UNIT I

1 Fundamentals of Computer Organization

Syllabus

Basic Organization of Computers, Classification Micro, Mini, Mainframe and Super Computer. System Bus and Interconnection, PCI, Computer Function, I-Cycle, Interrupt and Class of Interrupts.

Contents

1.1 Basic Organization of Computers

- A digital computer or simply computer in its simplest form is a fast electronic calculating machine that accepts digitized information from the user, processes it according to a sequence of instructions stored in the internal storage, and provides the processed information to the user.
- The sequence of instructions stored in the internal storage is called **computer program** and internal storage is called **computer memory**.
- The computer consists of five functionally independent units :
  - Input
  - Memory
  - Arithmetic and logic
  - Output and
  - Control units.
- The Fig. 1.1.1 (a) and (b) show these five functional units of a computer and its physical locations in the computer.
- The input unit accepts the digital information from user with the help of input devices such as keyboard, mouse, microphone etc.
- The information received from the input unit is either stored in the memory for later use or immediately used by the arithmetic and logic unit to perform the desired operations.

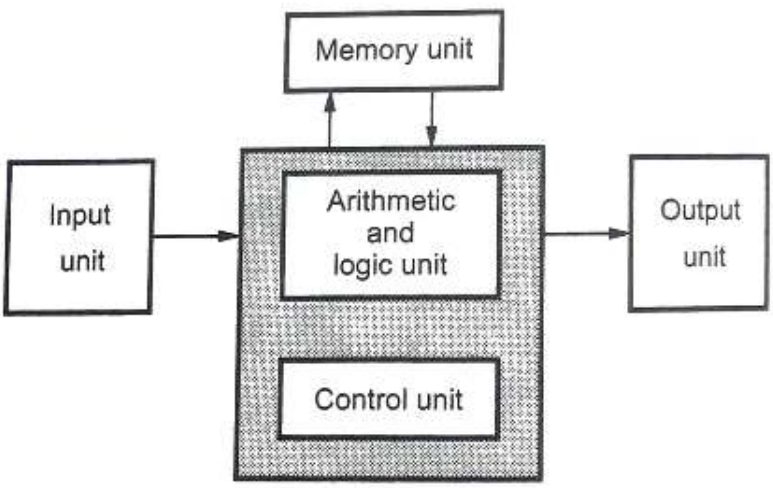


Fig. 1.1.1 (a)

- The program stored in the memory decides the processing steps and the processed output is sent to the user with the help of output devices or it is stored in the memory for later reference.
- All the above mentioned activities are co-ordinated and controlled by the control unit.
- The arithmetic and logic unit in conjunction with control unit is commonly called **Central Processing Unit (CPU)**.

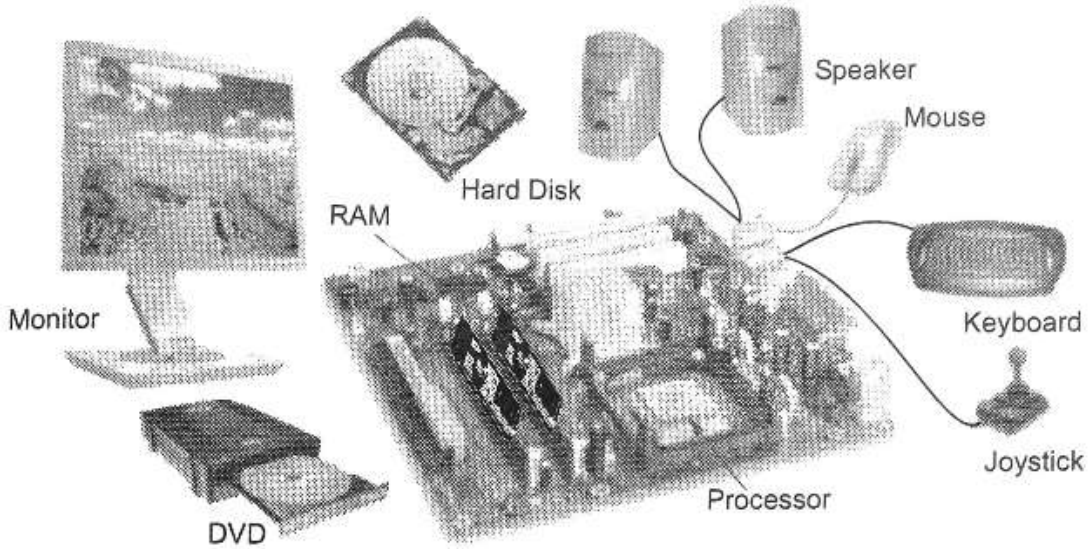
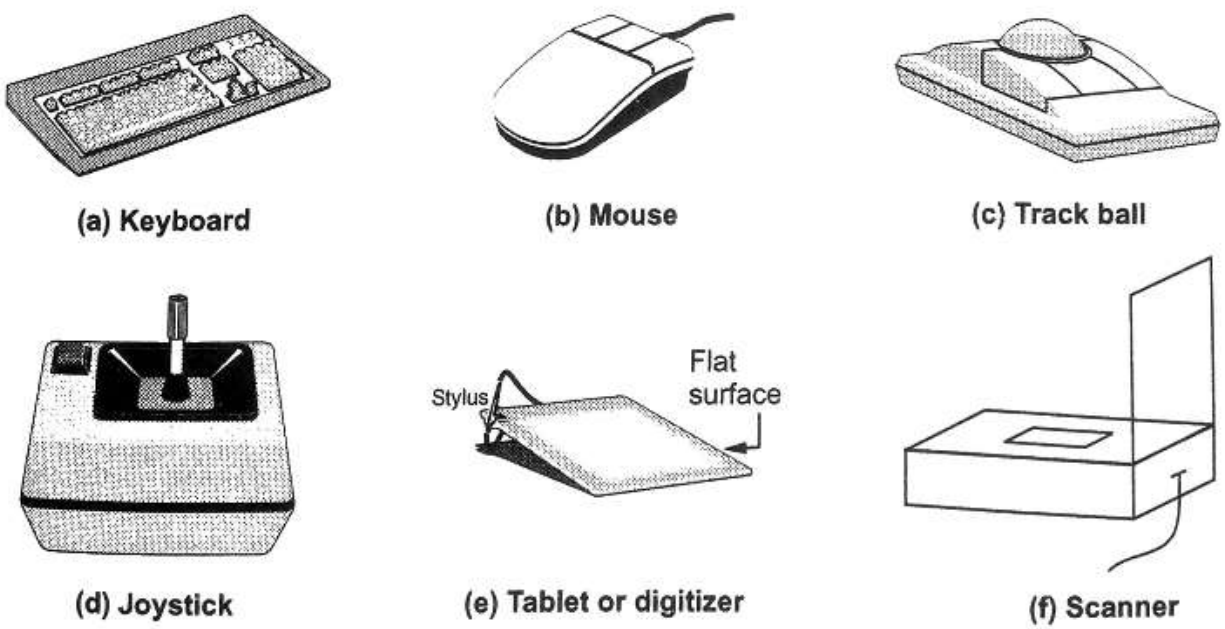


Fig. 1.1.1 (b) Types of hardware devices

1.1.1 Input Unit

- A computer accepts a digitally coded information through input unit using input devices.
- The most commonly used input devices are keyboard and mouse.
- The keyboard is used for entering text and numeric information.
- Mouse is used to position the screen cursor and thereby enter the information by selecting option.
- Apart from keyboard and mouse there are many other input devices are available, which include joysticks, trackball, spaceball, digitizers and scanners.



(a) Keyboard (b) Mouse (c) Track ball

(d) Joystick (e) Tablet or digitizer (f) Scanner

Fig. 1.1.2 Input devices

1.1.2 Memory Unit

- The memory unit is used to store programs and data.
- Usually, two types of memory devices are used to form a memory unit : Primary **storage memory** device and **secondary storage memory** device.
- The primary memory, commonly called **main memory** is a fast memory used for the storage of programs and active data (the data currently in process).
- The main memory is a semiconductor memory.
- It consists of a large number of semiconductor storage cells, each capable of storing one bit of information.
- These cells are read or written by the central processing unit in a group of fixed size called **word**.
- The main memory is organized such that the contents of one word, containing n bits, can be stored or retrieved in one write or read operation, respectively.
- To access data from a particular word from main memory each word in the main memory has a **distinct address**. This allows to access any word from the main memory by specifying corresponding address.
- The number of bits in each word is referred to as the **word length** of the computer. Typically, the word length varies from 8 to 64 bits.
- The number of such words in the main memory decides the **size of memory** or **capacity of the memory**.
- The size of computer main memory varies from few million words to tens of million words.
- An important characteristics of a memory is an **access time** (the time required to access one word). The access time for main memory should be as small as possible. Typically, it is of the order of 10 to 100 nanoseconds. The access time depends on the type of memory. In Randomly Accessed Memories (RAMs), fixed time is required to access any word in the memory. In sequential access memories this time is not fixed.
- The main memory consists of only randomly accessed memories. These memories are fast but they are small in capacities and expensive. Therefore, the computer uses the secondary storage memories such as magnetic tapes, magnetic disks for the storage of large amount of data.

Stored program concept

- Today's computer are built on two key principles
  1. Instructions are represented as numbers.
  2. Programs can be stored in memory to be read or written just like numbers.

- These principles lead to the stored-program concept.
- According to stored-program concept, memory can contain the program (source code), the corresponding compiled machine code, editor program and even the compiler that generated the machine code.

1.1.3 Arithmetic and Logic Unit

- The arithmetic and logic unit (ALU) is responsible for performing arithmetic operations such as add, subtract, division and multiplication and logical operations such as ANDing, ORing, Inverting etc.
- To perform these operations, operands from the main memory are brought into the high speed storage elements called **registers** of the processor.
- Each register can store one word of data and they are used to store frequently used operands.
- After performing operation, the result is either stored in the register or memory location.

1.1.4 Output Unit

- The output unit sends the processed results to the user using output devices such as video monitor, printer, plotter, etc.
- The video monitors display the output on the CRT screen whereas printers and plotters give the hard-copy output.
- Printers are classified according to their printing methodology : **Impact printers** and **non-impact printers.**

1.1.5 Control Unit

- The control unit co-ordinates and controls the activities amongst the functional units.
- Control unit fetches the instructions stored in the main memory, identify the operations and the devices involved in it and accordingly generate control signals to execute the desired operations.
- It uses control signals or timing signals to determine when a given action is to take place.
- It controls input and output operations, data transfers between the processor, memory and input/output devices using timing signals.

- The control and the arithmetic and logic units of a computer are usually many times faster than other devices connected to a computer system. This enables them to control a number of external input/output devices.

Review Questions

1. *Explain different functional units of a digital computer.*
2. *What is a stored program concept ? Explain the functional units of a stored program digital computer, along with a block diagram.*

1.2 Classification of Computers

- According to size, cost computational power and application computers are classified as :
  - Microcomputers
  - Minicomputers
  - Desktop computers
  - Personal computers
  - Portable notebook computers
  - Workstations
  - Mainframes or enterprise systems
  - Servers
  - Super computers.

Microcomputers :

- As the name implies micro-computers are smaller computers.
- They contain only one central processing unit.
- One distinguishing feature of a microcomputer is that the CPU is usually a single integrated circuit called a **microprocessor**.
- Microcomputer is the integration of microprocessor and supporting peripherals (memory and I/O devices).
- The word length depends on the microprocessor used and is in the range of 8 bits to 32 bits.
- These type of computers are used for small industrial control, process control and where storage and speed requirements are moderate.

Minicomputers :

- Minicomputers are the scaled up version of the microcomputers with the moderate speed and storage capacity.
- These are designed to process smaller data words, typically 32-bit words.
- This type of computers are used for scientific calculations, research, data processing application and many other.

Desktop Computers :

- The desktop computer are the computers which are usually found on a home or office desk.
- They consist of processing unit, storage unit, visual display and audio as output units and keyboard and mouse as input units.
- Usually storage unit of such computer consists of hard disks, CD-ROMs and diskettes.

Personal Computers :

- The personal computers are the most common form of desktop computers.
- They found wide use in homes, schools and business offices.

Portable Notebook Computers :

- Portable notebook computers are the compact version of personal computers.
- The laptop computers are the good example of portable notebook computer.

Workstations :

- Workstations have higher computation power than personal computers.
- They have high resolution graphics terminals and improved input/output capabilities.
- Workstations are used in engineering applications and in interactive graphics applications.

Mainframes or Enterprise Systems :

- Mainframe computers are implemented using two or more Central Processing Units (CPU).
- These are designed to work at very high speeds with large data word lengths, typically 64 bits or greater.
- The data storage capacity of these computers is very high.
- This type of computers are used for complex scientific calculations, large data processing applications, Military defense control and for complex graphics applications (e.g. : For creating walkthroughs with the help of animation software).

Servers :

- These computers have large storage unit and faster communication links.
- The large storage unit allows to store sizable database and fast communication links allow faster communication of data blocks with computers connected in the network.
- These computers serve major role in Internet communication.

Supercomputers :

- These computers are basically multiprocessor computers used for the large-scale numerical calculations required in applications such as weather forecasting, robotic engineering, aircraft design and simulation.

Review Question

1. *Explain the various types of computers and their applications.*

1.3 Computer Function

- The basic function of computer is to execute program, sequence of instructions.
- Instructions are stored in the computer memory.
- Instructions are executed to process data which is loaded into the computer memory through input unit.
- After processing the data, the result is either stored back into the computer memory for further reference or it is sent to the outside world through the output port.
- All functional units of the computer contribute to execute a program.
- The functions of different computer units are,
  - The input unit accepts data and instructions from the outside world to machine. It is operated by control unit.
  - The memory unit stores both, data and instructions.
  - The arithmetic-logic unit performs arithmetic and logical operations.
  - The control unit fetches and interprets the instructions in memory and causes them to be executed.
  - The output unit transmits final results and messages to the outside world.
- To perform execution of instruction, the processor contains a number of registers used for temporary storage of data and some special function registers, as shown in Fig. 1.3.1.

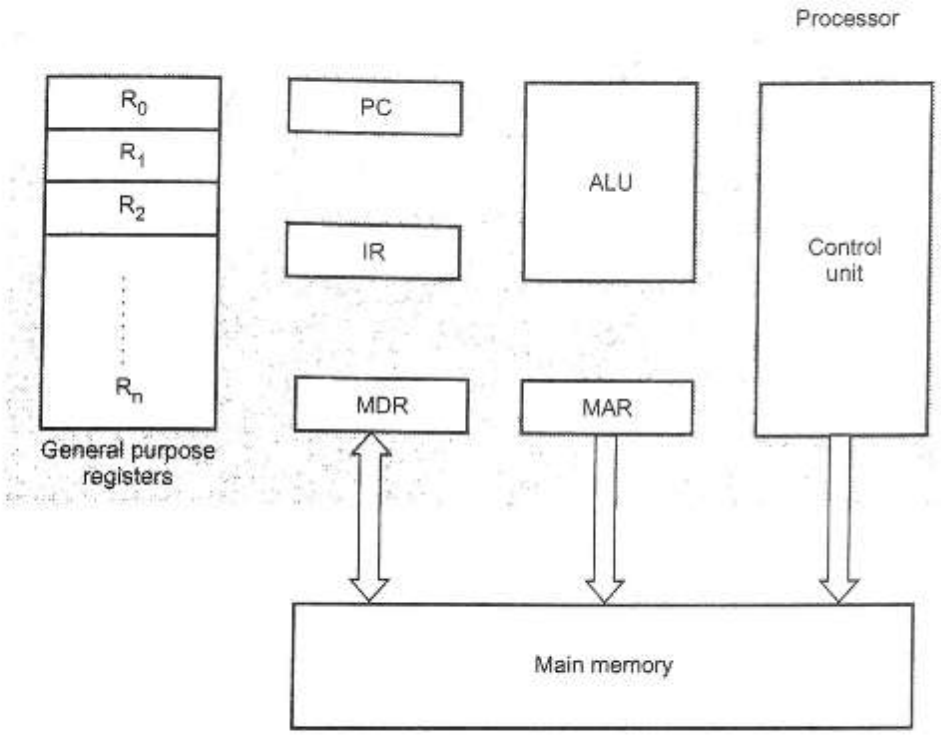


Fig. 1.3.1 Connections between the processor and the main memory

- The special function registers include Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR) and Memory Data Register (MDR).

Program Counter (PC) :

- A program is a series of instructions stored in the memory. These instructions tell the CPU exactly how to get the desired result.
- It is important that these instructions must be executed in a proper order to get the correct result.
- The sequence of instruction execution is monitored by the program counter.
- It keeps track of which instruction is being executed and what the next instruction will be.

Instruction Register (IR) :

- It is used to hold the instruction that is currently being executed.
- The contents of IR are available to the control unit, which generate the timing signals that control the various processing elements involved in executing the instruction.

Memory Address Register (MAR) and Memory Data Register (MDR) :

- These registers are used to handle the data transfer between the main memory and the processor.
- The MAR holds the address of the main memory to or from which data is to be transferred.

- The MDR sometimes also called MBR (Memory Buffer Register) contains the data to be written into or read from the addressed word of the main memory.

General purpose registers :

- These are used to **hold the operands** for arithmetic and logic operations and/or used to store the result of the operation.
- Since the access time of these registers is lowest, these are used to **store frequently used data.**

Execution of program :

- PC is set to point to the first instruction of the program.
- The contents of PC are transferred to the MAR, whose output is connected to the address lines of memory and simultaneously Read Control Signal is sent to the memory.
- After the access time of memory, the addressed word is received from memory and stored into the MDR.
- The received instruction code, i.e. the contents of MDR are transferred to IR and now instruction is ready to be decoded and executed.
- If the instruction involved an operation to be performed by the ALU and operand is available in general-purpose registers the operation is performed by the ALU.
- If operand is not available in the general-purpose registers, it is reside in the memory. It has to be fetched by sending its address to the MAR and initiating a Read cycle.
- When the operand is read from the memory into the MDR, it is transferred from the MDR to the ALU.
- ALU then performs the desired operation and stores the result either in the general purpose registers or in the memory.
- If the result of operation is to be stored in the memory, then the result is sent to the MDR. The address of the memory location where the result is to be stored is sent to the MAR, and a write cycle is initiated.
- During the execution of current instruction, the contents of PC are incremented so that they points to the next instruction to be executed.
- As soon as current instruction is executed, next instruction is fetched and cycle is repeated.
- Like memory, computer also communicates with input/output devices. Thus, computer also has some instruction to handle I/O data transfer.

Example 1.3.1 *State the operations involved in the execution of ADD R1, R0 instruction.*

Solution : The instruction Add R1, R0 adds the operand in R1 register to the operand in R0 register and stores the sum into R0 register. Let us see the steps involved in the execution of this instruction.

1. Fetch the instruction from the memory into IR register of the processor.
2. Decode the instruction.
3. Add the contents of R1 and R0 and store the result in the R0.

Example 1.3.2 *State the operations involved in the execution of Add LOCA, R0.*

Solution : The instruction Add LOCA, R0 adds the operand at memory location LOCA to the operand in register R0 and stores result in the register R0. The steps involved in the execution of this instruction are :

1. Fetch the instruction from the memory into the IR register of the processor.
2. Decode the instruction.
3. Fetch the second operand from memory location LOCA and add the contents of LOCA and the contents of register R0.
4. Store the result in the R0.

Review Questions

1. *Explain different functional units of a computer. Mention the function of the processor registers i) PC ii) MAR iii) IR.*
2. *Draw and explain the connections between the processor and the main memory.*

1.4 System Bus and Interconnections

- The central processing unit, memory unit and I/O unit are the hardware components/modules of the computer. They work together with communicating each other and have paths for connecting the modules together. The collection of paths connecting the various modules is called the **interconnection structure**.
- The design of this interconnection structure will depend on the exchanges that must be made between modules.
- A group of wires, called **bus** is used to provide necessary signals for communication between modules.
- A bus that connects major computer components/modules (CPU, memory, I/O) is called a **system bus**. The system bus is a set of conductors that connects the CPU, memory and I/O modules.

- Usually, the system bus is separated into three functional groups :
  - Data Bus
  - Address Bus
  - Control Bus

1. Data Bus :

- The data bus consists of 8, 16, 32 or more parallel signal lines. These lines are used to send data to memory and output ports, and to receive data from memory and input port. Therefore, data bus lines are **bi-directional.**
- This means that CPU can read data on these lines from memory or from a port, as well as send data out on these lines to a memory location or to a port.
- The data bus is connected in parallel to all peripherals.
- The communication between peripheral and CPU is activated by giving output enable pulse to the peripheral.
- Outputs of peripherals are floated when they are not in use.

2. Address Bus :

- It is an **unidirectional** bus.
- The address bus consists of 16, 20, 24 or more parallel signal lines.
- On these lines the CPU sends out the address of the memory location or I/O port that is to be written to or read from.
- Here, the communication is one way, the address is send from CPU to memory and I/O port and hence these lines are unidirectional.

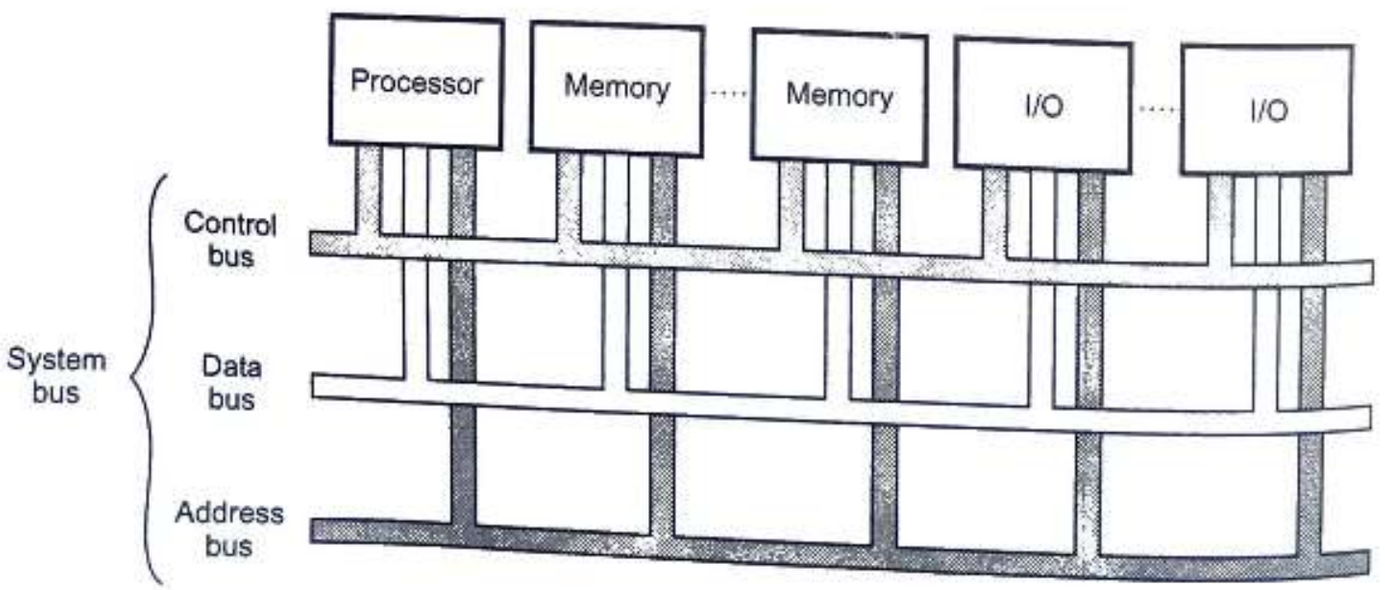


Fig. 1.4.1 Bus interconnection scheme

3. Control Bus :

- The control lines regulate the activity on the bus.
- The CPU sends signals on the control bus to enable the outputs of addressed memory devices or port devices.
- Fig. 1.4.1 shows bus interconnection scheme.

1.4.1 Single Bus Structure

- Another way to represent the same bus connection scheme is shown in Fig. 1.4.2.

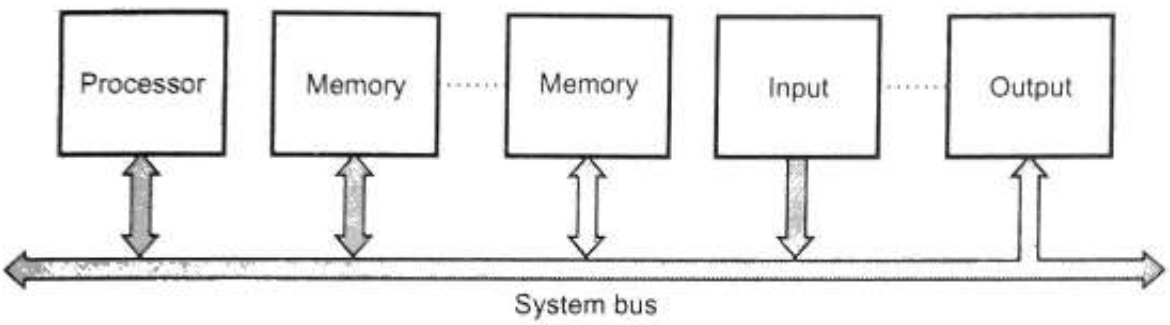


Fig. 1.4.2 Single bus structure

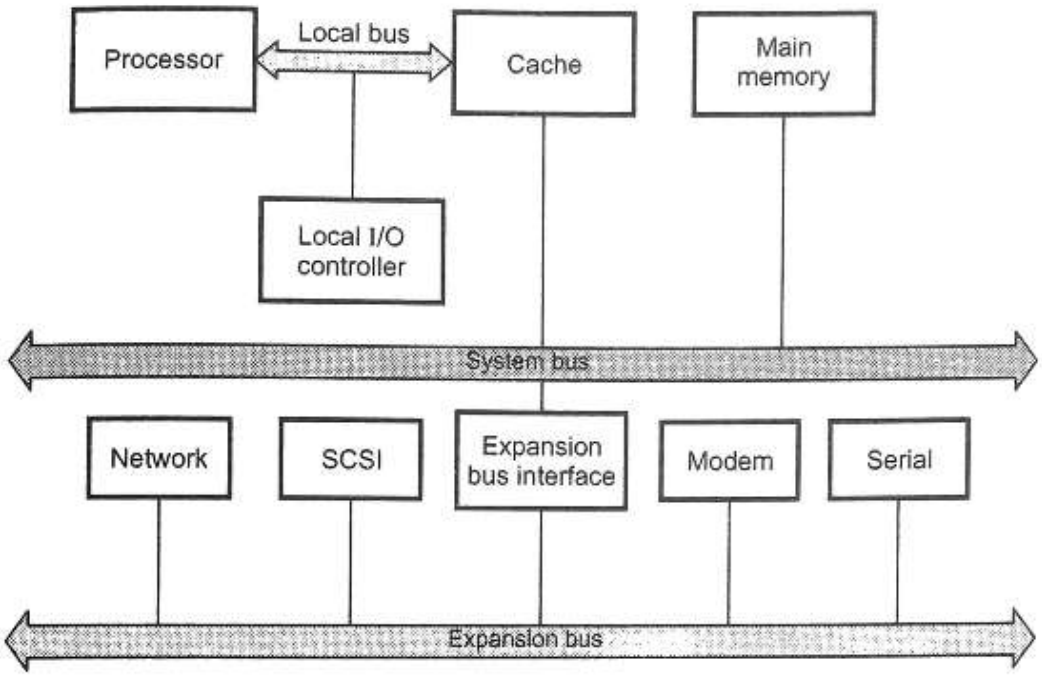
- Here, address bus, data bus and control bus are shown by single bus called **system bus**. Hence such interconnection bus structure is called **single bus structure.**
- In a single bus structure all units are connected to **common bus** called **system bus.**
- However, with single bus only two units can communicate with each other at a time.
- The bus control lines are used to arbitrate multiple requests for use of the bus.
- The main advantage of single bus structure is its low cost and its flexibility for attaching peripheral devices.
- The complexity of bus control logic depends on the amount of translation needed between the system bus and CPU, the timing requirements, whether or not interrupt management is included and the size of the overall system.
- For a small system, control signals of the CPU could be used directly to reduce handshaking logic. Drivers and receivers would not be needed for the data and address lines.
- But large systems with several interfaces would need bus driver and receiver circuits connected to the bus in order to maintain adequate signal quality.
- In most of the processors, multiplexed address and data buses are used to reduce the number of pins. During first part of bus cycle, address is present on this bus.

Afterwards, the same bus is used for data transfer purpose. So latches are required to hold the address sent by the CPU initially.

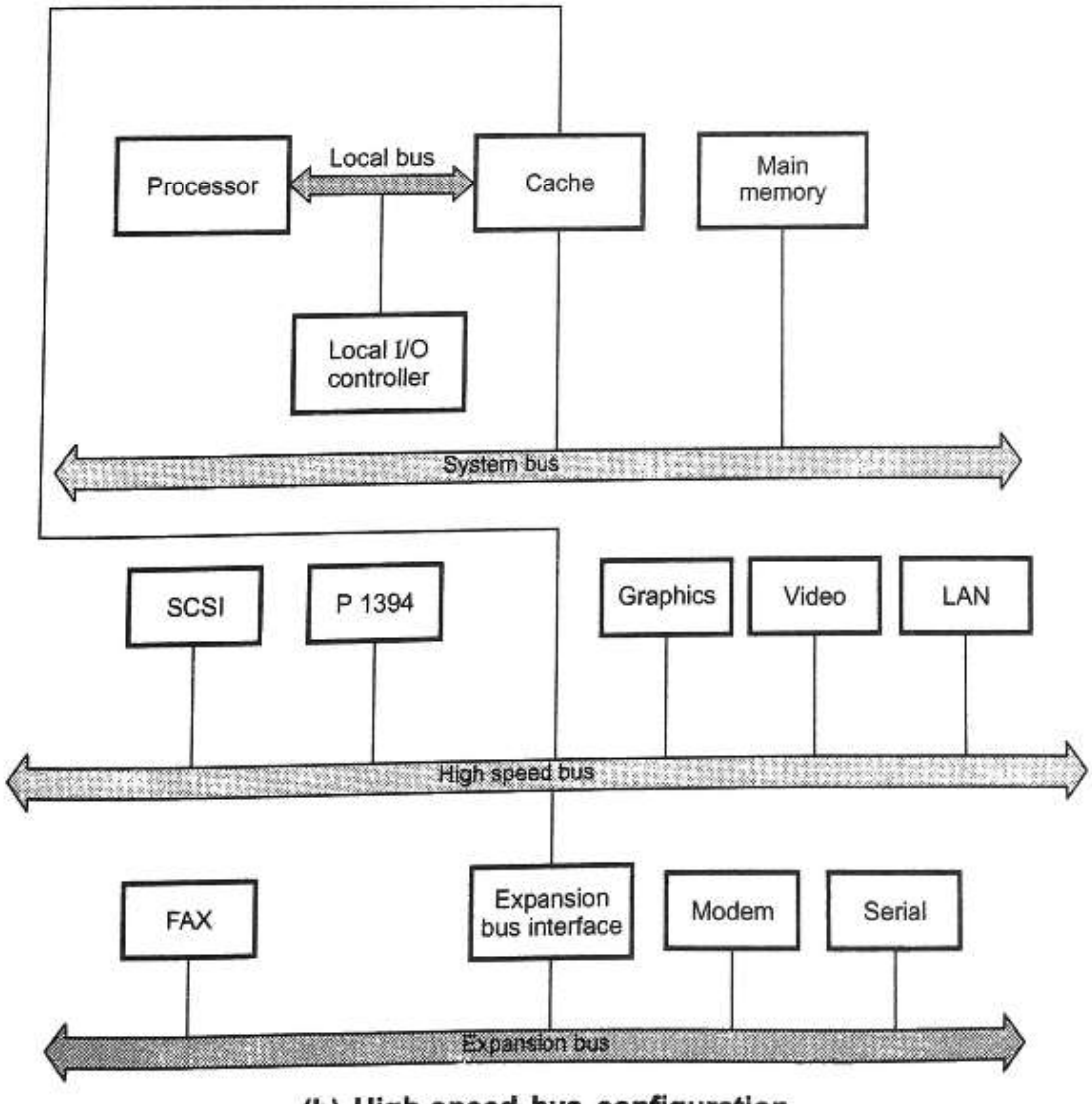
- Interrupt priority management is optional in a system. It is not required in systems which use software priority management.
- The complex system includes hardware for managing the I/O interrupts to increase efficiency of a system.
- Many manufacturers have made priority management devices. Programmable Interrupt Controller (PIC) is the IC designed to fulfil the same task.

1.4.2 Multiple Bus Structures

- The performance of computer system suffers when large number of devices are connected to the bus.
- This is because of the two major reasons :
  1. When more devices are connected to the common bus we need to share the bus amongst these devices. The sharing mechanism co-ordinates the use of bus to different devices. This co-ordination requires finite time called **propagation delay**. When control of the bus passes from one device to another frequently, these propagation delays are noticeable and affect the performance of computer system.
  2. When the aggregate data transfer demand approaches the capacity of the bus, the bus may become a bottleneck. In such situations we have to increase the data rate of the bus or we have to use wider bus.
- Now-a-days the data transfer rates for video controllers and network interfaces are growing rapidly.
- The need of high speed shared bus is impractical to satisfy with a single bus. Thus, most computer systems use the multiple buses.
- These buses have the hierarchical structure.
- Fig. 1.4.3 shows two bus configurations. The traditional bus connection uses three buses : Local bus, system bus and expanded bus.
- The high speed bus configuration uses high-speed bus along with the three buses used in the traditional bus connection.
- Here, cache controller is connected to high-speed bus.
- This bus supports connection to high-speed LANs, such as Fiber Distributed Data Interface (FDDI), video and graphics workstation controllers, as well as interface controllers to local peripheral buses including **Small Computer System Interface SCSI** and P1394.



(a) Traditional bus configuration



(b) High-speed bus configuration

Fig. 1.4.3

1.4.3 PCI Bus

In early 1992, Intel formed an another industrial group in relation to be PC bus. The main intension behind the formation of group was to overcome the weaknesses in the ISA and EISA buses. They designed PCI (Peripheral Component Interface) specifications in June 1992 and updated in April 1993.

1.4.3.1 Features

1. It is designed to economically meet the I/O requirements of modern systems. It requires very few chips to implement and support other buses attached to the PCI bus.
2. It bypasses the standard I/O bus, uses the system bus to increase the bus clock speed and take full advantage of the CPU's data path.
3. It has an ability to function with a 64-bit data bus.
4. It has high bandwidth. The information is transferred across the PCI bus at 33 MHz, at the full data width of the CPU. When the bus is used in conjunction with a 32-bit CPU, the bandwidth is 132 Mbytes/sec. It is calculated as follows :

   33 MHz × 32-bit = 1,056 Mbits/sec

   1,056 Mbits/sec ÷ 8 = 132 Mbytes/sec
5. PCI bus is designed to support a variety of microprocessor based configurations including both single and multiprocessor systems.
6. The PCI bus can operate concurrently with the processor bus. The CPU can be processing data in a external cache while the PCI bus is busy transferring information between other parts of the system.
7. The PCI bus is processor - independent bus that can function as a mezzanine or peripheral bus.
8. It makes use of synchronous timings and centralized arbitration scheme.
9. It delivers better system performance for high - speed I/O subsystems (e.g. graphic display adapters, network interface controllers, disk controllers and so on).
10. The PCI interface contains a 256 bytes configuration memory which allows the computer to interrogate the PCI interface. This feature allows the system to automatically configure itself for the PCI plug-board and hence it is referred to as plug-and-play.

