lecture Hours/Week: 3 **Exam Marks: 50**
Total Number of Lecture Hours: 40 **Exam Hours 03**
Credits – 04

Course objectives: This course will enable students to
1. Explain the basic organization of a computer system.
2. Demonstrate functioning of different sub systems, such as processor, Input/output, and memory.
3. Describe the architectural features and instructions of 32-bit microcontroller ARM Cortex M3.
4. Apply the knowledge gained for Programming ARM Cortex M3 for different applications.
5. Understand the basic hardware components and their selection method based on the characteristics and attributes of an embedded system

Module – 1

Basic Structure of Computers: Basic Operational Concepts, Bus Structures, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement.

Text Book 1: Chapter 1 – 1.3, 1.4, 1.6 (1.6.1-1.6.4, 1.6.7), Chapter 2 – 2.2 to 2.10

Input/Output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Direct Memory Access, Buses, Interface Circuits, Standard I/O Interfaces – PCI Bus, SCSI Bus, USB.

Text Book 1: Chapter 4 – 4.1, 4.2, 4.4, 4.5, 4.6, 4.7

Text Books:

- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, 5th Edition, Tata McGraw Hill, 2002. (Listed topics only from Chapters 1, 2, 4, 5, 8).

- 2. Andrew N Sloss, Dominic System and Chris Wright, “ARM System Developers Guide”, Elsevier, Morgan Kaufman publisher, 1st Edition, 2008.

INDEX SHEET

Section 1: 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.

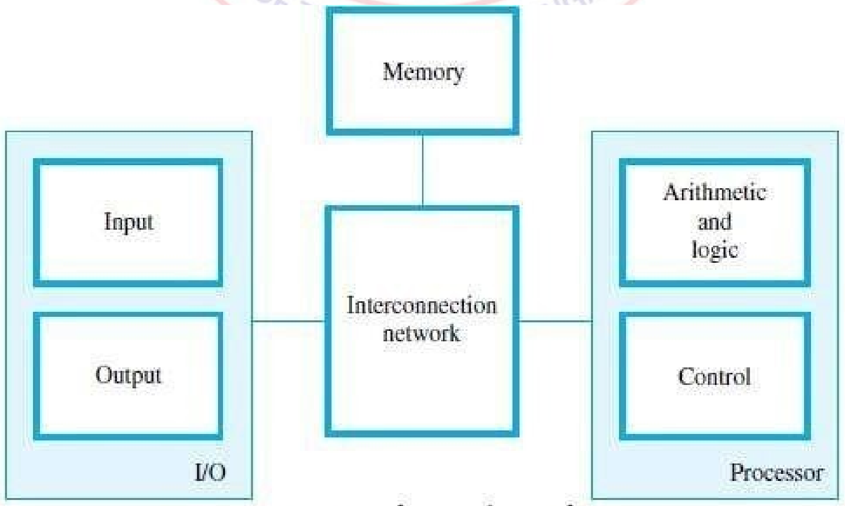


Figure 1.1 Basic functional units of a computer.

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 - Instruction-Register, PC - Program Counter, MAR-Memory Address Register, MDR- Memory Data Register)

STEPS TO EXECUTE AN INSTRUCTION

- ➢ The address of first instruction (to be executed) gets loaded into PC.
- ➢ The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal tomemory.
- ➢ After certain amount of elapsed time, the first instruction is read out of memory and placed intoMDR.
- ➢ Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded &executed.
- ➢ 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.
- ➢ Likewise required number of operands is fetched into processor.
- ➢ Finally, ALU performs the desired operation.
- ➢ If the result of this operation is to be stored in the memory, then the result is sent to the MDR.
- ➢ The address of the location where the result is to be stored is sent to the MAR and a Write cycle isinitiated.

- At some point during execution, contents of PC are incremented to point to next instruction in the program.

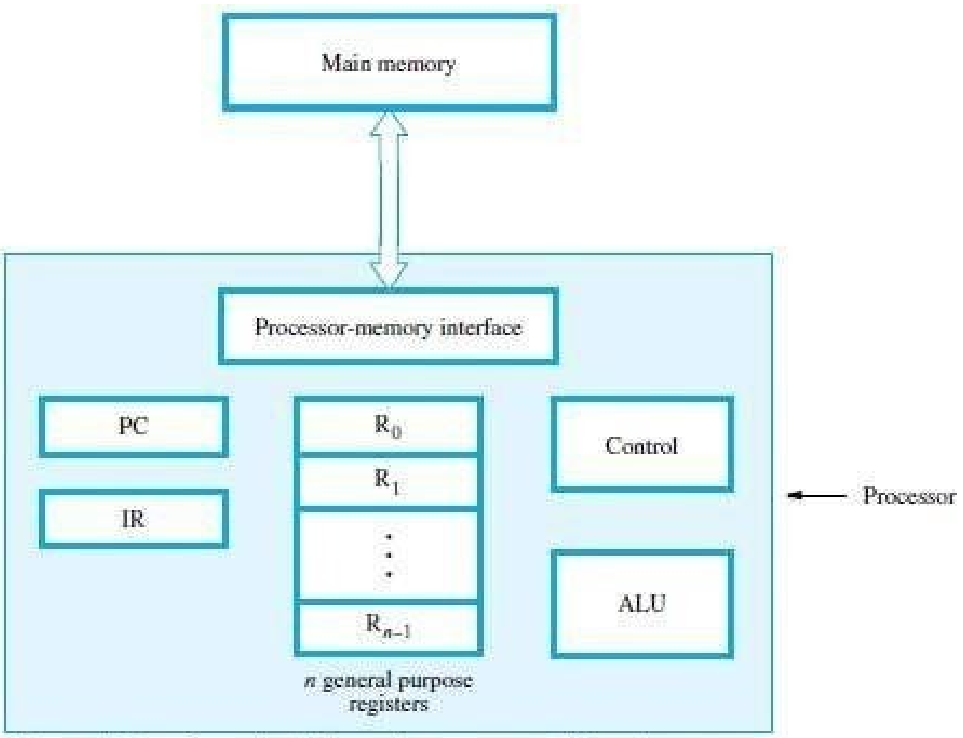


Figure 1.2 Connection between the processor and the main memory.

Section 2:

1.2 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.

- 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.

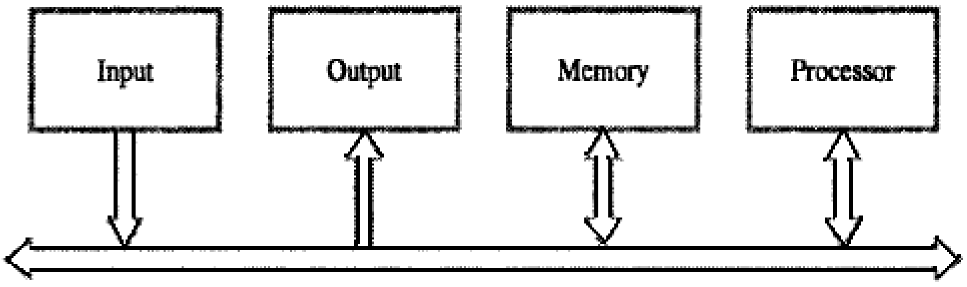


Figure 1.3 Single-bus structure.

Section 3:

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. (HLL- High LevelLanguage).
- For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

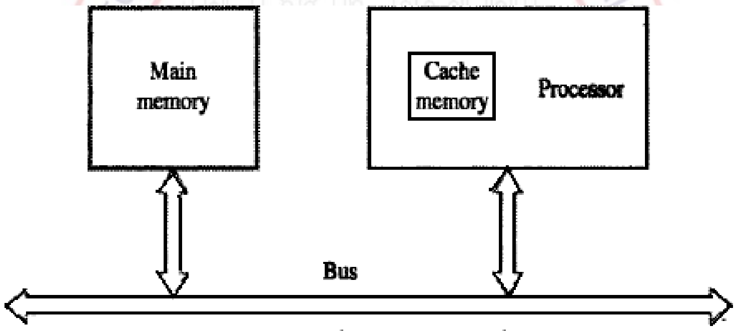


Figure 1.5 The processor cache.

Let us 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 minimizedwhich is achieved by using the cache.

1.3.1 PROCESSOR CLOCK

- Processor circuits are controlled by a timing signal called a **Clock**.
- The clock defines regular time intervals called **Clock Cycles**.
- To execute a machine instruction, the processor divides the action to be performed into a sequenceof basic steps such that each step can be completed in one clock cycle.
- Let P = Length of one clock cycle R = Clock rate.
- Relation between P and R is given by

$$R = \frac{1}{P}$$

- R is measured in cycles per second and Cycles per second is also called Hertz (Hz)

1.3.2 BASIC PERFORMANCE EQUATION

- Let T = Processor time required to executed a program.
  N = Actual number of instruction executions.
  S = Average number of basic steps needed to execute one machine instruction. R = Clock rate in cycles per second.
- The program execution time is given by

$$T = \frac{N \times S}{R} \quad \ldots\ldots(1)$$

Eq1 is referred to as the basic performance equation.

- To achieve high performance, the computer designer must reduce the value of T, which means reducing N and S, and increasing R.
  - ➢ The value of N is reduced if source program is compiled into fewer machine instructions.
  - ➢ The value of S is reduced if instructions have a smaller number of basic steps to perform.
  - ➢ The value of R can be increased by using a higher frequency clock.
- Care has to be taken while modifying values since changes in one parameter may affect the other.

1.3.3 CLOCK RATE

- There are 2 possibilities for increasing the clock rate R:
  1) Improving the IC technology makes logic-circuits faster.
     This reduces the time needed to compute a basic step. (IC-integrated circuits).
     This allows the clock period P to be reduced and the clock rate R to be increased.
  2) Reducing the amount of processing done in one basic step also reduces the clock period P.
- In presence of a cache, the percentage of accesses to the main-memory is small.

Hence, much of performance-gain expected from the use of faster technology can be realized.The value of T will be reduced by same factor as R is increased." S & N" are not affected.

1.3.4 PERFORMANCE MEASUREMENT

- Benchmark refers to standard task used to measure how well a processor operates.
- The Performance Measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC- System Performance Evaluation Corporation).
- SPEC Rating is given by

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

- SPEC rating = 50- The computer under test is 50 times as fast as reference-computer.
- The test is repeated for all the programs in the SPEC suite.Then, the geometric mean of the results is computed.
- Let SPECi = Rating for program „i' in the suite.
  Overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i \right)^{\frac{1}{n}}$$

where n = no. of programs in the suite.

INSTRUCTION SET: CISC AND RISC

RISC	CISC
Simple instructions taking one cycle.	Complex instructions taking multiple cycle.
Instructions are executed by hardwired control unit.	Instructions are executed by microprogrammed control unit.
Few instructions.	Many instructions.
Fixed format instructions.	Variable format instructions.
Few addressing modes, and most instructions have register to register addressing mode.	Many addressing modes.
Multiple register set.	Single register set.
Highly pipelined.	No pipelined or less pipelined.

Problem 1:

A program contains 1000 instructions. Out of that 25% instructions requires 4 clock cycles,40% instructions requires 5 clock cycles and remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine.

Solution:

N = 1000
25% of N= 250 instructions require 4 clock cycles.
40% of N =400 instructions require 5 clock
cycles.35% of N=350 instructions require 3
clock cycles.

T = (N*S)/R= (250*4+400*5+350*3)/1X10^9 =(1000+2000+1050)/1*10^9= 4.05 μs.

Problem 2:

For the following processor, obtain the performance.
Clock rate = 800 MHz
No. of instructions executed = 1000
Average no of steps needed / machine instruction = 20

Solution:

$$T = \frac{N \times S}{R}$$ = (1000*20)/800 * 10^6=25 micro sec or 25*10^{-6} sec

Section 4:

MACHINE INSTRUCTIONS &PROGRAMS

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 2k-1 as the addresses of successive-locations in the memory).
- ✓ If 2k = no. of addressable locations;then 2k addresses constitute the address-space of the computer. For example, a 24-bit address generates an address-space of 224 locations (16 MB).

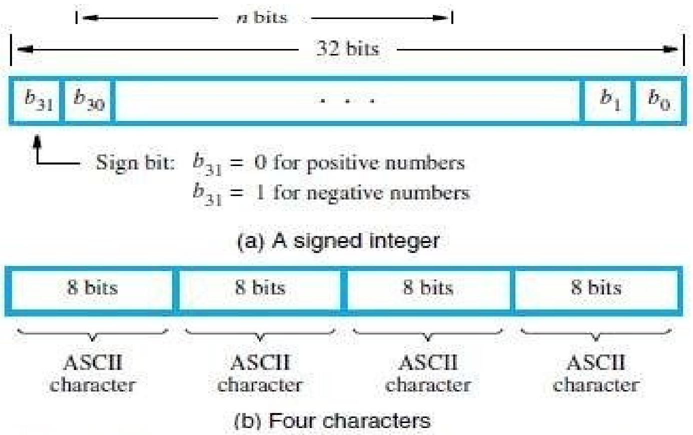


Figure 2.2 Examples of encoded information in a 32-bit word.

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).

- ✓ Big-Endian: Lower byte-addresses are used for the more significant bytes of the word.
- ✓ 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.

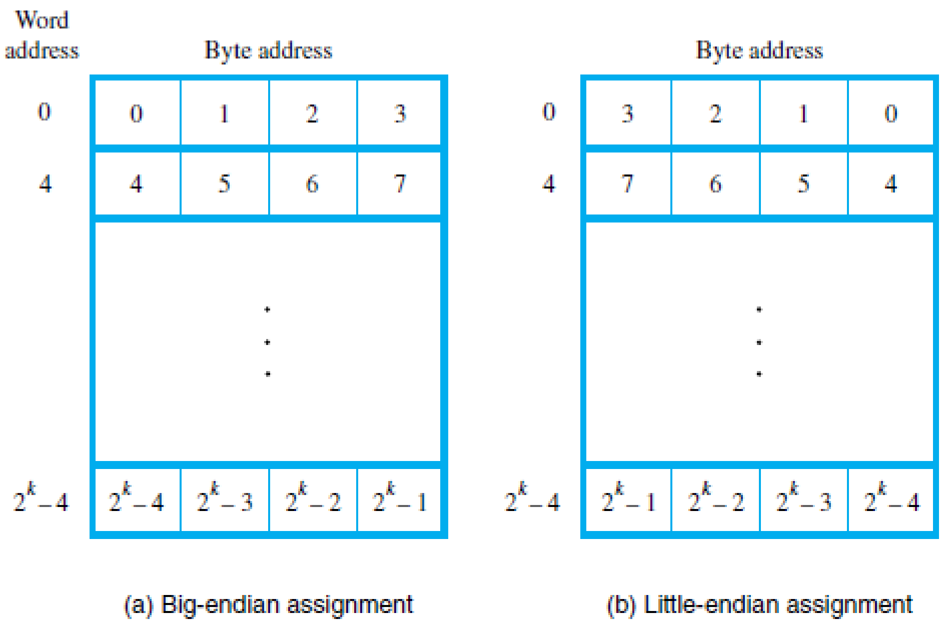

Word address
Byte address
0
4
2^k-4
0 1 2 3
4 5 6 7
2^k-4 2^k-3 2^k-2 2^k-1
(a) Big-endian assignment
Byte address
0
4
2^k-4
3 2 1 0
7 6 5 4
2^k-1 2^k-2 2^k-3 2^k-4
(b) Little-endian assignment

Figure 2.3 Byte and word addressing.