1.4.3.2 PCI Configurations

As mentioned earlier, PCI bus is designed to support single processor as well as multiprocessor systems. Fig. 1.4.4 (a) shows a typical use of PCI in a single processor system.

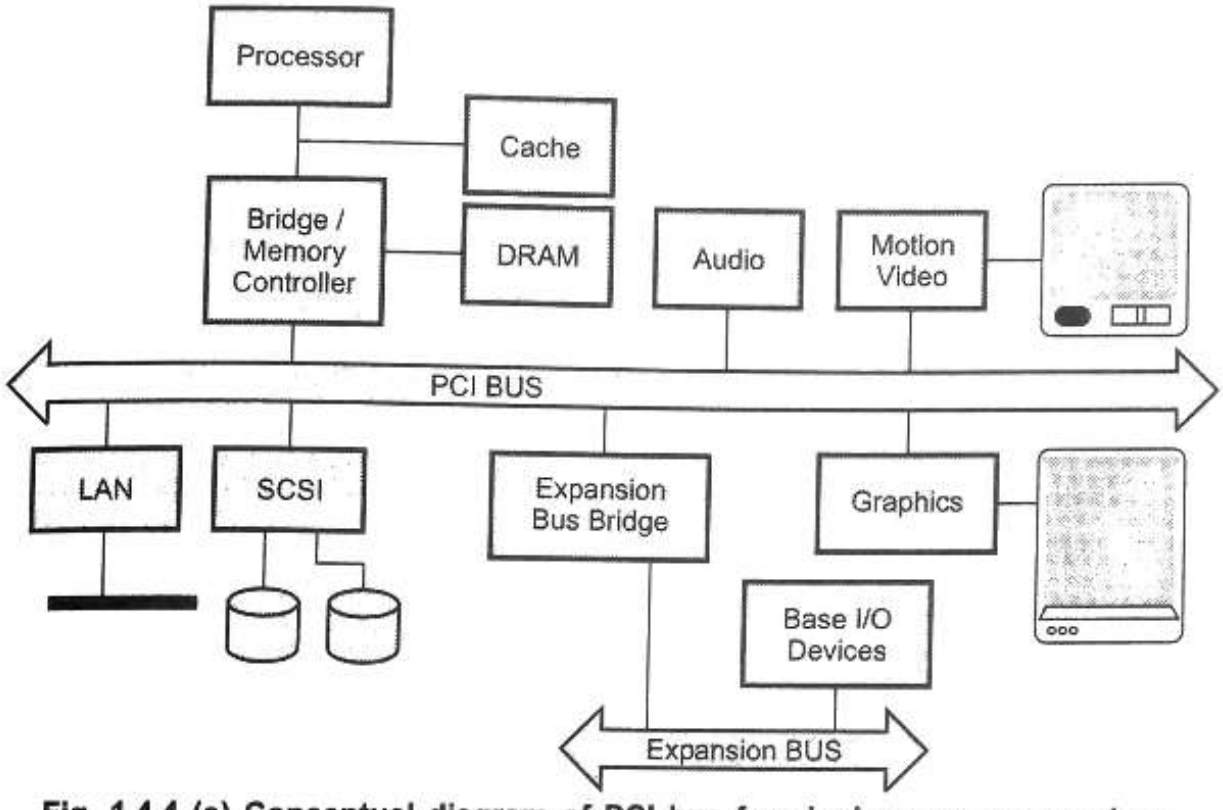


Fig. 1.4.4 (a) Conceptual diagram of PCI bus for single processor system

Notice that the processor bus is separate and independent of the PCI bus. The processor connects to the PCI bus through an integrated circuit called a PCI bridge. The memory controller and PCI bridge provides tight coupling with the processor and delivers data at high speeds.

Fig. 1.4.4 (b) shows a typical use of PCI in a multiprocessor system. As shown in the Fig. 1.4.4 in multiprocessor systems one or more PCI configurations may be connected

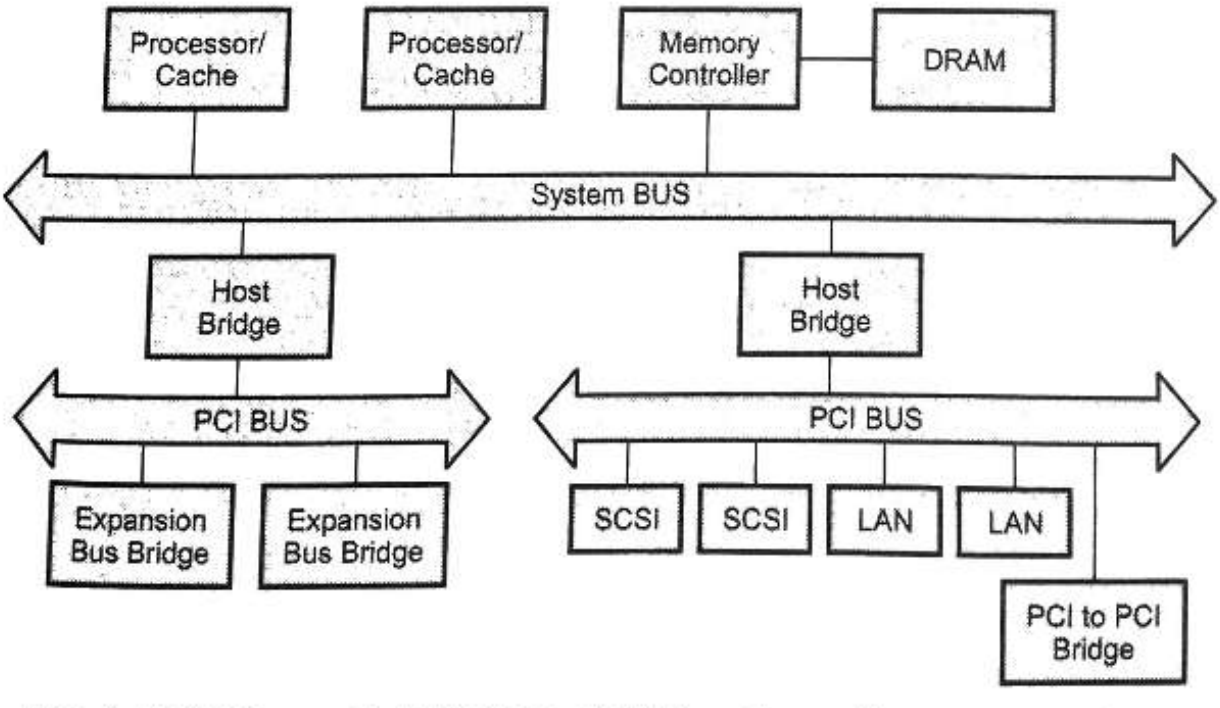


Fig. 1.4.4 (b) Conceptual diagram of PCI bus for multiprocessor system

by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

1.4.3.3 PCI Bus Signals

PCI signals are functionally divided into the following groups :

- **System Signals :** System signals include clock and reset signals (Test clock, clock, Test Reset, Reset).
- **Address and Data Signals :** Address and data signals include 64 lines that are time - multiplexed for addresses and data lines (AD0 - AD63). The other signals from this group such as parity, command and byte enable signals are used to interpret and validate the signal lines that carry the addresses and data (PAR, C/BE).

Interface Control Signals : Interface control signals control the timing of transactions and provide co-ordination among initiators and targets. These are as follows :

$\overline{\text{FRAME}}$: The current master uses this signal to indicate the start and duration of a transaction. The signal is asserted at the start and deasserted when the initiator is ready to begin the final data phase.

$\overline{\text{IRDY}}$: The current master (initiator) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{TRDY}}$: The target device (selected device) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{STOP}}$: This signal indicates that the current target wishes the initiator to stop the current transaction.

$\overline{\text{LOCK}}$: The signal indicates an automatic operation that may require multiple transactions.

IDSEL : This initialization device select signal is used as a chip select during configuration read and write transactions.

$\overline{\text{DEVSEL}}$: This device select signal is used by the target to indicate device has selected.

- **Arbitration Signals :** Unlike other PCI signals, these are not shared lines. Rather, each PCI master has its own Request and Grant arbitration signals that connect it directly to the PCI bus arbiter.
- **Error Reporting Signals :** These are used to report parity and system errors ($\overline{\text{PERR}}$, $\overline{\text{SERR}}$).
- **Interrupt Signals :** Like bus arbitration signals these are not shared signals. Rather, each PCI device has its own interrupt signals ($\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$).

by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

1.4.3.3 PCI Bus Signals

PCI signals are functionally divided into the following groups :

- **System Signals :** System signals include clock and reset signals (Test clock, clock, Test Reset, Reset).
- **Address and Data Signals :** Address and data signals include 64 lines that are time - multiplexed for addresses and data lines (AD0 - AD63). The other signals from this group such as parity, command and byte enable signals are used to interpret and validate the signal lines that carry the addresses and data (PAR, C/BE).

Interface Control Signals : Interface control signals control the timing of transactions and provide co-ordination among initiators and targets. These are as follows :

$\overline{\text{FRAME}}$: The current master uses this signal to indicate the start and duration of a transaction. The signal is asserted at the start and deasserted when the initiator is ready to begin the final data phase.

$\overline{\text{IRDY}}$: The current master (initiator) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{TRDY}}$: The target device (selected device) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{STOP}}$: This signal indicates that the current target wishes the initiator to stop the current transaction.

$\overline{\text{LOCK}}$: The signal indicates an automatic operation that may require multiple transactions.

IDSEL : This initialization device select signal is used as a chip select during configuration read and write transactions.

$\overline{\text{DEVSEL}}$: This device select signal is used by the target to indicate device has selected.

- **Arbitration Signals :** Unlike other PCI signals, these are not shared lines. Rather, each PCI master has its own Request and Grant arbitration signals that connect it directly to the PCI bus arbiter.
- **Error Reporting Signals :** These are used to report parity and system errors ($\overline{\text{PERR}}$, $\overline{\text{SERR}}$).
- **Interrupt Signals :** Like bus arbitration signals these are not shared signals. Rather, each PCI device has its own interrupt signals ($\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$).

by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

1.4.3.3 PCI Bus Signals

PCI signals are functionally divided into the following groups :

- **System Signals :** System signals include clock and reset signals (Test clock, clock, Test Reset, Reset).
- **Address and Data Signals :** Address and data signals include 64 lines that are time - multiplexed for addresses and data lines (AD0 - AD63). The other signals from this group such as parity, command and byte enable signals are used to interpret and validate the signal lines that carry the addresses and data (PAR, C/BE).

Interface Control Signals : Interface control signals control the timing of transactions and provide co-ordination among initiators and targets. These are as follows :

$\overline{\text{FRAME}}$: The current master uses this signal to indicate the start and duration of a transaction. The signal is asserted at the start and deasserted when the initiator is ready to begin the final data phase.

$\overline{\text{IRDY}}$: The current master (initiator) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{TRDY}}$: The target device (selected device) uses this signal to indicate that it is ready to read or write valid data.

$\overline{\text{STOP}}$: This signal indicates that the current target wishes the initiator to stop the current transaction.

$\overline{\text{LOCK}}$: The signal indicates an automatic operation that may require multiple transactions.

IDSEL : This initialization device select signal is used as a chip select during configuration read and write transactions.

$\overline{\text{DEVSEL}}$: This device select signal is used by the target to indicate device has selected.

- **Arbitration Signals :** Unlike other PCI signals, these are not shared lines. Rather, each PCI master has its own Request and Grant arbitration signals that connect it directly to the PCI bus arbiter.
- **Error Reporting Signals :** These are used to report parity and system errors ($\overline{\text{PERR}}$, $\overline{\text{SERR}}$).
- **Interrupt Signals :** Like bus arbitration signals these are not shared signals. Rather, each PCI device has its own interrupt signals ($\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$).

- **Cache Support Signals :** These signals support snoopy cache protocol ($\overline{SBO}$, SDONE).
- **JTAG/Boundary Scan Signals :** These signals support testing procedures defined in IEEE standard 149.1 (TCK - Test clock, TDI - Test Input, TDO - Test Output, TMS - Test Mode Select, TRST - Test Reset).

1.4.3.4 Data Transfer

Every data transfer on the PCI bus is a single transaction consisting of one address phase and one or more data phases. Let us see the typical read cycle to read 32-bit word from memory on the PCI bus. All the events during read cycle are synchronized with the falling edge of the clock cycle. The devices connected to the bus sample the bus lines on the rising edge at the beginning of a bus cycle. The events during the bus cycle are explained below and they are also labeled on the diagram.

1. The master asserts $\overline{FRAME}$ signal to indicate the start of a transaction. This signal remains asserted until initiator is ready to begin the data phase. The initiator puts the start address on the address bus, and initiates read cycle by activating C/$\overline{BE}$ lines.
2. At the beginning of clock2, the target device recognizes its address on the address bus.
3. The initiator ceases the address lines and it changes the information on the C/$\overline{BE}$ lines to designate which address lines are to be used for transfer for the currently addressed data from 1 to 4 bytes. The initiator also asserts IRDY to indicate that it is ready for the first data byte.
4. The selected target asserts $\overline{DEVSEL}$ to indicate that it has recognized its address and it responds by placing the requested data on the address bus. It then asserts $\overline{TRDY}$ to indicate that valid data is present on the bus.
5. The initiator reads the data at the beginning of clock 4 and changes the byte enable lines as needed in preparation for the next read.
6. The initiator deasserts $\overline{FRAME}$ signal indicating that it is second last data byte transfer.
7. The initiator deasserts $\overline{IRDY}$ and the target deasserts $\overline{TRDY}$ and $\overline{DEVSEL}$, returning bus to the idle state.

Similar kind of handshaking signals are activated during PCI write operation. Here, the data from initiator is copied into the targeted devices.

The Fig. 1.4.5 (b) shows how the $\overline{IRDY}$ and $\overline{TRDY}$ signals can be used by the initiator and target, respectively to pause the data transfer in the middle of a transaction. As shown in the Fig. 1.4.5 (b) the read cycle operation is same upto two data bytes transfer.

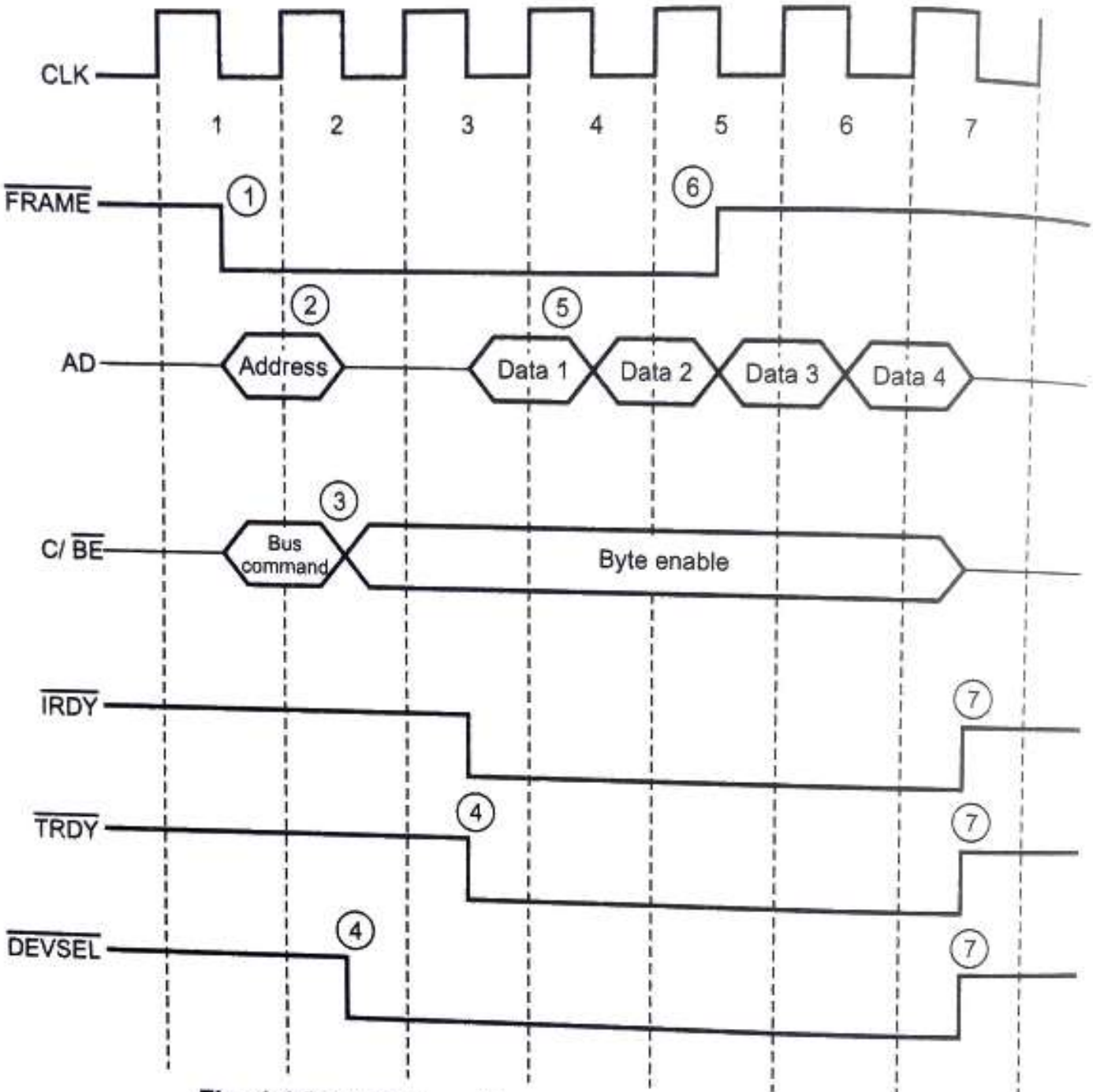


Fig. 1.4.5 (a) Timing diagram for PCI read operation

The target sends the third word in cycle 5. In Fig. 1.4.5 (b), it is assumed that the initiator is not able to receive it. Hence, it deasserts $\overline{\text{IRDY}}$. In response, the target holds the third data byte on the AD lines until $\overline{\text{IRDY}}$ is asserted again.

In cycle 6, it is assumed that the initiator is ready to receive next data bytes thus is asserts $\overline{\text{IRDY}}$ signal and reads third data byte. At this point, it is assumed that the target is not ready to transfer the fourth word. The target informs this by deasserting $\overline{\text{TRDY}}$ line. In cycle 8, target sends the fourth word and asserts $\overline{\text{TRDY}}$. Since $\overline{\text{FRAME}}$ signal is deasserted during the third data byte transfer, the transaction ends after the fourth data byte transfer.

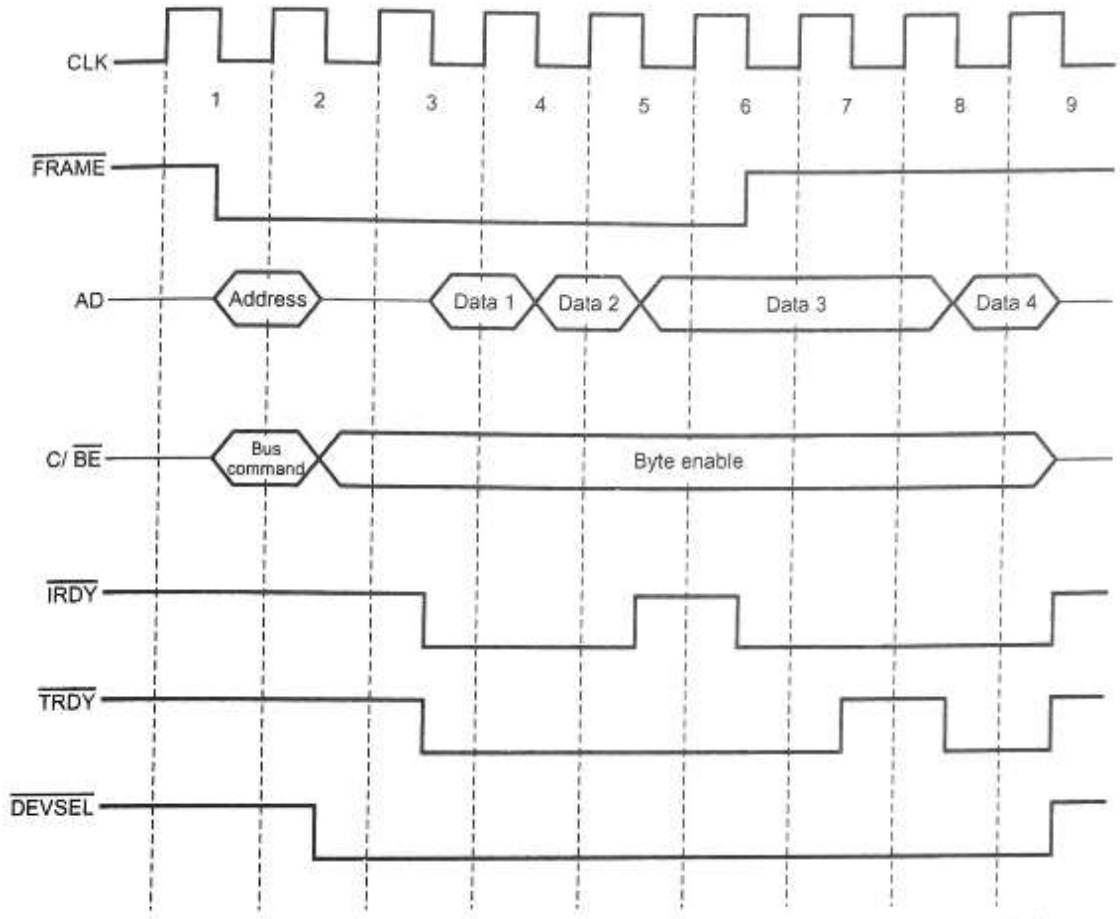


Fig. 1.4.5 (b) PCI read operation showing the role of IRDY/ TRDY

Device Configuration

When an I/O device is connected to a computer, it is necessary to configure both the device and the driver software for that device. To configure the device we may need to set switches and jumpers. On the other hand, to configure software we need to know the device's address, characteristics and so on. The PCI simplifies the process of configuration of devices by incorporating a small configuration ROM memory in each I/O device interface. This memory stores information about that device. The configuration ROMs of all devices are accessible in the configuration address space. When power up or reset, PCI initialization software reads these ROMs, determines the type of the devices and assigns addresses to them. During bus configuration operation, addresses are not yet assigned to the devices and hence they cannot accessed based on their addresses.

During a configuration operation, IDSEL (Initialization device select) signal is used to select the device rather than the device addresses. About of 32 address lines, AD11 to AD31, i.e. 21 address lines are used to connect IDSEL pin of each device. This restricts the number of I/O devices that can be connected to 21. The lower 11 address lines AD0 to AD10 are used to specify the type of operation and to access the contents of the device configuration ROM.

The configuration software scans all 21 locations in the configuration address space to identify which devices are present. It then assigns the addresses and priority to them. The PCI bus has four interrupt-request lines. By writing into a device configuration register, the software instructs the device as to which of these lines it can use to request an interrupt.

Review Questions

1. *What is a bus ? Explain single bus structure in an architecture.*
2. *What is a bus ? Explain single bus and multiple bus structure used to interconnect functional units in computer system.*
3. *List the features of PCI bus.*
4. *Write a note on PCI configurations.*
5. *Describe the PCI bus structure.*
6. *Explain the PCI commands.*
7. *Write a note on PCI arbitration.*
8. *Describe how a read operation is performed on a PCI bus.*

1.5 Interrupt and Class of Interrupts

- Sometimes it is necessary to have the computer automatically execute one of a collection of special routines whenever certain conditions exists within a program or the computer system e.g. It is necessary that computer system should give response to devices such as keyboard, sensor and other components when they request for service.
- This method provides an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine (Interrupt Service Routine) that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off. The event that causes the interruption is called **interrupt** and the special routine executed to service the interrupt is called **Interrupt Service Routine (ISR).**

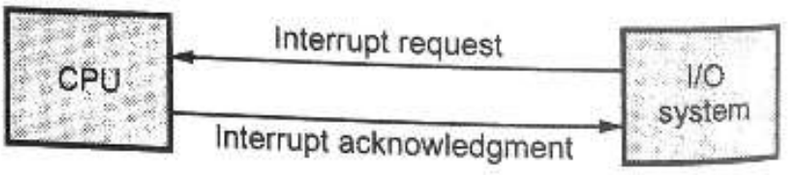


Fig. 1.5.1

- The interrupt service routine is different from subroutine because the address of ISR is predefined or it is available in Interrupt Vector Table (IVT), whereas subroutine address is necessarily to be given in subroutine CALL instruction. IRET instruction is used to return from the ISR whereas RET instruction is used to return from subroutine. IRET instruction restores flag contents along with CS and

IP in the IA-32 architecture; however RET instruction only restores CS and IP contents.

- An interrupt caused by an external signal is referred as a **hardware interrupt**.
- Conditional interrupts or interrupts caused by special instructions are called **software interrupts**.

1.5.1 Enabling and Disabling Interrupts

- Most of the processors provide the masking facility. In the processor those interrupts which can be masked under software control are called **maskable interrupts**.
- The interrupts which can not be masked under software control are called **non-maskable interrupts**.
- Maskable interrupts are enabled and disabled under program control. By setting or resetting particular flip-flops in the processor, interrupts can be masked or unmasked, respectively.
- When masked, processor does not respond to the interrupt even though the interrupt is activated.

1.5.2 Vector Interrupts

- When the external device interrupts the processor (interrupt request), processor has to execute interrupt service routine for servicing that interrupt. If the internal control circuit of the processor produces a CALL to a predetermined memory location which is the starting address of interrupt service routine, then that address is called **vector address** and such interrupts are called **vector interrupts**. For vector interrupts fastest and most flexible response is obtained since such an interrupt causes a direct hardware-implemented transition to the correct interrupt-handling program. This technique is called **vectoring**. When processor is interrupted, it reads the vector address and loads it into the PC.

1.5.3 Interrupt Nesting

- For some devices, a long delay in responding to an interrupt request may cause error in the operation of computer. Such interrupts are acknowledged and serviced eventhough processor is executing an interrupt service routine for another device.
- A system of interrupts that allows an interrupt service routine to be interrupted is known as **nested interrupts**.

1.5.4 Interrupt Priority

- When interrupt requests arrive from two or more devices simultaneously, the processor has to decide which request should be serviced first and which one should be delayed. The processor takes the decision with the help of interrupt priorities.
- It accepts the request having the highest priority.

1.5.5 Recognition of Interrupt and Response to Interrupt

- The CPU recognizes the interrupt when the external asynchronous input (interrupt input) is asserted (a signal is sent to the interrupt input) by an I/O device.
- In response to an interrupt a special sequence of actions are performed. These are as follows :

- When a processor is interrupted, it stops executing its current program and calls a special routine which "services" the interrupt. The event that causes the interruption is called **interrupt** and the special routine which is executed is called **interrupt service routine.**

1. The processor completes its current instruction. No instruction is cut-off in the middle of its execution.
2. The program counter's current contents are stored on the stack. Remember, during the execution of an

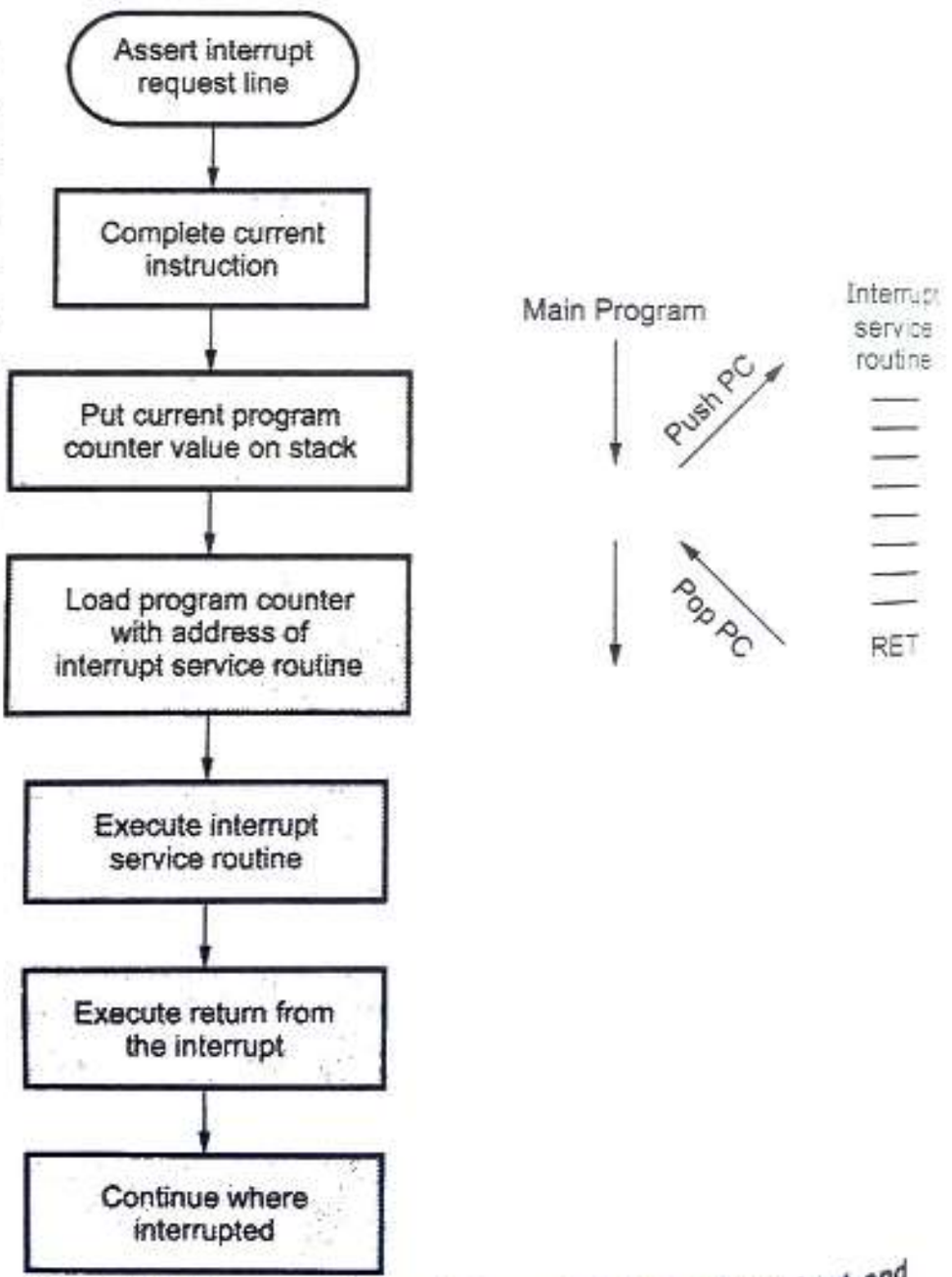


Fig. 1.5.2 Response to an interrupt with the flowchart and diagram

instruction the program counter is pointing to the memory location for the next instruction.

3. The program counter is loaded with the address of an interrupt service routine.
4. Program execution continues with the instruction taken from the memory location pointed by the new program counter contents.
5. The interrupt program continues to execute until a return instruction is executed.
6. After execution of the RET instruction processor gets the old address (the address of the next instruction from where the interrupt service routine was called.) of the program counter form the stack and puts it back into the program counter. This allows the interrupted program to continue executing at the instruction following the one where it was interrupted. Fig. 1.5.2 shows the response to an interrupt with the flowchart and diagram.

Review Questions

1. *What is interrupt ?*
2. *What is a non-maskable interrupt ? What is the action performed on receipt of a NMI ?*
3. *What are vectored interrupts ?*
4. *What do you mean by interrupt nesting ?*
5. *What is priority interrupt ?*

1.6 Instruction Cycle (I - Cycle)

- An instruction cycle involves three subcycles,
  - **Fetch :** The fetch phase reads the next instruction from memory into the CPU.
  - **Decode :** The decode phase interprets the opcode by decoding it.
  - **Execute :** The execute phase performs the indicated operation.
- Fig. 1.6.1 shows the basic instruction cycle.
- Actually, processor checks for valid interrupt request after each instruction cycle. If any valid interrupt request is present, processor saves the current process state and services the interrupt.
- If the operands on which the instruction works are present within the processor-registers, a memory access is not required. But if the execution of an instruction involves one or more operands in memory, each requires a memory access.
- For fetching the indirect addresses, one or more instruction sub cycles are required.

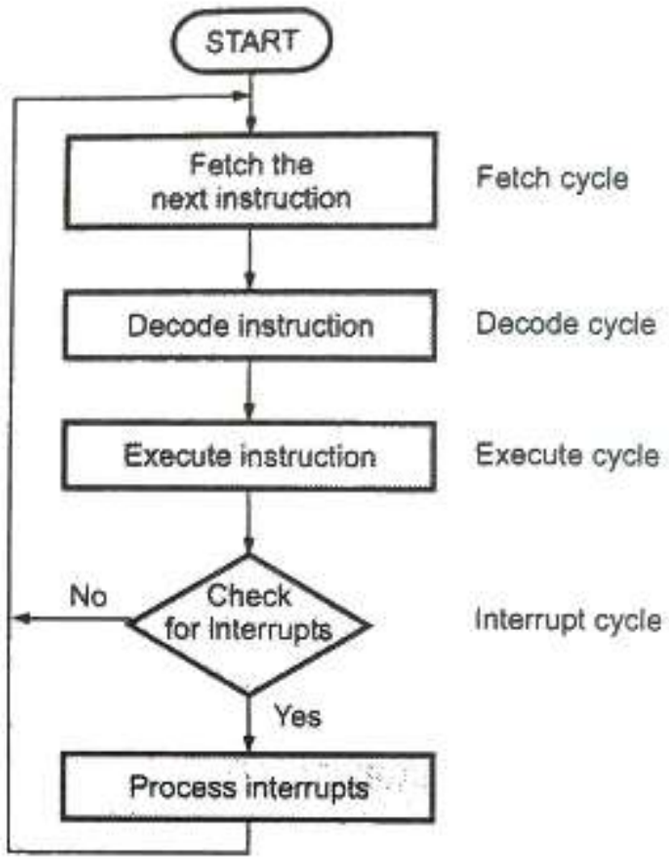


Fig. 1.6.1 Basic instruction cycle with interrupt cycle

- After fetching the instruction, it is decoded and if any indirect addressing is involved, the required operands are fetched using indirect addressing.
- Also, after performing the operation on the operands according to the opcode, a similar process may be needed to store the result in memory.
- Following execution, interrupt processing may be required before fetchingthe next instruction. The same process can be viewed as shown in Fig. 1.6.2.

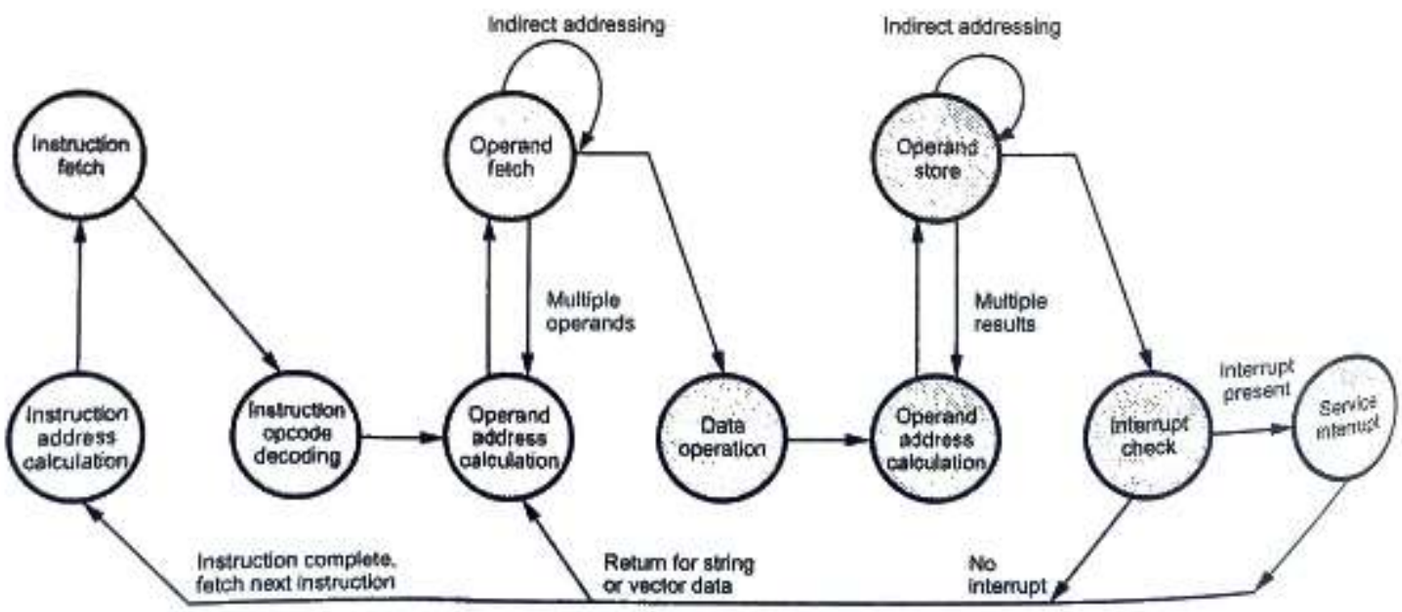


Fig. 1.6.2 Instruction cycle state diagram

1.6.1 Fetch Cycle

- During the **fetch cycle** an instruction opcode is read from memory. The following sequence of operations take place during this cycle.
  - The address of an instruction to be fetched from memory is in PC. This address is moved to AR.
  - The contents of AR (i.e. address) is placed on the address bus.
  - The control unit generates the control signal to read the contents of the memory.
- The contents of the addressed memory location are placed on the data bus and copied into the DR.
- The data in DR is moved to IR.
- Meanwhile, the PC is incremented by one for pointing the next data in memory.

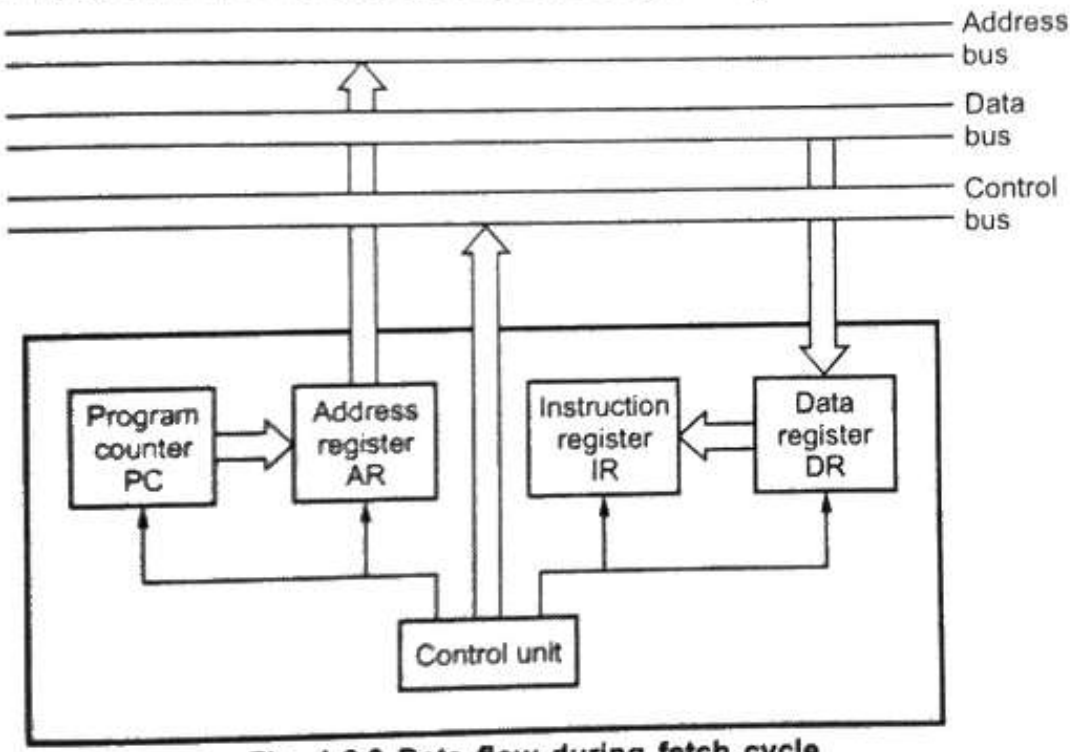


Fig. 1.6.3 Data flow during fetch cycle

1.6.2 Decode Cycle

During decode cycle, the IR contents (i.e. opcode) are decoded by ID. During decoding operation, it is determined that the operands are present within registers of CPU or indirect addressing is required. For indirect addressing, indirect cycle is performed. The DR contains the address reference. This is transferred to the AR. The control unit generates the control signal to read the desired address of the operand into DR as shown in Fig. 1.6.4.

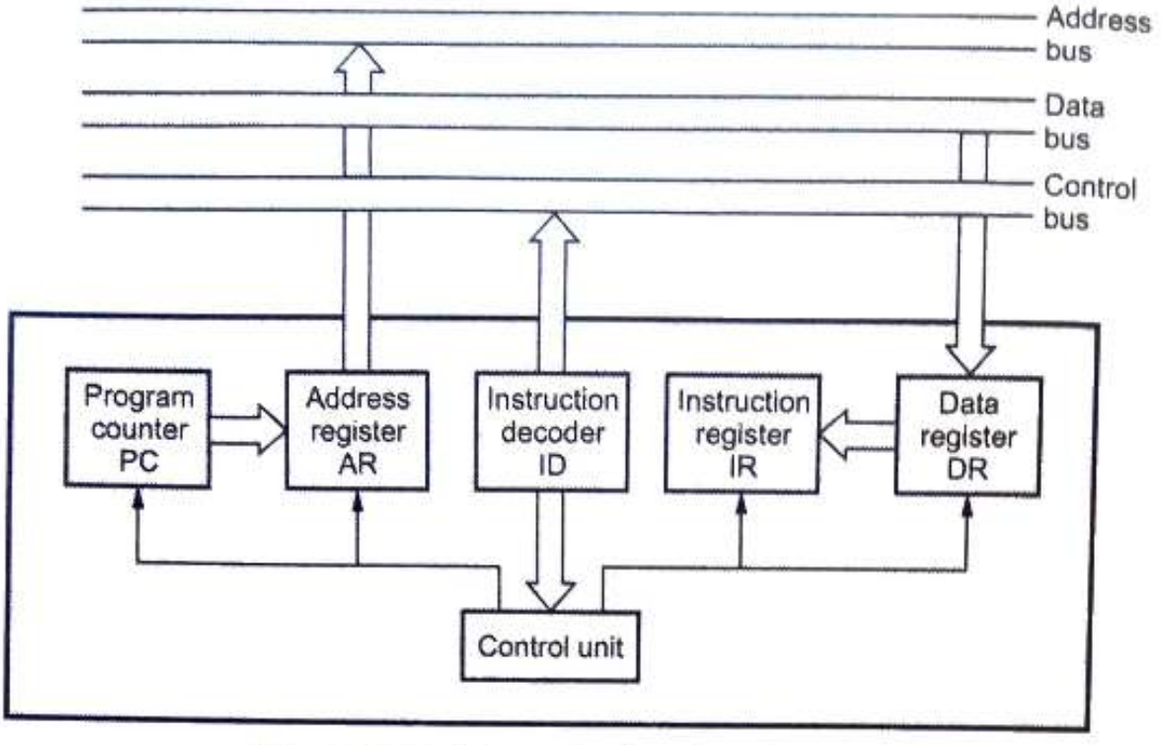


Fig. 1.6.4 Data flow during decode cycle

1.6.3 Interrupt Cycle

- Fig. 1.6.5 shows the data flow during interrupt cycle.

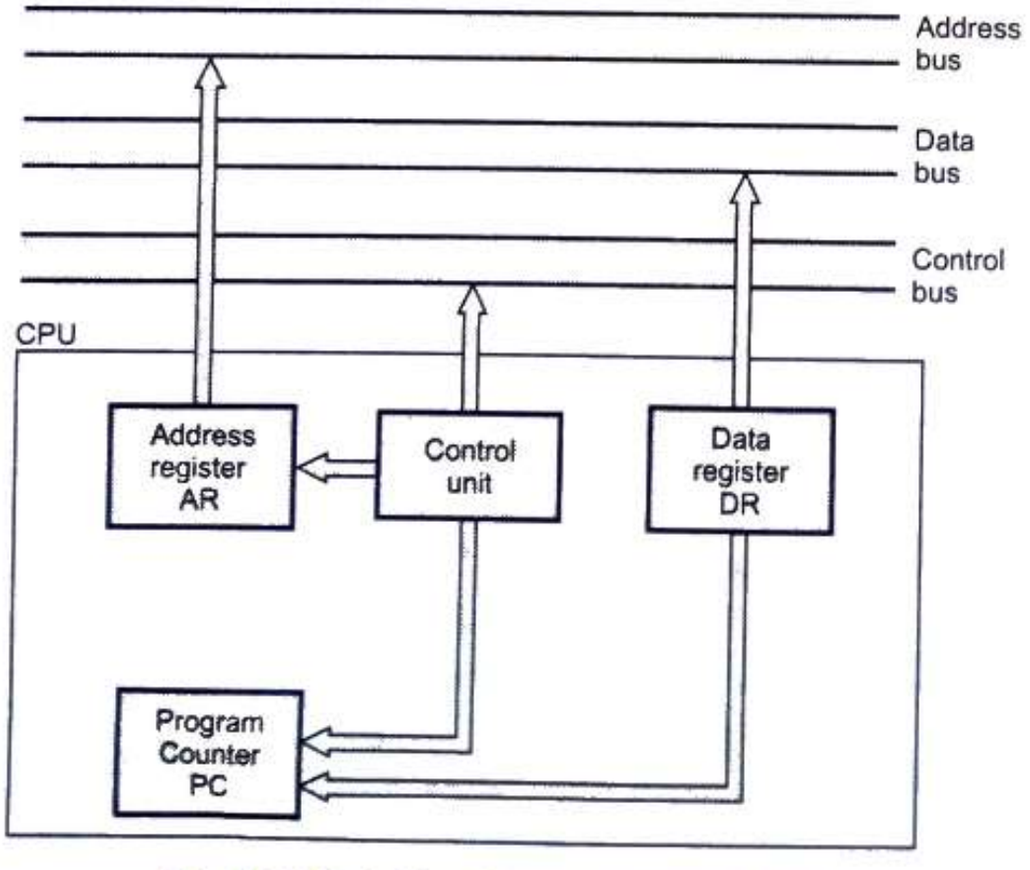


Fig. 1.6.5 Data flow during interrupt cycle

- During interrupt cycle, the current contents of PC are saved so that the CPU can resume normal activity after servicing the interrupt. So, the contents of the program counter, PC are transferred to the data register DR, to be written into memory. The special memory location is reserved for this purpose. For example, stack.
- This memory location or address is loaded into the address register, AR from the control unit. It might, for example, be a stack pointer. The interrupt service routine address is loaded into the PC. So, the next instruction cycle will begin by fetching the first instruction from the interrupt service routine.

Review Questions

1. *Explain the instruction cycle.*

   OR

   Explain instruction cycle states of a processor with suitable diagram.
2. *Explain sequence of events that occur in fetch cycle symbolically with diagrams at each stage.*
3. *Explain sequence of events that occur in decode cycle.*
4. *With the help of neat diagram explain the data flow during interrupt cycle.*

❑❑❑

UNIT I

2 Number Systems

Syllabus

Number systems, Decimal Number system, Binary number system, Octal & Hexadecimal number system, 1's & 2's complement, Binary Fixed Point Representation.

Contents

2.1 Introduction

- Number system is a basis for counting various items.
- The decimal number system has 10 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- Modern computers communicate and operate with binary numbers which use only the digits 0 and 1.
- When decimal quantities are represented in the binary form, they take more digits.
- For large decimal numbers people have to deal with very large binary strings and therefore, they do not like working with binary numbers. This fact gave rise to three new number systems : **Octal**, **Hexadecimal** and **Binary Coded Decimal (BCD)**.

Review Questions

1. *Explain various number systems.*
2. *Name the number system used in computers.*

2.2 Representation of Numbers in Different Radix

2.2.1 Decimal Number System

- In decimal number system we can express any decimal number in units, tens, hundreds, thousands and so on.
- When we write a decimal number say, 5678.9, we know it can be represented as 5000 + 600 + 70 + 8 + 0.9 = 5678.9
- The decimal number 5678.9 can also be written as 5678.9_{10}, where the 10 subscript indicates the **radix or base.**
- The position of a digit with reference to the decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number being represented.
- The leftmost digit, which has the greatest weight is called the **most significant digit** and the rightmost digit, which has the least weight, is called the **least significant digit.**
- Fig. 2.2.1 shows decimal digit and its weights expressed as a power of 10.

	10^3	10^2	10^1	10^0		10^{-1}
	5	6	7	8	.	9
In power of 10	5×10^3	6×10^2	7×10^1	8×10^0	.	9×10^{-1}
	MSD					LSD

Fig. 2.2.1 Representation of a decimal number

2.2.2 Binary Number System

- Binary system with its two digits is a **base-two system.**
- The two binary digits (bits) are 1 and 0.
- In binary system, weight is expressed as a power of 2.
- The Fig. 2.2.2 (a) shows representation of binary number 1101.101 in power of 2.
- By adding each digit of a binary number in a power of 2 we can find the decimal equivalent of the given binary number.

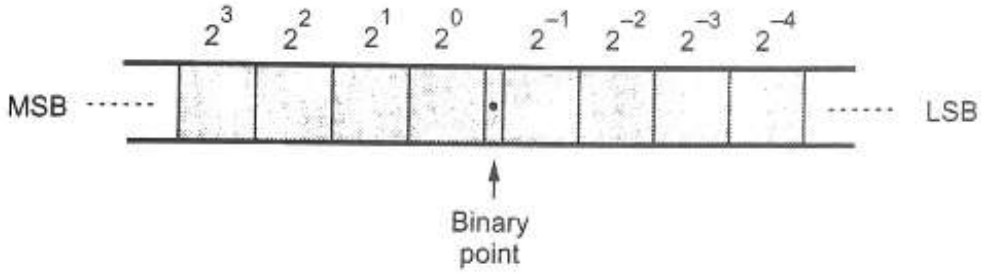


Fig. 2.2.2 Binary position values as a power of 2

	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
N =	1	1	0	1	.	1	0	1
	1×2^3	1×2^2	0×2^1	1×2^0	.	1×2^{-1}	0×2^{-2}	1×2^{-3}

$$N = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = (13.625)_{10}$$

Fig. 2.2.2 (a)

2.2.3 Octal Number System

- The octal number system uses first eight digits of decimal number system : 0, 1, 2, 3, 4, 5, 6 and 7. As it uses 8 digits, its base is 8.
- For example, the octal number 5632.471 can be represented in power of 8 as shown in Fig. 2.2.2 (b).

	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
N =	5	6	3	2	.	4	7	1
	5×8^3	6×8^2	3×8^1	2×8^0	.	4×8^{-1}	7×8^{-2}	1×8^{-3}

$$N = 5\times8^3 + 6\times8^2 + 3\times8^1 + 2\times8^0 + 4\times8^{-1} + 7\times8^{-2} + 1\times8^{-3} = (2970.611328)_{10}$$

Fig. 2.2.2 (b)

- By adding each digit of an octal number in a power of 8 we can find the decimal equivalent of the given octal number.

2.2.4 Hexadecimal Number System

- The hexadecimal number system has a base of 16 having 16 characters : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- Table 2.2.1 shows the relationship between decimal, binary, octal and hexadecimal.
- For example, 3FD.84 can be represented in power of 16 as shown below.
- By adding each digit of a hexadecimal number in a power of 16 we can find decimal equivalent of the given hexadecimal number.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 2.2.1 Relation between decimal, binary, octal and hexadecimal numbers

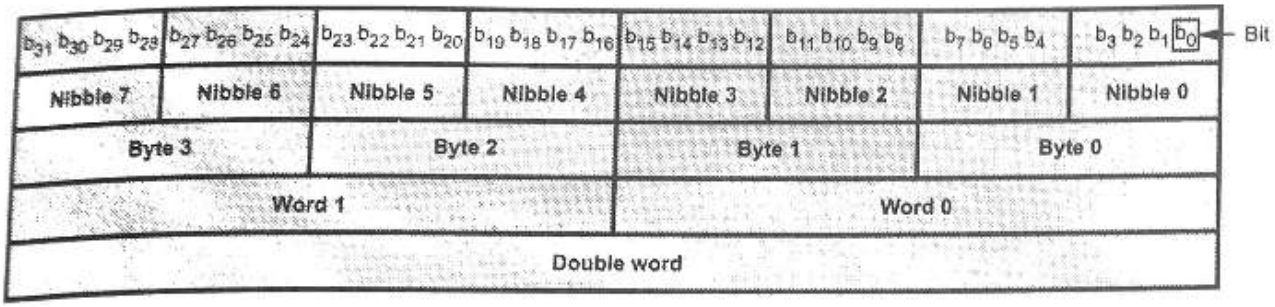
	16^2	16^1	16^0		16^{-1}	16^{-2}
N =	3	F	D	.	8	4
	3×16^2	$F \times 16^1$	$D \times 16^0$	.	8×16^{-1}	4×16^{-2}

$$N = 3 \times 16^2 + F \times 16^1 + D \times 16^0 + 8 \times 16^{-1} + 4 \times 16^{-2}$$

$$= 3 \times 16^2 + 15 \times 16^1 + 13 \times 16^0 + 8 \times 16^{-1} + 4 \times 16^{-2} = (1021.515625)_{10}$$

2.2.5 Format of a Binary Number

- A single digit in the binary number is called **bit.**
- The following figure shows the format of binary number. Four binary digits form a **nibble,** eight binary digits form a **byte,** sixteen binary digits form a word and thirty-two binary digits form a double-word.



Nibble : 4-bits can represent 2^4 = 16 distinct values

Byte : 8-bits can represent 2^8 = 256 distinct values

Word : 16-bits can represent 2^{16} = 65536 distinct values

Double word : 32-bits can represent 2^{32} = 4294967296 distinct values

2.2.6 Counting in Radix (Base) r

- Each number system has r set of characters. For example, in decimal number system r equals to 10 has 10 characters from 0 to 9, in binary number system r equals to 2 has 2 characters 0 and 1 and so on.
- In general we can say that, a number represented in radix r, has r characters in its set and r can be any value. This is illustrated in Table 2.2.2.

Radix (Base) r	Characters in set
2 (Binary)	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3, 4, 5
7	0, 1, 2, 3, 4, 5, 6
8 (Octal)	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
:	
:	
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Table 2.2.2 Radix and character set

Example 2.2.1 *Find the decimal equivalent of $(231.23)_4$.*

Solution :
$$N = 2 \times 4^2 + 3 \times 4^1 + 1 \times 4^0 + 2 \times 4^{-1} + 3 \times 4^{-2}$$
$$= 32 + 12 + 1 + 0.5 + 0.1875$$
$$= \mathbf{45.6875_{10}}$$

Example 2.2.2 *Count from 0 to 9 in radix 5.*

Solution : The Table 2.2.2 indicates that radix 5 has 5 characters. A count sequence from 0 decimal to 9 decimal is

00, 01, 02, 03, 04, 10, 11, 12, 13, 14

Example 2.2.3 *What is the largest binary number that can be expressed with 12-bits ? What is the equivalent decimal and hexadecimal ?*

Solution : $(1111\ 1111\ 1111)_2$, $(4095)_{10}$, $(FFF)_H$.

2.3 Conversion of Numbers from One Radix to Another Radix

2.3.1 Binary to Octal Conversion

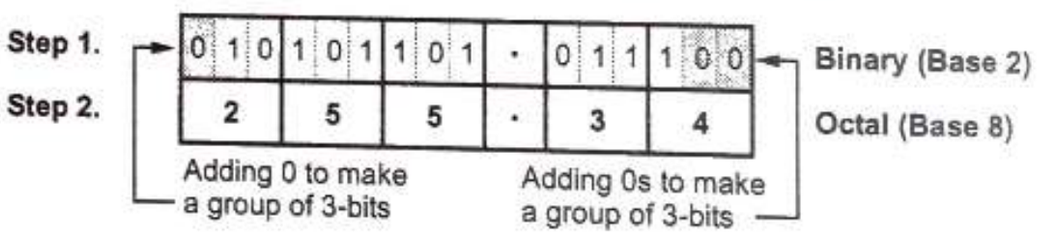
- The base for octal number is 8 and the base for binary number is 2.
- The base for octal number is the third power of the base for binary numbers. Therefore, by grouping 3 digits of binary numbers and then converting each group digit to its octal equivalent we can convert binary number to its octal equivalent.

Example 2.3.1 *Convert 10101101.0111 to octal equivalent.*

Solution :

Step 1 : Make group of 3-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 2 : Write equivalent octal number for each group of 3-bits.



$\therefore \quad (10101101.0111)_2 = (255.34)_8$

2.3.2 Octal to Binary Conversion

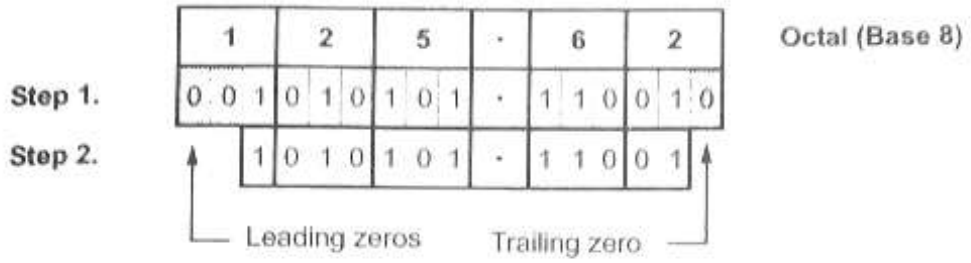
- Conversion from octal to binary is a reversal of the process explained in the previous section. Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

Example 2.3.2 *Convert $(125.62)_8$ to binary.*

Solution :

Step 1 : Write equivalent 3-bit binary number for each octal digit.

Step 2 : Remove any leading or trailing zeros.



$\therefore \quad (125.62)_8 = (1010101.11001)_2$

2.3.3 Binary to Hexadecimal Conversion

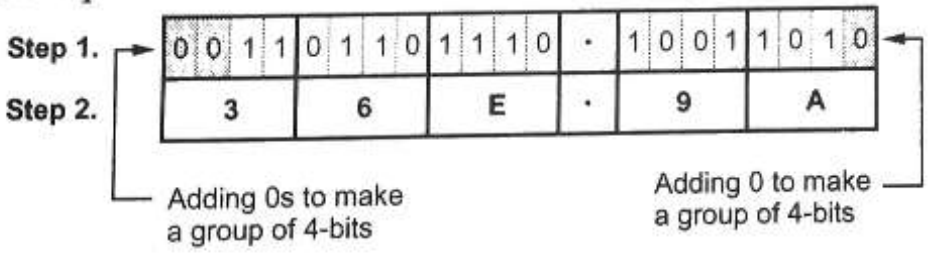
- The base for hexadecimal numbers is 16 and the base for binary numbers is 2.
- The base for hexadecimal number is the fourth power of the base for binary numbers. Therefore, by grouping 4 digits of binary numbers and then converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

Example 2.3.3 *Convert 1101101110 · 1001101 to hexadecimal equivalent.*

Solution :

Step 1 : Make group of 4-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 2 : Write equivalent hexadecimal number for each group of 4-bits.



$\therefore \quad (1101101110.1001101)_2 = (36E.9A)_{16}$

2.3.4 Hexadecimal to Binary Conversion

- Conversion from hexadecimal to binary is a reversal of the process explained in the previous section. Each digit of the hexadecimal number is individually converted to its binary equivalent to get hexadecimal to binary conversion of the number.

Example 2.3.4 *Convert $(8A9.B4)_{16}$ to binary.*

Solution :

Step 1 : Write equivalent 4-bit binary number of each hexadecimal digit.

Step 2 : Remove any leading or trailing zeros.

	8	A	9	·	B	4
Step 1.	1 0 0 0	1 0 1 0	1 0 0 1	·	1 0 1 1	0 1 0 0
Step 2.	1 0 0 0	1 0 1 0	1 0 0 1	·	1 0 1 1	0 1

Trailing zeros

$\therefore \quad (8A9.B4)_{16} = (1000\ 1010\ 1001.101101)_2$

2.3.5 Octal to Hexadecimal Conversion

The easiest way to convert octal number to hexadecimal number is given below :

1. Convert octal number to its binary equivalent.
2. Convert binary number to its hexadecimal equivalent.

Example 2.3.5 *Convert $(615.25)_8$ to its hexadecimal equivalent.*

Solution :

Step 1 : Write equivalent 3-bit binary number for each octal digit.

Step 2 : Make group of 4-bits starting from LSB for integer part and MSB fractional part by adding 0s at the end, if required.

Step 3 : Write equivalent octal number for each group to 4-bits.

	6	1	5	·	2	5	
Step 1.	1 1 0	0 0 1	1 0 1	·	0 1 0	1 0 1	Octal (Base 8)
Step 2.	0 0 0 1	1 0 0 0	1 1 0 1	·	0 1 0 1	0 1 0 0	Binary (Base 2)
Step 3.	1	8	D	·	5	4	Hex (Base 16)

Adding 0s to make a group of 4-bits

Adding 0s to make a group of 4-bits

2.3.6 Hexadecimal to Octal Conversion

The easiest way to convert hexadecimal number to octal number is given below :

1. Convert hexadecimal number to its binary equivalent.
2. Convert binary number to its octal equivalent.

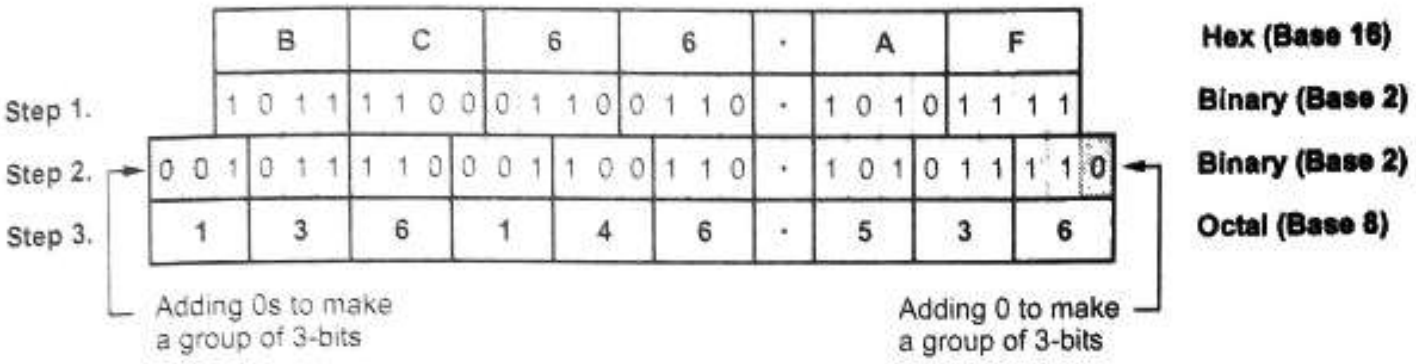
Example 2.3.6 *Convert $(BC66.AF)_{16}$ to its octal equivalent.*

Solution :

Step 1 : Write equivalent 4-bit binary number for each hexadecimal digit.

Step 2 : Make group of 3-bits starting from LSB for integer part and MSB for fractional part by adding 0s at the end, if required.

Step 3 : Write equivalent octal number for each group of 3-bits.



2.3.7 Converting Any Radix to Decimal

- In general, numbers can be represented as,

$$N = A_{n-1}r^{n-1} + A_{n-2}r^{n-2} + \ldots + A_1r^1 + A_0r^0 + A_{-1}r^{-1} + A_{-2}r^{-2} + \ldots C_{-m}r^{-m}$$

where N = Number in decimal
A = Digit
r = Radix or base of a number system
n = The number of digits in the integer portion of number
m = The number of digits in the fractional portion of number

- From this general equation we can convert number with any radix into its decimal equivalent. This is illustrated using following example.

Example 2.3.7 *Convert $(3102.12)_4$ to its decimal equivalent.*

Solution : $N = 3\times4^3 + 1\times4^2 + 0\times4^1 + 2\times4^0 + 1\times4^{-1} + 2\times4^{-2}$

$= 192 + 16 + 0 + 2 + 0.25 + 0.125 = \mathbf{210.375_{10}}$

Example 2.3.8 *Determine the value of base x, if :* $(193)_x = (623)_8$

Solution : $(193)_x = (623)_8$

Converting octal into decimal : $6\times8^2 + 2\times8 + 3 = (403)_{10} = (623)_8$

$\therefore \quad (193)_x = 1\times x^2 + 9\times x + 3\times x^0 = (403)_{10}$

$\therefore \quad x^2 + 9x + 3 = 403 \qquad \therefore \; x = 16 \text{ or } x = -25$

Negative is not applicable $\qquad \therefore \; x = 16$

$\therefore \quad (193)_{16} = \mathbf{(623)_8}$

2.3.8 Conversion of Decimal Number to Any Radix Number

Step 1 : Convert integer part.

Step 2 : Convert fractional part.

- The conversion of integer part is accomplished by successive division method and the conversion of fractional part is accomplished by successive multiplication method.

Steps in Successive Division Method

1. Divide the integer part of decimal number by desired base number, stating quotient (Q) and remainder (R).
2. Consider quotient as a new decimal number and repeat step 1 until quotient becomes 0.
3. List the remainders in the reverse order.

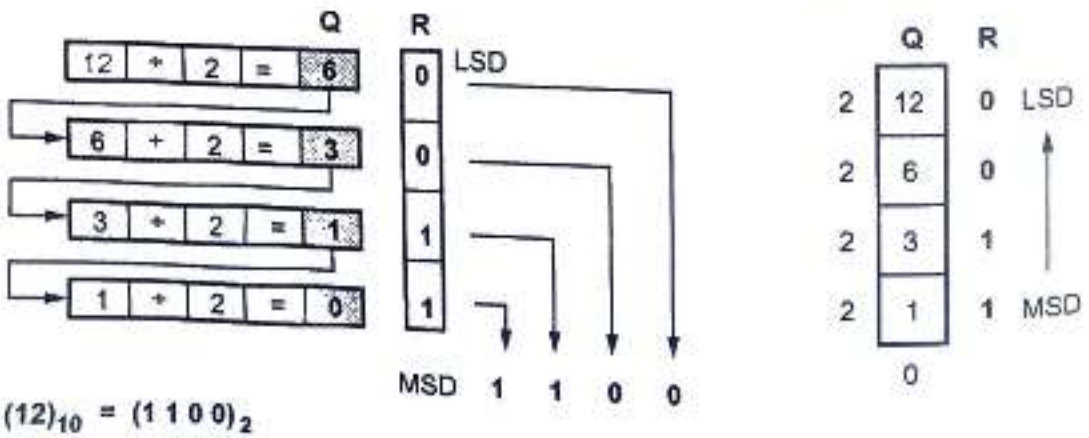
Steps in Successive Multiplication Method

1. Multiply the fractional part of decimal number by desired base number.
2. Record the integer part of product as carry and fractional part as new fractional part.
3. Repeat steps 1 and 2 until fractional part of product becomes 0 or until you have as many digits as necessary for your application.
4. Read carries downwords to get desired base number.

Example 2.3.9 *Convert 12.125 decimal into binary.*

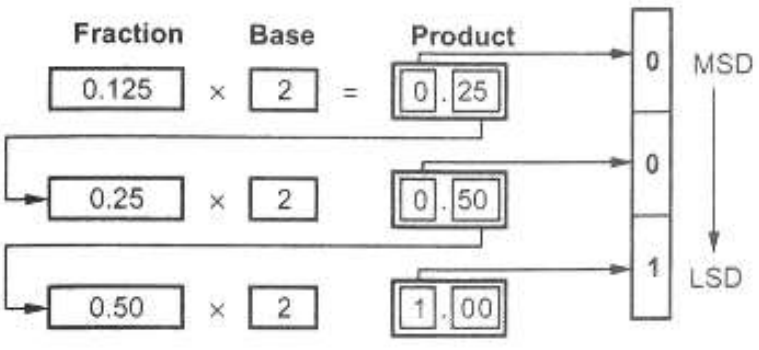
Solution :

Integer part : Conversion of integer part by successive division method.



$(12)_{10} = (1\ 1\ 0\ 0)_2$

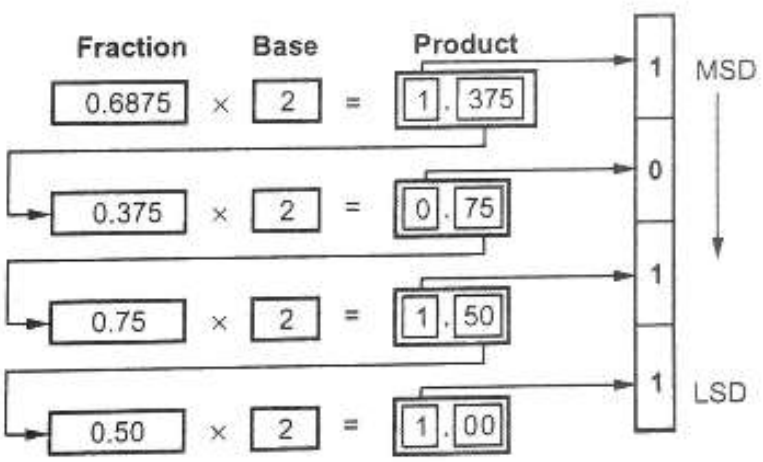
Fractional part : Conversion of fractional part by successive multiplication method.