Consider a 32-bit integer (in hex): 0x12345678 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 characterin the string.
  2) A separate memory word location or register can contain a number indicating the length ofthe 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.

Section 5:

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).

REGISTER TRANSFER NOTATION (RTN)

- The possible locations in which transfer of information occurs are: 1) Memory-location 2) Processor register & 3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	R1 = [LOC]	Contents of memory-location LOCare transferred into register R1.
Processor	R0, R1 ,R2	[R3] = [R1]+[R2]	Add the contents of register R1 &R2and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	R1 =DATAIN	Contents of I/O register DATAIN are transferred into register R1.

ASSEMBLY LANGUAGE NOTATION

- To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents ofregister R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES

Instruction Type	Syntax	Example	Description	Instructions for Operation C<-[A]+[B]
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcode [no Source/Destination]	Push	Locations of all operandsare defined implicitly. The operands are stored ina pushdown stack.	Not possible

Access to data in the registers is much faster than to data stored in memory-locations.

- Let Ri represent a general-purpose register. The instructions: *Load A,Ri*
  Store Ri,A
  Add A,Ri

are generalizations of the Load, Store and Add Instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

- In processors, where arithmetic operations as allowed only on operands that are in registers, the taskC<-[A]+[B] can be performed by the instruction sequence:

Move A,Ri
Move B,Rj
Add Ri,Rj
Move Rj,C

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

- The program is executed as follows:
  1) Initially, the address of the first instruction is loaded into PC (Figure 2.8).
  2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called *Straight-Line sequencing***.**
  3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.
- There are 2 phases for Instruction Execution:
  1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.
  2) Execute Phase: The contents of IR is examined to determine which operation is to be performed. The specified-operation is then performed by the processor.

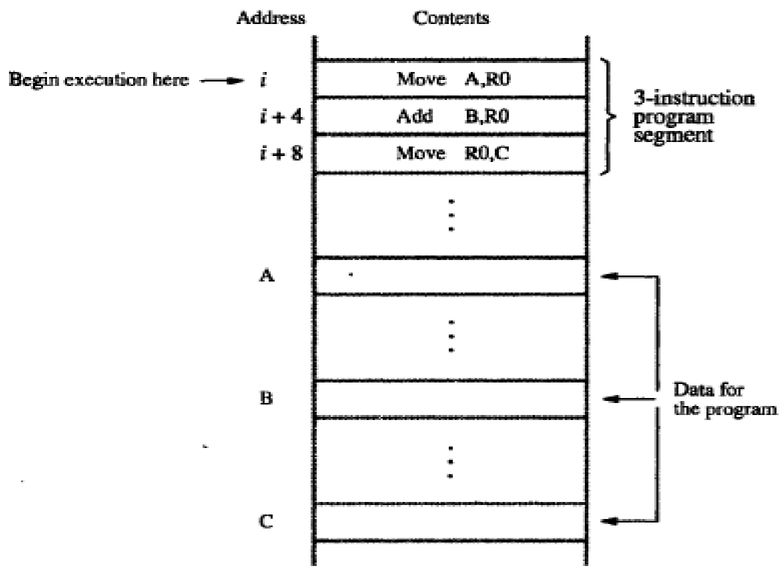


Figure 2.8 A program for C ← [A] + [B].

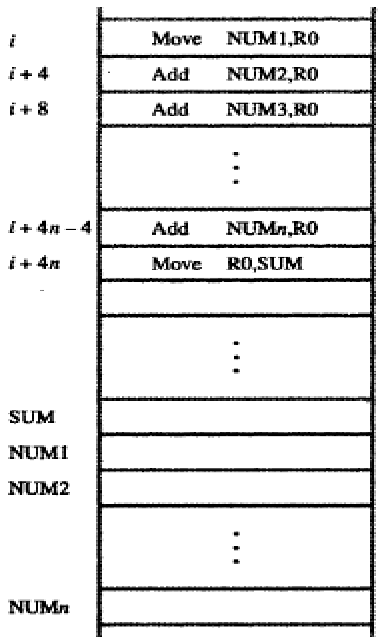


Figure 2.9 A straight-line program for adding n numbers.

Program Explanation

- Consider the program for adding a list of n numbers (Figure 2.9).
- The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2…..NUMn.
- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory-location SUM.

Section 6:

ADDRESSING MODES

- The different ways in which the location of an operand is specified in an instruction are referred to as **Addressing Modes** (Table 2.1).

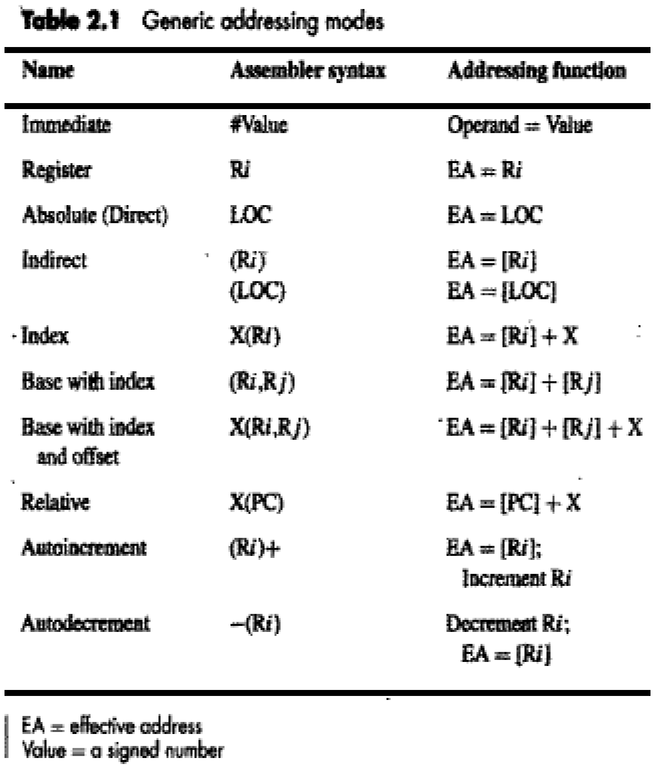


Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC

Indirect	(R*i*)
	(LOC)
		EA = [R*i*]
		EA = [LOC]

Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

IMPLEMENTATION OF VARIABLE AND CONSTANTS

- **Variable** is represented by allocating a memory-location to hold its value.
- Thus, the value can be changed as needed using appropriate instructions.
- There are 2 accessing modes to access the variables:
  1) Register Mode
  2) Absolute Mode

Register Mode

- The operand is the contents of a register.
- The name (or address) of the register is given in the instruction.
- Registers are used as temporary storage locations where the data in a register are accessed.
- For example, the instruction

Move R1, R2 ;Copy content of register R1 into register R2.

Absolute (Direct) Mode

- The operand is in a memory-location.
- The address of memory-location is given explicitly in the instruction.
- The absolute mode can represent global variables in the program.
- For example, the instruction

Move LOC, R2 ;Copy content of memory-location LOC into register R2.

Immediate Mode

- The operand is given explicitly in the instruction.
- For example, the instruction

Move #200, R0 ;Place the value 200 in register R0.

- Clearly, the immediate mode is only used to specify the value of a source-operand.

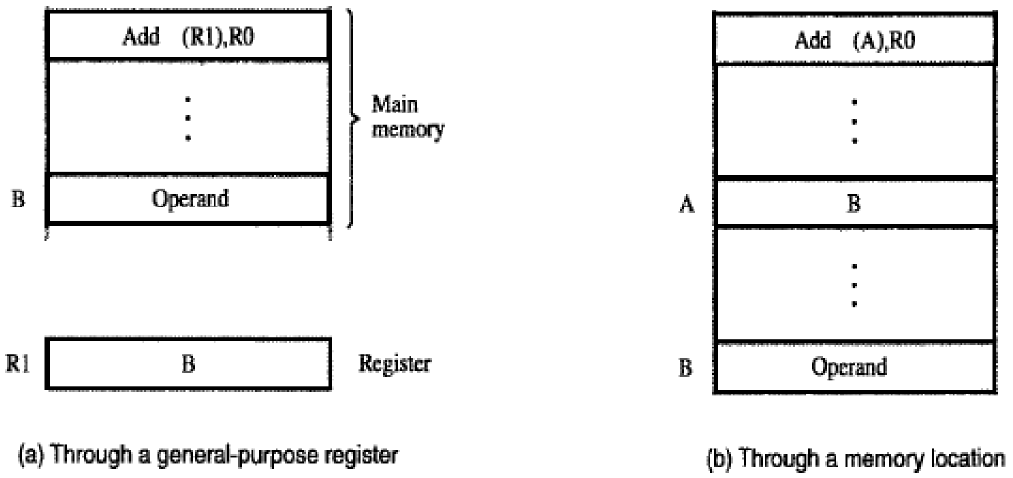
INDIRECTION AND POINTERS

- Instruction does not give the operand or its address explicitly.
- Instead, the instruction provides information from which the new address of the operand can be determined.
- This address is called **Effective Address (EA)** of the operand.

Indirect Mode

- The EA of the operand is the contents of a register(or memory-location).
- The register (or memory-location) that contains the address of an operand is called a **Pointer**.
- We denote the indirection by
  - → name of the register or
  - → new address given in the instruction.

  E.g: *Add (R1),R0* ;The operand is in memory. Register R1 gives the effective-address (B) ofthe operand. The data is read from location B and added to contents of register R0.



Add (R1),R0
Main memory
B
Operand
R1
B
Register
(a) Through a general-purpose register
Add (A),R0
A
B
B
Operand
(b) Through a memory location

Figure 2.11 Indirect addressing.

- To execute the Add instruction in fig 2.11 (a), the processor uses the value which is in register R1, asthe EA of the operand.
- It requests a read operation from the memory to read the contents of location B. The value read isthe desired operand, which the processor adds to the contents of register R0.
- Indirect addressing through a memory-location is also possible as shown in fig 2.11(b). In this case, the processor first reads the contents of memory-location A, then requests a second read operation using the value B as an address to obtain the operand.

INDEXING AND ARRAYS

- A different kind of flexibility for accessing operands is useful in dealing with lists and arrays.

Index mode

- The operation is indicated as X(Ri)

  where X=the constant value which defines an offset(also called a displacement).

  Ri=the name of the index register which contains address of a new location.
- The effective-address of the operand is given by EA=X+[Ri]
- The contents of the index-register are not changed in the process of generating the effective-address.
- The constant X may be given either
  - → as an explicit number or
  - → as a symbolic-name representing a numerical value.

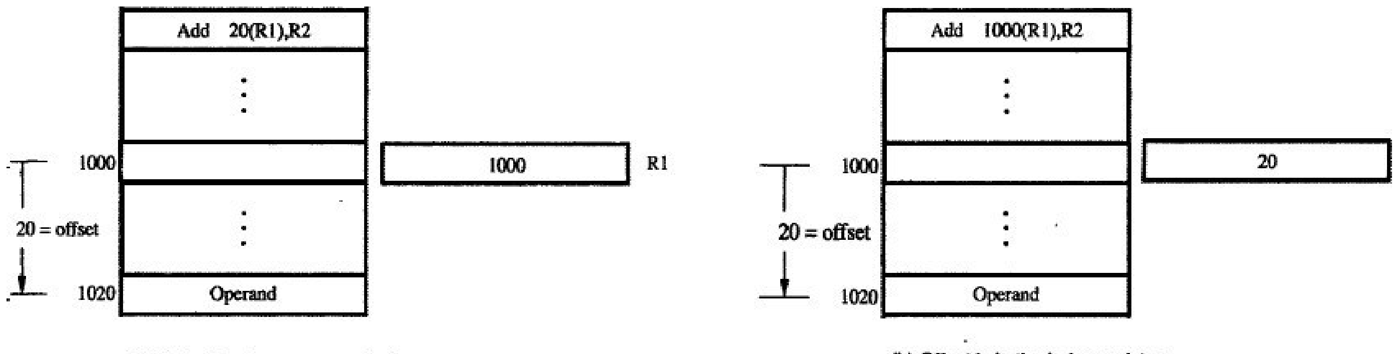


Figure 2.13 Indexed addressing.

• Fig(a) illustrates two ways of using the Index mode. In fig(a), the index register, R1, contains the address of a memory-location, and the value X defines an offset(also called a displacement) from this address to the location where the operand is found.
• To find EA of operand:

Eg: Add 20(R1), R2

EA=>1000+20=1020

• An alternative use is illustrated in fig(b). Here, the constant X corresponds to a memory address, andthe contents of the index register define the offset to the operand. In either case, the effective-addressis the sum of two values; one is given explicitly in the instruction, and the other is stored in aregister.

Base with Index Mode

• Another version of the Index mode uses 2 registers which can be denoted as(Ri, Rj)
• Here, a second register may be used to contain the offset X.
• The second register is usually called the *base register*.
• The effective-address of the operand is given by EA=[Ri]+[Rj]
• This form of indexed addressing provides more flexibility in accessing operands because both components of the effective-address can be changed.

Base with Index & Offset Mode

• Another version of the Index mode uses 2 registers plus a constant, which can be denoted asX(Ri, Rj)
• The effective-address of the operand is given by EA=X+[Ri]+[Rj]
• This added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (Ri, Rj) part of the addressing-mode. In other words, this mode implements a 3-dimensional array.

RELATIVE MODE

• This is similar to index-mode with one difference:

The effective-address is determined using the PC in place of the general purpose register Ri.

• The operation is indicated as X(PC).
• X(PC) denotes an effective-address of the operand which is X locations above or below the current contents of PC.
• Since the addressed-location is identified "relative" to the PC, the name Relative mode is associatedwith this type of addressing.
• This mode is used commonly in conditional branch instructions.
• An instruction such as

Branch > 0 LOOP ;Causes program execution to go to the branch target locationidentified by name LOOP if branch condition is satisfied.

ADDITIONAL ADDRESSING MODES

1) Auto Increment Mode

➢ Effective-address of operand is contents of a register specified in the instruction (Fig: 2.16).
➢ After accessing the operand, the contents of this register are automatically incremented topoint to the next item in a list.
➢ Implicitly, the increment amount is 1.
➢ This mode is denoted as

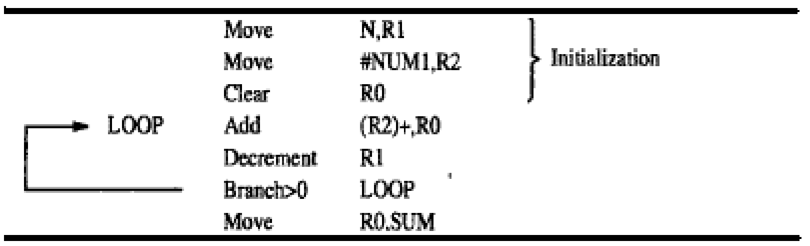
(Ri)+ ;where Ri=pointer-register.