$(0.125)_{10} = (0.001)_2$ $(12.125)_{10} = (1100.001)_2$

Example 2.3.10 *Convert $(0.6875)_{10}$ to binary.*

Solution :

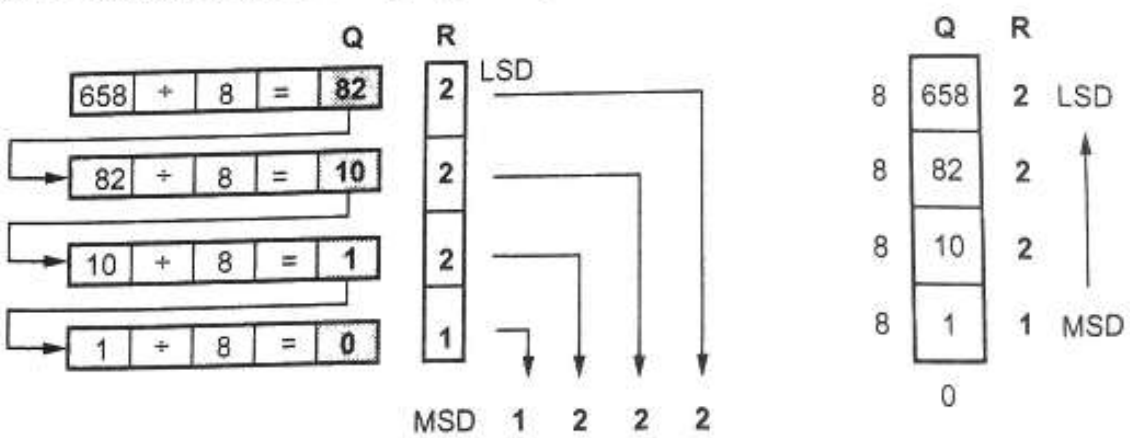


$\therefore (0.6875)_{10} = (0.1011)_2$

Example 2.3.11 *Convert 658.825 decimal into octal.*

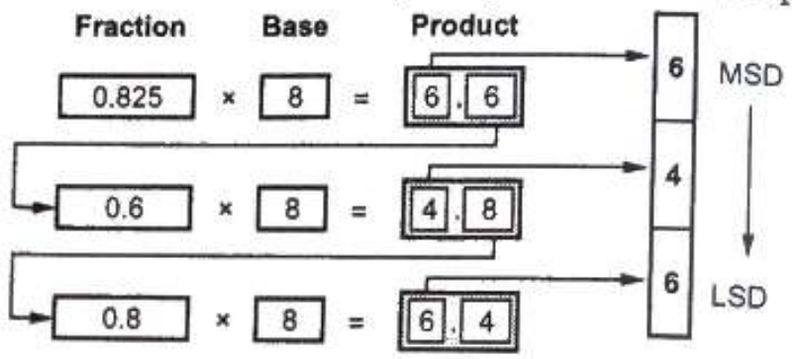
Solution :

Integer part : Conversion of integer part by successive division method.



$(658)_{10} = (1222)_8$

Fractional part : Conversion of fractional part by successive multiplication method.



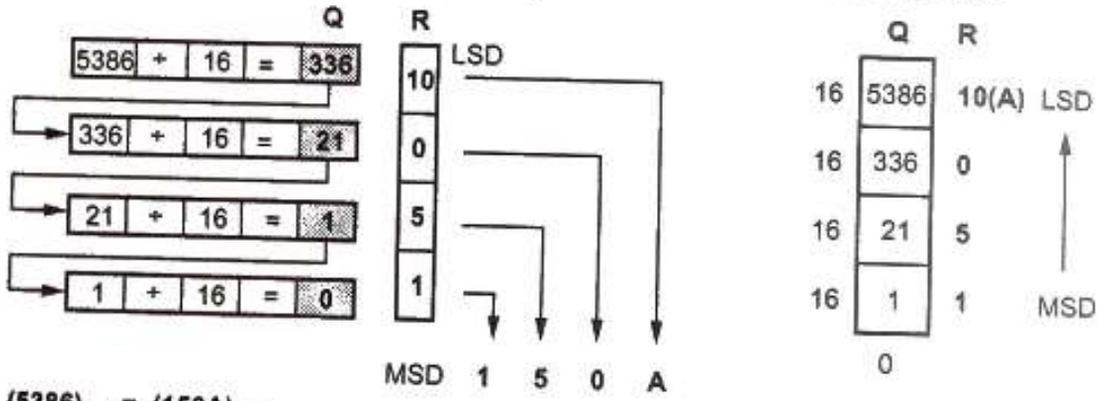
$(0.825)_{10} = (0.646)_8$ $\therefore (658.825)_{10} = (1222.646)_8$

In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 8 until we have as many digits as necessary for our application.

Example 2.3.12 *Convert 5386.345 decimal into hexadecimal.*

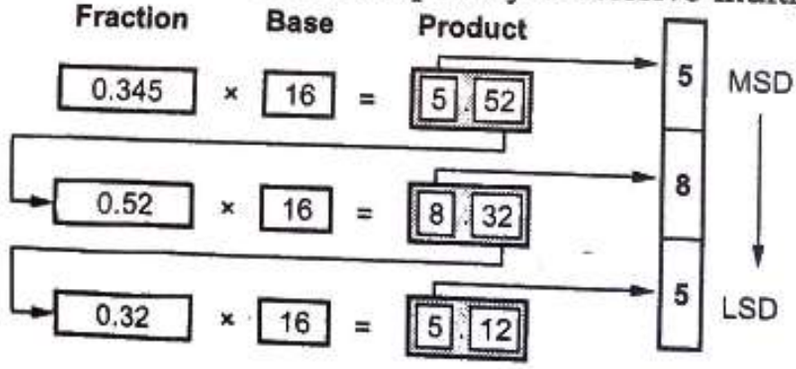
Solution :

Integer part : Conversion of integer part by successive division method.



$(5386)_{10} = (150A)_{16}$

Fractional part : Conversion of fractional part by successive multiplication method.



$(0.345)_{10} = (0.585)_{16}$ $\therefore (5386.345)_{10} = (150A.585)_{16}$

In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 16 until we have as many digits as necessary for our application.

Example 2.3.13 *The solution to the quadratic equation $x^2 - 11x + 22 = 0$ is $x = 3$ and $x = 6$. What is the base of the numbers ?*

Solution : Given quadratic equation is $x^2 - 11x + 22 = 0$ and it is given that $x = 3$ and $x = 6$

$$\therefore \quad (x - 3)(x - 6) = x^2 - 11x + 22 \qquad ...(1)$$

The numbers 3 and 6 are same in any base, whose value is more than 6.

i.e., $(3)_b = (3)_{10}$

$(6)_b = (6)_{10}$ where b = Base

Solving equation (1) we have,

$$(x^2 - 9x + 18)_{10} = (x^2 - 11x + 22)_b$$

Comparing coefficients of x we have

$$(-9)_{10} = (-11)_b$$

$$\therefore \quad (9)_{10} = (11)_b$$

$$\therefore \quad 9 = b^1 + b^0 = b + 1$$

$$\therefore \quad b = 8$$

or comparing constants we have,

$$(18)_{10} = (22)_b$$

$$\therefore \quad 18 = 2b + 2$$

$$\therefore \quad b = 8$$

Example 2.3.14 *Determine the value of b for the following :*

i) $(292)_{10} = (1204)_b$ ii) $(16)_{10} = (100)_b$

Solution : **i)** $(292)_{10} = 1 \times b^3 + 2 \times b^2 + 0 \times b^1 + 4 \times b^0$

$$292 = b^3 + 2b^2 + 4$$

$$\therefore \quad b = 6$$

ii) $(16)_{10} = 1 \times b^2 + 0 \times b^1 + 0 \times b^0 = b^2$

$$\therefore \quad b = 4$$

Example 2.3.15 *Convert $(101101.1101)_2$ to decimal and hexadecimal form.*

Solution :

$$D = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$

$$= 32 + 8 + 4 + 1 + 0.5 + 0.25 + 0.0625 = \mathbf{45.8125}$$

0	0	1	0	1	1	0	1	.	1	1	0	1	Binary number
2			D					.	D				Hex number

$\therefore$ $(101101.1101)_2 = (45.8125)_{10} = (2D.D)_{16}$

Examples for Practice

Example 2.3.16 *Find the octal equivalent of the hexadecimal number DC.BA.*

[Ans. : $(334.564)_8$**]**

Example 2.3.17 *Convert* $(126)_{10}$ *to octal number and binary number.*

[Ans. : $(176)_8$, $(1111110)_2$**]**

Example 2.3.18 *Convert the* $(153.513)_{10}$ *to octal.*

[Ans. : $(231.4065)_8$**]**

Example 2.3.19 *Perform the following code conversions :*
$(1010.10)_{16} \rightarrow (?)_2 \rightarrow (?)_8 \rightarrow (?)_{10}$

[Ans. : $(1000000010000.0001)_2$, $(10020.04)_8$, $(4112.0625)_{10}$**]**

Example 2.3.20 *Convert* $(1947)_{10}$ *into its equivalent octal and hexadecimal representations*

[Ans. : $(1947)_{10} = (79B)_{16} = (3633)_8$**]**

2.4 Complements

2.4.1 1's Complement Representation

- The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

Example 2.4.1 *Find 1's complement of* $(11010100)_2$.

Solution :

1	1	0	1	0	1	0	0	Number
▽	▽	▽	▽	▽	▽	▽	▽	NOT operation
0	0	1	0	1	0	1	1	1's complement of number

2.4.2 2's Complement Representation

- The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as,

2's complement = 1's complement + 1

Example 2.4.2 *Find 2's complement of* $(11000100)_2$.

Solution :

1	1	0	0	0	1	0	0	Number
					1	1		Carry
0	0	1	1	1	0	1	1	1's complement of number
+							1	Add 1
0	0	1	1	1	1	0	0	2's complement of number

Example 2.4.3 *Represent the decimal number – 200 and 200 using 2's complement binary form.*

Solution : To represent 200 and – 200 we need 9-bit number system.

+ 200	=	0	1	1	0	0	1	0	0	0	
– 200	=	1	0	0	1	1	0	1	1	1	1's complement
	+									1	Add 1
		1	0	0	1	1	1	0	0	0	2's complement

Fig. 2.4.1

2.5 Binary Fixed Point Representation

- In real life, we deal with real numbers - Numbers with fractional part. Most modern computer have native (hardware) support for **floating point** numbers. However, the use of floating point is not necessarily the only way to represent fractional numbers.
- The use of fixed point data type is used widely in Digital Signal Processing (DSP) and game applications, where performance is sometimes more important than precision.
- The key to represent fractional numbers in fixed point representation, is the concept of **binary point.** A binary point is like the decimal point in a decimal system. It acts as a divider between the integer and the fractional part of a number.
- In a decimal system, a decimal point denotes the position in a numeral that the coefficient should multiply by $10^0 = 1$. For example, in the numeral 36.5, the coefficient 6 has a weight of $10^0 = 1$. But what happen to the 5 to the right of decimal point ? We know that it carries a weight of 10^{-1}. We know the numeral "36.5" represents the value "thirty six and a half" because,

  $3*10^1 + 6*10^0 + 5*10^{-1} = 36.5$

- The very same concept of decimal point can be applied to our binary representation, making a **"binary point".** As in the decimal system, a binary point represents the coefficient of the term $2^0 = 1$. All digits (or bits) to the left of the

binary point carries a weight of $2^0, 2^1, 2^2$ and so on. Digits (or bits) on the right of binary point carries a weight of $2^{-1}, 2^{-2}, 2^{-3}$ and so on.

- For example, the number : 100100.1_2 represents the value :

2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	0	0	1	0	0	1	0	...

$$= 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1}$$

$$= 32 + 4 + 0.5$$

$$= \mathbf{36.5}$$

- Consider the binary representation of number 73. It is 1001001. We can realize that the bit pattern of 73 and 36.5 is exactly the same. The **only difference**, is the position of binary point. In the case of 73_{10}, there is no binary point. Alternatively, we can say the binary point is located at the far right, at position 0. (Think in decimal 73 and 73.0 represents the same number.)
- In the case of 36.5_{10}, binary point is located one position to the left of 73_{10} :

2^6	2^5	2^4	2^3	2^2	2^1	2^0	DP	2^{-1}	
1	0	0	1	0	0	1	0	0	= 73

2^5	2^4	2^3	2^2	2^1	2^0	DP	2^{-1}	
1	0	0	1	0	0	.	1	= 73/2 = 36.5

- Thus, we can say that shifting binary number to the right by 1 bit position is equivalent to dividing the number by 2.
- In general, mathematically, given a fixed binary point position, shifting the bit pattern of a number to the right by 1 bit divide the number by 2. Similarly, shifting a number to the left by 1 bit multiplies the number by 2.
- To define a fixed point type conceptually, all we need are two parameters :
  - Width of the number representation and
  - Binary point position within the number.
- We will use the notation fixed<w,b>, where w denotes the number of bits used as a whole (the width of a number) and b denotes the position of binary point counting from the least significant bit (counting from 0).

- For example, fixed<8, 3> denotes a 8-bit fixed point number, of which 3 right most bits are fractional.
- Therefore, the bit pattern :

0	0	0	1	0	1	1	0

represents a real number : 00010.110_2

$$= 1*2^1 + 1*2^{-1} + 1*2^{-1}$$

$$= 2 + 0.5 + 0.25$$

$$= \mathbf{2.75}$$

- The fixed point numbers are indeed a close relative to integer representation. The two only differs in the position of binary point.
- In fact, you might even consider integer representation as a "special case" of fixed point numbers, where the binary point is at position 0. All the arithmetic operations a computer can operate on integer can therefore be applied to fixed point number as well.
- The fixed point arithmetic is much faster than floating point arithmetic.
- The disadvantage of fixed point number, is that of course the loss of range and precision when compare with floating point number representations.

Review Question

1. *Write a note on binary fixed point representation.*

UNIT II

3 Communication Interface

Syllabus

Basic Peripherals & their interfacing with 8086/8088, Semiconductor Memory Interfacing-Dynamic RAM Interfacing-Interfacing I/O ports-PIO-8255, Modes of operation-interfacing Analog-Digital Data converter-stepper motor interfacing.

Contents

3.1 Basic Peripherals and their Interfacing with 8086/8088

- The most of the microprocessors support isolated I/O system. It partitions memory from I/O, via software, by having instructions that specifically access (address) memory and others that specifically access I/O.
- When these instructions are decoded by the microprocessor, an appropriate control signal is generated to activate either memory or I/O operation.
- In 8086, M/$\overline{IO}$ signal is used for this purpose. The 8086 outputs a logic '0' on the M/$\overline{IO}$ line for an I/O operation and a logic '1' for memory operation.
- In 8086, it is possible to connect 64 K I/O ports in the system since 8086 can send 16-bit address for I/O. The 8086 gives 8-bit or 16-bit I/O addresses depending on the addressing mode used. This means it can select one of the 256 or 65536 I/O ports.
- Input/output devices can be interfaced with microprocessor systems in two ways:
  1. I/O mapped I/O 2. Memory mapped I/O.

3.1.1 I/O Mapped I/O

- The 8086 has four special instructions IN, INS, OUT, and OUTS to transfer data through the input/output ports in I/O mapped I/O system. The IN instruction copies data from a port to the accumulator.
- If an 8-bit port is read, the data will go to AL and if an 16-bit port is read the data will goto AX.
- On the other hand, the OUT instruction copies a byte from AL or a word from AX to the specified port.
- The M/$\overline{IO}$ signal is always low when 8086 is executing these instructions. So M/$\overline{IO}$ signal is used to generate separate addresses for input/output.
- Only 256 (2^8) I/O addresses can be generated when direct addressing method is used. By using indirect address method this range can be extended up to 65536 (2^{16}) addresses.

3.1.2 Memory Mapped I/O

- In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device; an I/O device is connected as if it is a memory register.
- The 8086 uses same control signals and instructions to access I/O as those of memory. Here $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is high, indicating memory bus cycle.

3.1.3 I/O Device Selection

- To select an appropriate I/O device it is necessary to do following things :
  1. Decode the address bus to generate unique signal corresponding to the device address on the address bus.
  2. When device address signal and control signal ($\overline{IORC}$ or $\overline{IOWC}$) both are low, generate device select signal.
  3. Use device select signal to activate the interfacing device (I/O port).
- Fig. 3.1.1 shows the absolute decoding circuit for the I/O device. The IC 74LS138, 3:8 decoder along with OR gate is used to generate device address signal. This signal is ORed with control signal ($\overline{IORC}$ or $\overline{IOWC}$) to generate device select signal.

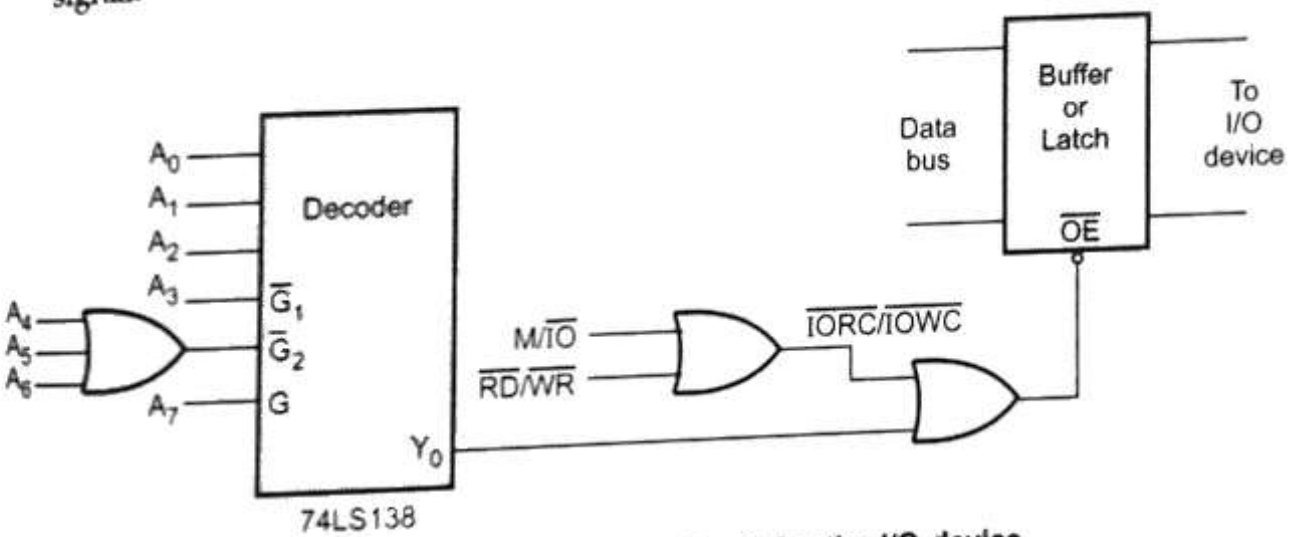


Fig 3.1.1 Absolute decoding circuit for the I/O device

- To generate device select signal (Y_0) low, the address on the address bus must be as given below :

$A_2\ A_1\ A_0 = 000$; Activates Y_0 output

$A_3\ A_4\ A_5\ A_6 = 0000$; Makes $\overline{G}_1$ and $\overline{G}_2$ low to enable output of decoder

$A_7 = 1$; Makes G high to enable output of decoder

Note **Decoder output is enabled only when control signals G_1 and G_2 are low and control signal G is high. Therefore, the address of this I/O device is 80H as shown in the Table 3.1.1.**

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
1	0	0	0	0	0	0	0	80H

Table 3.1.1

3.1.4 Interfacing 8-bit Input Port

- The microprocessor 8086 accepts 8-bit or 16-bit data from the input port. Let us see interfacing of 8-bit input port. Fig. 3.1.2 shows the circuit diagram to interface input port (buffer) which is used to read the status of 8 switches. The address for this input device is 80H as device select signal goes low when address is 80H.

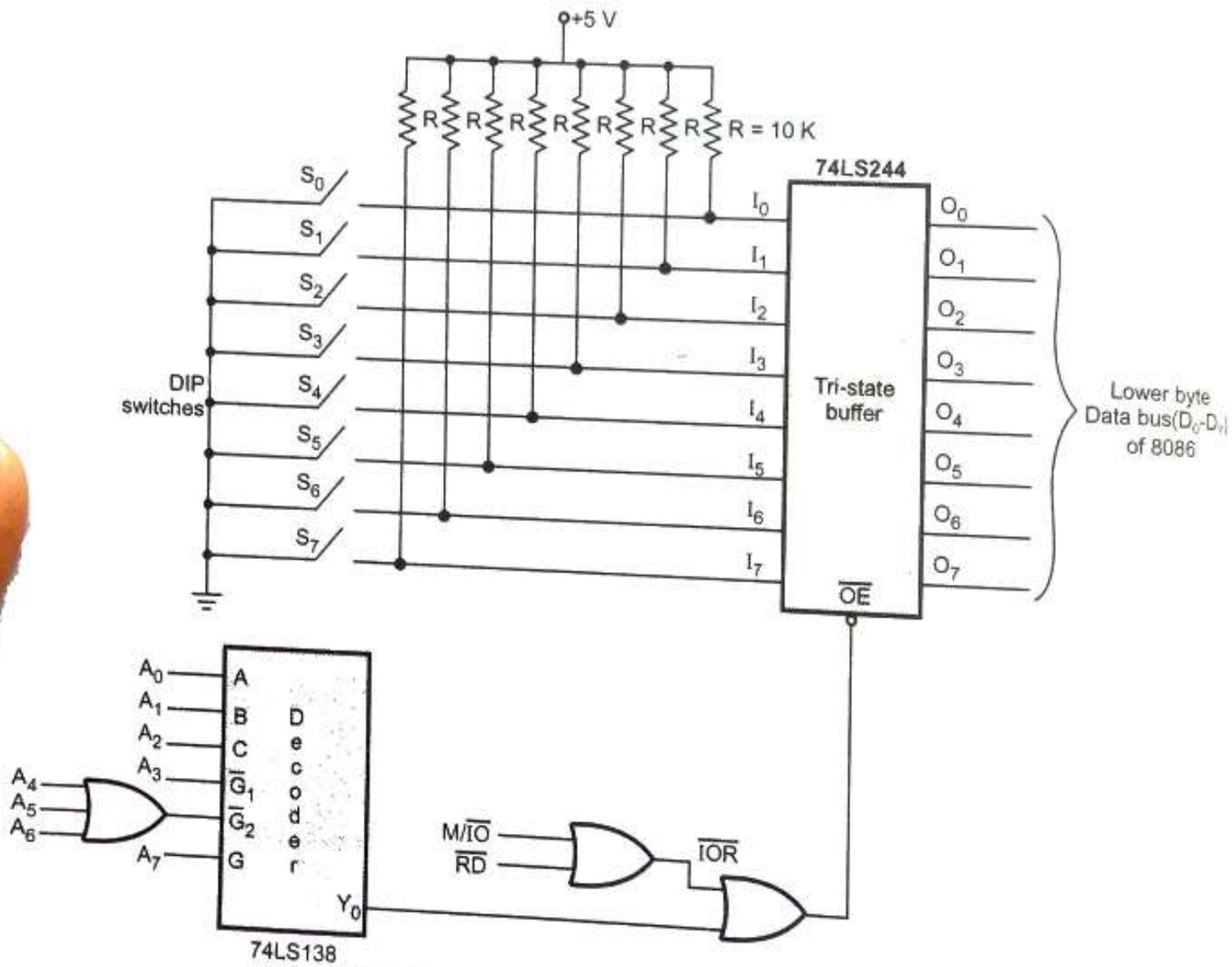


Fig. 3.1.2 Circuit diagram to interface 8-bit input port

- When the switch is in the released position, the status of line is high otherwise status is low. With this information microprocessor can check a particular key is pressed or not.
- The following program checks whether the switch 2 is pressed or not.

Program :

```
IN      AL,80H      ; Read status of all switches
AND     AL,02H      ; Mask bit positions for other switches
JZ      NEXT        ; If Z = 0, i.e. switch 2 is pressed then
                    ; program control is transferred to label NEXT.
```

Example 3.1.1 *Explain 74138 decoder configurations to enable ports at address E8H to EFH.*

Solution : Fig. 3.1.3 shows the configuration of 74138 decoder to enable ports at address E8H to EFH. The A_2, A_1, A_0 lines are decoded to generate eight port enable signals. To generate port address range from E8H to EFH 74138 decoder is enabled only when A_7, A_6, A_5 and A_3 lines are high and A_4 line is low. This is illustrated in Table. 3.1.2.

Address map

A7	A6	A5	A4	A3	A2	A1	A0	Address
1	1	1	0	1	0	0	0	E8H
1	1	1	0	1	0	0	1	E9H
1	1	1	0	1	0	1	0	EAH
1	1	1	0	1	0	1	1	ABH
1	1	1	0	1	1	0	0	ECH
1	1	1	0	1	1	0	1	EDH
1	1	1	0	1	1	1	0	EEH
1	1	1	0	1	1	1	1	EFH

Table 3.1.2

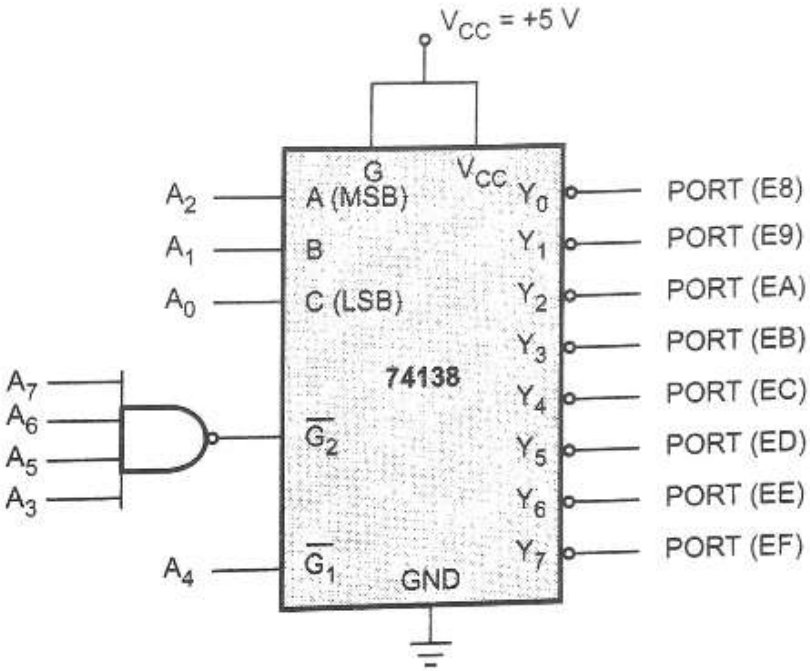


Fig. 3.1.3

3.1.5 Interfacing 16-bit Input Port

- Let us see interfacing of 16-bit input port. Fig. 3.1.4 shows the circuit diagram to interface 16-bit input port (buffer) which is used to read the status of 16 switches. The address for this input port is 80H as device select signal goes low when address is 80H. (See Fig. 3.1.4 on next page)
- The following program checks whether the switch 15 is pressed or not.

Program :

```
IN    AX,80H      ; Read status of all switches
AND AX,8000H      ; Mask bit positions for other switches
JZ    NEXT        ; if Z = 0, i.e. switch 15 is pressed then
                  ; program control is transferred to label NEXT.
```

3.1.6 Interfacing 8-bit Output Device

- The microprocessor 8086 sends 8-bit or 16-bit data to the output port according to instruction used for data transfer.

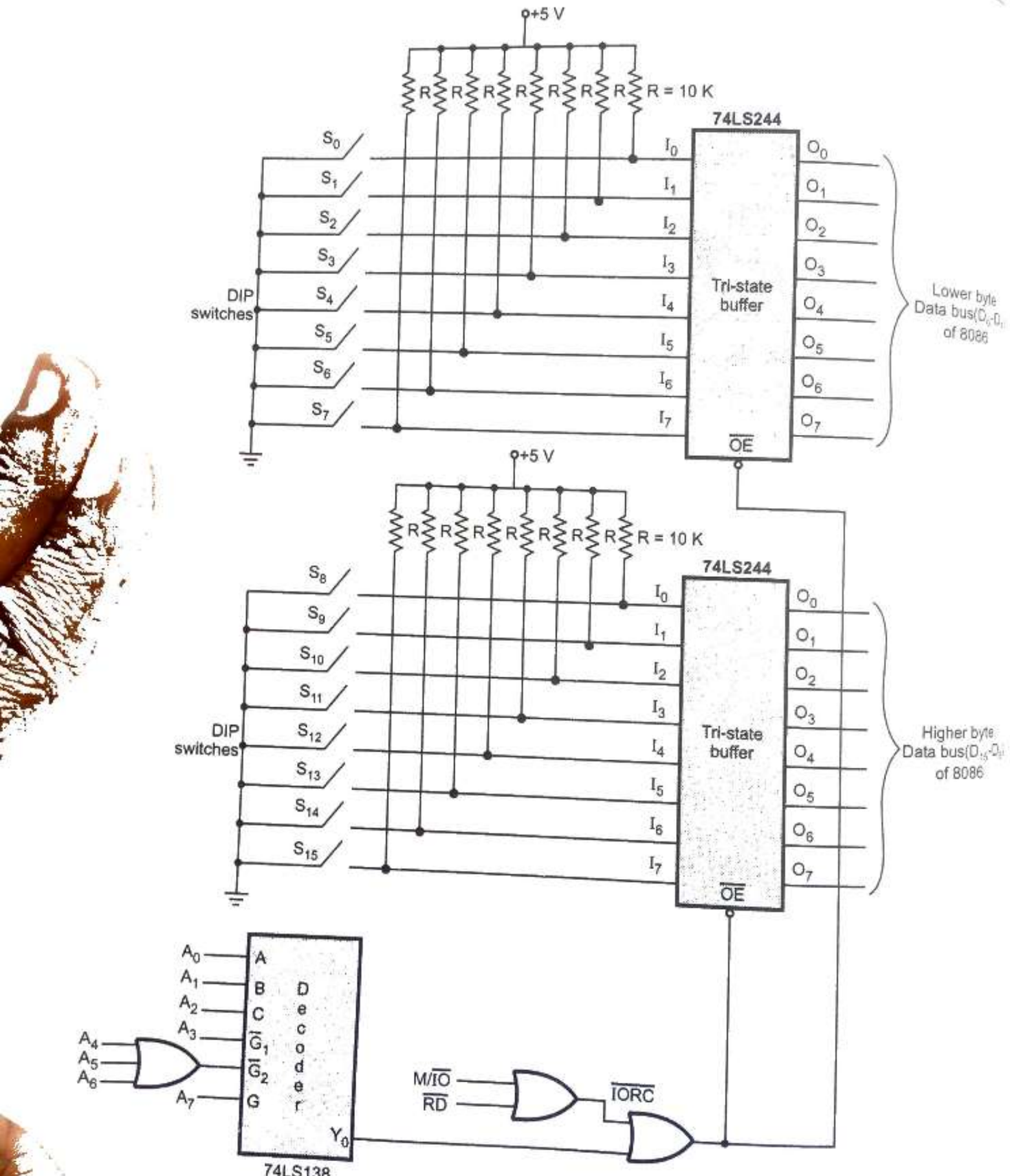


Fig. 3.1.4 Circuit diagram to interface 16-bit input port

- Fig. 3.1.5 shows the circuit diagram to interface 8-bit output port (latch) which is used to send the signal for glowing the LEDs. LED will glow when output pin status is low.
- The IC 74LS138 and OR gate are used to generate device select signal. The latch enable signal is active high. So NOR gate is used to generate latch enable signal, which goes high when Y_2 and $\overline{IOWC}$ both are low.

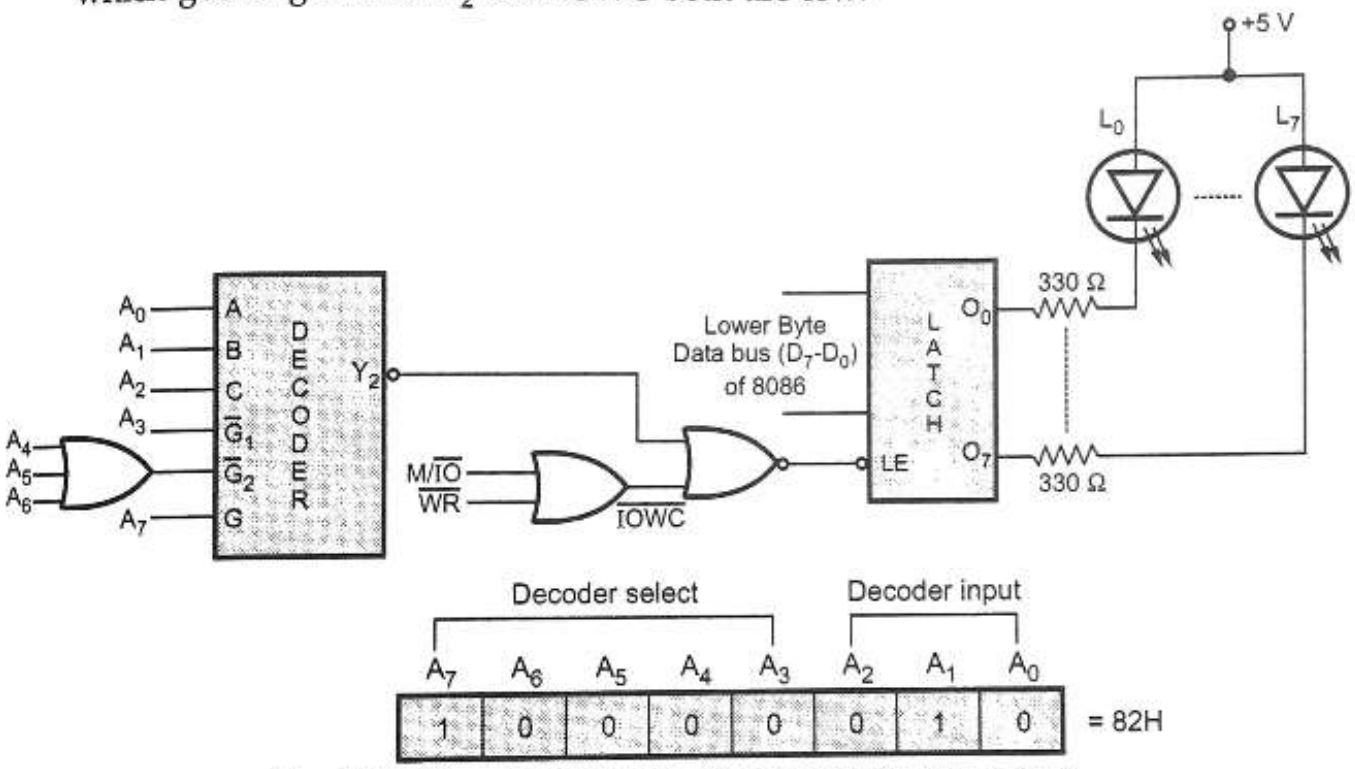


Fig. 3.1.5 Circuit diagram to interface 8-bit output port

- The following program glows the LEDs L_0, L_2 and L_5.

L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0	
1	1	0	1	1	0	1	0	= DAH

- The code (data) DAH must be sent on the latch to glow LEDs L_0, L_2 and L_5.

Program :

```
MOV AL,0DAH     ; Loads the data in the accumulator.
OUT 82H,AL      ; Sends the data on the latch.
```

3.1.7 Interfacing 16-bit Output Device

- Fig 3.1.6 shows the circuit diagram to interface 16-bit output port (latch) which is used to send the signal for glowing 16 LEDs. LED will glow when output pin status is low.

- The IC 74LS138 and OR gate are used to generate device select signal. The latch enable signal is active high. So NOR gate is used to generate latch enable signal which goes high when Y_2 and $\overline{IOWC}$ both are low.
- The following program glows the LEDs L_{14}, L_{11}, L_8, L_5, L_2 and L_0.

L_{15}	L_{14}	L_{13}	L_{12}	L_{11}	L_{10}	L_9	L_8	L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0	
1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	0	= B6DAH

- The code (data) B6DAH must be sent on the latch to glow LEDs L_{14}, L_{11}, L_8, L_5, L_2 and L_0.

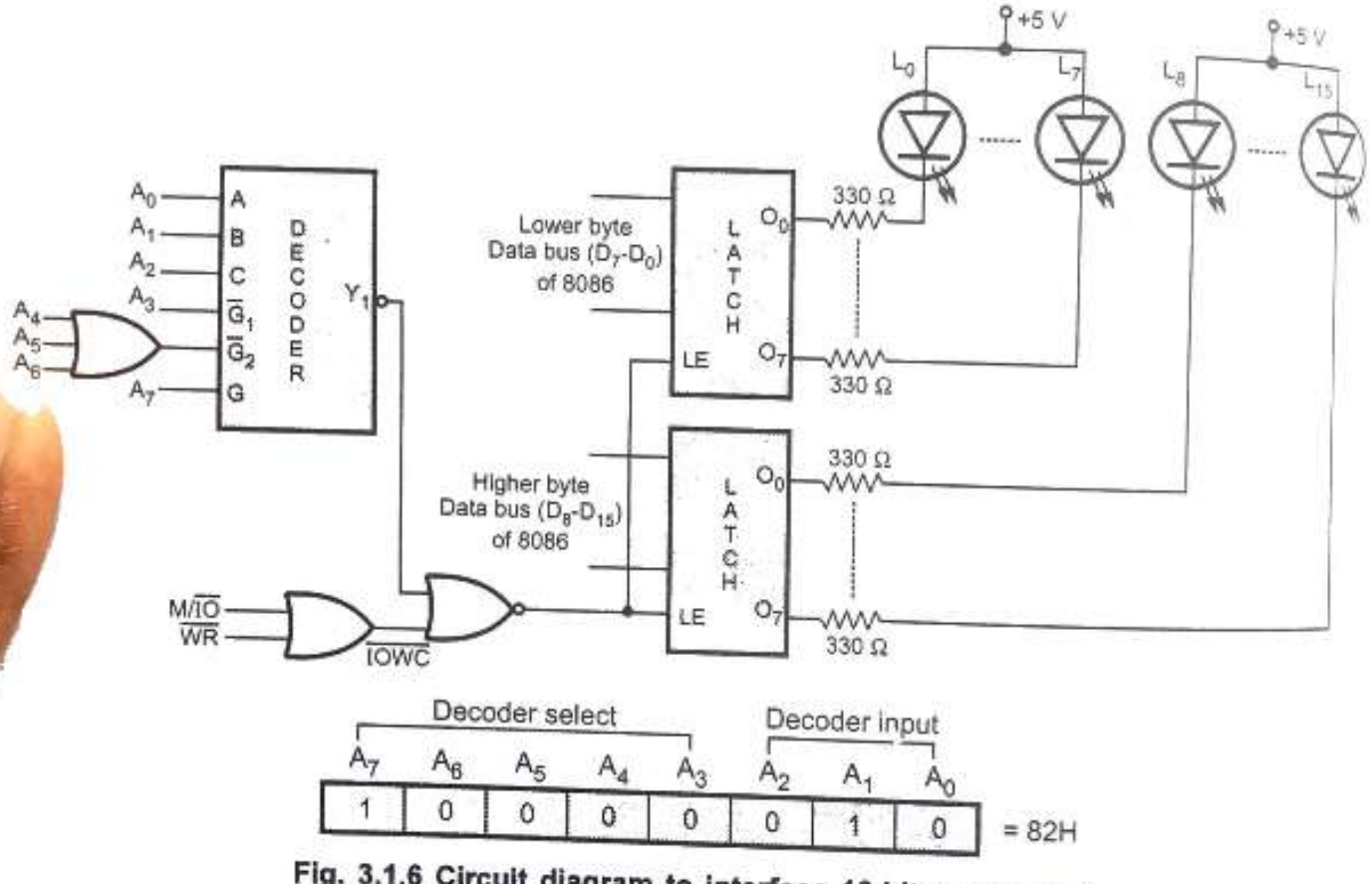


Fig. 3.1.6 Circuit diagram to interface 16-bit output port

Program :

```
MOV AX,0B6DAH    ; Loads the data in the accumulator.
OUT 82H,AX       ; Sends the data on the latch.
```

Note In the previous examples, we have 8-bit port addresses so only lower eight address lines are used for address decoding. For 16-bit port addresses we have to decode lower sixteen address lines to generate device select signal. The remaining part of the circuit is same.

- Fig. 3.1.7 shows the combined circuit for I/O interfacing. For this circuit the address of input port is 80H and address of output port is 82H. The following program displays the status of switches on the LEDs.

Program :

```
IN    80H     ; Read status of all switches.
OUT   82H     ; Send status on the output port.
```

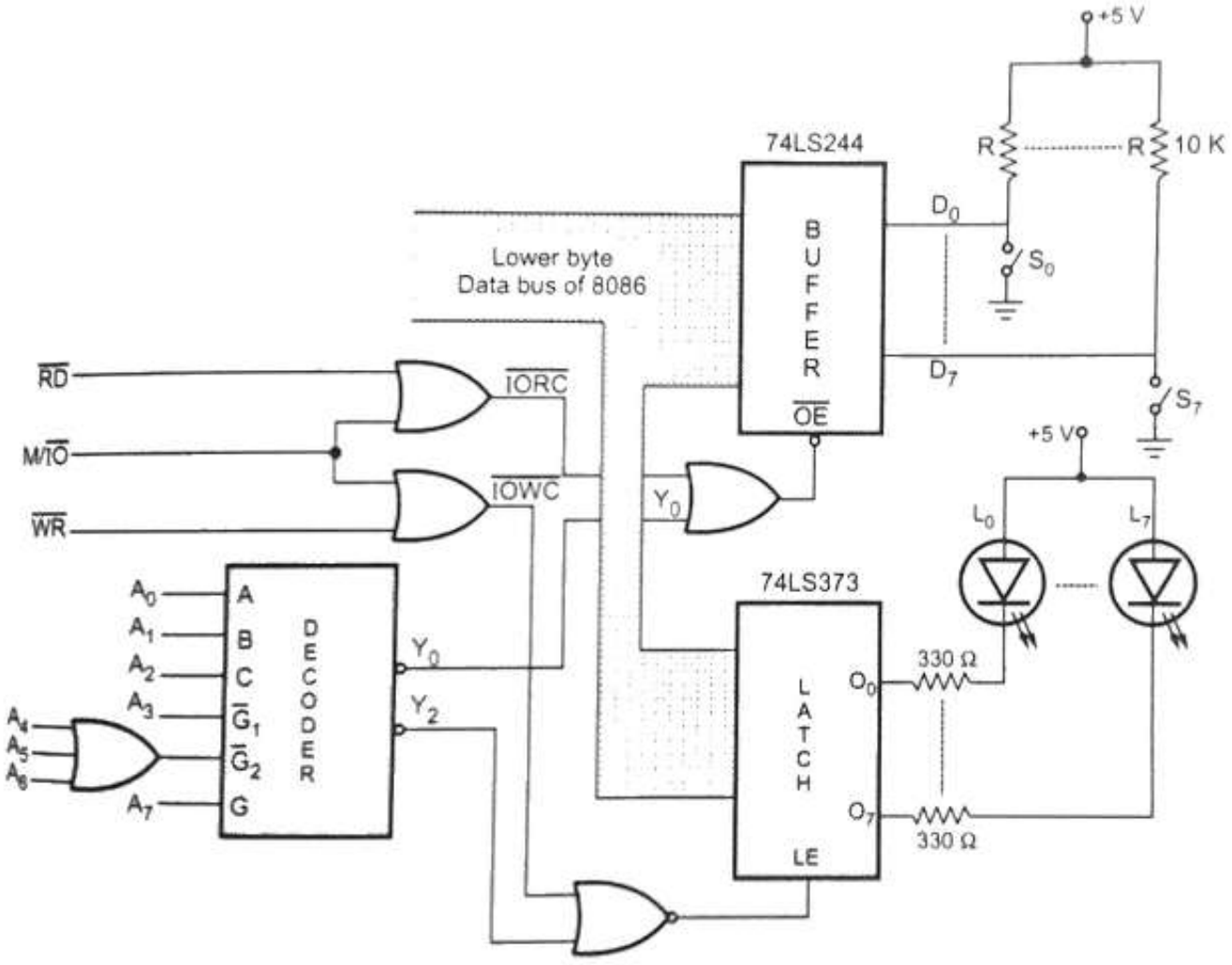


Fig. 3.1.7 I/O interfacing using I/O mapped I/O

Example 3.1.2 *Refer Fig. 3.1.7 and write a program that will check the switch1 status and do accordingly.*

i) S_1 = 0 : Blink Lower nibble LEDs. ii) S_1 = 1 : Blink Higher nibble LEDs.

Assume delay routine is available.

Solution : Input port address = 80H

Output port address = 82H

- Fig. 3.1.7 shows the combined circuit for I/O interfacing. For this circuit the address of input port is 80H and address of output port is 82H. The following program displays the status of switches on the LEDs.

Program :

```
IN      80H     ; Read status of all switches.
OUT     82H     ; Send status on the output port.
```

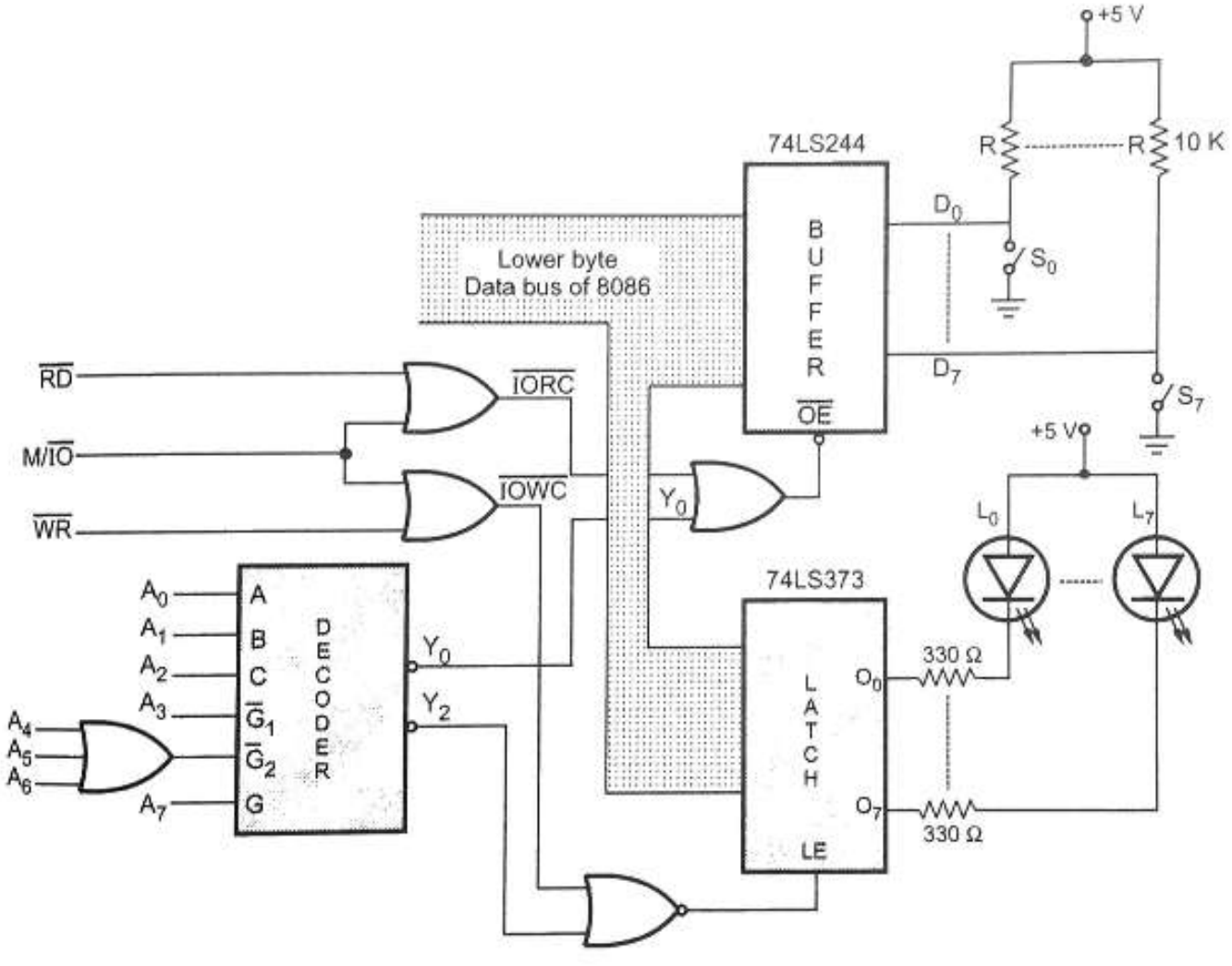


Fig. 3.1.7 I/O interfacing using I/O mapped I/O

Example 3.1.2 *Refer Fig. 3.1.7 and write a program that will check the switch1 status and do accordingly.*

i) $S_1 = 0$: Blink Lower nibble LEDs. ii) $S_1 = 1$: Blink Higher nibble LEDs.

Assume delay routine is available.

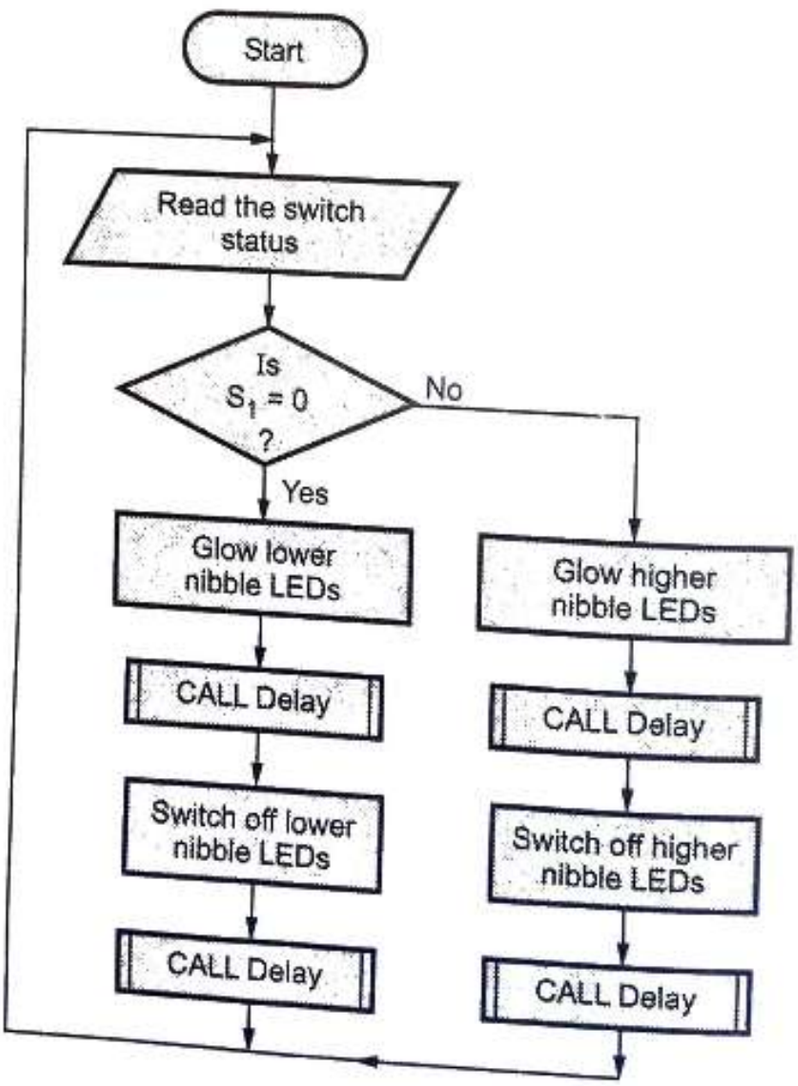
Solution : Input port address = 80H

Output port address = 82H

Source Program :

```
START :  IN AL,80H        ; Read status of switches
         AND AL,01H       ; Masks Bit 1 to Bit 7
         JNZ HIGHER       ; If sw1 is not pressed goto blink higher nibble
         MOV AL,0F0H      ; Load bit pattern to glow lower nibble LEDs
         OUT 82H,AL       ; Send it to output port
         CALL Delay       ; Call delay subroutine
         MOV AL,0FFH      ; Load bit pattern to switch off all LEDs
         OUT 82H,AL       ; Send it to output port
         CALL Delay       ; Call delay subroutine
         JMP  START       ; JUMP to START
HIGHER : MOV AL,0FH       ; Load bit pattern to glow higher nibble LEDs
         OUT 82H,AL       ; Send it to output port
         CALL  Delay      ; Call delay subroutine
         MOV AL,0FFH      ; Load bit pattern to switch off all LEDs
         OUT 82H,AL       ; Send it to output port
         CALL  Delay      ; Call delay subroutine
         JMP START        ; JUMP to START
```

Flowchart :



Example 3.1.3 *Write an 8086 ALP to read a byte of data from port A and port B. Add the data and save the result in a memory location.*

Solution :

```
.model small
.stack 100
.data
LOC        DB     0
.code
    START :        MOV AX, @ data      ; [Initialize data segment]
                   MOV DS, AX
                   IN AL, PORTA        ; Get byte from port A
                   MOV BL, AL          ; Save the data byte
                   IN AL, PORTB        ; Get byte from port B
                   ADD AL, BL          ; Add two bytes
                   MOV LOC, AL         ; Save the result
                   END START
                   END
```

Example 3.1.4 *Write an ALP using 8086 instructions to read a byte of data from port A and display its parity status as 00H or FFH for odd and even parity respectively, on port B.*

Solution :

```
          IN AL, PORTA        ; Read a byte from port A
          MOV CX, 08          ; Initialize counter with value 8
          MOV DL, 00H         ; Initialize counter to count number of 1s in the byte
BACK :    SHR AL, 1           ; Shift byte right so that LSB in the carry
          JNC SKIP            ; if CY = 0 skip next instruction
          INC DL              ; Otherwise increment count
SKIP :    LOOP BACK           ; Repeat unit CX ≠ 0
          SHR DL, 1           ; Check whether count is even or odd
          JNC SKIP1           ; if CY = 0 (count even) goto skip 1
          MOV AL, 00H         ; Otherwise send 00 on port B
          OUT PORTB, AL
          JMP LAST
SKIP1 :   MOV AL, FFH         ; if CY ≠ 0 (count odd) send
          OUT PORTB, AL       ; FFH on port B
LAST :    HLT                 ; Stop
```

3.1.8 I/O Interfacing with 16-bit Port Address

- As mentioned earlier, for 16-bit port address we have to decode lower sixteen address lines to generate device select signal.
- Fig. 3.1.8 shows the interfacing of 8-bit input port (buffer) and 8-bit output port (latch) having 16-bit address. The address of input port is 0080H and address of output port is 0082H.

Example :

- Refer Fig. 3.1.8 and write a program to display the status of switches on the LEDs

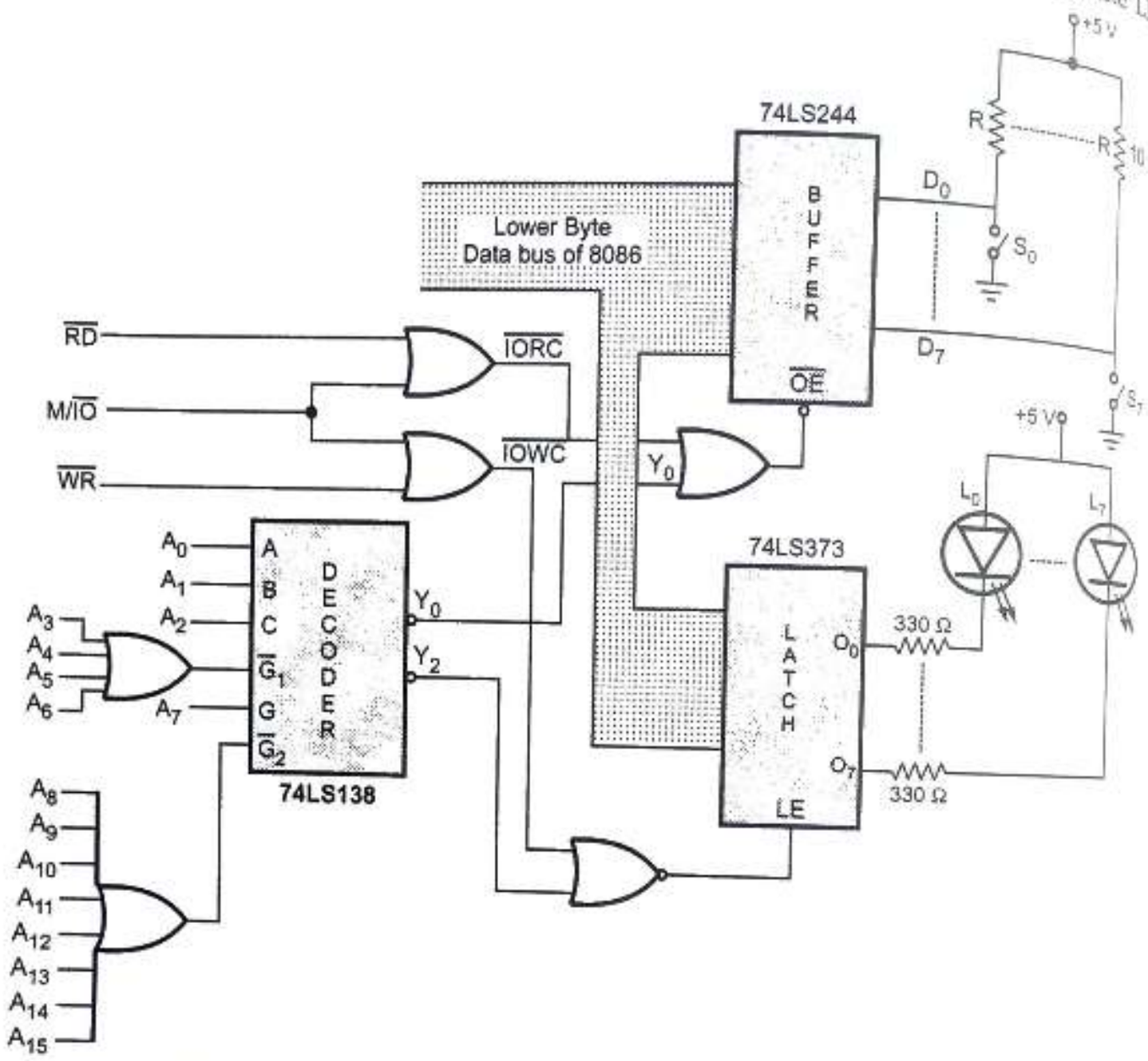


Fig. 3.1.8 Interfacing of 8-bit input port (buffer) and 8-bit output port (latch)

Program :

```
MOV DX,0080H    ; Load 16-bit address of the input port in DX
IN AL,DX        ; Read byte from input port in AL
MOV DX,0082H    ; Load 16-bit address of the output port in DX
OUT DX,AL       ; Send byte to output port from AL
```

3.1.9 I/O Interfacing with Memory Mapped I/O

- As mentioned earlier, in memory mapped I/O interfacing, the 8086 uses 20 address lines to identify an I/O device ; an I/O device is connected as if it is a memory register.
- Fig. 3.1.9 shows the I/O interfacing with memory mapped I/O. As shown in the Fig. 3.1.9, the 8086 uses same control signals and instructions to access I/O as

those of memory. Here $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is high, indicating memory bus cycle.

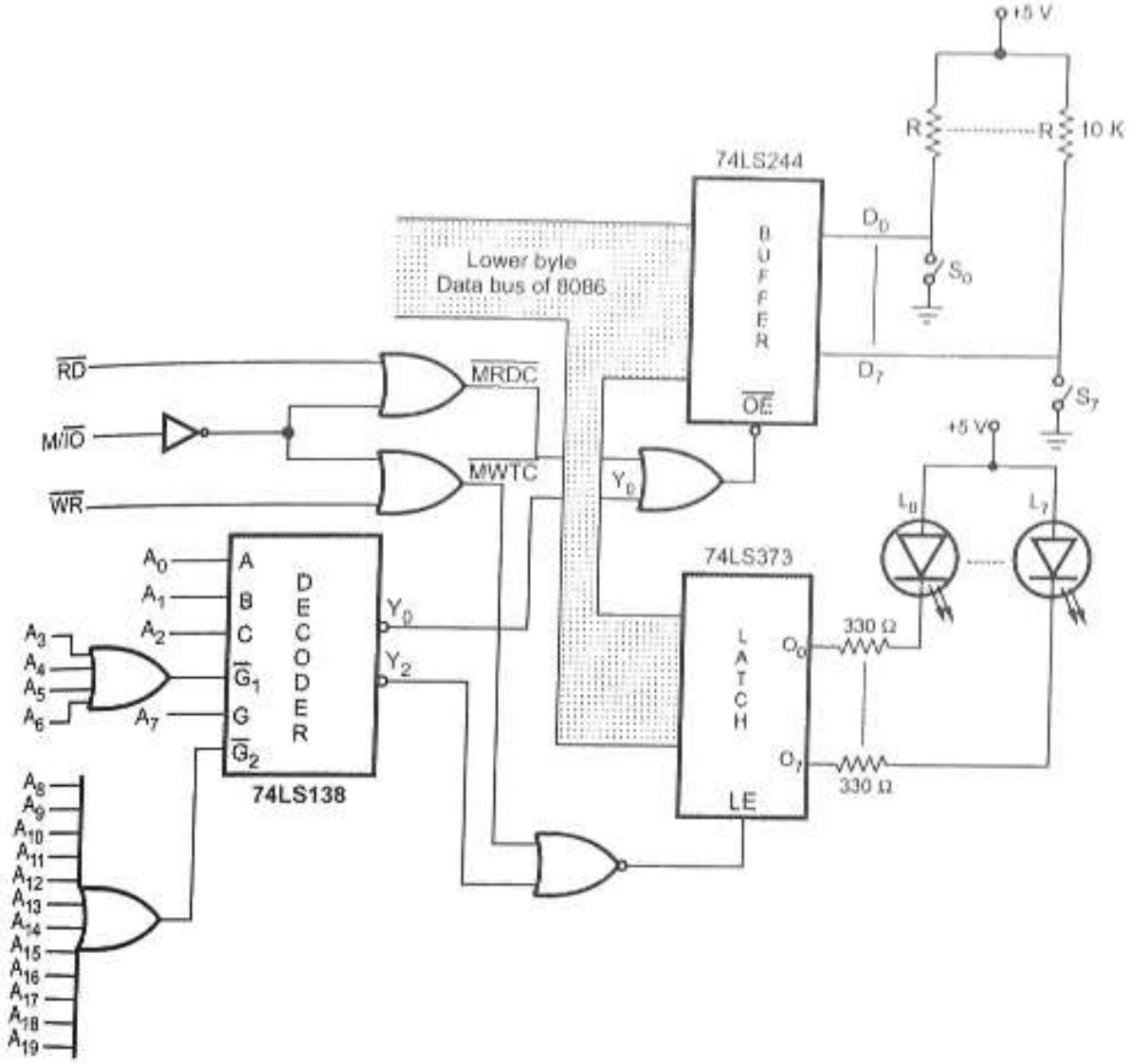


Fig. 3.1.9 I/O Interfacing with memory mapped I/O

Example :

Refer Fig. 3.1.9 and write a program to display the status of switches on the LEDs.

Program :

```
MOV AL,[0080H]      ; Read byte from input port in AL
MOV [0082],AL       ; Send byte to output port from AL
```

Review Questions

1. *Explain the I/O data transfer using I/O ports with the help of neat diagram.*
2. *Explain the I/O interfacing techniques : a) I/O mapped I/O b) Memory mapped I/O.*

3. *Draw and explain the interfacing of*
   a) An 8-bit input port *b) An 16-bit input port*
   c) An 8-bit output port *d) An 16-bit output port*
   with microprocessor 8086.

3.2 Semiconductor Memory Interfacing

- The 8086 is a 16-bit microprocessor, it can transfer 16-bit data. So in addition to byte, word (16-bit) has to be stored in the memory. This is stored by using two consecutive memory locations, one for least significant byte and other for most significant byte.
- The address of word is the address of least significant byte.
- To implement this, the entire memory is divided into two memory banks : $Bank_0$ and $Bank_1$. Fig. 3.2.1 shows the interfacing diagram to these memory banks.

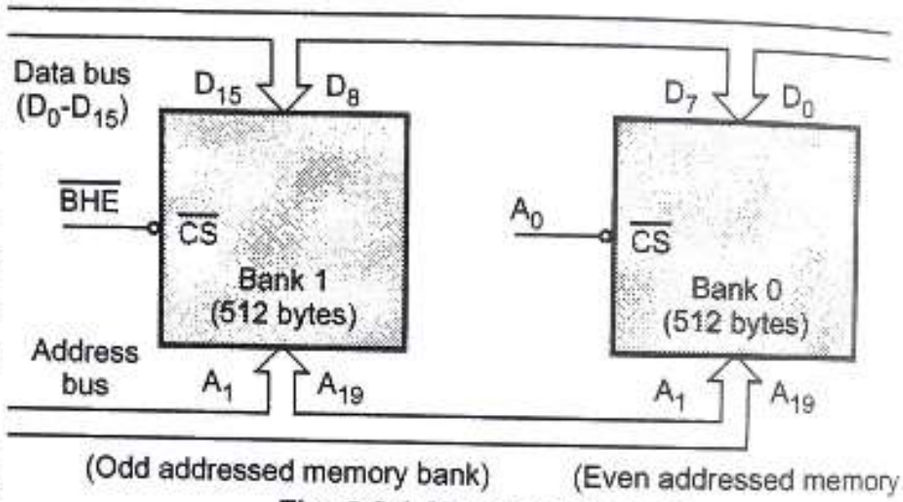


Fig. 3.2.1 Memory Interfacing

- Bank 0 is selected only when A_0 is zero and Bank 1 is selected only when $\overline{BHE}$ is zero.
- A_0 is zero for all even addresses. So Bank 0 is usually referred as **even addressed memory** bank.
- $\overline{BHE}$ is used to access higher order memory bank, referred to as **odd addressed memory** bank.
- Together $\overline{BHE}$ and A_0 tell the interface how the data appears on bus. Four possible combinations are shown in the table.

Sr.No.	Operation	$\overline{BHE}$	A_0	Data lines used
1.	Read/Write a byte at an even address	1	0	$D_7 - D_0$
2.	Read/Write a byte at an odd address	0	1	$D_{15} - D_8$
3.	Read/Write a word at an even address	0	0	$D_{15} - D_0$

4.	Read/Write a word at an odd address	0	1	D_{15}-D_8 in first operation byte from odd bank is transferred.
		1	0	D_7-D_0 in second operation byte from even bank is transferred.

Note : To access odd addressed word two bus cycles are required.

3.2.1 Memory Interfacing in Minimum Mode

- Fig. 3.2.2 shows the typical minimum mode 8086 system. Here, interfacing of memory and I/O devices are shown with the basic minimum mode 8086 configuration.
- For interfacing memory module to 8086, it is necessary to have odd and even memory banks. This is implemented by using two EPROMs and two RAMs.
- Data lines D_{15}-D_8 are connected to odd bank of EPROM and RAM, and data lines D_7-D_0 are connected to even bank of EPROM and RAM.
- Address lines are connected to EPROM and RAM as per their capacities.
- $\overline{RD}$ signal is connected to the output enable ($\overline{OE}$) signals of EPROMs and RAMs. $\overline{WR}$ signal is connected to $\overline{WR}$ signal of RAMs.
- Two separate decoders are used to generate chip select signals for memory and I/O devices. These chip select signals are logically ORed with either $\overline{BHE}$ or A_0 to generate final chip select signals.
- For generating final chip select signals for odd bank decoder outputs are logically ORed with $\overline{BHE}$ signal. On the other hand to generate final chip select signals for even bank decoder outputs are logically ORed with A_0 signal.
- The 16-bit I/O interface is shown in the Fig. 3.2.2. $\overline{RD}$ and $\overline{WR}$ signals are connected to the $\overline{RD}$ and $\overline{WR}$ signals of I/O device. Data lines D_{15}-D_0 are connected to the data lines of I/O device. The chip select signal for I/O device is generated using separate decoder whose output is enabled only when M/$\overline{IO}$ signal is low. (See Fig. 3.2.2 on next page.)

Example 3.2.1 *Design an 8086 based system with the following specifications.*

i) 8086 in minimum mode. ii) 4 K ROM iii) 128 RAM

Draw the complete schematic of the design indicating address map.

Solution : The 8086 is a 16-bit microprocessor. It can access 16-bit data simultaneously. For interfacing memory module to 8086 CPU, it is necessary to have odd and even memory banks. This can be achieved by using two 2 kbyte ROMs and two 64 byte RAMs, one for odd bank and another for even bank.