2) Auto Decrement Mode

➢ The contents of a register specified in the instruction are first automatically decremented andare then used as the effective-address of the operand.
➢ This mode is denoted as

-(Ri) ;where Ri=pointer-register.

➢ These 2 modes can be used together to implement an important data structure called a stack.



```
              Move        N,R1        }
              Move        #NUM1,R2    } Initialization
              Clear       R0          }
      LOOP    Add         (R2)+,R0
              Decrement   R1
              Branch>0    LOOP
              Move        R0,SUM
```

Figure 2.16 The Autoincrement addressing mode used in the program of Figure 2.12.

ASSEMBLY LANGUAGE

• We generally use symbolic-names to write a program.
• A complete set of symbolic-names and rules for their use constitute an **Assembly Language**.
• The set of rules for using the mnemonics in the specification of complete instructions and programs is called the **Syntax** of the language.
• Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an **Assembler.**
• The user program in its original alphanumeric text formal is called a **Source Program**, and the assembled machine language program is called an **Object Program**.

For example:

MOVE R0,SUM ;The term MOVE represents OP code for operation performed by instruction.
ADD #5,R3 ;Adds number 5 to contents of register R3 & puts the result back into registerR3.

ASSEMBLER DIRECTIVES

• **Directives** are the assembler commands to the assembler concerning the program being assembled.
• These commands are not translated into machine opcode in the object-program
• **EQU** informs the assembler about the value of an identifier (Figure: 2.18).
Ex*: SUM EQU 200* ;Informs assembler that the name SUM should be replaced by the value 200.
• **ORIGIN** tells the assembler about the starting-address of memory-area to place the data block.
Ex*:* ORIGIN 204 ;Instructs assembler to initiate data-block at memory-locations starting from 204.
• **DATAWORD** directive tells the assembler to load a value into the location.
Ex: *N DATAWORD 100 ;*Informs the assembler to load data 100 into the memory-location N(204).
• **RESERVE** directive is used to reserve a block of memory.
Ex: *NUM1 RESERVE 400 ;*declares a memory-block of 400 bytes is to be reserved for data.
• **END** directive tells the assembler that this is the end of the source-program text.

• **RETURN** directive identifies the point at which execution of the program should be terminated.
• Any statement that makes instructions or data being placed in a memory-location may be given a **label**. The label(say N or NUM1) is assigned a value equal to the address of that location.

	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	START

Figure 2.18 Assembly language representation for the program in Figure 2.17.

GENERAL FORMAT OF A STATEMENT

• Most assembly languages require statements in a source program to be written in the form:

Label	*Operation*	*Operands*	*Comment*

1) **Label** is an optional name associated with the memory-address where the machine language instruction produced from the statement will be loaded.
2) **Operation Field** contains the OP-code mnemonic of the desired instruction or assembler.
3) **Operand Field** contains addressing information for accessing one or more operands, depending on the type of instruction.
4) **Comment Field** is used for documentation purposes to make program easier to understand.

ASSEMBLY AND EXECUTION OF PRGRAMS

• Programs written in an assembly language are automatically translated into a sequence of machine instructions by the **Assembler**.
• **Assembler Program**
→ replaces all symbols denoting operations & addressing-modes with binary-codes used in machine instructions.
→ replaces all names and labels with their actual values.
→ assigns addresses to instructions & data blocks, starting at address given in ORIGIN directive
→ inserts constants that may be given in DATAWORD directives.
→ reserves memory-space as requested by RESERVE directives.
• **Two Pass Assembler** has 2 passes:
1) **First Pass:** Work out all the addresses of labels.
➢ As the assembler scans through a source-program, it keeps track of all names of numerical-values that correspond to them in a symbol-table.
2) **Second Pass:** Generate machine code, substituting values for the labels.
➢ When a name appears a second time in the source-program, it is replaced with its value fromthe table.

• The assembler stores the object-program on a magnetic-disk. The object-program must be loadedinto the memory of the computer before it is executed. For this, a **Loader Program** is used.
• **Debugger Program** is used to help the user find the programming errors.
• Debugger program enables the user
→ to stop execution of the object-program at some points of interest &
→ to examine the contents of various processor-registers and memory-location.

Section 7:

BASIC INPUT/OUTPUT OPERATIONS

• Consider the problem of moving a character-code from the keyboard to the processor (Figure: 2.19).For this transfer, buffer-register DATAIN & a status control flags(SIN) are used.
• When a key is pressed, the corresponding ASCII code is stored in a **DATAIN** register associated withthe keyboard.
➢ **SIN=1** → When a character is typed in the keyboard. This informs the processor that a valid character is in DATAIN.
➢ **SIN=0** → When the character is transferred to the processor.
• An analogous process takes place when characters are transferred from the processor to the display.For this transfer, buffer-register DATAOUT & a status control flag SOUT are used.
➢ **SOUT=1** → When the display is ready to receive a character.
➢ **SOUT=0** → When the character is being transferred to DATAOUT.
• The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry

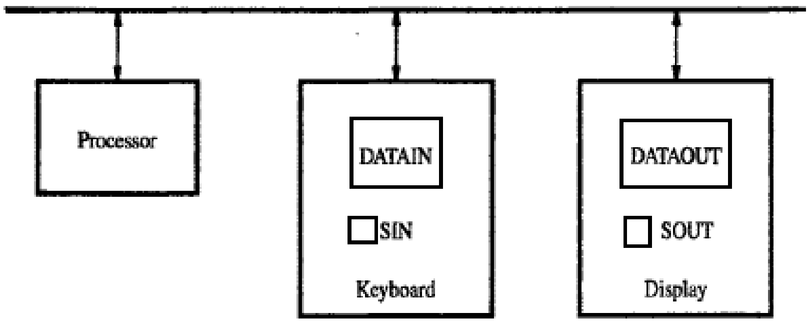
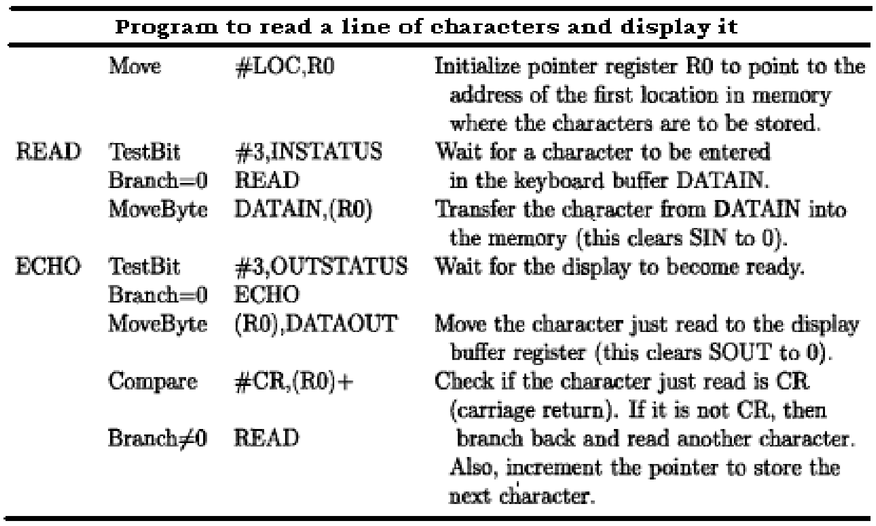


Figure 2.19 Bus connection for processor, keyboard, and display.

commonly known as a **device interface**.

MEMORY-MAPPED I/O



Program to read a line of characters and display it

```
       Move        #LOC,R0         Initialize pointer register R0 to point to the
                                     address of the first location in memory
                                     where the characters are to be stored.
READ   TestBit     #3,INSTATUS     Wait for a character to be entered
       Branch=0    READ              in the keyboard buffer DATAIN.
       MoveByte    DATAIN,(R0)     Transfer the character from DATAIN into
                                     the memory (this clears SIN to 0).
ECHO   TestBit     #3,OUTSTATUS    Wait for the display to become ready.
       Branch=0    ECHO
       MoveByte    (R0),DATAOUT    Move the character just read to the display
                                     buffer register (this clears SOUT to 0).
       Compare     #CR,(R0)+       Check if the character just read is CR
                                     (carriage return). If it is not CR, then
       Branch≠0    READ              branch back and read another character.
                                     Also, increment the pointer to store the
                                     next character.
```

Figure 2.20 A program that reads a line of characters and displays it.

- Some address values are used to refer to peripheral device buffer-registers such as DATAIN & DATAOUT.
- No special instructions are needed to access the contents of the registers; data can be transferred between these registers and the processor using instructions such as Move, Load or Store.
- For example, contents of the keyboard character buffer DATAIN can be transferred to register R1 inthe processor by the instruction

    MoveByte DATAIN,R1
- The MoveByte operation code signifies that the operand size is a byte.
- The **Testbit** instruction tests the state of one bit in the destination, where the bit position to betested is indicated by the first operand.

STACKS

- A **stack** is a special type of data structure where elements are inserted from one end and elementsare deleted from the same end. This end is called the **top** of the stack (Figure: 2.14).
- The various operations performed on stack:
    1) Insert: An element is inserted from top end. Insertion operation is called **push** operation.
    2) Delete: An element is deleted from top end. Deletion operation is called **pop** operation.
- A processor-register is used to keep track of the address of the element of the stack that is at the topat any given time. This register is called the **Stack Pointer (SP)**.
- If we assume a byte-addressable memory with a 32-bit word length,
    1) The push operation can be implemented as

        Subtract #4, SP Move
        NEWITEM, (SP)
    2) The pop operation can be implemented as

        Move (SP), ITEM
        Add #4, SP

Routine for a safe pop and push operation as follows:

```
SAFEPOP   Compare    #2000,SP      Check to see if the stack pointer contains
          Branch>0   EMPTYERROR      an address value greater than 2000. If it
                                     does, the stack is empty. Branch to the
                                     routine EMPTYERROR for appropriate
                                     action.
          Move       (SP)+,ITEM    Otherwise, pop the top of the stack into
                                     memory location ITEM.
```

(a) Routine for a safe pop operation

```
SAFEPUSH   Compare    #1500,SP        Check to see if the stack pointer
           Branch≤0   FULLERROR         contains an address value equal
                                        to or less than 1500. If it does, the
                                        stack is full. Branch to the routine
                                        FULLERROR for appropriate action.
           Move       NEWITEM,−(SP)   Otherwise, push the element in memory
                                        location NEWITEM onto the stack.
```

(b) Routine for a safe push operation

Figure 2.23 Checking for empty and full errors in pop and push operations.

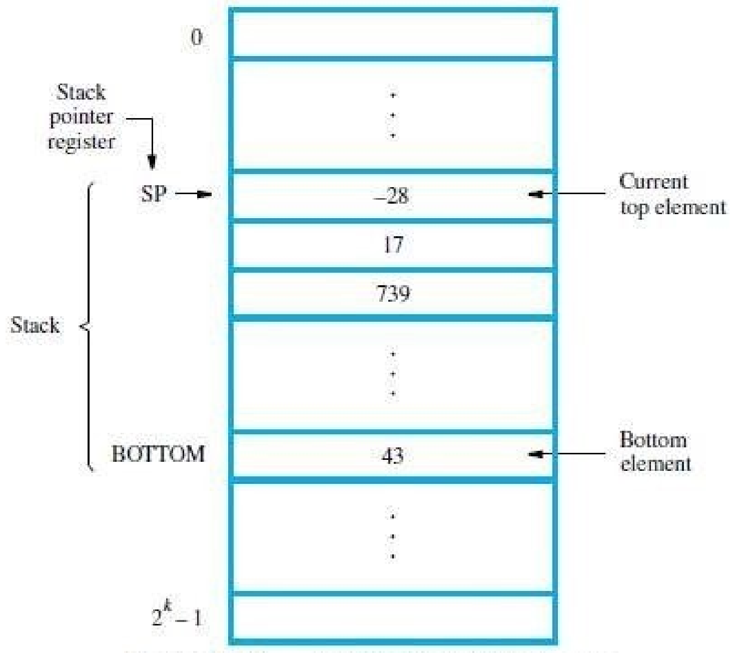


Figure 2.14 A stack of words in the memory.

QUEUE

- Data are stored in and retrieved from a queue on a FIFO basis.
- Difference between stack and queue?
  1) One end of the stack is fixed while the other end rises and falls as data are pushed and popped.
  2) In stack, a single pointer is needed to keep track of top of the stack at any given time.
     In queue, two pointers are needed to keep track of both the front and end for removal and insertion respectively.
  3) Without further control, a queue would continuously move through the memory of acomputer in the direction of higher addresses. One way to limit the queue to a fixed region in memory is to use a circular buffer.

SUBROUTINES

- A subtask consisting of a set of instructions which is executed many times is called a **Subroutine**.
- A Call instruction causes a branch to the subroutine (Figure: 2.16).
- At the end of the subroutine, a return instruction is executed
- Program resumes execution at the instruction immediately following the subroutine call
- The way in which a computer makes it possible to call and return from subroutines is referred to asits **Subroutine Linkage** method.
- The simplest subroutine linkage method is to save the return-address in a specific location, whichmay be a register dedicated to this function. Such a register is called the **Link Register**.
- When the subroutine completes its task, the Return instruction returns to the calling-program by branching indirectly through the link-register.
- The **Call Instruction** is a special branch instruction that performs the following operations:
  → Store the contents of PC into link-register.
  → Branch to the target-address specified by the instruction.
- The **Return Instruction** is a special branch instruction that performs the operation:
  → Branch to the address contained in the link-register.

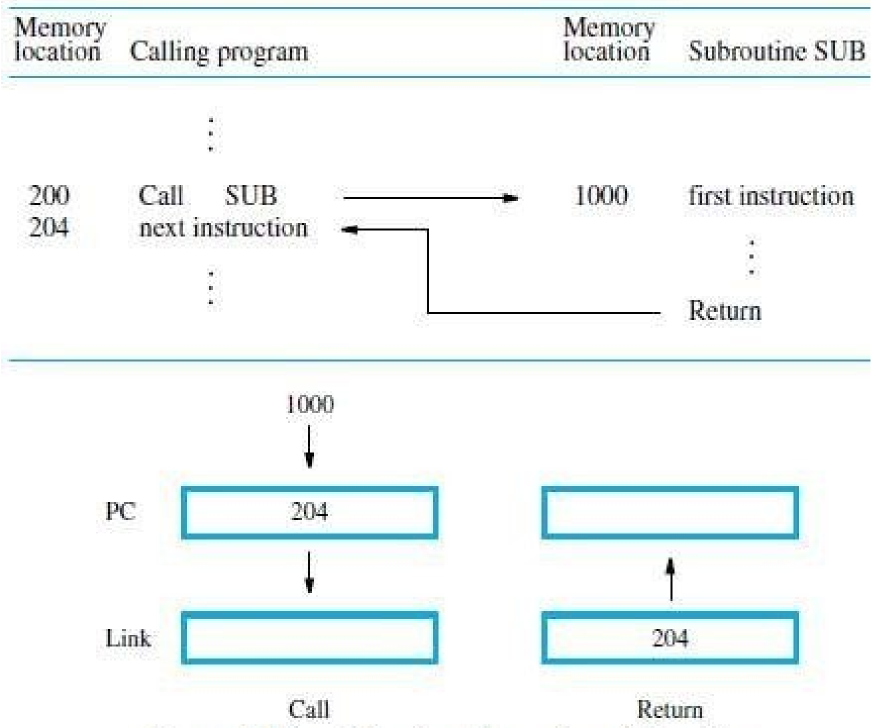


Figure 2.16 Subroutine linkage using a link register.

Section 8:

LOGIC INSTRUCTIONS

- Logic operations such as AND, OR, and NOT applied to individual bits.
- These are the basic building blocks of digital-circuits.
- This is also useful to be able to perform logic operations is software, which is done using instructionsthat apply these operations to all bits of a word or byte independently and in parallel.
- For example, the instruction

  Not dst

SHIFT AND ROTATE INSTRUCTIONS

- There are many applications that require the bits of an operand to be shifted right or left somespecified number of bit positions.
- The details of how the shifts are performed depend on whether the operand is a signed number orsome more general binary-coded information.
- For general operands, we use a logical shift.

  For a number, we use an arithmetic shift, which preserves the sign of the number.

LOGICAL SHIFTS

- Two logical shift instructions are
  1) Shifting left (LShiftL) &
  2) Shifting right (LShiftR).
- These instructions shift an operand over a number of bit positions specified in a count operandcontained in the instruction.

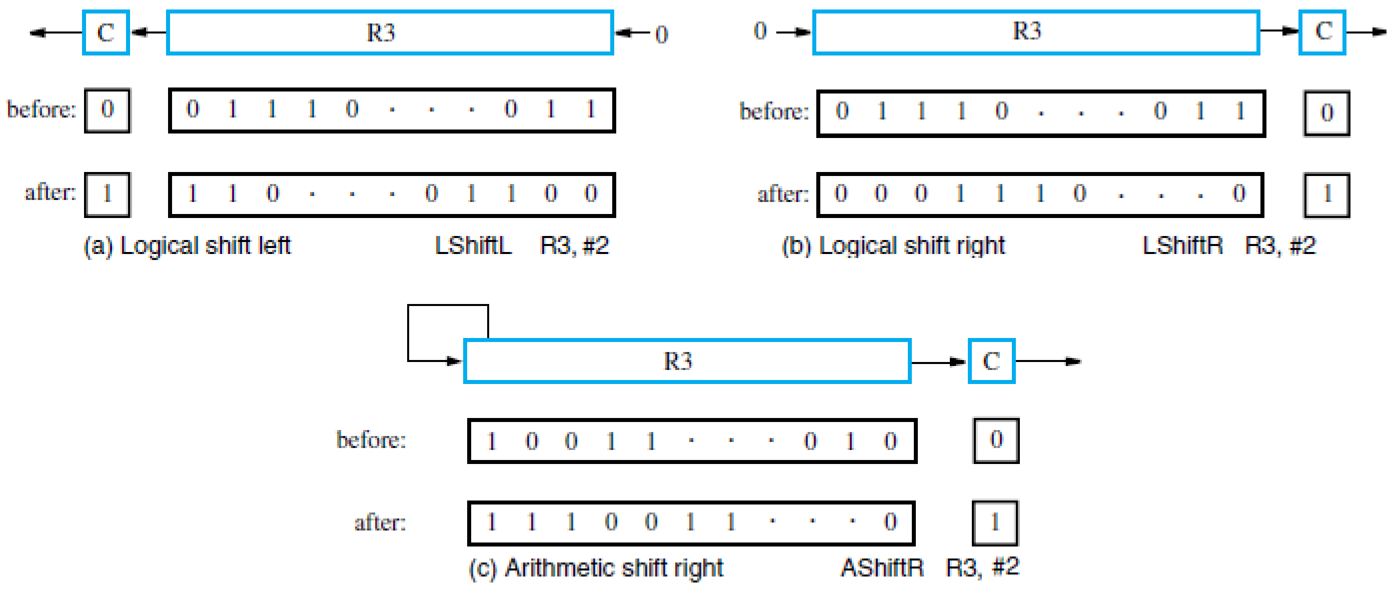


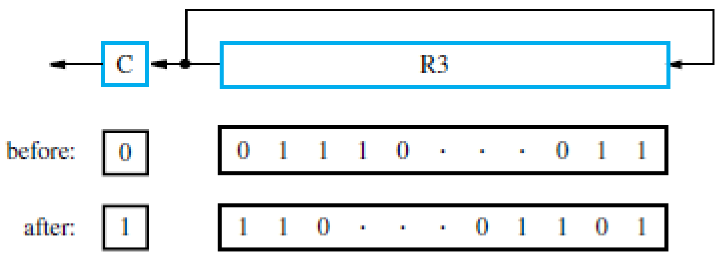
Figure 2.23 Logical and arithmetic shift instructions.

```
Move        #LOC,R0      R0 points to data.
MoveByte    (R0)+,R1     Load first byte into R1.
LShiftL     #4,R1        Shift left by 4 bit positions.
MoveByte    (R0),R2      Load second byte into R2.
And         #$F,R2       Eliminate high-order bits.
Or          R1,R2        Concatenate the BCD digits.
MoveBvte    R2.PACKED    Store the result.
```

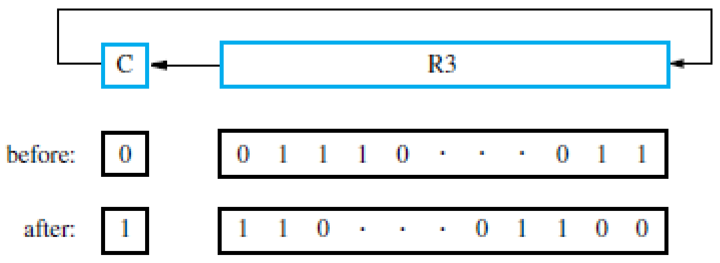
Figure 2.31 A routine that packs two BCD digits.

ROTATE OPERATIONS

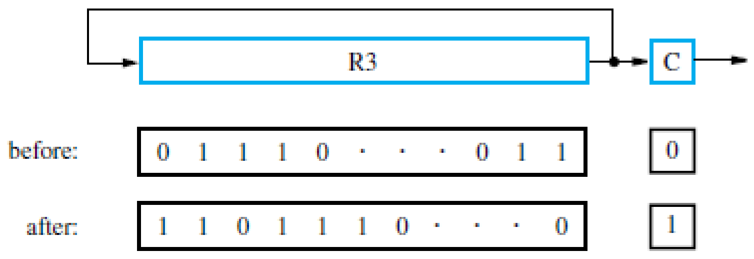
- In shift operations, the bits shifted out of the operand are lost, except for the last bit shifted outwhich is retained in the Carry-flag C.
- To preserve all bits, a set of rotate instructions can be used.
- They move the bits that are shifted out of one end of the operand back into the other end.
  - Two versions of both the left and right rotate instructions are usually provided.In one version, the bits of the operand is simply rotated. In the other version, the rotation includes the C flag.



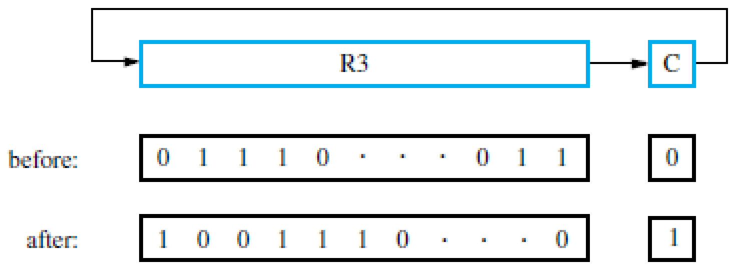
(a) Rotate left without carry RotateL R3, #2



(b) Rotate left with carry RotateLC R3, #2



(c) Rotate right without carry RotateR R3, #2



(d) Rotate right with carry RotateRC R3, #2

Figure 2.25 Rotate instructions.

ENCODING OF MACHINE INSTRUCTIONS

- To be executed in a processor, an instruction must be encoded in a binary-pattern. Such encoded instructions are referred to as **Machine Instructions**.
- The instructions that use symbolic-names and acronyms are called *assembly language instructions*.
- We have seen instructions that perform operations such as add, subtract, move, shift, rotate, andbranch. These instructions may use operands of different sizes, such as 32-bit and 8-bit numbers.
- Let us examine some typical cases.

The instruction

Add R1, R2 ;Has to specify the registers R1 and R2, in addition to the OP code. If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register addressing-mode is used for each operand.

The instruction

Move 24(R0), R5 ;Requires 16 bits to denote the OP code and the two registers, andsome bits to express that the source operand uses the Index addressing mode and that the index value is 24.

- In all these examples, the instructions can be encoded in a 32-bit word (Fig 2.39).
- The OP code for given instruction refers to type of operation that is to be performed.
- Source and destination field refers to source and destination operand respectively.
- The "Other info" field allows us to specify the additional information that may be needed such as anindex value or an immediate operand.
- Using multiple words, we can implement complex instructions, closely resembling operations in high- level programming languages. The term complex instruction set computers (CISC) refers to processors that use
- CISC approach results in instructions of variable length, dependent on the number of operands andthe type of addressing modes used.
- In RISC (reduced instruction set computers), any instruction occupies only one word.
- The RISC approach introduced other restrictions such as that all manipulation of data must be doneon operands that are already in registers.

Ex: Add R1,R2,R3

- In RISC type machine, the memory references are limited to only Load/Store operations.

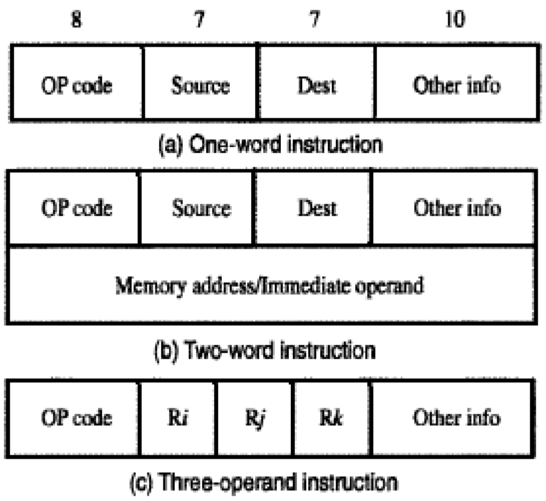


Figure 2.39 Encoding instructions into 32-bit words.

Section 9:

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.

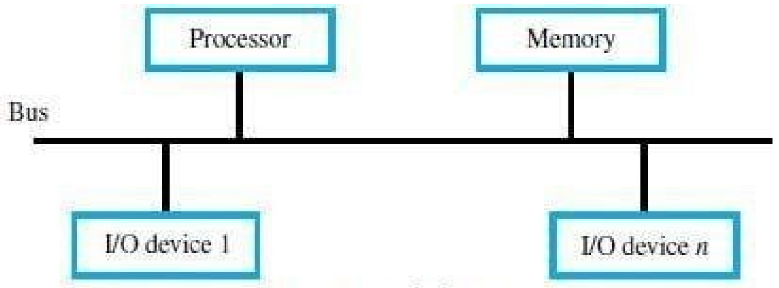


Figure 7.1 A single-bus structure.

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 address appears 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 adisplay/printer.

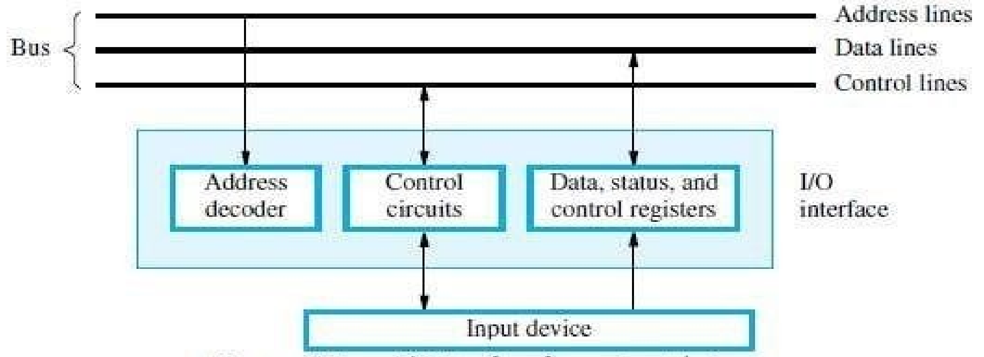


Figure 7.2 I/O interface for an input device.

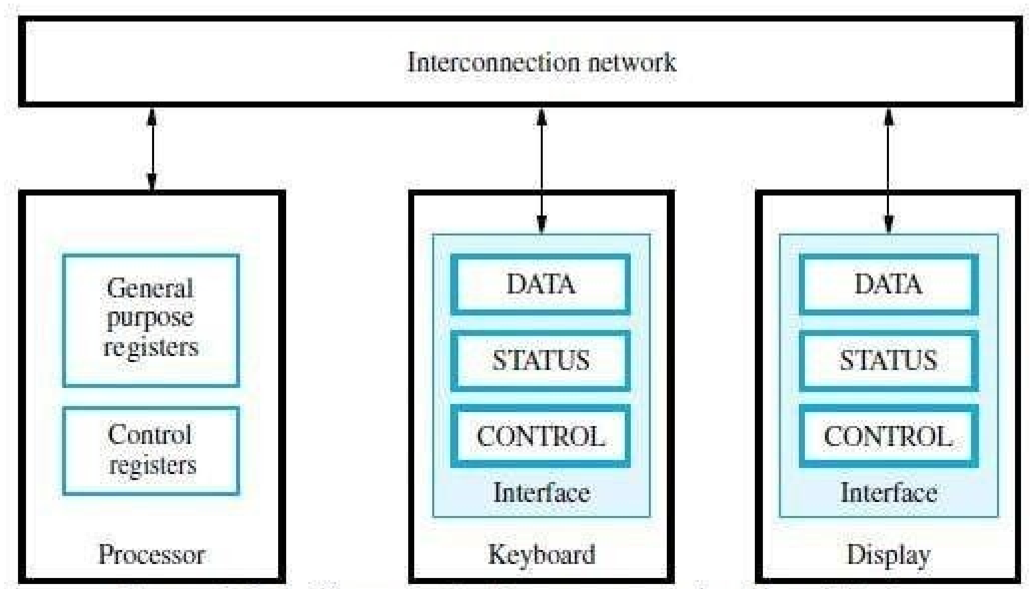


Figure 3.2 The connection for processor, keyboard, and display.

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/O device. (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.

Section 10:

INTERRUPTS

- There are many situations where other tasks can be performed while waiting for an I/O device to become 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).