- As 64 kbyte RAM needs 16 address lines, A_1 to A_{16} lines are used and 2 K ROM needs 11 address lines, A_1 to A_{11}. A_0 and $\overline{BHE}$ are used to select ever and odd memory banks respectively. Fig. 3.2.3 shows the interface between 8086 and two memory chips. (See Fig. 3.2.3 on page number 3 - 17)

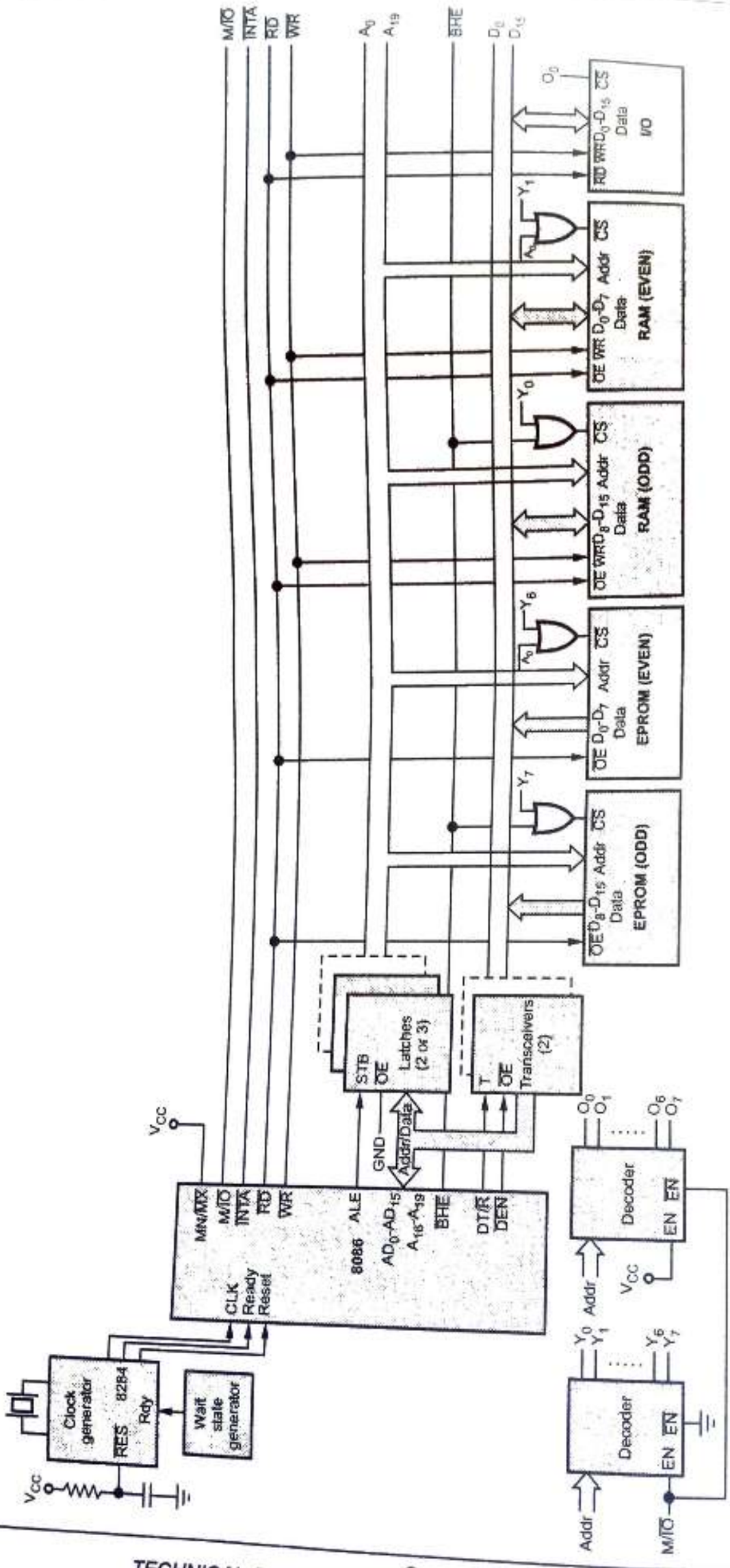


Fig. 3.2.2 Minimum mode 8086 system

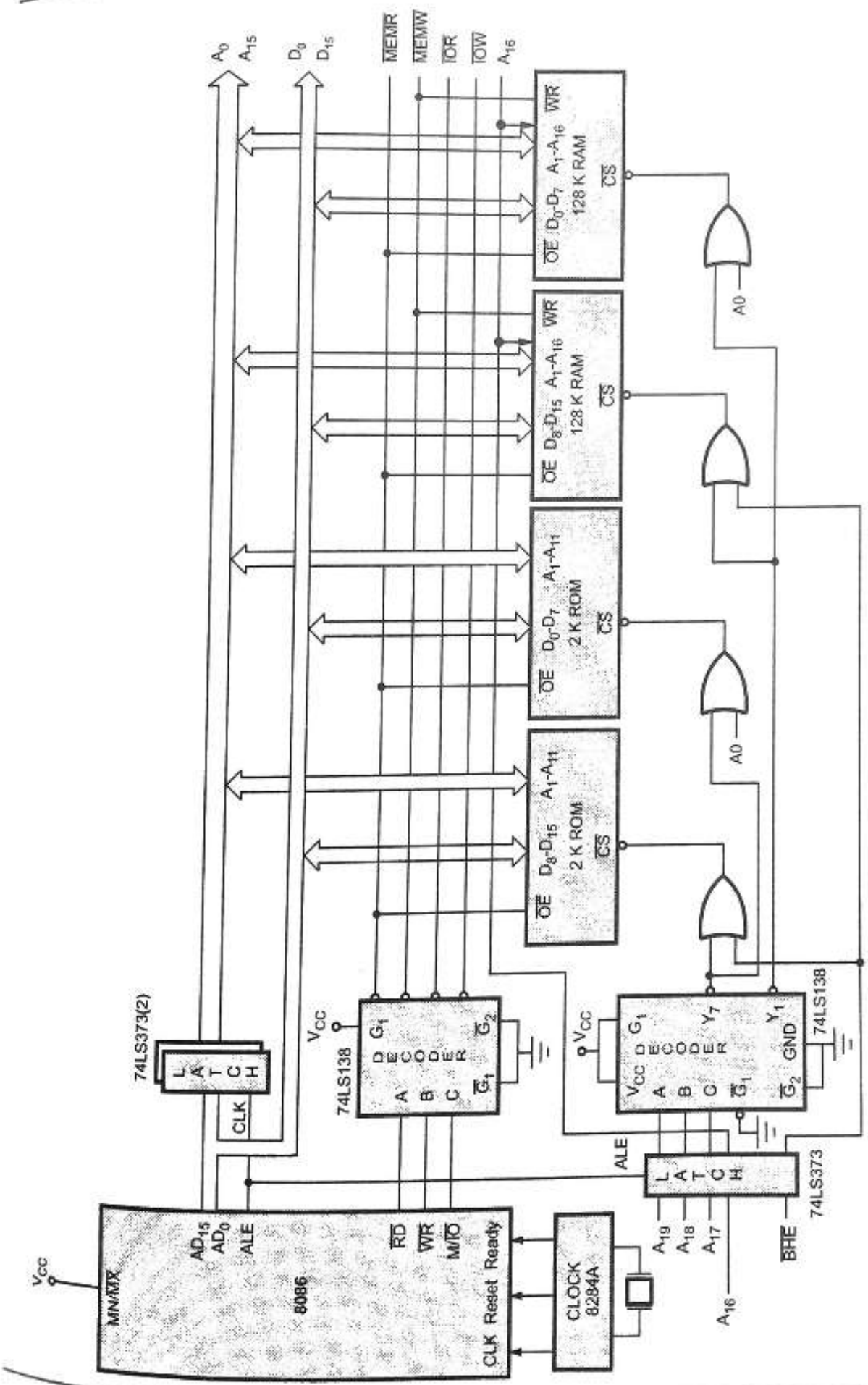


Fig. 3.2.3 Interfacing 128 K RAM and 2K ROM with 8086 in minimum mode

Memory map :

$\overline{BHE}$	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address	Memory
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	FF000H	Even
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FFFFEH	ROM1
0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	FF001H	Odd
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFFH	ROM2
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20000H	Even
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	3FFFEH	RAM1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	20001H	Odd
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFFH	RAM2

3.2.2 Memory Interfacing in Maximum Mode

- Fig. 3.2.4 shows the typical maximum mode 8086 system. Here interfacing of memory and I/O devices are shown with the basic maximum mode configuration.
- The connections for memory and I/O devices are similar to that of minimum mode configuration. However, the generation of control signals from 8086 is done by external bus controller 8288. (See Fig. 3.2.4 on next page.)

Review Questions

1. *Explain memory organization of 8086.*
2. *Explain the memory interfacing in minimum mode of 8086.*
3. *Explain the memory interfacing in maximum mode of 8086.*

3.3 Dynamic RAM Interfacing

- DRAM needs refreshing after every 2 ms or 4 ms. On the other hand, DRAM contains more memory cells as compared to static RAM per unit area and they are available in much larger sizes : upto 16 M × 1. In this section we see the interfacing of DRAM to the processor, i.e. DRAM memory system.
- Let us see the pin connection for a typical 64 K × 4 DRAM (TMS4464). We know that, to address 64 K memory registers we require 16 address lines. But in DRAM to reduce total number of address lines, address is given in two phases. This allows DRAM to multiplex lower address and higher address.
- The DRAM TMS4464 (64 K×4) has eight address lines (A_0 - A_7). When **Column Address Strobe** ($\overline{CAS}$) is low these address lines give column address and when **Row Address Strobe** ($\overline{RAS}$) is low these address lines give row address. First,

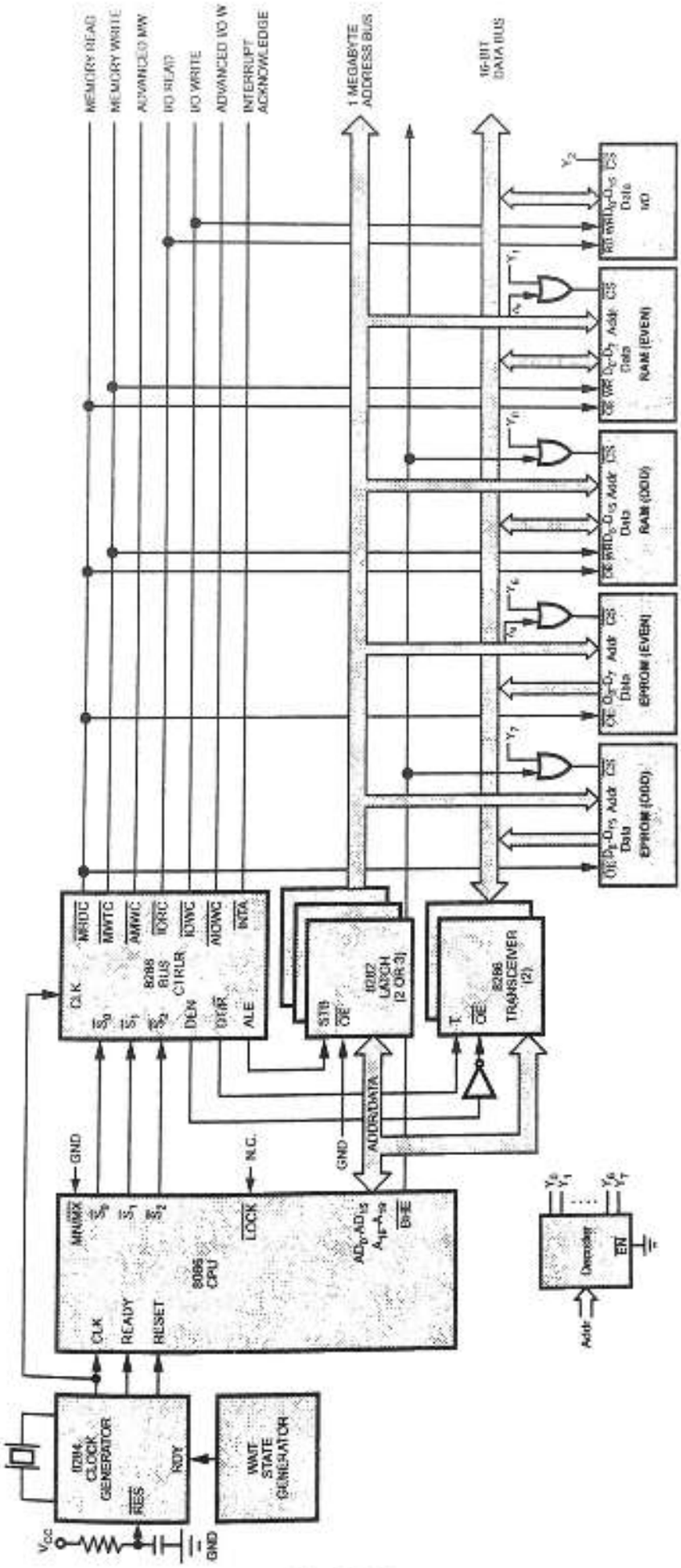
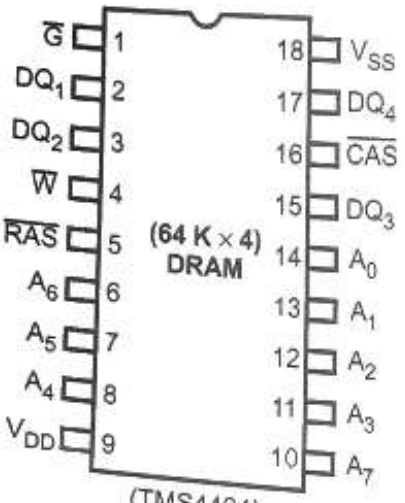

MEMORY READ
MEMORY WRITE
ADVANCED MW
I/O READ
I/O WRITE
ADVANCED I/O W
INTERRUPT ACKNOWLEDGE
1 MEGABYTE ADDRESS BUS
16-BIT DATA BUS
8284 CLOCK GENERATOR
WAIT STATE GENERATOR
8086 CPU
8288 BUS CONTROLLER
8282 LATCH (2 OR 3)
8286 TRANSCEIVER (2)
RAM (EVEN)
RAM (ODD)
EPROM (EVEN)
EPROM (ODD)
GND
N.C.
ADDR/DATA
CLK
READY
RESET
LOCK
Decoder
EN
Addr

Fig. 3.2.4

A_0 - A_7 are placed on the address bus and strobed into an internal row latch by asserting $\overline{RAS}$. Then, the address lines A_8 - A_{15} are placed on the same eight address inputs and strobed into an internal column latch by asserting $\overline{CAS}$.



(a) Pin diagram

Pin Definitions	
A_0 - A_7	Address lines
$\overline{CAS}$	Column address strobe
DQ_1 - DQ_4	Data lines
$\overline{RAS}$	Row address strobe
V_{DD}	+5V supply
$\overline{W}$	Write enable
V_{SS}	Ground

(b) Pin definitions

Fig. 3.3.1

- Therefore, 16-bit address gets latched into two internal 8-bit latches. The internally latched 16-bit access the contents of one of the 4-bit memory locations.
- Fig. 3.3.2 shows the circuit for multiplexing A_0 - A_7 and A_8 - A_{15} address lines. Here, $\overline{RAS}$ line is used to strobe row address into the DRAM. When $\overline{RAS}$ is low row address is strobed and when $\overline{RAS}$ is high column address is strobed into the DRAM.

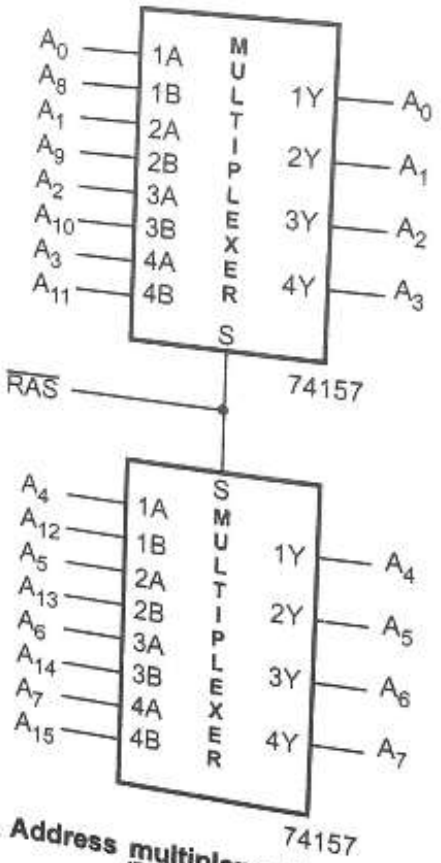


Fig. 3.3.2 Address multiplexer for 64 × 4 DRAM (TMS4464)

3.3.1 DRAM Controller

- In most systems, a DRAM controller integrated circuit is used to do address multiplexing and the generation of the DRAM control signals. In this section we see the details of 82C08 (DRAM controller).

- The 82C08 can control upto 1 Mbyte of memory for 8086. It contains an address multiplexer that multiplexes an 18-bit address onto 9 address connections.
- Fig. 3.3.3 shows pin diagram of DRAM controller. AL_0 - AL_8 and AH_0 - AH_8 are the address inputs for DRAM controller and AO_0 - AO_8 are the address outputs from the DRAM controller.
- The 82C08, DRAM controller has internal circuitry which generates the $\overline{CAS}$ and $\overline{RAS}$ signals for the DRAM. The 82C08 generates these signals internally using the CLK, $\overline{S}_1$ and $\overline{S}_0$ signals.
- The processor provides $\overline{S}_1$ and $\overline{S}_0$ signals as $\overline{RD}$ and $\overline{WR}$ inputs to the 82C08 DRAM controller.
- The $\overline{AACK}/\overline{XACK}$ signal of 80C08 is an acknowledge output that is used to indicate the ready condition for the microprocessor. This signal is usually connected to the READY input of the microprocessor or the clock generator READY input.

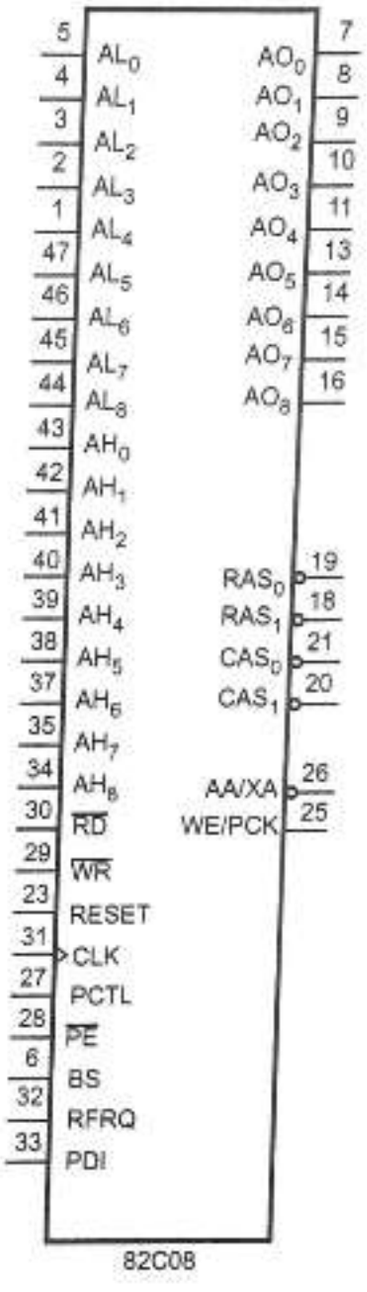


Fig. 3.3.3 Pin diagram of 82C08 DRAM controller

- Fig. 3.3.4 shows the interface of 1Mbyte DRAM memory to the 8086 with the help of DRAM controller. As DRAM controller accepts 18 address lines it can address up to 256 K memory locations. Here, 1 Mbyte DRAM memory is interfaced using a series of four 256 K × 8 bit DRAM modules. The PAL 16L8 is used to generate bank select signals and other control signals, as shown in the Fig. 3.3.4. The A_{19} signal selects the upper bank or the lower bank through the BS (**Bank Select**) input to the 82C08 DRAM controller. (See Fig. 3.3.4 on next page)

Review Questions

1. *Write a note of DRAM interfacing.*
2. *What is the role of DRAM controller ?*

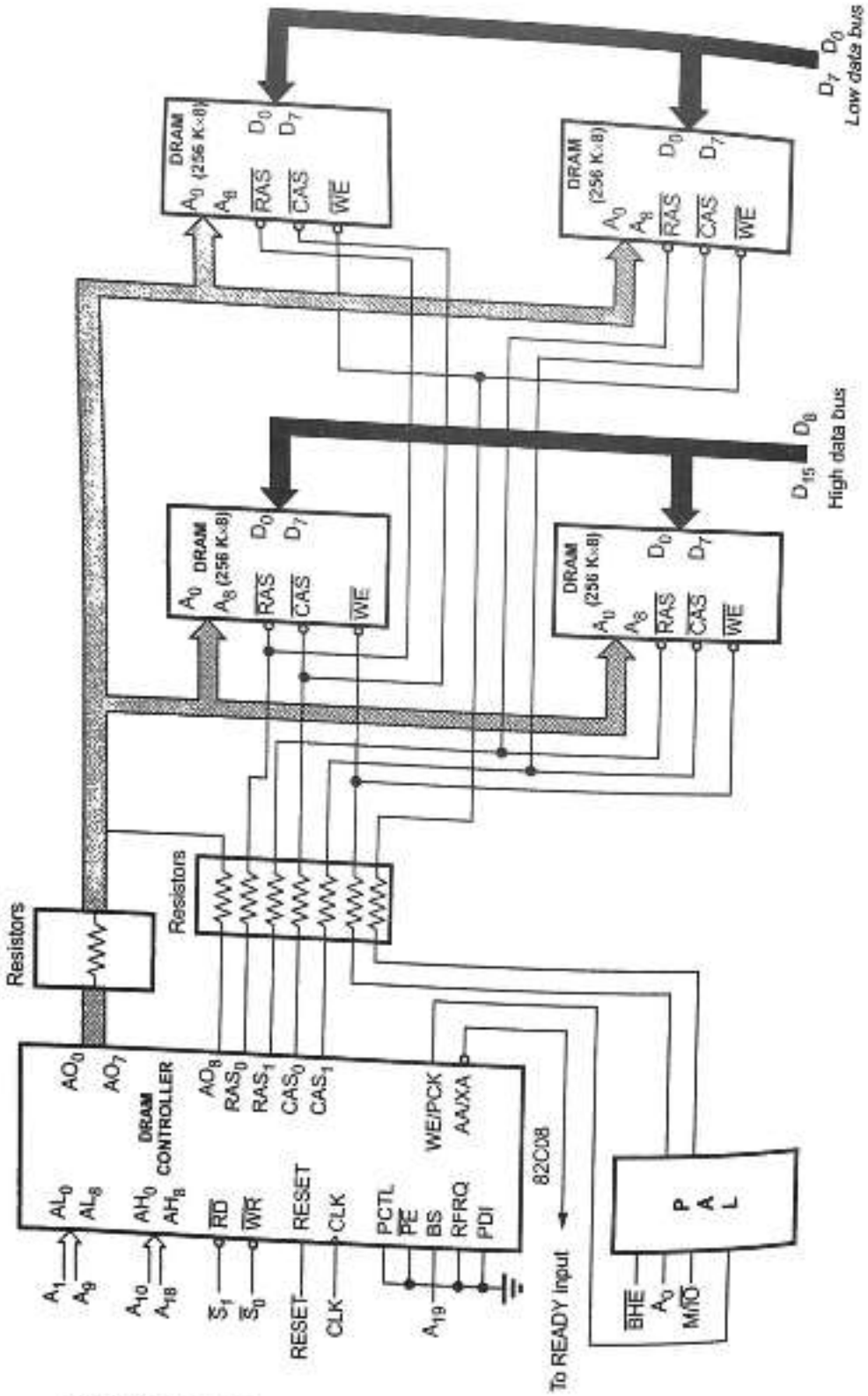


Fig. 3.3.4 The interface of 1 Mbyte DRAM memory to 8086

3.4 Programmable Peripheral Interface (PPI) - 8255

3.4.1 Features

1. The 8255A is a widely used, programmable, parallel I/O device or parallel communication interface.
2. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O.
3. It is compatible with all Intel and most other microprocessors.
4. It is completely TTL compatible.
5. It has three 8-bit ports (24 I/O pins) : Port A, Port B and Port C, which are arranged in two groups of 12 pins. Each port has an unique address and data can be read from or written to a port. In addition to the address assigned to the three ports, another address is assigned to the control register into which control words are written for programming the 8255 to operate in various modes.
6. Its bit set/reset mode allows setting and resetting of individual bits of Port C.
7. The 8255 can operate in 3 I/O modes :

   Mode 0 : Simple input/output

   Mode 1 : Input/Output with handshake

   Mode 2 : Bi-directional I/O data transfer

   a) In **Mode** 0, Port A and Port B can be configured as simple 8-bit input or output ports without handshaking. The two halves of Port C can be programmed separately as 4-bit input or output ports.

   b) In **Mode 1**, two groups each of 12 pins are formed. Group A consists of Port A and the upper half of Port C while Group B consists of Port B and the lower half of Port C. Ports A and B can be programmed as 8-bit input or output ports with three lines of Port C in each group used for handshaking.

   c) In **Mode 2**, only Port A can be used as a bidirectional port. The handshaking signals are provided on five lines of Port C (PC_3 - PC_7). Port B can be used in mode 0 or in mode 1.
8. All I/O pins of 8255 has 2.5 mA d.c. driving capacity (i.e. sourcing current of 2.5 mA).

3.4.2 Pin Diagram

- Fig. 3.4.1 shows the pin diagram of 8255.

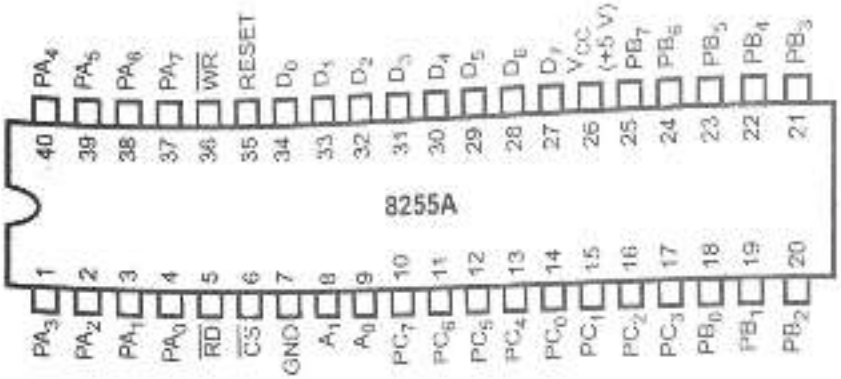


Fig. 3.4.1 Pin diagram of 8255A

Pin symbols	Function
D_0-D_7 (Data bus)	These bi-directional, tri-state data bus lines are connected to the system data bus. They are used to transfer data and control word from microprocessor (8085) to 8255 or to receive data or status word from 8255 to the 8085.
PA_0-PA_7 (Port A)	These 8-bit bi-directional I/O pins are used to send data to output device and to receive data from input device. It functions as an 8-bit data output latch/buffer, when used in output mode and an 8-bit data input buffer, when used in input mode.
PB_0-PB_7 (Port B)	These 8-bit bi-directional I/O pins are used to send data to output device and to receive data from input device. It functions as an 8-bit data, output latch/buffer when used in output mode and an 8-bit data input buffer, when used in input mode.
PC_0-PC_7	These 8-bit bi-directional I/O pins are divided into two groups PC_L (PC_3-PC_0) and PC_U (PC_7-PC_4). These groups individually can transfer data in or out when programmed for simple I/O, and used as handshake signals when programmed for handshake or bi-directional modes.
$\overline{RD}$ (Read)	When this pin is low, the CPU can read the data in the ports or the status word, through the data buffer.
$\overline{WR}$ (Write)	When this input pin is low, the CPU can write data on the ports or in the control register through the data bus buffer.
$\overline{CS}$ (Chip Select)	This is an active low input which can be enabled for data transfer operation between the CPU and the 8255.
RESET	This is an active high input used to reset 8255. When RESET input is high, the control register is cleared and all the ports are set to the input mode. Usually RESET OUT signal from 8085 is used to reset 8255.
A_0 and A_1	These input signals along with $\overline{RD}$ and $\overline{WR}$ inputs control the selection of the control/status word registers or one of the three ports.

Address		Port
A_1	A_0	
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control register

Table 3.4.1 Port selection

3.4.3 Block Diagram

- Fig. 3.4.2 shows the internal block diagram of 8255A. It consists of data bus buffer, control logic and Group A and Group B controls.

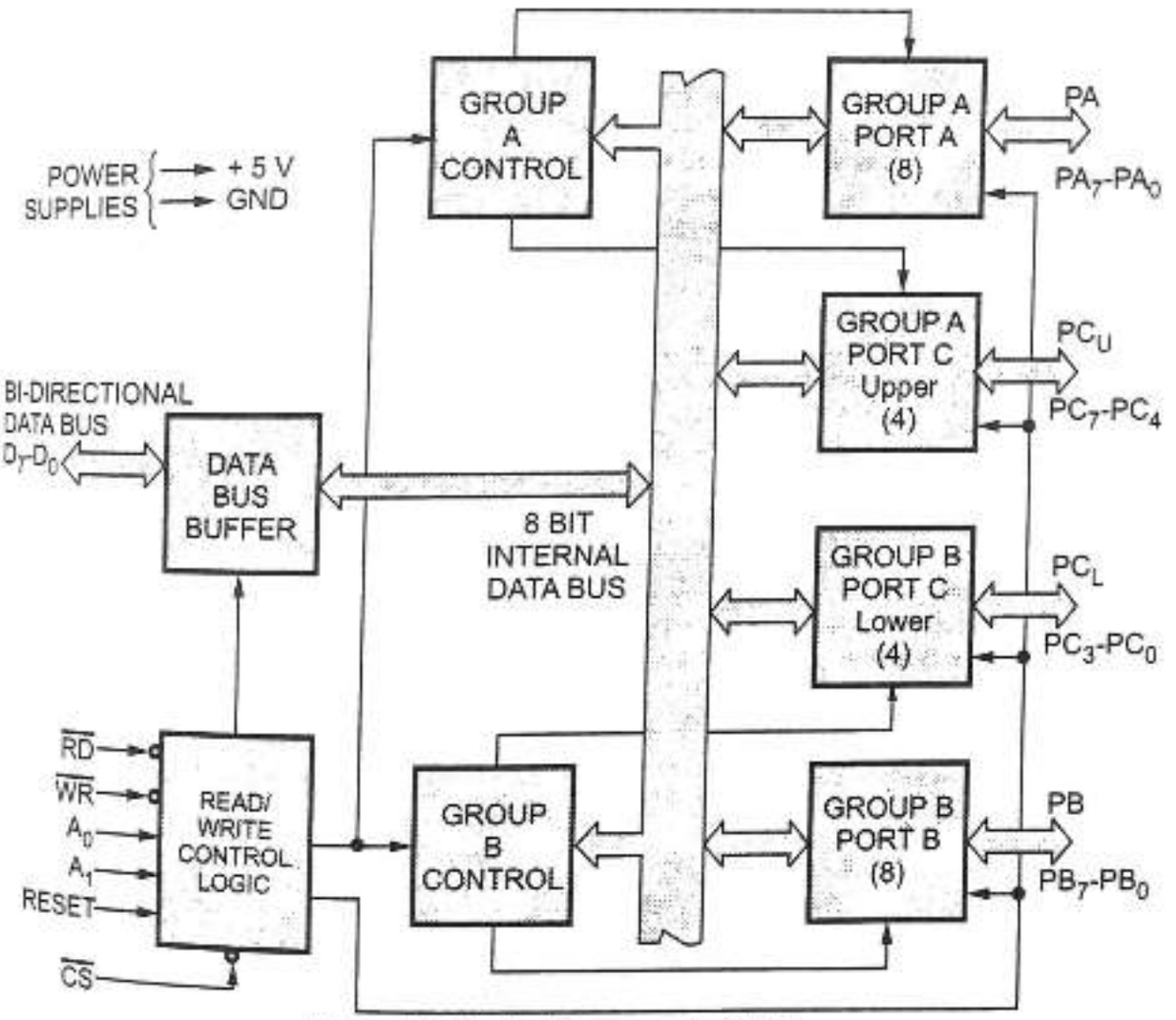


Fig. 3.4.2 Block diagram of 8255A

3.4.3.1 Data Bus Buffer

- This tri-state bi-directional buffer is used to interface the internal data bus of 8255 to the system data bus.

- Input or Output instructions executed by the CPU either Read data from, or Write data into the buffer.
- Output data from the CPU to the ports or control register, and input data to the CPU from the ports or status register are all passed through the buffer.

3.4.3.2 Control Logic

- The control logic block accepts control bus signals as well as inputs from the address bus and issues commands to the individual group control blocks (Group A control and Group B control). It issues appropriate enabling signals to access the required data/control words or status word.
- The input pins for the control logic section are described here.

3.4.3.3 Group A and Group B Controls

- Each of the Group A and Group B control blocks receives control words from the CPU and issues appropriate commands to the ports associated with it.
- The Group A control block controls Port A and PC_7-PC_4 while the Group B control block controls Port B and PC_3-PC_0.

Port A : This has an 8-bit latched and buffered output and an 8-bit input latch. It can be programmed in three modes : mode 0, mode 1 and mode 2.

Port B : This has an 8-bit data I/O latch/buffer and an 8-bit data input buffer. It can be programmed in mode 0 and mode 1.

Port C : This has one 8-bit unlatched input buffer and an 8-bit output latch/buffer. Port C can be separated into two parts and each can be used as control signals for ports A and B in the handshake mode. It can be programmed for bit set/reset operation.

3.4.4 Operating Modes

3.4.4.1 Bit Set-Reset (BSR) Mode

- The individual bits of Port C can be set or reset by sending out a single OUT instruction to the control register. When Port C is used for control/status operation, this feature can be used to set or reset individual bits.

3.4.4.2 I/O Modes

Mode 0 : Simple input/output

- In this mode, Ports A and B are used as two simple 8-bit I/O ports and Port C as two 4-bit ports.

- Each port (or half - port, in case of C) can be programmed to function as simply an input port or an output port.
- The input/output features in mode 0 are as follows :
  1. Outputs are latched. 2. Inputs are buffered, not latched.
  3. Ports do not have handshake or interrupt capability.

Mode 1 : Input/Output with handshake

- In this mode, input or output data transfer is controlled by handshaking signals.
- Handshaking signals are used to transfer data between devices whose data transfer speeds are not same.
- For example, computer can send data to the printer with large speed but printer can't accept data and print data with this rate. So computer has to send data with the speed with which printer can accept. This type of data transfer is achieved by using handshaking signals alongwith data signals.
- Fig. 3.4.3 shows data transfer between computer and printer using handshaking signals. These handshaking signals are used to tell computer whether printer is ready to accept the data or not. If printer is ready to accept the data then after sending data on data bus, computer uses another handshaking signal ($\overline{STB}$) to tell printer that valid data is available on the data bus.

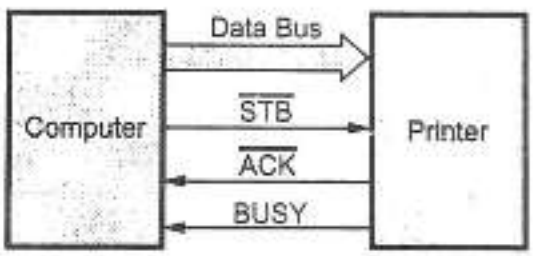


Fig. 3.4.3 Data transfer between computer and printer using handshaking signals handshaking signals.

- The 8255 mode 1 which supports handshaking has following features.
  1. Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports.
  2. Each port uses three lines from Port C as handshake signals. The remaining two lines of Port C can be used for simple I/O functions.
  3. Input and output data are latched.
  4. Interrupt logic is supported.

Mode 2 : Bi-directional I/O data transfer

- This mode allows bi-directional data transfer (transmission and reception) over a single 8-bit data bus using handshaking signals. This feature is available only in Group A with Port A as the 8-bit bidirectional data bus ; and PC_3 - PC_7 are used for handshaking purpose.
- In this mode, both inputs and outputs are latched.