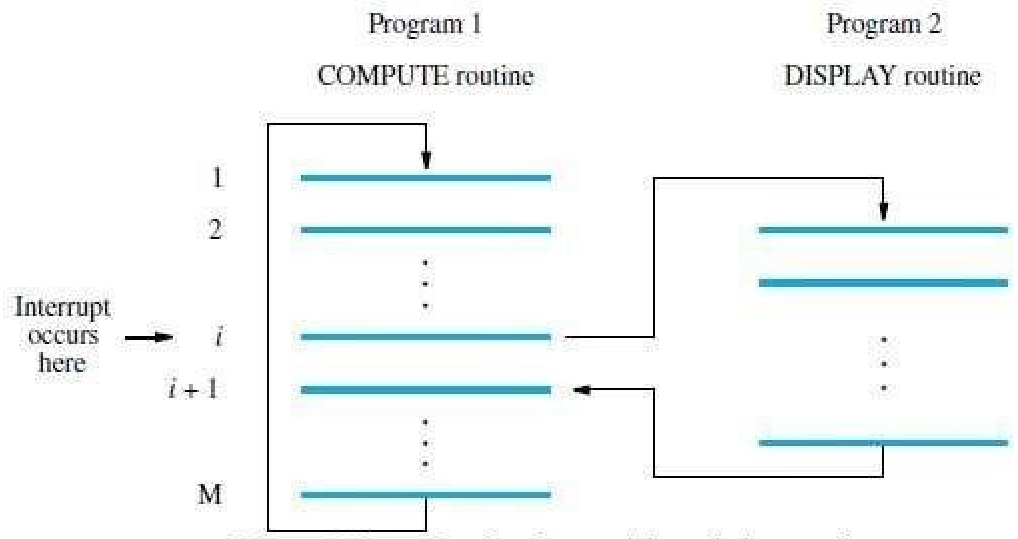


Figure 3.6 Transfer of control through the use of interrupts.

- 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.

Difference between Subroutine & ISR

Subroutine	ISR
A subroutine performs a function required by the program from which it is called.	ISR may not have anything in common with program being executed at time INTR is received
Subroutine is just a linkage of 2 or more function related to each other.	Interrupt is a mechanism for coordinating I/O transfers.

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 gates 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.

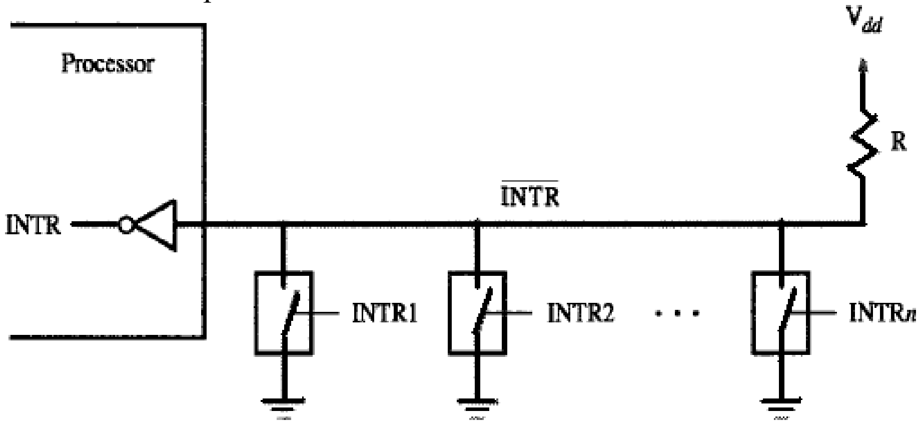


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

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:
  4) How can the processor recognize the device requesting an interrupt?
  5) How can the processor obtain the starting address of the appropriate ISR?
  6) Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  7) 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.