- Due to use of a single 8-bit data bus for bi-directional data transfer, the data sent out by the CPU through Port A appears on the bus connecting it to the peripheral, only when the peripheral requests it.
- The remaining lines of Port C i.e. PC_0-PC_2 can be used for simple I/O functions.
- The Port B can be programmed in mode 0 or in mode 1. When Port B is programmed in mode 1, PC_0-PC_2 lines of Port C are used as handshaking signals.

3.4.5 Control Word Formats

- A high on the RESET pin causes all 24 lines of the three 8-bit ports to be in the input mode. All flip-flops are cleared and the interrupts are reset. This condition is maintained even after the RESET goes low.
- The ports of the 8255 can then be programmed for any other mode by writing a single control word into the control register, when required.

3.4.5.1 For Bit Set/Reset Mode

- Fig. 3.4.4 shows bit set/reset control word format.

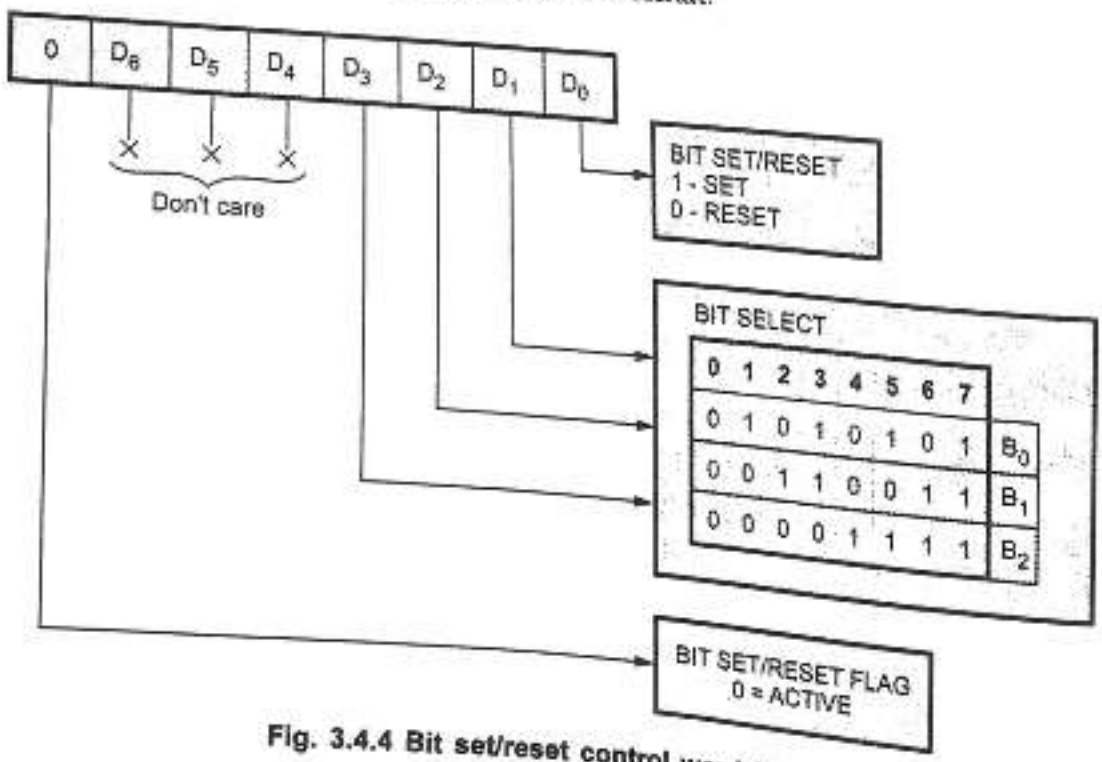


Fig. 3.4.4 Bit set/reset control word format

- The eight possible combinations of the states of bits D_3 - D_1 (B_2 B_1 B_0) in the Bit Set-Reset format (BSR) determine particular bit in PC_0 - PC_7 being set or reset as per he status of bit D_0.
- A BSR word is to be written for each bit that is to be set or reset.

- For example, if bit PC_3 is to be set and bit PC_4 is to be reset, the appropriate BSR words that will have to be loaded into the control register will be, $0\times\times\times0111$ and $0\times\times\times1000$, respectively, where $\times$ is don't care.
- The BSR word can also be used for enabling or disabling interrupt signals generated by Port C when the 8255 is programmed for Mode 1 or 2 operation. This is done by setting or resetting the associated bits of the interrupts. This is described in detail in next section.

3.4.5.2 For I/O Mode

- The mode definition format for I/O mode is shown in Fig. 3.4.5. The control words for both, mode definition and bit set-reset are loaded into the same control register, with bit D_7 used for specifying whether the word loaded into the control register is a mode definition word or bit set-reset word. If D_7 is high, the word is

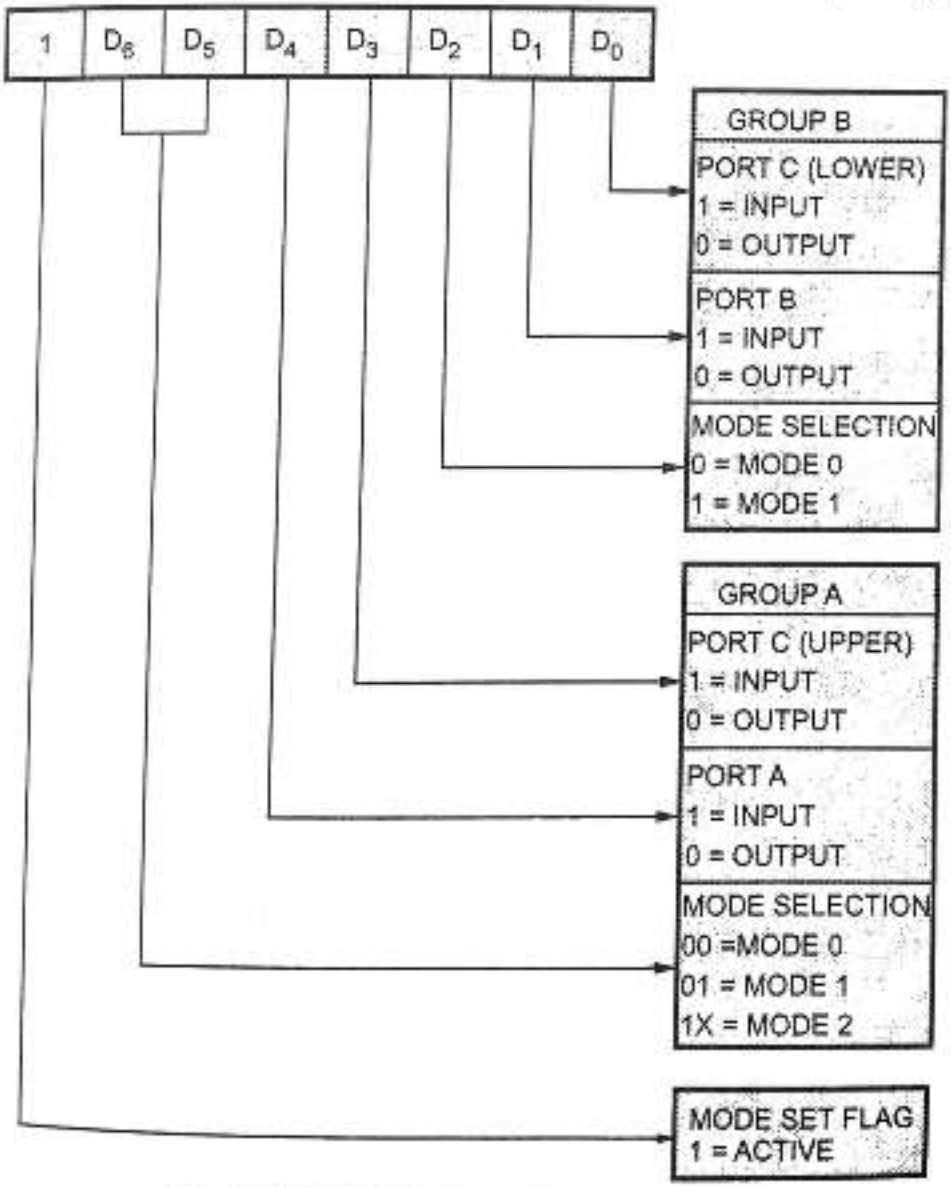


Fig. 3.4.5 8255 Mode definition format

taken as a mode definition word, and if it is low, it is taken as a bit set-reset word. The appropriate bits are set or reset depending on the type of operation desired, and loaded into the control register. (See Fig. 3.4.5 on previous page).

Example 3.4.1 *Write a program to initialize 8255 in the configuration given below :*

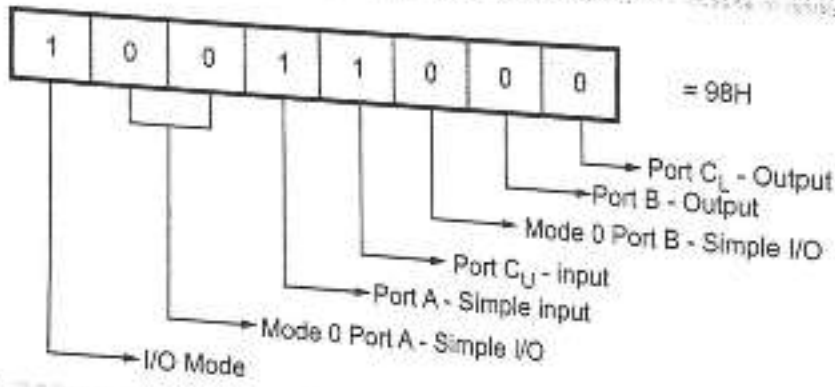
1. *Port A : Simple input*
2. *Port B : Simple output*
3. *Port C_L : Output*
4. *Port C_U : Input*

Assume address of the control word register of 8255 is 83H.

Solution :

Source program :

```
MOV AL,98H      ; Load control word
OUT 83H,AL      ; Send control word
```

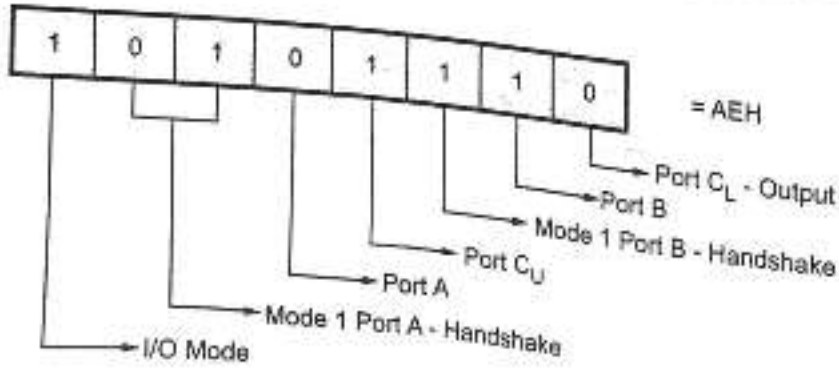


Example 3.4.2 *Write a program to initialize 8255 in the configuration given below :*

1. *Port A : Output with handshake*
2. *Port B : Input with handshake*
3. *Port C_L : Output*
4. *Port C_U : Input*

Assume address of the control word register of 8255 is 23H.

Solution :



Source program :

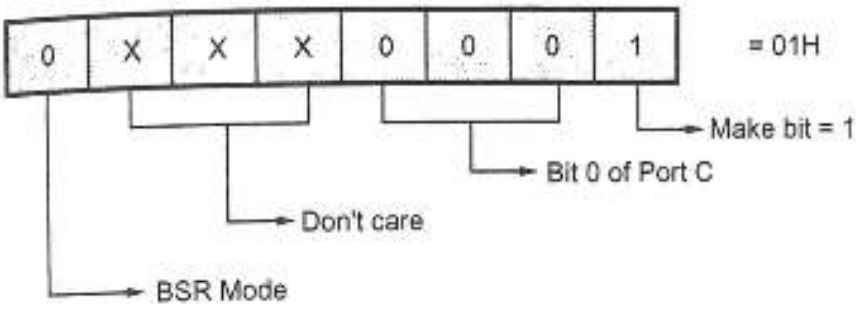
```
MOV AL,0AEH     ; Load control word
OUT 23H,AL      ; Send control word
```

Program : Blink port C bit 0 of 8255.

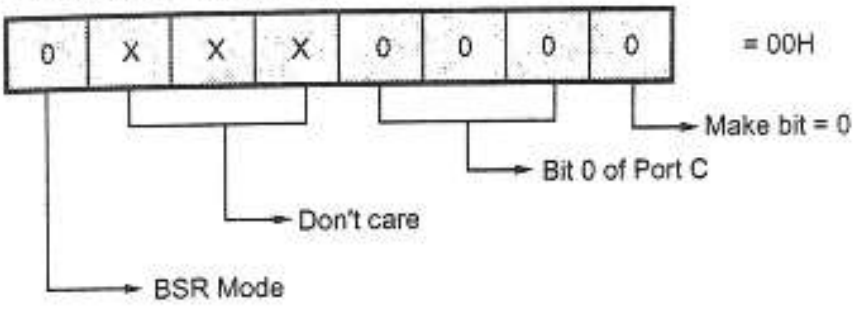
Program statement :

Write a program to blink Port C bit 0 of the 8255. Assume address of control word register of 8255 is 83H. Use bit set/reset mode.

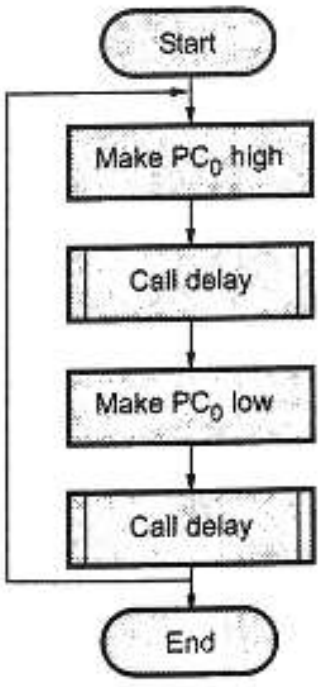
Solution : Control word to make bit 0 high.



Control word to make bit 0 low



Flowchart :



Source program :

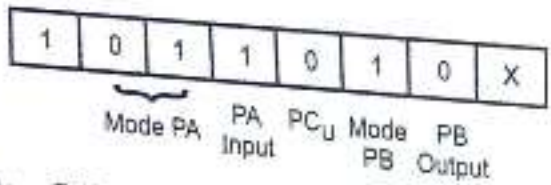
```
BACK:   MOV AL,01H  ; Load bit pattern to make PC0 high
        OUT 83H,AL  ; Send it to control word register
        CALL DELAY  ; Call Delay subroutine
        MOV AL,00H  ; Load bit pattern to make PC0 Low
        OUT 83H,AL  ; Send it to control word register
        CALL Delay  ; Call Delay subroutine
        JMP BACK    ; Repeat
```

Example 3.4.3 *Explain the control word format in I/O and BSR mode. Prepare a control word needed to initialize 8255 as :*

i) Port A handshake input *ii) Port B handshake output*

iii) Port C bits PC_6 and PC_7 as output.

Solution : Control word



Example 3.4.4 *Write a BSR control word subroutine to set bits PC7 and PC3 and reset them after 10 msec. Assume that a delay subroutine is available. Address for control word register = 83 H.*

Solution : Program

```
SR PROC NEAR
MOV AL, 07H ;   Load bit pattern to make PC3 high
OUT 83H, AL ;   Send it to control word register
MOV AL, 0FH ;   Load bit pattern to make PC7 high
OUT 83H, AL ;   Send it to control word register
CALL DELAY  ;   WAIT for 10 msec
MOV AL, 06H ;   Load bit pattern to make PC3 low
OUT 83H, AL ;   Send it to control word register
MOV AL, 0EH ;   Load bit pattern to make PC7 low
OUT 83H, AL ;   Send it to control word register
RET
ENDP
```

Example 3.4.5 *Consider the following addresses for PA = 30H, PB = 31H, PC = 32H, CWR = 33H. Write a set of instructions to set PC0 and PC2 using BSR mode.*

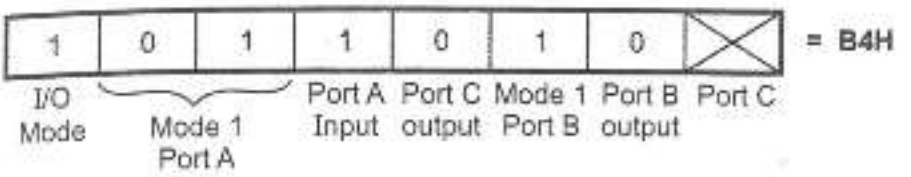
Solution :

```
MOV AL, 01H     ; Set PC0
OUT 33H, AL
MOV AL, 05H     ; Set PC2
OUT 33H, AL
```

Example 3.4.6 *Show the mode set control word need to initialize an 8255A as follows :*

- *Port A - handshake input ;*
- *Port B - handshake output ;*
- *Port C - bits PC6 and PC7 as outputs.*

Solution :



3.4.6 8255 Programming and Operation

3.4.6.1 Programming in Mode 0

- The Ports A, B and C can be configured as simple input or output ports by writing the appropriate control word in the control word register.
- In the control word, D_7 is set to '1' (to define a mode set operation) and D_6, D_5 and D_2 are all set to '0' to configure all the ports in Mode 0 operation.
- The status of bits D_4, D_3, D_1 and D_0 then determine (refer to Fig. 3.4.7) whether the corresponding ports are to be configured as input or output.
- For example in mode 0, if Port A and Port B are to operate as output ports with Port C lower as input and Port C upper as output, the control word that will have to be loaded into the control register will be as follows.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0	0	0	1	= 81H

- As mentioned earlier, this mode provides simple input and output operations for each of the three ports. No handshaking is required, data is simply written to or read from a specified port.

Input mode :

- Fig. 3.4.6 shows the timing diagram for mode 0 input mode.
- After initialization of 8255 in the input mode 0, CPU can read data through the input port by initiating read command with proper port address.
- Read command activates $\overline{RD}$ signal. Upon activation of $\overline{RD}$ signal CPU reads the data from the selected input port into the CPU register.

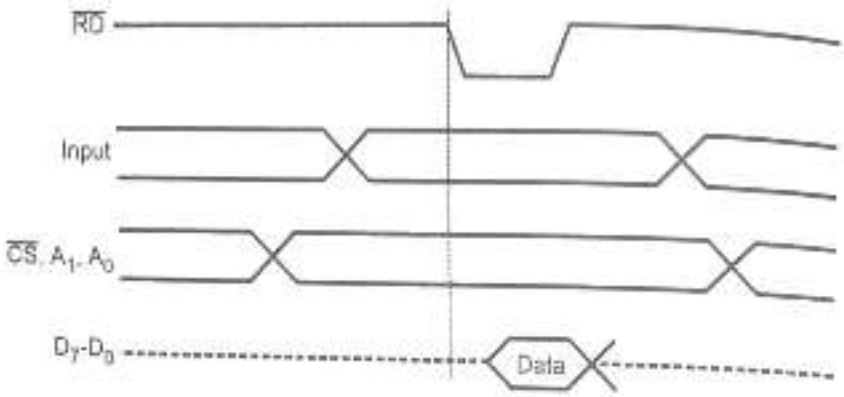


Fig. 3.4.6 Timing diagram for mode 0 input mode

Output mode : Fig. 3.4.7 shows the timing diagram for mode 0 output mode.

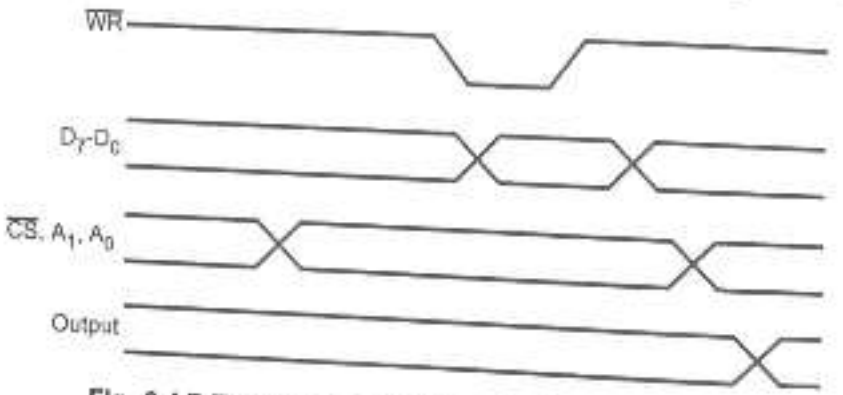


Fig. 3.4.7 Timing diagram for mode 0 output mode

After initialization of 8255 in the output mode 0, CPU can write data into the output port by initiating write command with proper port address. CPU sends data on the data bus and upon activation of $\overline{WR}$ signal, data on the data bus gets latched on the selected output port.

Mode 0 configurations :

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (Upper)	#	PORT B	PORT C (Lower)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT

0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

3.4.6.2 Programming in Mode 1 (Input / Output with Handshake)

- Both Group A and Group B can operate in Mode 1, either together or individually, with each port containing an 8-bit latched Input or Output data port, and a 4-bit port which is used for control and status of the 8-bit port.
- When Port A is to be programmed as an input port, PC_3 , PC_4 and PC_5 are used for control.
- PC_6 and PC_7 are not used and can be Input or Output, as programmed by bit D_3 of the control word.
- When Port A is programmed as an output port, PC_3 , PC_6 and PC_7 are used for control and PC_4 and PC_5 can be Input or Output, as programmed by bit D_3, of the control word.
- When port B is to be programmed as an input or output port, PC_0, PC_1 and PC_2 are used for control.

Mode 1 input control signals :

1. $\overline{STB}$ (Strobe Input) :

- This is an active low input signal for 8255 and output signal for the input device.
- The input device activates this signal to indicate CPU that the data to be read is already sent on the port lines of 8255 port. Upon activation of this signal 8255 loads the data from the input port lines into the input buffer of that port.

2. IBF (Input Buffer Full) :

- This is an active high output signal for 8255 and an input signal for input device.
- This signal is generated by 8255 in response to $\overline{STB}$ signal as an acknowledgment to input device.
- It also indicates to the input device that the input buffer is full and it is not ready to accept next byte from the input device. Therefore input device sends data on the port lines only when IBF signal is not active.
- The IBF signal is deactivated when CPU reads the data from input buffer of the respective port by activation of $\overline{RD}$ signal.

3. INTR (Interrupt Request) :

- This is an active high output signal generated by 8255. A 'high' on this output can be used to interrupt the CPU when an input device is requesting service.
- The 8255 sets the INTR when $\overline{STB}$ signal is 'one', IBF signal is 'one' and INTE is 'one', indicating CPU that the data from the input device is available in the input buffer.
- This signal is reset by the falling edge of the $\overline{RD}$ signal i.e. immediately after reading the data from the input buffer.
- INTE (Interrupt Enable) flip-flop is used to enable or disable INTR (Interrupt request) signal.
- If INTE flip-flop is set, the interrupt request is generated depending on the status of $\overline{STB}$ and IBF signals. If INTE flip-flop is reset, the interrupt request is not generated, allowing masking facility for the interrupt.

Mode 1 : Port A input operation

- Fig. 3.4.8 (a) shows Port A as an input port along with the control word and control signals (for handshaking with a peripheral). When the control word (as in Fig. 3.4.8 (a)) is loaded into the control register, Group A is configured in Mode 1 with Port A as an input port.
- Port A can accept parallel data from a peripheral (like a keyboard) and this data can be read by the CPU. The peripheral first loads data into Port by making the $\overline{STB}_A$ input low. This latches the data placed by the peripheral on the common data bus into Port A. Port A acknowledges reception of data by making IBF_A (Input Buffer Full) high. IBF_A is set when the $\overline{STB}_A$ input is made low, as shown in Fig. 3.4.8 (b).

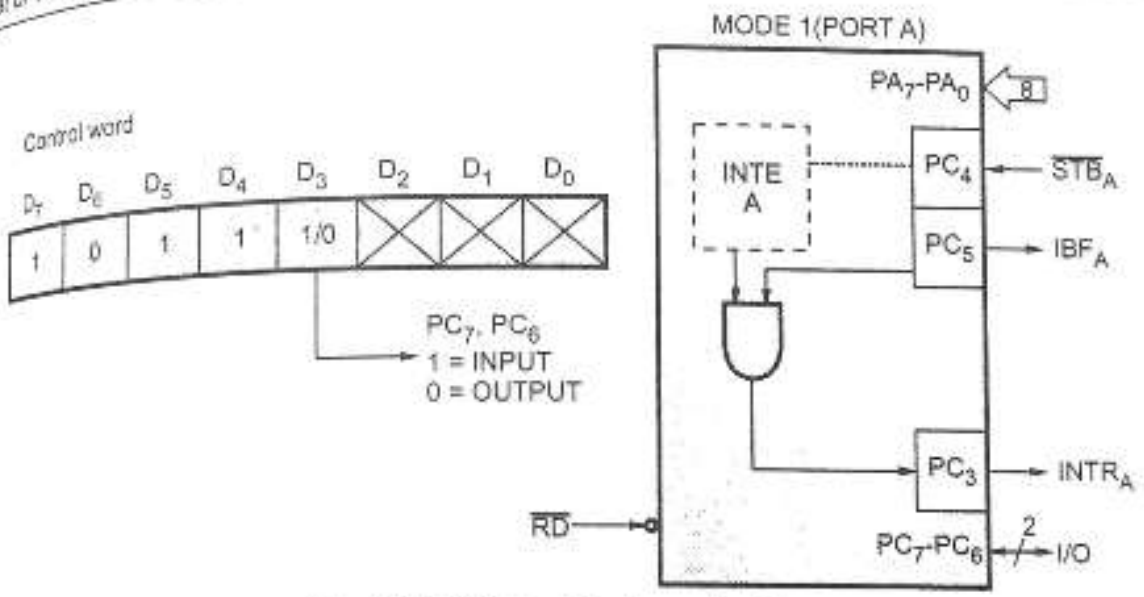


Fig. 3.4.8 (a) Port A in mode 1

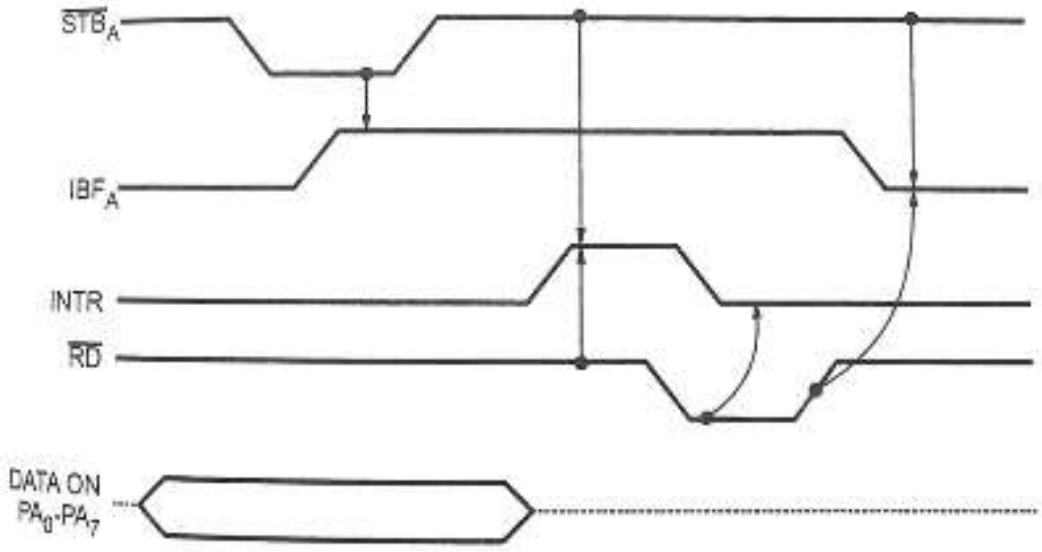


Fig. 3.4.8 (b) Timing diagram for port A in mode 1

- $INTR_A$ is an active high output signal which can be used to interrupt the CPU so that the CPU can suspend its current operation and read the data written into Port A by the peripheral. $INTR_A$ can be enabled or disabled by the $INTE_A$ flip-flop which is controlled by Bit Set-Reset operation of PC_4. $INTR_A$ is set (if enabled by setting the $INTE_A$ flip-flop) after the $\overline{STB}_A$ has gone high again and if IBF_A is high.
- On receipt of the interrupt, the CPU can be forced to read Port A. The falling edge of the $\overline{RD}$ input resets IBF_A and it goes low. This can be used to indicate to the peripheral that the input buffer is empty and that data can again be loaded into it.

Mode 1 : Port B input operation

- Fig. 3.4.9 shows Port B as an input port (when in Mode 1).

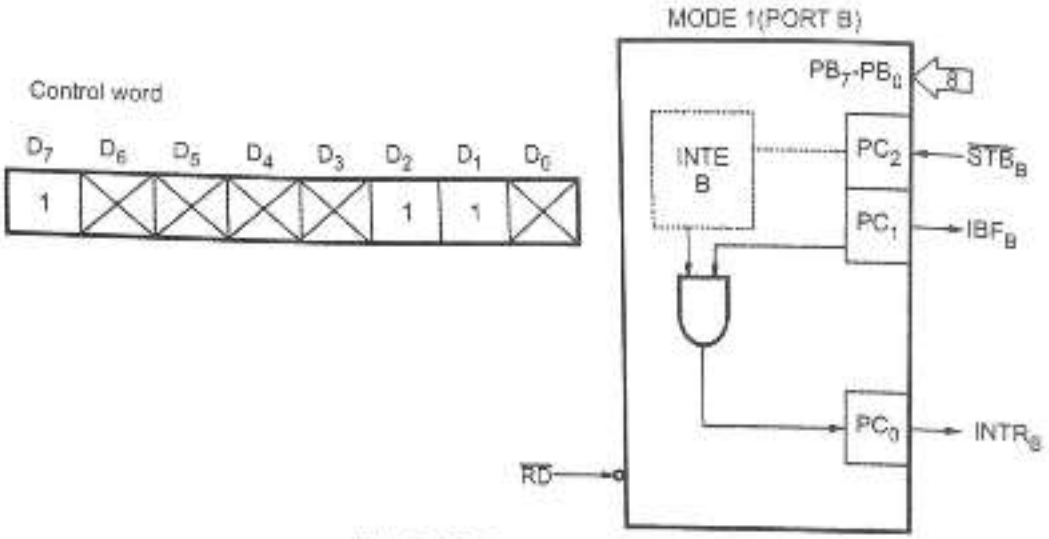


Fig. 3.4.9 Port B in mode 1

- The timing diagram and operation of Port B is similar to that of Port A except that it uses different bits of Port C for control.
- $INTE_B$ is controlled by bit set/reset of PC_2.
- If the CPU is busy with other system operations, it can read data from the input port when it is interrupted. This is often called interrupt driven I/O.
- However, if the CPU is otherwise not busy with other jobs, it can continuously poll (read) the status word to check for an IBF_A. This is often called program controlled I/O.
- The status word is accessed by reading Port C (A_1 A_0 must be 10, $\overline{RD}$ and $\overline{CS}$ must be low). The status word format when Ports A and B are input ports in Mode 1, is shown in Fig. 3.4.10.

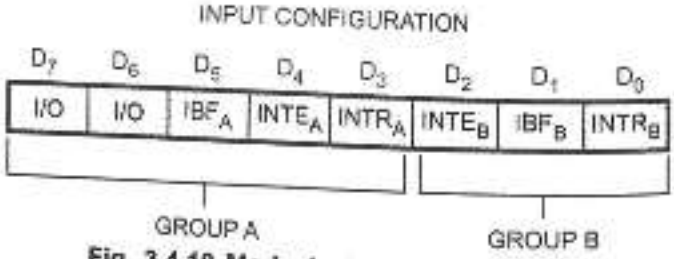


Fig. 3.4.10 Mode 1 status word (input)

Mode 1 : Output control signals

1. $\overline{OBF}$ (Output Buffer Full) :

- This is an active low output signal for 8255 and input signal for the output device. The 8255 activates this signal to indicate output device that data is available on the output port. Upon activation of $\overline{OBF}$ signal, output device reads data from the output port and acknowledges it by $\overline{ACK}$ signal.
- The $\overline{OBF}$ signal is activated at the rising edge of the $\overline{WR}$ signal and de-activated at the falling edge of the $\overline{ACK}$ signal.

2. $\overline{ACK}$ (Acknowledge Input) :

- This is an active low input signal for 8255 and output signal for the output device.
- The output device generates this signal to indicate 8255 that the data from port A or Port B has been accepted.

3. INTR (Interrupt Request) :

- This is an active high output signal generated by 8255. A 'high' on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU.
- The 8255 sets the INTR when $\overline{ACK}$ signal is 'one', $\overline{OBF}$ is 'one' and INTE is 'one', indicating that the output device is ready to accept next data byte. This signal is reset by the falling edge of the $\overline{WR}$ signal i.e. immediately after sending the data to the output port.
- INTE (Interrupt Enable) flip-flop is used to enable or disable INTR (Interrupt Request) signal. If INTE flip-flop is set, the interrupt request is generated depending on the status of $\overline{ACK}$ and $\overline{OBF}$ signals. If INTE flip-flop is reset, the interrupt request is not generated, allowing masking facility for the interrupt.

Mode 1 : Port A output operation

- Fig. 3.4.11 (a) shows Port A configured as an output port (When in Mode 1) along with the control word and control signals (for handshaking with a peripheral).
- When the control word (as in Fig. 3.4.11 (a)) is loaded into the control register, Group A is configured in Mode 1 with Port A as an output port. The CPU can send data to a peripheral (like a display device) through Port A of the 8255.
- The $\overline{OBF}_A$ output (Output Buffer Full) goes low on the rising edge of the $\overline{WR}$ signal (when the CPU writes data into the 8255).