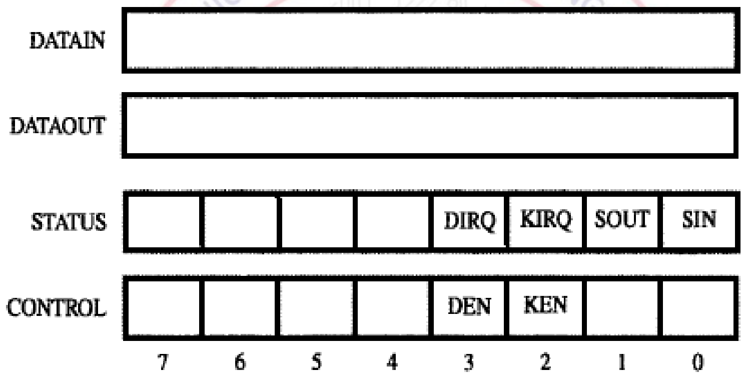


Figure 4.3 Registers in keyboard and display interfaces.

```
      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 processorover 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 toISR.
- 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 INTRsignal.
- 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).

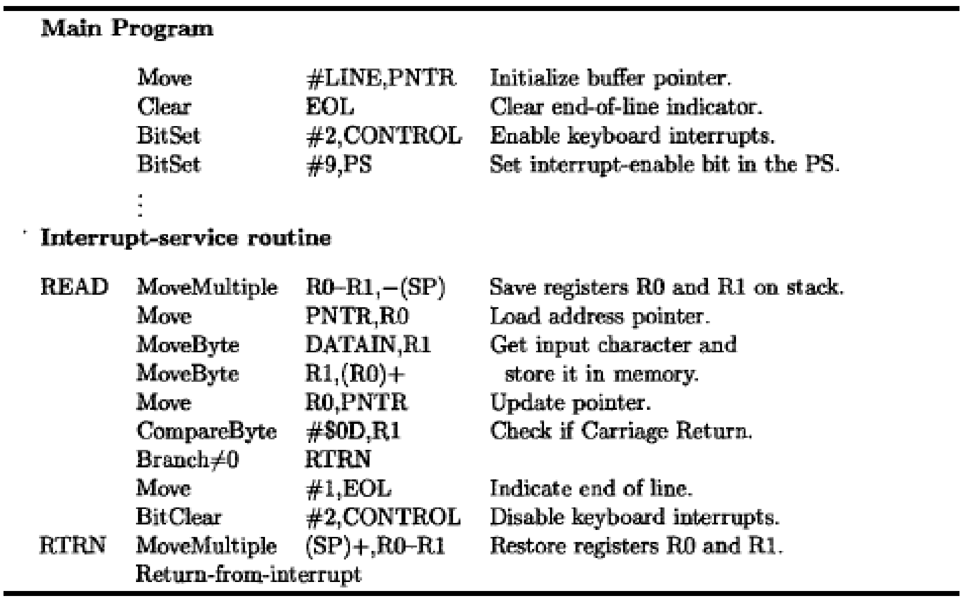
1) At Device-end

IE bit in a control-register determines whether device is allowed to generate aninterrupt-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 foreach 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 ofthe 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 **SupervisorMode**.
- 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 **PrivilegedException.**

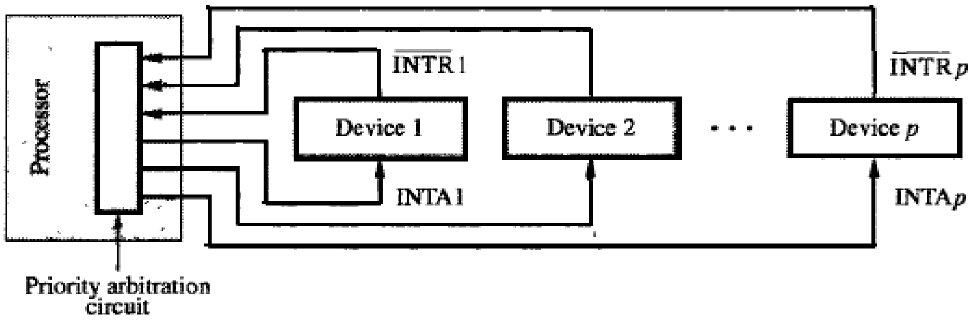


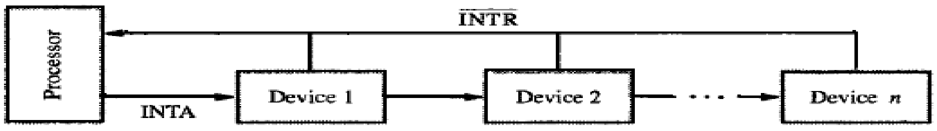
Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

SIMULTANEOUS REQUESTS

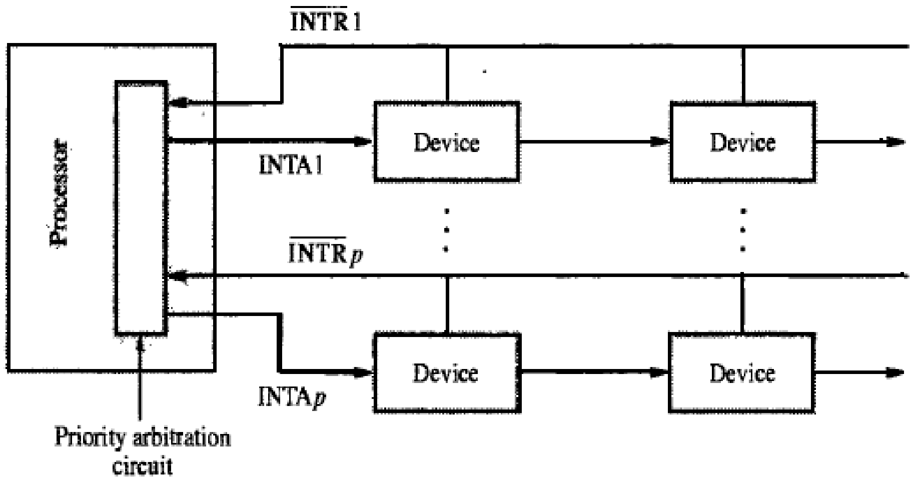
- The processor must have some mechanisms to decide which request to servicewhen simultaneous requests arrive.
- INTR line is common to all devices (Figure 4.8a).
- INTA line is connected in a daisy-chain fashion.
- INTA signal propagates serially through devices.
- When several devices raise an interrupt-request, 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 differentpriority level.
- Within a group, devices are connected in a daisy chain. (Figure 4.8b).



(a) Daisy chain



(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 executionof every instruction(using debugging-program as ESR).
- Debugging-program enables user to examine contents of registers, memory-locations andso on.
- On return from debugging-program, next instruction in program being debugged isexecuted, 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 priorityof processor.
- An attempt to execute privileged-instruction will produce a **Privilege Exception**.
- As a result, processor switches to supervisor-mode & begins to execute an appropriateroutine in OS.

Section 11:

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 anotherprogram.

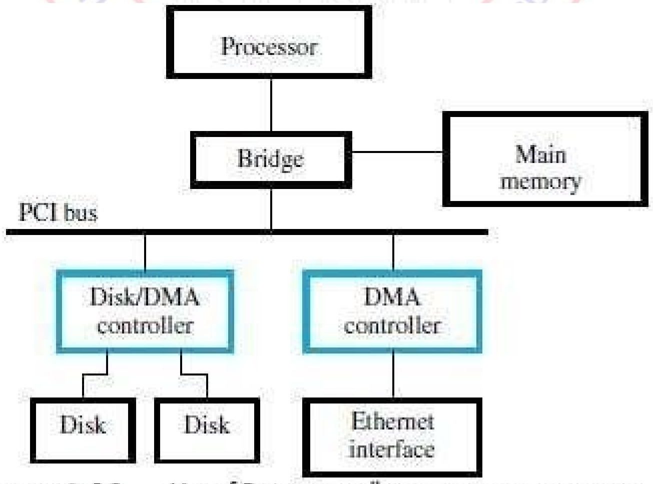


Figure 8.13 Use of DMA controllers in a computer system.

- 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.

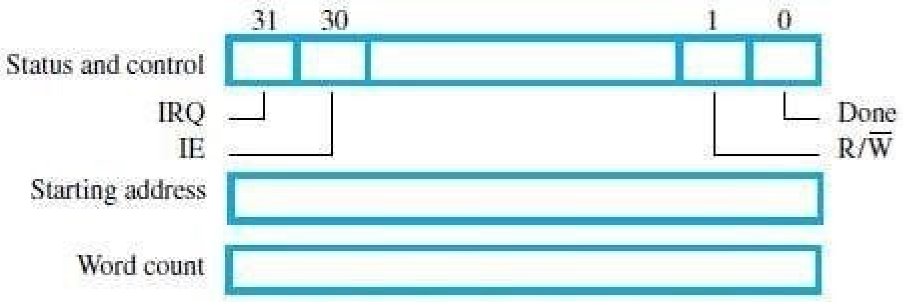


Figure 8.12 Typical registers in a DMA controller.

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 thanprocessor 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 without any interruption. This is known as **Block Mode** (or burst mode).

BUS ARBITRATION

- The device that is allowed to initiate data-transfers on bus at any given time is called **bus-master.**
- There can be only one bus-master at any given time.
- **Bus Arbitration** is the process by which
  → next device to become the bus-master is selected &
  → bus-mastership is transferred to that device.
- The two approaches are:
  1) Centralized Arbitration: A single bus-arbiter performs the required arbitration.
  2) Distributed Arbitration: All devices participate in selection of next bus-master.
- A conflict may arise if both the processor and a DMA controller or two DMAcontrollers try to use thebus at the same time to access the main-memory.
- To resolve this, an arbitration procedure is implemented on the bus to coordinatethe activities of alldevices requesting memory transfers.
- The bus arbiter may be the processor or a separate unit connected to the bus.

CENTRALIZED ARBITRATION

- A single bus-arbiter performs the required arbitration (Figure: 4.20).
- Normally, processor is the bus-master.
- Processor may grant bus-mastership to one of the DMA controllers.
- A DMA controller indicates that it needs to become bus-master by activating BR line.
- The signal on the BR line is the logical OR of bus-requests from all devices connected toit.
- Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free.
- BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.
- If DMA controller-1 is requesting the bus, Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.
- Current bus-master indicates to all devices that it is using bus by activating BBSY line.
- The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers.
- Arbiter ensures that only 1 request is granted at any given time according to a priority scheme. (BR Bus-Request, BG Bus-Grant, BBSY Bus Busy).

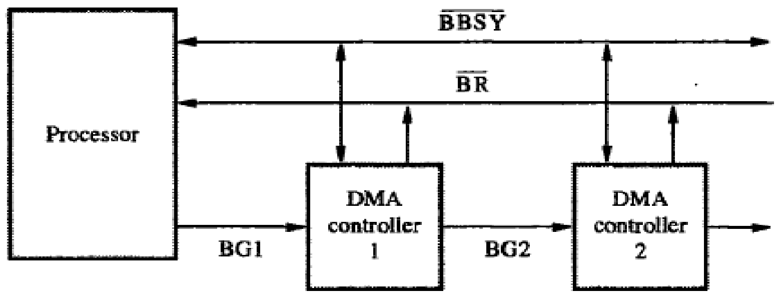


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

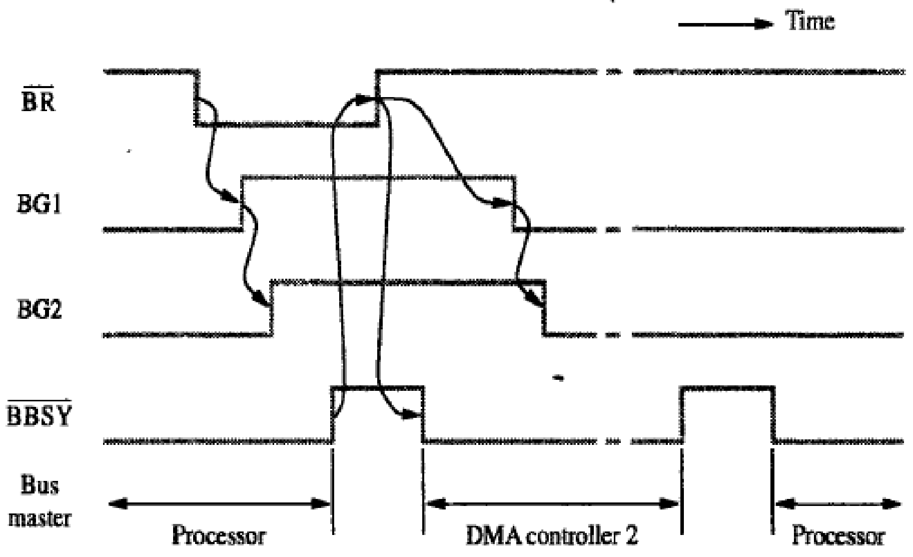


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

- The timing diagram shows the sequence of events for the devices connected to theprocessor.
- DMA controller-2
  - → requests and acquires bus-mastership and
  - → later releases the bus. (Figure: 4.21).
- After DMA controller-2 releases the bus, the processor resources bus-mastership.

DISTRIBUTED ARBITRATION

- All device participate in the selection of next bus-master (Figure 4.22).
- Each device on bus is assigned a 4-bit identification number (ID).
- When 1 or more devices request bus, they
  - → assert Start-Arbitration signal &

→ place their 4-bit ID numbers on four open-collector lines ARB 0 through ARB 3

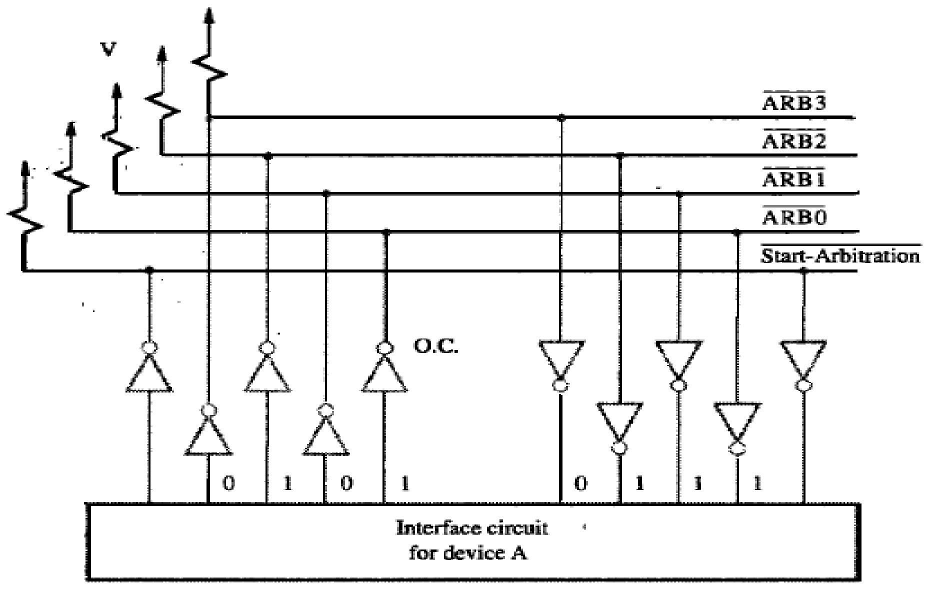


Figure 4.22 A distributed arbitration scheme.

- A winner is selected as a result of interaction among signals transmitted over these lines.
- Net-outcome is that the code on 4 lines represents request that has the highest ID number.
- **Advantage:**
  This approach offers higher reliability since operation of bus is not dependent on anysingle device.

For example:
- Assume 2 devices A & B have their ID 5 (0101), 6 (0110) and their code is 0111.
- Each device compares the pattern on the arbitration line to its own ID starting fromMSB.
- If the device detects a difference at any bit position, it disables the drivers at that bitposition.
- Driver is disabled by placing ”0” at the input of the driver.
- In e.g. “A” detects a difference in line ARB1, hence it disables the drivers on linesARB1 & ARB0.
- This causes pattern on arbitration-line to change to 0110. Thismeans that “B” has won contention.

Section 12:

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 thebuses.
- 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 dataon 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 thebus.
  - ➢ 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 & electricalcharacteristics.
- Clock pulse width(t1-t0) must be longer than max. propagation-delay b/w devicesconnected 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 changingstate.
- 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 itsinput-buffer
  - For data to be loaded correctly into a storage device, data must be available at inputof that device for a period greater than setup-time of device.

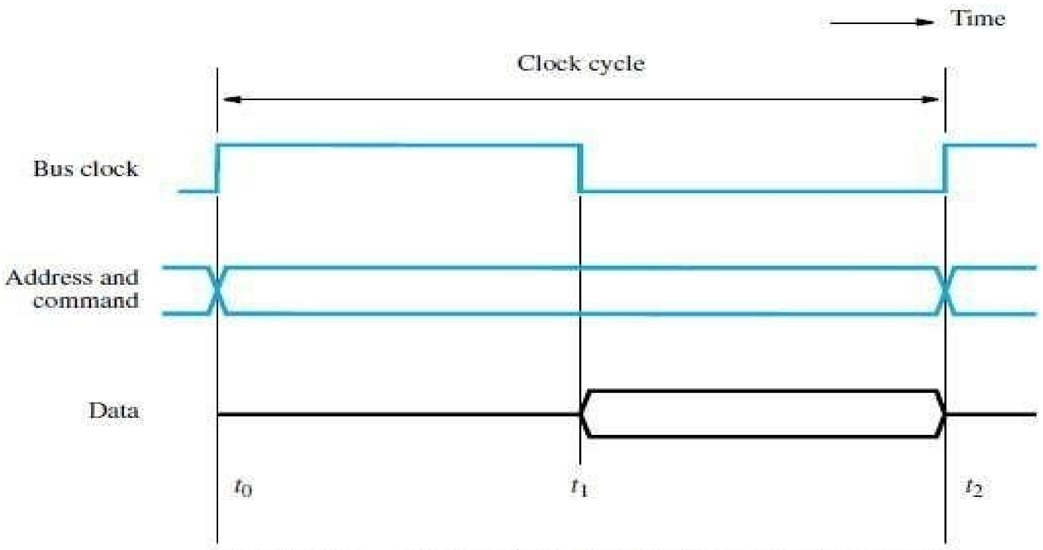


Figure 7.3 Timing of an input transfer on a synchronous bus.

A Detailed Timing Diagram for the Read-operation

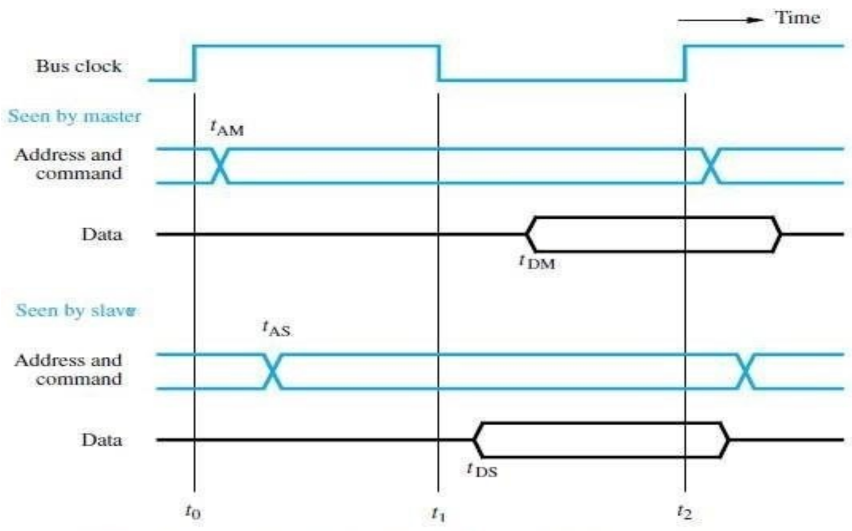


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

- 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 clockperiod (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 thatbuffers.

Disadvantages

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

Multiple Cycle Transfer for Read-operation

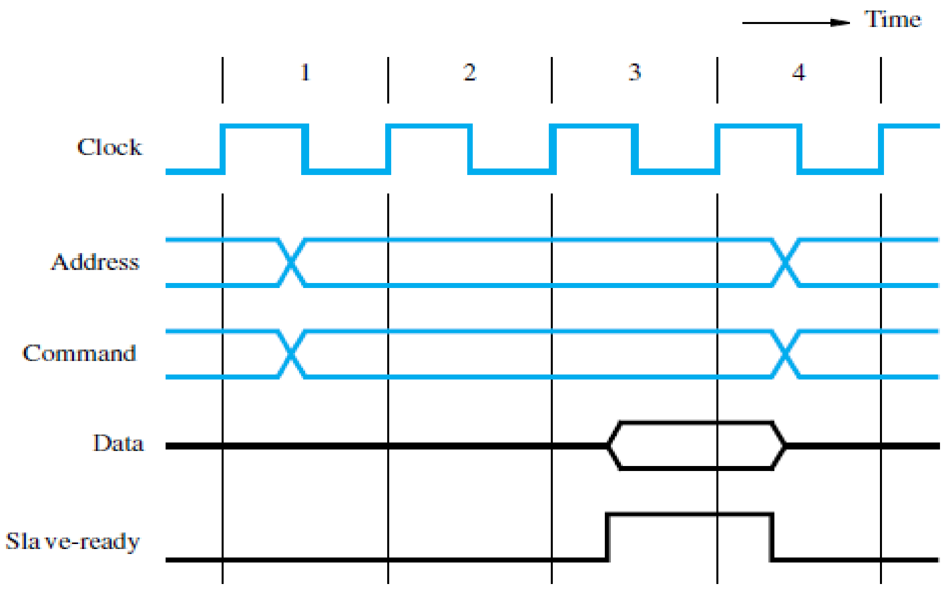


Figure 7.5 An input transfer using multiple clock cycles.

- 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 makesaccession 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 isready.
  - ➢ 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 atdifferent time
  - ➢ Therefore, the delay t1 – t0 should be larger than the maximum possible busskew.
- 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, itremoves data and SR signal from the bus. This completes the input transfer.

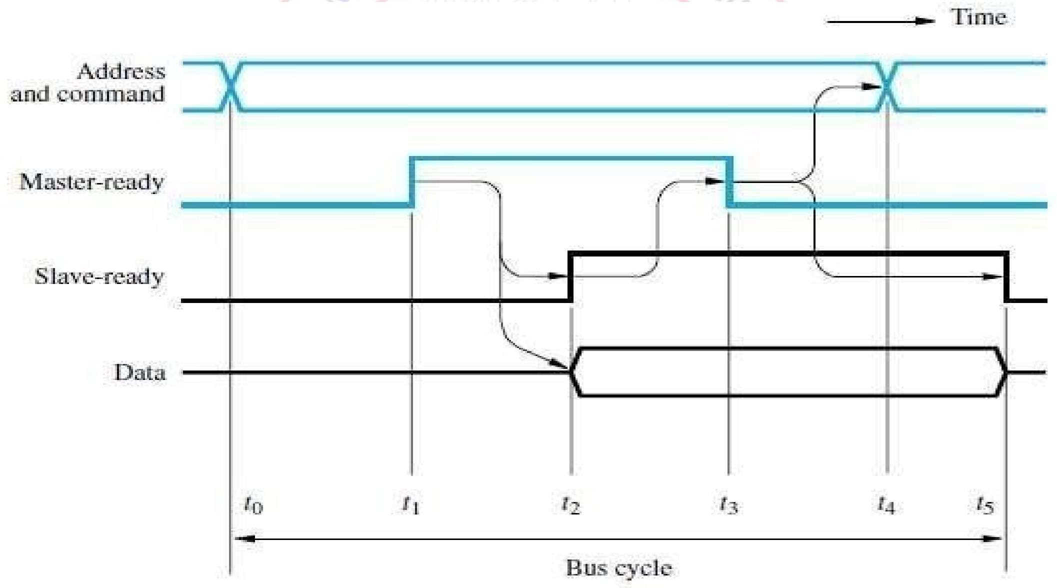


Figure 7.6 Handshake control of data transfer during an input operation.

- 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.

INTERFACE-CIRCUITS

- An **I/O Interface** consists of the circuitry required to connect an I/O device to acomputer-bus.
- On one side of the interface, we have bus signals.

  On the other side, we have a data path with its associated controls to transferdata between theinterface and the I/O device known as **port**.
- Two types are:

  1. Parallel Port transfers data in the form of a number of bits (8 or 16)simultaneously to orfrom the device.

  2. Serial Port transmits and receives data one bit at a time.
- Communication with the bus is the same for both formats.
- The conversion from the parallel to the serial format, and vice versa, takes placeinside the interface-circuit.
- In parallel-port, the connection between the device and the computer uses

  → a multiple-pin connector and

  → a cable with as many wires.
- This arrangement is suitable for devices that are physically close to the computer.
- In serial port, it is much more convenient and cost-effective where longer cables areneeded.

Functions of I/O Interface

1) Provides a storage buffer for at least one word of data.
2) Contains status-flags that can be accessed by the processor to determinewhether the bufferis full or empty.
3) Contains address-decoding circuitry to determine when it isbeing addressed by theprocessor.
4) Generates the appropriate timing signals required by the bus control scheme.
5) Performs any format conversion that may be necessary to transfer data between the bus and the I/O device (such as parallel-serial conversion in the case of a serial port).

PARALLEL-PORT

KEYBOARD INTERFACED TO PROCESSOR

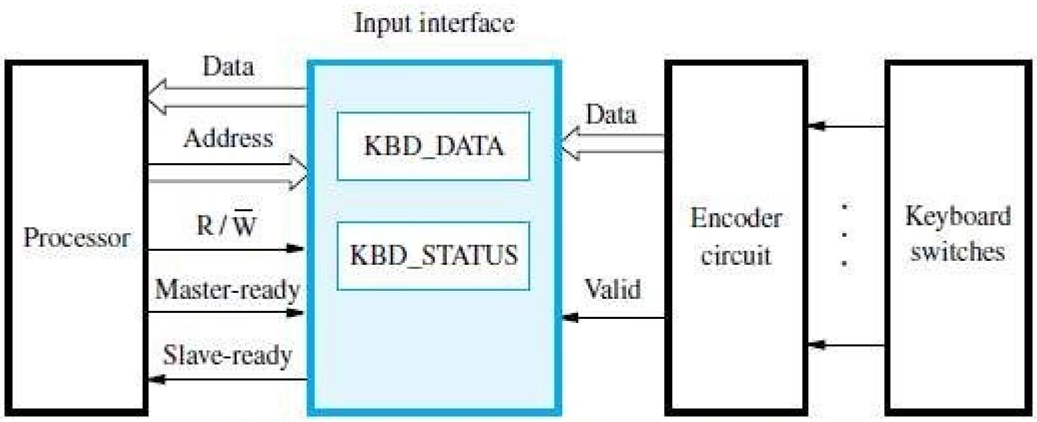


Figure 7.10 Keyboard to processor connection.

The output of the encoder consists of
→ bits representing the encoded character and
→ one signal called **valid**, which indicates the key is pressed.
• The information is sent to the interface-circuits (Figure 7.10).
• Interface-circuits contain
1) Data register DATAIN &
2) Status-flag SIN.
• When a key is pressed, the Valid signal changes from 0 to1.
Then, SIN=1 when ASCII code is loaded into DATAIN.
SIN = 0 when processor reads the contents of the DATAIN.
• The interface-circuit is connected to the asynchronous bus.
• Data transfers on the bus are controlled using the handshake signals:
1) Master ready &
2) Slave ready.

INPUT-INTERFACE-CIRCUIT

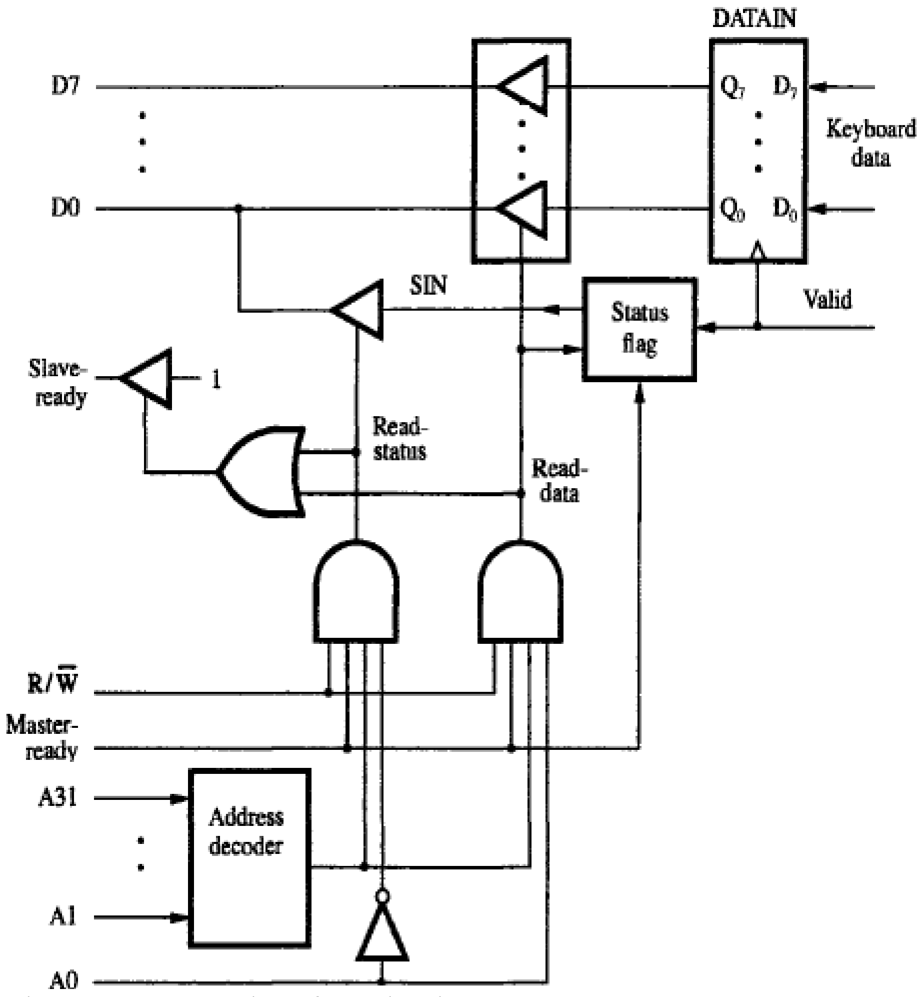


Figure 4.29: Input-interface-circuit

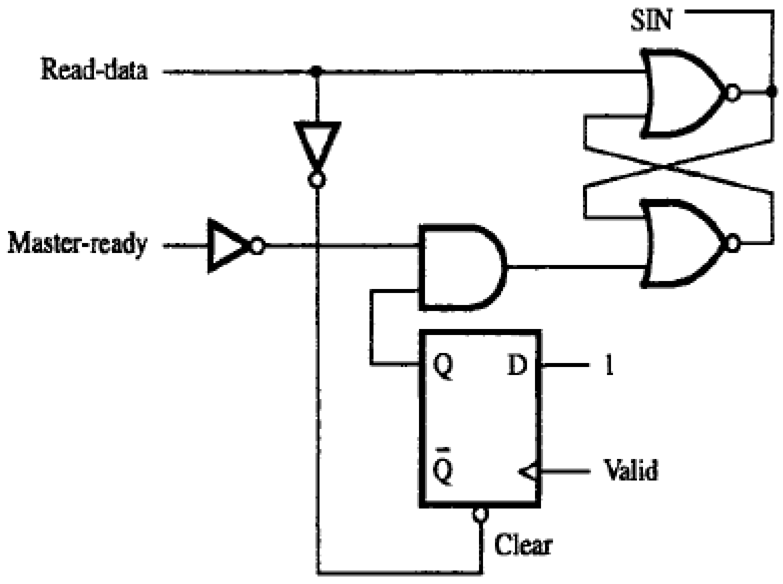


Figure 4.30 Circuit for the status flag block in Figure 4.29.

Output-lines of DATAIN are connected to the data-lines of bus by means of 3-state drivers (Fig 4.29).
• Drivers are turned on when
→ processor issues a read signal and
→ address selects DATAIN.
• SIN signal is generated using a status-flag circuit (Figure 4.30).
SIN signal is connected to line D0 of the processor-bus using a 3-state driver.
• Address-decoder selects the input-interface based on bits A1 through A31.
• Bit A0 determines whether the status or data register is to be read, when Master-ready is active.

- Processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1.

PRINTER INTERFACED TO PROCESSOR

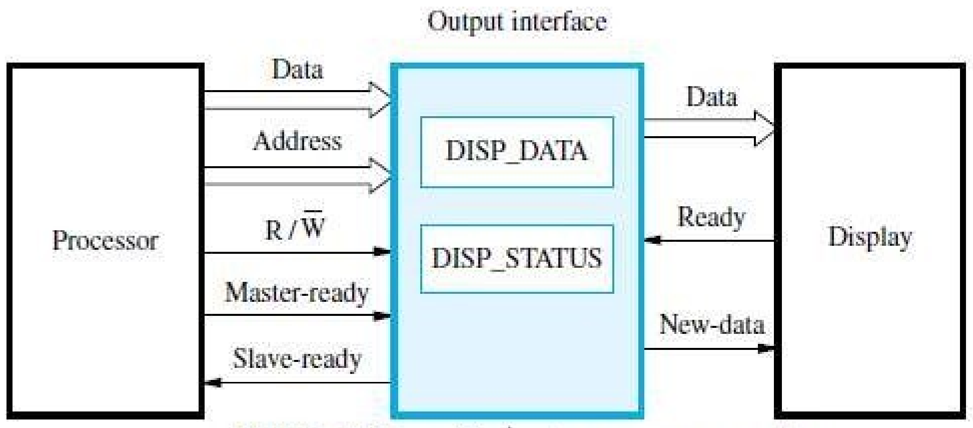


Figure 7.13 Display to processor connection.

Keyboard is connected to a processor using a parallel-port.
• Processor uses
→ memory-mapped I/O and
→ asynchronous bus protocol.
• On the processor-side of the interface, we have:
→ Data-lines

→ Address-lines
→ Control or R/W line
→ Master-Ready signal and
→ Slave-Ready signal.
• On the keyboard-side of the interface, we have:
→ Encoder-circuit which generates a code for the key pressed.
→ Debouncing-circuit which eliminates the effect of a key.
→ Data-lines which contain the code for the key.
→ Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAIN and SIN to be set to 1.

GENERAL 8 BIT PARALLEL PROCESSING

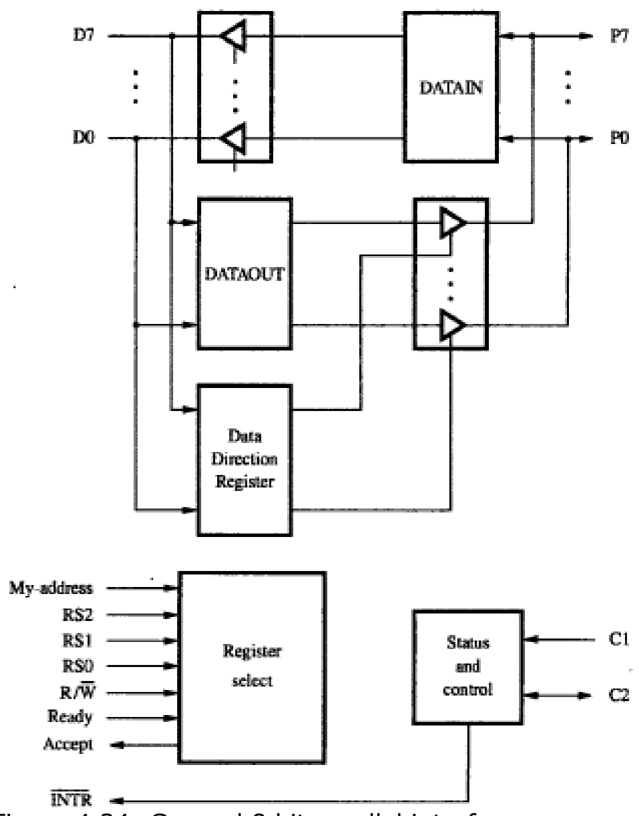


Figure 4.34: General 8 bit parallel interface

Data-lines **P7** through **PO** can be used for either input or output purposes (Figure 4.34).
• For increased flexibility,
→ some lines can be used as inputs and
→ some lines can be used as outputs.
• The **DATAOUT** register is connected to data-lines via 3-state drivers that are controlled by a **DDR**.
• The processor can write any 8-bit pattern into DDR. (DDR Data Direction Register).
• If DDR=1,
Then, data-line acts as an output-line;
Otherwise, data-line acts as an input-line.
• Two lines, **C1** and **C2** are used to control the interaction between interface-circuit and I/0 device. Two lines, C1 and C2 are also programmable.
• Line C2 is bidirectional to provide different modes of signaling, including the handshake.

• The **Ready** and **Accept** lines are the handshake control lines on the processor-bus side.
Hence, the Ready and Accept lines can be connected to Master-ready and Slave-ready.
• The input signal **My-address** should be connected to the output of an address-decoder.
The address-decoder recognizes the address assigned to the interface.
• There are 3 register select lines: **RS0-RS2**.
Three register select lines allows up to eight registers in the interface.
• An interrupt-request **INTR** is also provided.
INTR should be connected to the interrupt-request line on the computer-bus.

Section 13
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 (Figure4.38).
- 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 thedevices.

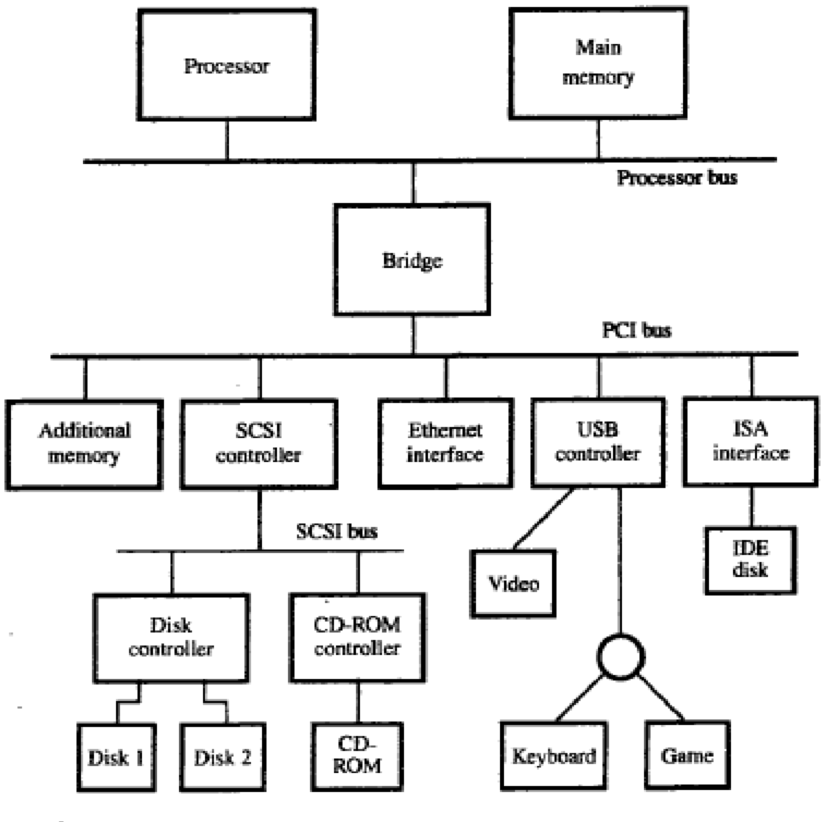


Figure 4.38 An example of a computer system using different interface standards.

- 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 thecomputer-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 rangingfrom keyboard to game control to internal connection.

- IDE (Integrated Device Electronics) disk is compatible with ISA which showsthe 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 bysending a sequence ofdata-words from successive memory-locations.
- During **write-operation**,
  - ➢ When the processor sends an address, a sequence of data-words iswritten 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 datais called a transaction.

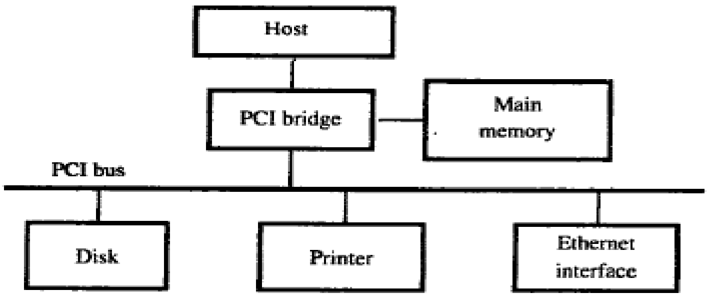


Figure 4.39 Use of a PCI bus in a computer system.

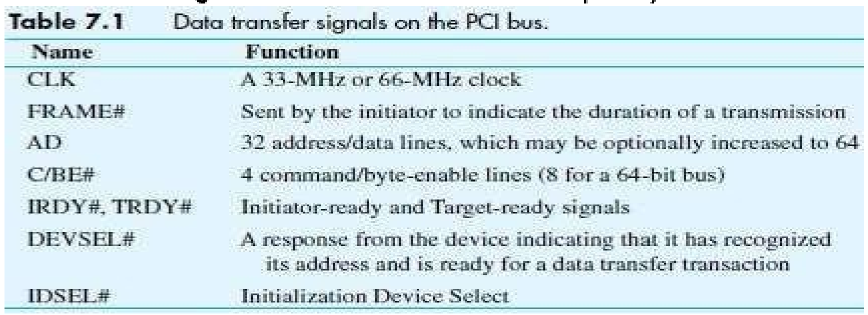


Table 7.1 Data transfer signals on the PCI bus.

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**’

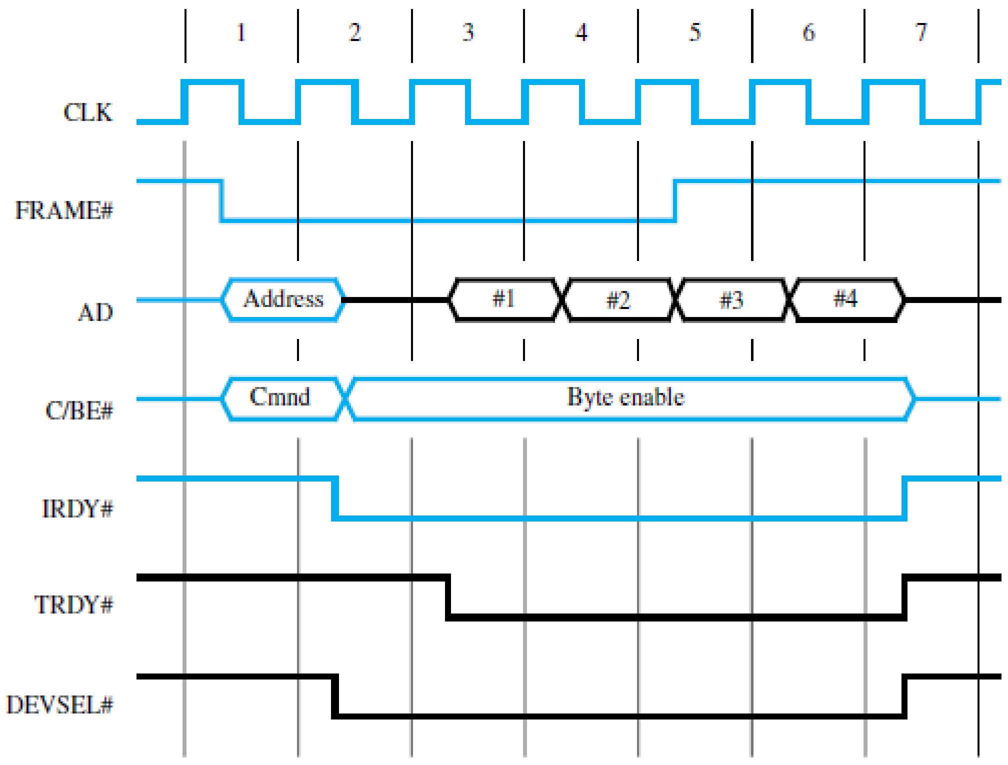


Figure 7.19 A Read operation on the PCI bus.

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 3 more 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 the content 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

SCSI Bus

• SCSI stands for Small Computer System Interface.
• SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute).
• SCSI bus the several options. It may be,

Narrow bus	It has 8 data-lines & transfers 1 byte at a time.
Wide bus	It has 16 data-lines & transfer 2 byte at a time.
Single-Ended Transmission	Each signal uses separate wire.
HVD (High Voltage Differential)	It was 5v (TTL cells)
LVD (Low Voltage Differential)	It uses 3.3v

• Because of these various options, SCSI connector may have 50, 68 or 80 pins. Thedata transfer rate ranges from 5MB/s to 160MB/s 320Mb/s, 640MB/s. The transfer rate depends on,
1) Length of the cable
2) Number of devices connected.
• To achieve high transfer rate, the bus length should be 1.6m for SEsignaling and 12m for LVD signaling.
• The SCSI bus us connected to the processor-bus through the SCSIcontroller. The data are stored on a disk in blocks called sectors.
Each sector contains several hundreds of bytes. These data will not bestored in contiguous memory-location.
• SCSI protocol is designed to retrieve the data in the first sector or any other selectedsectors.
• Using SCSI protocol, the burst of data are transferred at high speed.
• The controller connected to SCSI bus is of 2 types. They are1) Initiator * 2) Target

1) Initiator

➢ It has the ability to select a particular target & to send commandsspecifying the operation tobe performed.
➢ They are the controllers on the processor side.

2) Target

➢ The disk controller operates as a target.
➢ It carries out the commands it receive from the initiator.

➢ The initiator establishes a logical connection with the intended target.

Steps for Read-operation

1) The SCSI controller contends for control of the bus (initiator).
2) When the initiator wins the arbitration-process, the initiator
   → selects the target controller and
   → hands over control of the bus to it.
3) The target starts an output operation. The initiator sends a command specifyingthe required read-operation.
4) The target
   → sends a message to initiator indicating that it will temporarily suspendconnection b/w them.
   → then releases the bus.
5) The target controller sends a command to the disk drive to move the read headto the first sector involved in the requested read-operation.

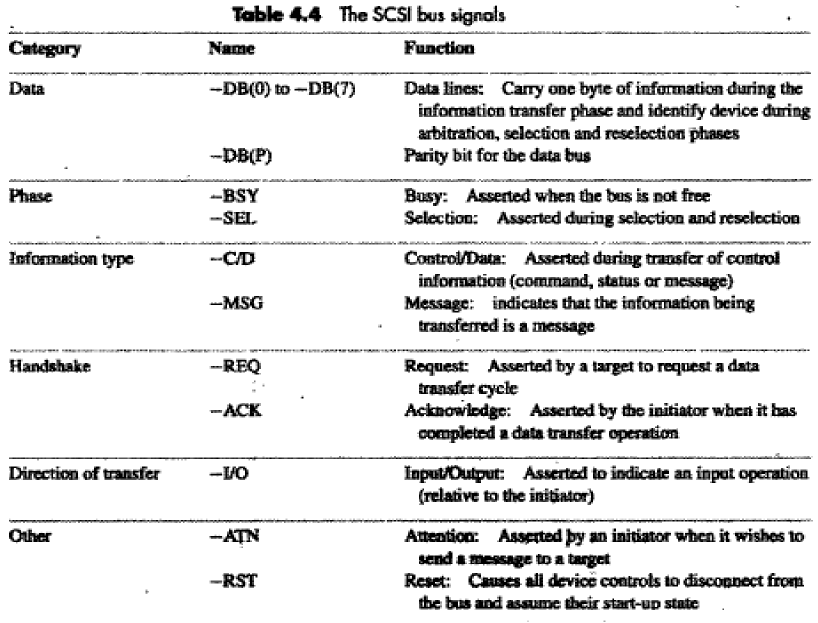
6. The target
   → transfers the contents of the data buffer to the initiator and
   → then suspends the connection again.

7) The target controller sends a command to the disk drive to perform another seekoperation.
8) As the initiator controller receives the data, it stores them into the main-memory using the DMA approach.
9) The SCSI controller sends an interrupt to the processor indicating that the data are nowavailable.

BUS SIGNALS OF SCSI

• The bus has no address-lines. Instead, it has data-lines to identify the bus-controllers involved in the selection/reselection/arbitration-process.
• For narrow bus, there are 8 possible controllers numbered from 0 to 7. For awide bus, there are 16controllers.
• Once a connection is established b/w two controllers, there is no further needfor addressing & the data-lines are used to carry the data.
All signal names are proceeded by minus sign.
• This indicates that the signals are active or that the data-line is equal to 1, when they are in the low voltage state.

Table 4.4 The SCSI bus signals



Category	Name	Function
Data	−DB(0) to −DB(7)	Data lines: Carry one byte of information during the information transfer phase and identify device during arbitration, selection and reselection phases
	−DB(P)	Parity bit for the data bus
Phase	−BSY	Busy: Asserted when the bus is not free
	−SEL	Selection: Asserted during selection and reselection
Information type	−C/D	Control/Data: Asserted during transfer of control information (command, status or message)
	−MSG	Message: indicates that the information being transferred is a message
Handshake	−REQ	Request: Asserted by a target to request a data transfer cycle
	−ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	−I/O	Input/Output: Asserted to indicate an input operation (relative to the initiator)
Other	−ATN	Attention: Asserted by an initiator when it wishes to send a message to a target
	−RST	Reset: Causes all device controls to disconnect from the bus and assume their start-up state

PHASES IN SCSI BUS

- The phases in SCSI bus operation are:
  1) Arbitration
  2) Selection
  3) Information transfer
  4) Reselection

1) Arbitration

- When the –BSY signal is in inactive state,
  → the bus will be free &
  → any controller can request the use of bus.
- SCSI uses distributed arbitration scheme because
  each controller may generate requests at the same time.
- Each controller on the bus is assigned a fixed priority.
- When –BSY becomes active, all controllers that are requesting the bus
  → examines the data-lines &
  → determine whether highest priority device is requesting bus at thesame time.
- The controller using the highest numbered line realizes that it has won the arbitration-process.
- At that time, all other controllers disconnect from the bus & wait for –BSY to becomeinactive again.

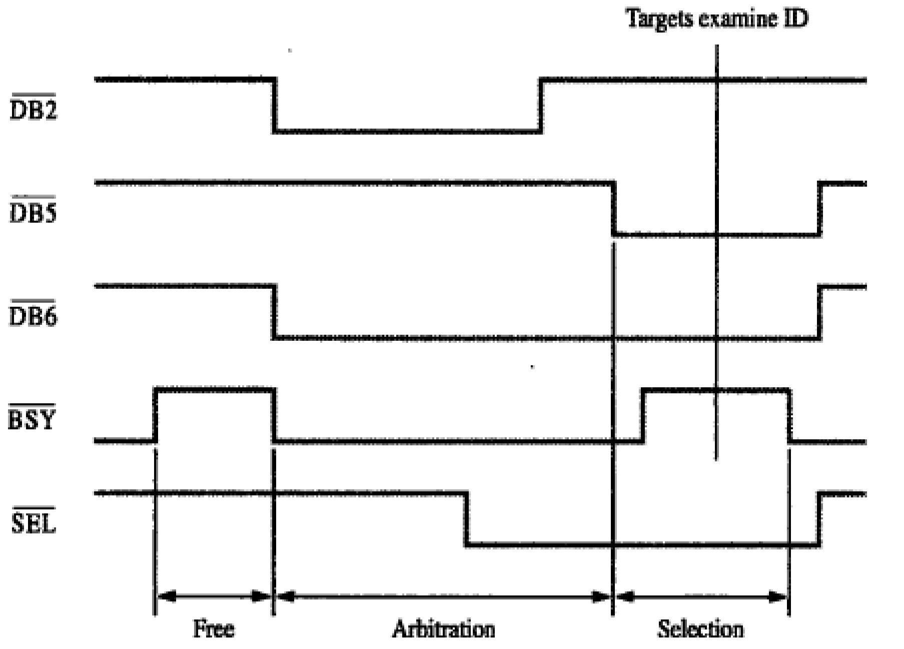


Figure 4.42 Arbitration and selection on the SCSI bus. Device 6 wins arbitration and selects device 2.

2) Information Transfer

- The information transferred between two controllers may consist of
  → commands from the initiator to the target
  → status responses from the target to the initiator or
  → data-transferred to/from the I/0 device.

- Handshake signaling is used to control information transfers, with the targetcontroller taking the roleof the bus-master.

3) Selection

- Here, Device

→ wins arbitration and
→ asserts –BSY and –DB6 signals.

- The Select Target Controller responds by asserting –BSY.
- This informs that the connection that it requested is established.

4) Reselection

- The connection between the two controllers has been reestablished, with thetarget in control of thebus as required for data transfer to proceed.

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 availableon 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 the computer-box.

Port Limitation

➢ Normally, the system has a few limited ports.
➢ To add new ports, the user must open the computer-box to gainaccess to the internal expansion 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 anytime, while the system is 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 offunctionality.
- 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 computer system, the data generated by keyboard are called asynchronous.

The data generated from keyboard depends upon the speed of the humanoperator which is about 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 handledby 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 anytime, 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 orspeaker).

• A hub copies a message that it receives from its upstream connection to all itsdownstream ports.

• As a result, a message sent by the host computer is broadcast to all I/O devices, but onlytheaddressed-device will respond to that message.

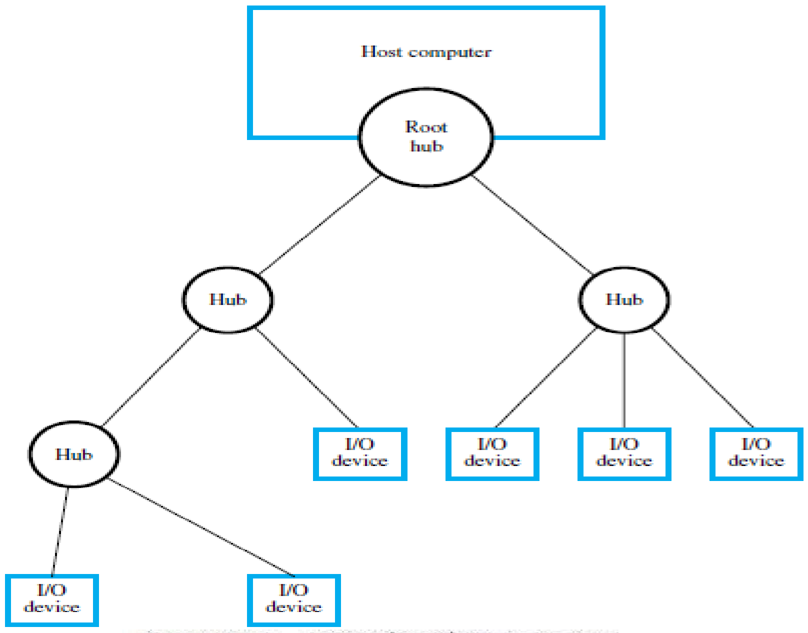


Figure 7.17 Universal Serial Bus tree structure.

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, andaddresses 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 itrecords this fact as partof its own status information.

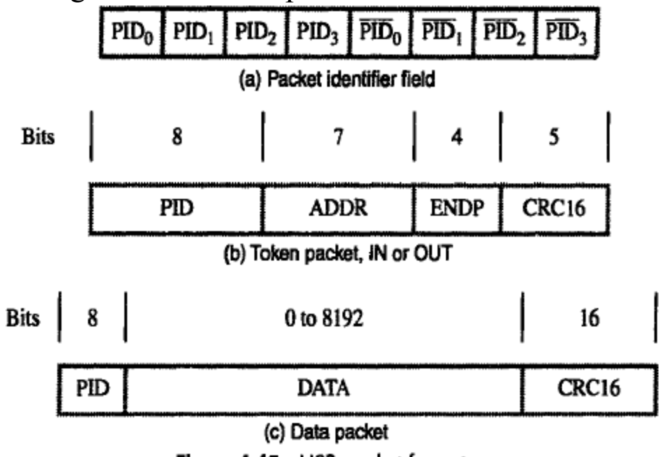
• Periodically, the host polls each hub to

→ 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 ofcommands 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)** whichidentifies type of that packet.
- 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



PID_0 PID_1 PID_2 PID_3 $\overline{PID_0}$ $\overline{PID_1}$ $\overline{PID_2}$ $\overline{PID_3}$
(a) Packet identifier field
Bits 8 7 4 5
PID ADDR ENDP CRC16
(b) Token packet, IN or OUT
Bits 8 0 to 8192 16
PID DATA CRC16
(c) Data packet

Figure 4.45 USB packet formats.