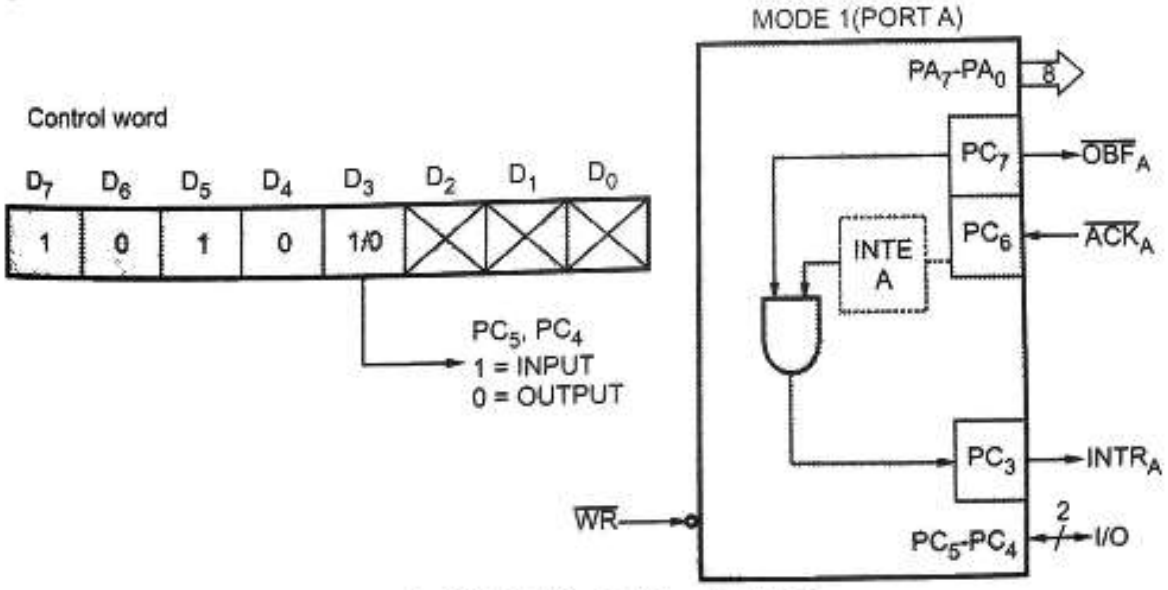


Fig. 3.4.11 (a) Port A in mode 1

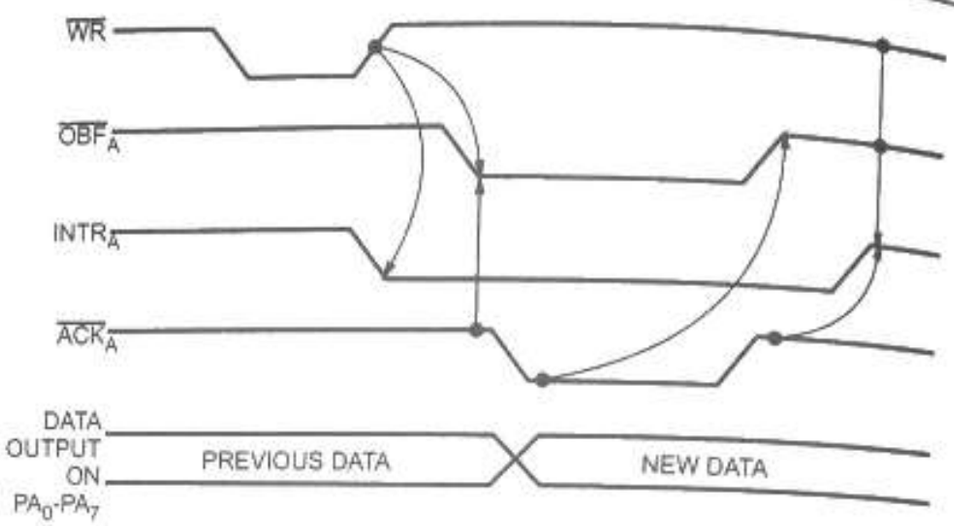


Fig. 3.4.11 (b) Port A in mode 1 (Output)

- The $\overline{OBF}_A$ output from 8255 can be used as a strobe input to the peripheral to latch the contents of Port A.
- The peripheral responds to the receipt of data by making the $\overline{ACK}_A$ input of the 8255 low, thus acknowledging that it has received the data sent by the CPU through Port A. The $\overline{ACK}_A$ low sets the OBF_A signal, which can be polled by the CPU through OBF_A of the status word to load the next data when it is high again.
- $INTR_A$ is an active high output of the 8255 which is made high (if the associated $INTE_A$ flip-flop is set) when $\overline{ACK}_A$ is made high again by the peripheral, and when $\overline{OBF}_A$ goes high again (see timing diagram in Fig. 3.4.12 (b)). It can be used to interrupt the CPU whenever the output buffer is empty. It is reset by the falling edge of $\overline{WR}$ when the CPU writes data onto Port A. It can be enabled or disabled by writing a '1' or a '0' respectively to PC_6 in the BSR mode.

Mode 1 : Port B output operation

- Fig. 3.4.12 shows Port B as an output port when in Mode 1. The operation of Port B is similar to that of Port A.
- $INTR_A$ is controlled by writing a '1' or a '0' to PC_2 in the BSR mode.

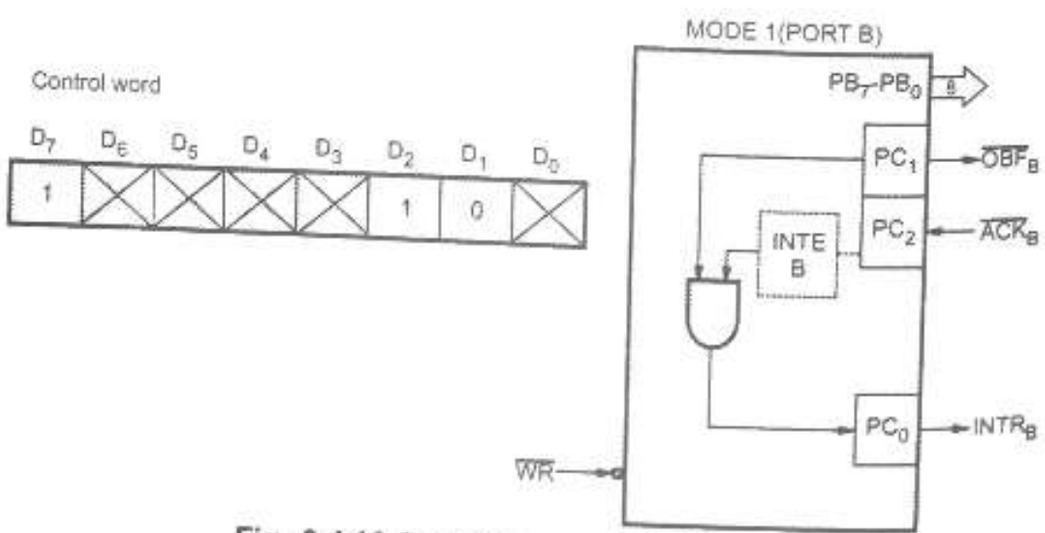


Fig. 3.4.12 Port B in mode 1 (Output)

- The status word is accessed by issuing a Read to Port C. The format of the status word when Ports A and B are Output ports in Mode 1 is shown in Fig. 3.4.13.

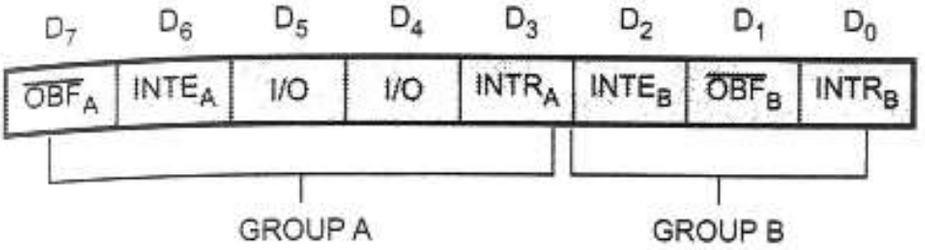


Fig. 3.4.13 Mode 1 status word (Output)

3.4.6.3 Programming in Mode 2 (Strobes Bi-directional Bus I/O)

- When the 8255 is operated in Mode 2 (by loading the appropriate control word), Port A can be used as a bi-directional 8-bit I/O bus using for handshaking.
- Port B can be programmed in Mode 0 or in Mode 1. When Port B is programmed in mode 1, PC_0 - PC_2 lines of Port C are used as handshaking signals.
- Fig. 3.4.14 shows the control word that should be loaded into the control port to configure 8255 in Mode 2.

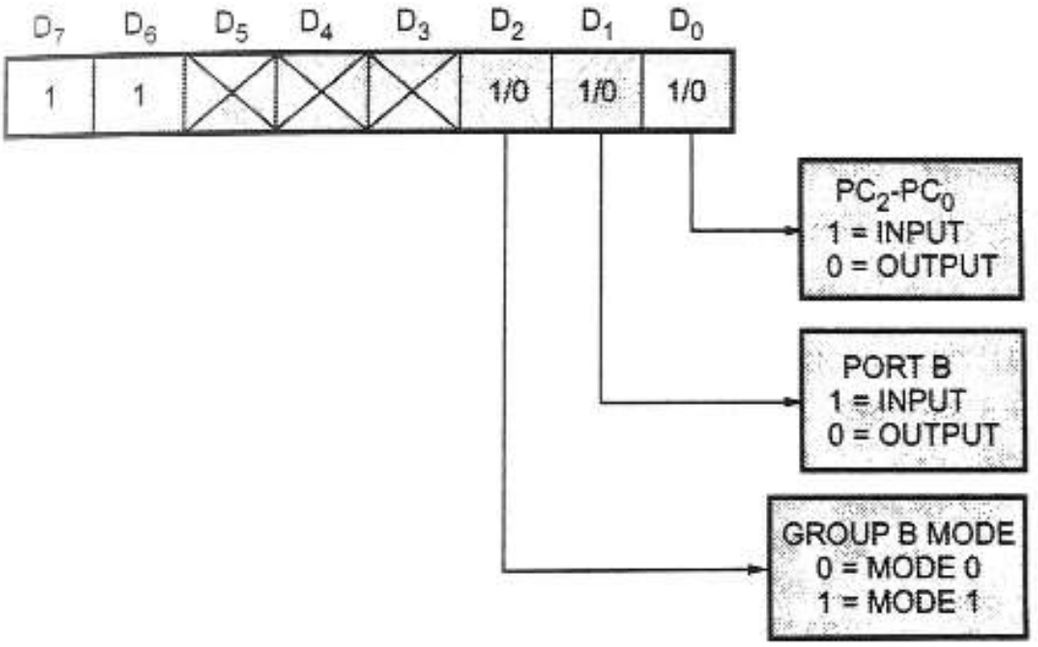


Fig. 3.4.14 Mode 2 control word

Mode 2 : Control signals

INTR (Interrupt Request) : A 'high' on this output can be used to interrupt the CPU for input or output operations.

Output control signals :

$\overline{OBF}_A$ (Output Buffer Full)

This is an active low output which indicates that the CPU has written data into Port A.

$\overline{ACK}_A$ (Acknowledge)

This is an active low input signal (generated by the peripheral) which enables the tri-state output buffer of Port A and makes Port A data available to the peripheral. In Mode 2, Port A outputs are in tri-state until enabled.

INTE 1

This is the flip-flop associated with Output Buffer Full. INTE 1 can be used to enable or disable the interrupt by setting or resetting PC_6 in the BSR Mode.

Input control signals :

$\overline{STB}$ (Strobe Input)

This is an active low input signal which enables Port A to latch the data available at its input.

IBF (Input Buffer Full Flip-Flop)

This is an active high output which indicates that data has been loaded into the input latch of Port A.

INTE 2

This is an Interrupt enable flip-flop associated with Input Buffer Full. It can be controlled by setting or resetting PC_4 in the BSR mode.

Mode 2 : Port A operation

Fig. 3.4.15 shows Port A and associated control signals when 8255 is in Mode 2. Interrupts are generated for both output and input operations on the same $INTR_A$ (PC_3) line.

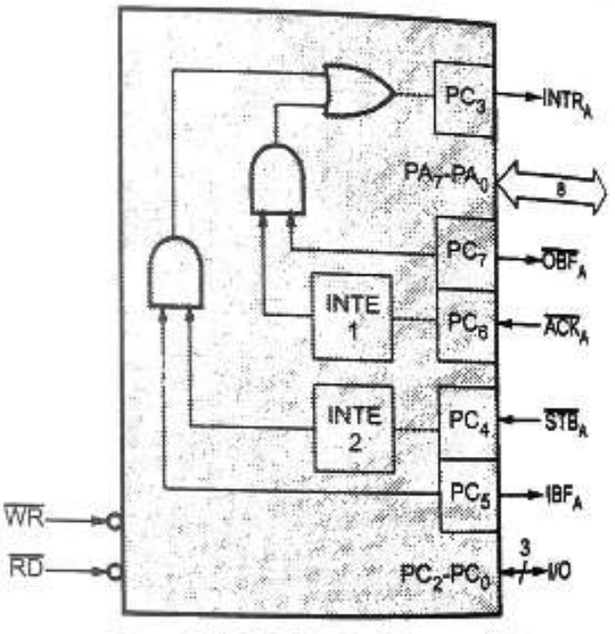


Fig. 3.4.15 Mode 2 operation

Status word in mode 2

- The status word for Mode 2 (accessed by reading Port C) is shown in Fig. 3.4.16.
- D_7-D_3 of the status word carry information about $\overline{OBF}_A$, $INTE_1$, IBF_A, $INTE_2$, $INTR_A$.
- The status of the bits D_2 - D_0 depend on the mode setting of Group B. If B is programmed in Mode 0, D_2 - D_0 are the same as PC_2 - PC_0 (simple I/O

however if B is in Mode 1, D_2 - D_0 carry information about the control signals for Port B (as in Fig. 3.4.11 or Fig. 3.4.14), depending upon whether Port B is an input port or output port respectively.

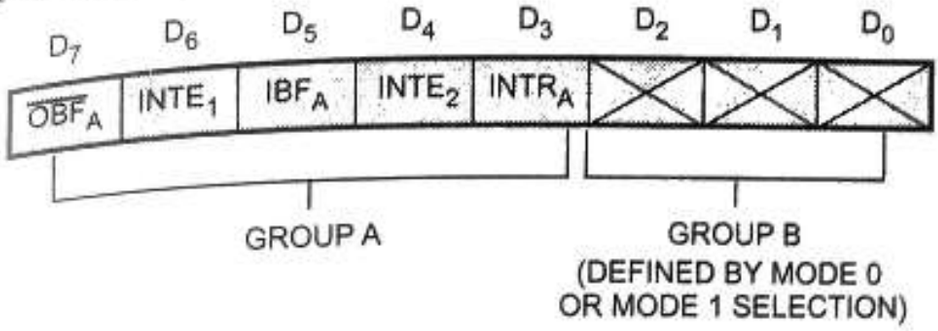


Fig. 3.4.16 Status word for mode 2

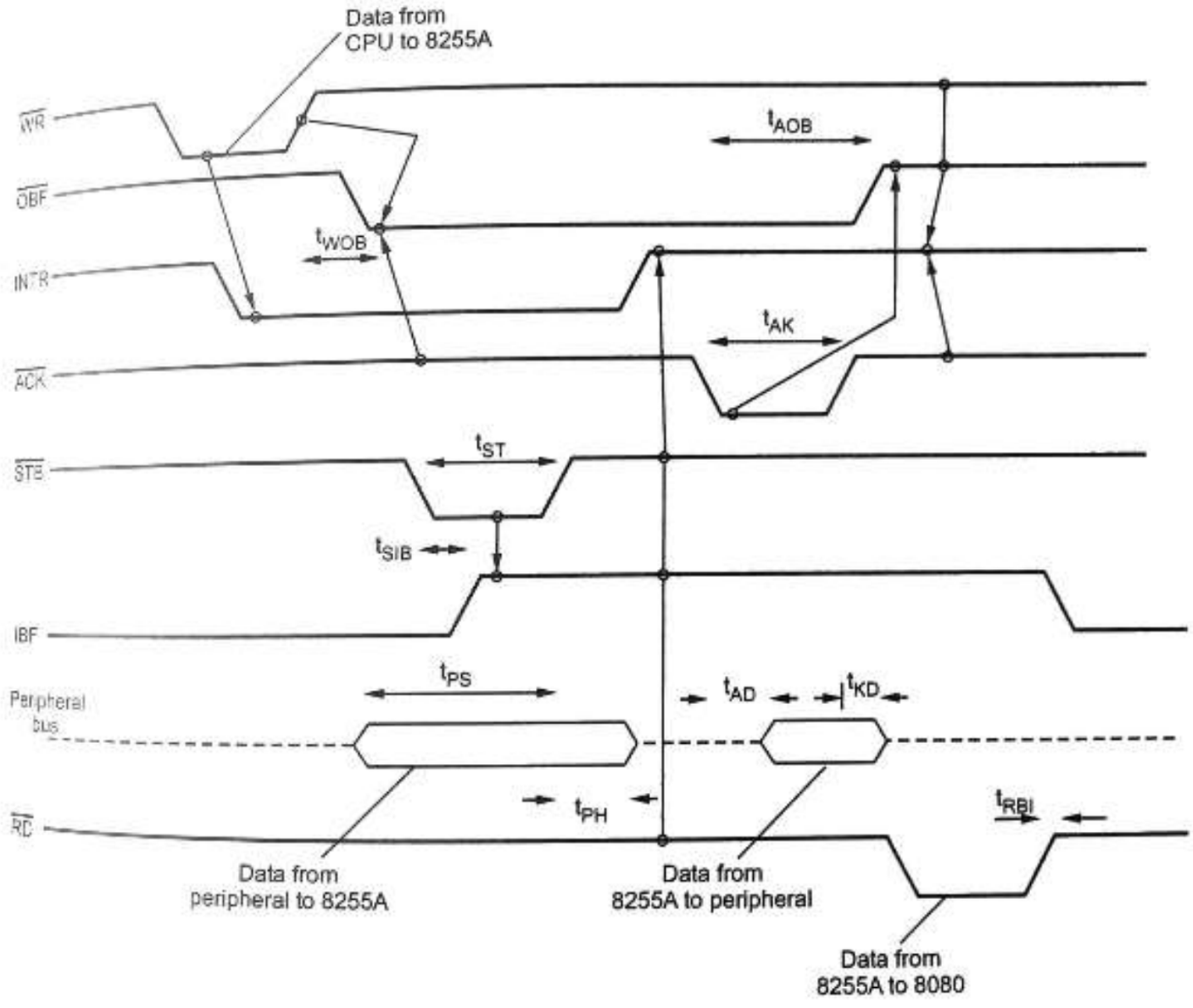


Fig. 3.4.17 Timing diagram for mode 2

3.4.7 Interfacing of 8255 with 8086

3.4.7.1 Interfacing 8255 to 8086 in I/O Mapped I/O Mode

- The 8086 has four special instructions IN, INS, OUT and OUTS to transfer data through the input/output ports in I/O mapped I/O system.
- $M/\overline{IO}$ signal is always low when 8086 is executing these instructions. So $M/\overline{IO}$ signal is used to generate separate addresses for, memory and input/output.
- Only 256 (2^8) I/O addresses can be generated when direct addressing method is used. By using indirect address method this range can be extended up to 65536 (2^{16}) addresses.
- Fig. 3.4.18 shows the interfacing of 8255 with 8086 in I/O mapped I/O technique. Here, $\overline{RD}$ and $\overline{WR}$ signals are activated when $M/\overline{IO}$ signal is low, indicating I/O bus cycle.
- Only lower data bus (D_0-D_7) is used as 8255 is 8-bit device.
- Reset out signal from clock generator is connected to the reset signal of the 8255.
- In case of interrupt driven I/O INTR signal (PC_3 or PC_0) from 8255 is connected to INTR input of 8086.

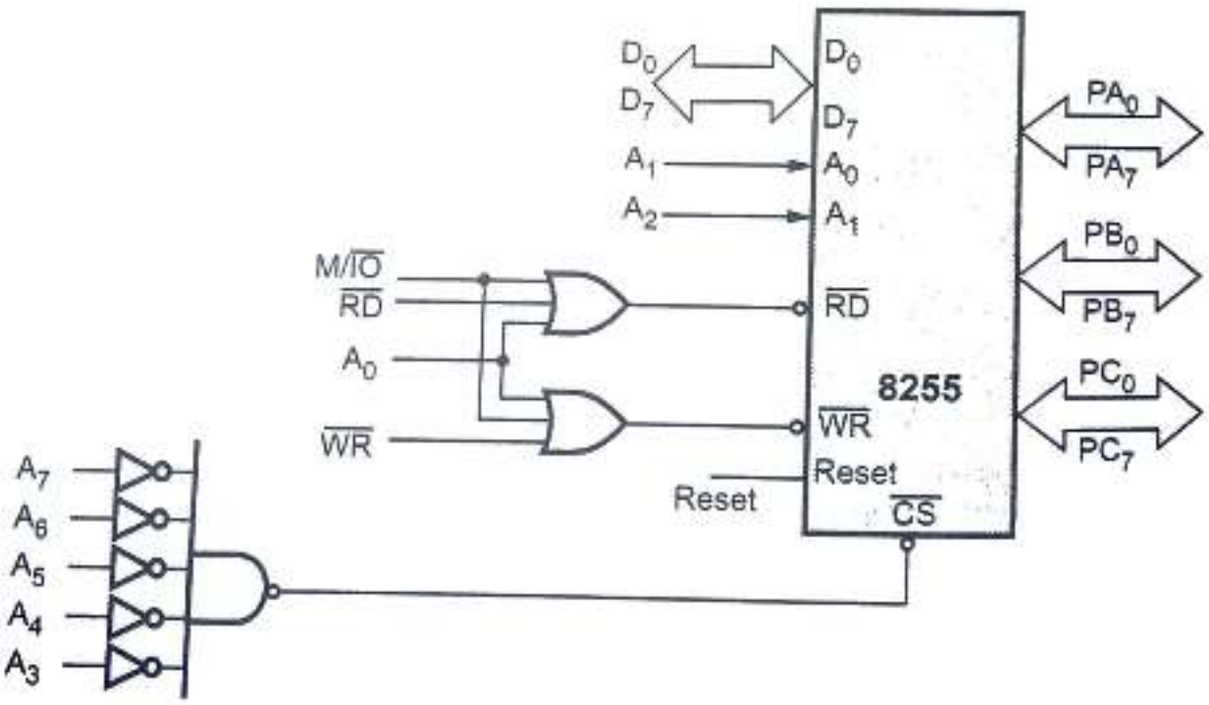


Fig. 3.4.18 I/O mapped I/O (Direct address method)

I/O map :

Port / Control register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port A	0	0	0	0	0	0	0	0	00H
Port B	0	0	0	0	0	0	1	0	02H
Port C	0	0	0	0	0	1	0	0	04H
Control register	0	0	0	0	0	1	1	0	06H

Note It is assumed that the direct addressing is used.

Example 3.4.7 *Interface an 8255 with 8086 so as to have Port-A address BCD1H, Port-B address BCD3H, Port-C address BCD5H and control word register address BCD7H.*

Solution :

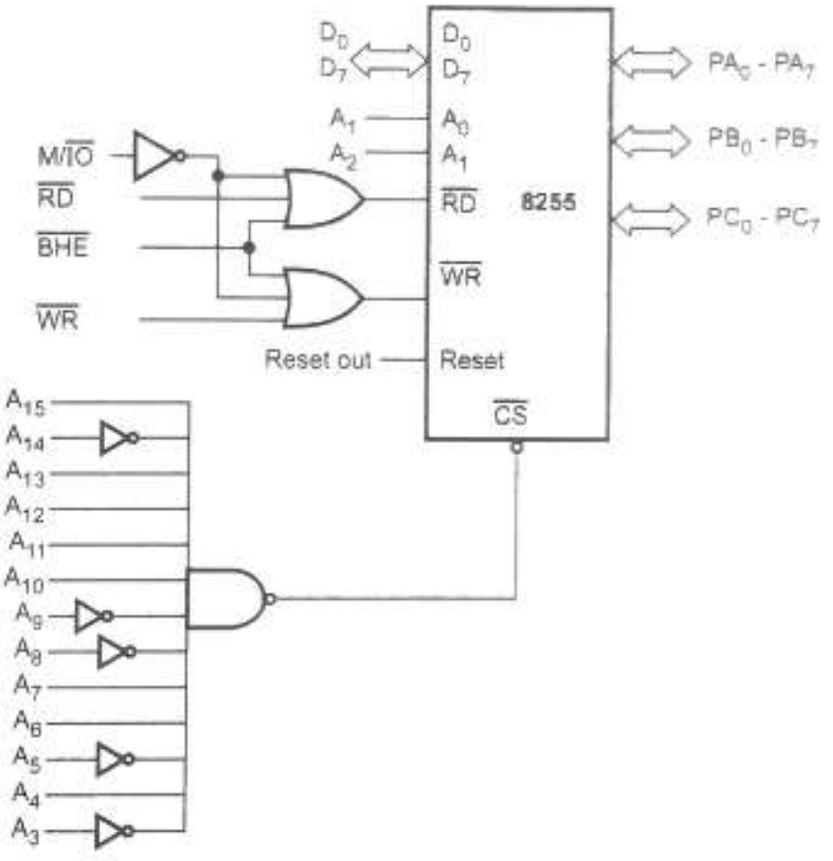


Fig. 3.4.19 I/O mapped I/O interfacing with indirect address method

3.4.7.2 Interfacing 8255 to 8086 in Memory Mapped I/O

- In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device ; an I/O device is connected as if it is a memory register.
- The 8086 uses same control signals and instructions to access I/O as those of memory.
- Fig. 3.4.20 shows the interfacing of 8255 with 8086 in memory mapped I/O technique. Here $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is high, indicating memory bus cycle.

- Address lines A_0 - A_1 are used by 8255 for internal decoding. To get absolute address, all remaining address lines (A_3 - A_{19}) are used to decode the address of 8255.
- Other signal connections are same as in I/O mapped I/O.

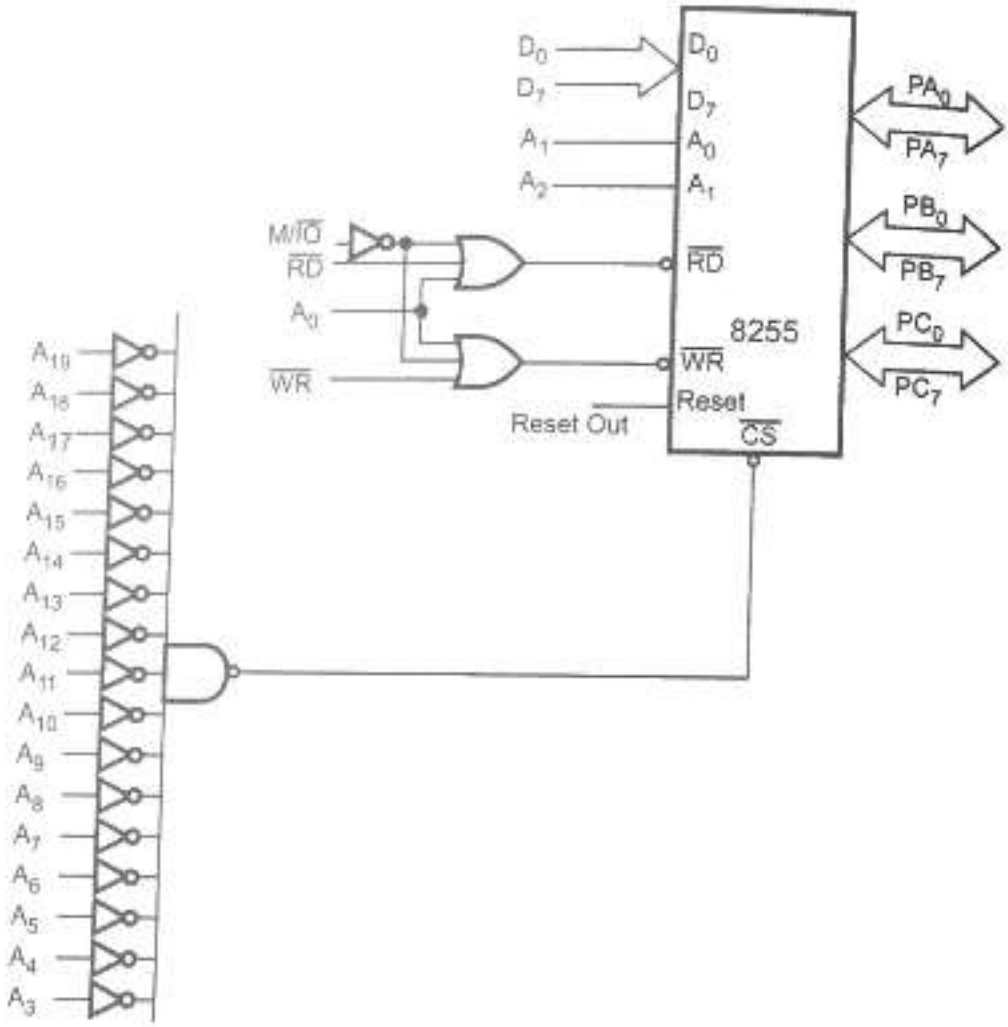


Fig. 3.4.20 Memory mapped I/O

I/O map :

Register	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Port A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000H
Port B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	00002H
Port C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	00004H
Control register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	00006H

Example 3.4.8 *Interface 8255 PPI with 8086 microprocessor in maximum mode. Draw interfacing diagram and mention address map for 8255.*

Solution : Address map for 8255

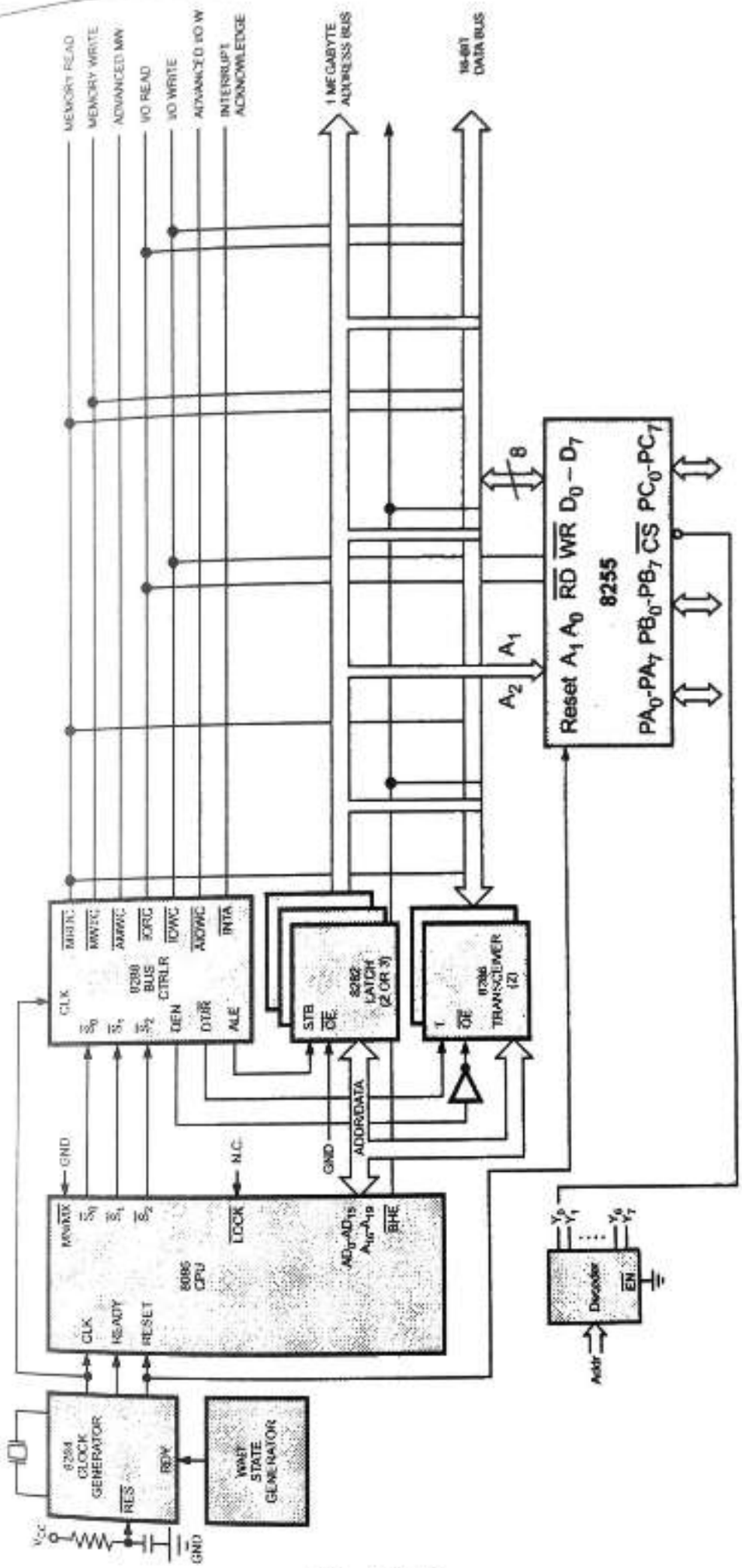

MEMORY READ
MEMORY WRITE
ADVANCED MW
I/O READ
I/O WRITE
ADVANCED I/O W
INTERRUPT ACKNOWLEDGE
1 MEGABYTE ADDRESS BUS
16-BIT DATA BUS
8
Reset A_1 A_0 $\overline{RD}$ $\overline{WR}$ $D_0 - D_7$
8255
PA_0-PA_7 PB_0-PB_7 $\overline{CS}$ PC_0-PC_7
A_1
A_2
8288 BUS CTRLR
8282 LATCH (2 OR 3)
8286 TRANSCEIVER (2)
ADDR/DATA
GND
N.C.
8086 CPU
LOCK
CLK
READY
RESET
8284 CLOCK GENERATOR
WAIT STATE GENERATOR
Decoder
EN
Addr
Vcc
GND

Fig. 3.4.21

Port/Register	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Port A	0	0	0	0	0	0	0	0	00H
Port B	0	0	0	0	0	0	1	0	02H
Port C	0	0	0	0	0	1	0	0	04H
Control register	0	0	0	0	0	1	1	0	06H

Review Questions

1. *What are the features of 8255 ?*
2. *Explain in detail about the parallel communication interface.*
3. *Explain important signals of 8255.*
4. *Discuss the organization and architecture of 8255 programmable peripheral interface IC with a functional block diagram.*
5. *Draw and explain the functional diagram of parallel communication interfacing chip.*
6. *Explain the I/O mode operation of 8255 in detail.*
7. *With a neat diagram discuss the various modes of operation of 8255.*
8. *What are the basic modes of operations in parallel communication interfaces ?*
9. *What are the operating modes of 8255 ?*
10. *Draw the control word format of 8255.*
11. *Discuss the different modes of operation of 8255 parallel communication interface with suitable diagrams.*
12. *Describe mode 1 input and mode 1 output operation of 8255 with timing diagrams.*
13. *Explain the mode 0 operation of 8255 programmable peripheral interface.*
14. *Explain the parallel communication interface with microprocessor.*
15. *Interface 8255 (PPI) with 8086 microprocessor in minimum mode. Draw interfacing diagram and mention address map for 8255.*
16. *Draw and explain in brief 8255A system connection.*
17. *Draw the interfacing scheme of 8255 and 8086 in I/O mapped I/O mode.*
18. *Draw the interfacing scheme of 8255 and 8086 in memory mapped I/O mode.*

3.5 Digital to Analog Interface

- Fig. 3.5.1 (See Fig. 3.5.1 on next page) shows the interfacing of DAC 0808 with microprocessor 8086. Here, programmable peripheral interface, 8255 is used as parallel port to send the digital data to DAC.

I/O Map for 8255

Port / Register	Address
Port A	0 0
Port B	0 2
Port C	0 4
Control Register	0 6

Program :

```
MOV AL,80 H      ; Initialization control word for 8255 to
OUT 06, AL       ; Configure all ports as output ports
MOV AL, data     ; Load 8-bit data to be sent at the input of 0808 DAC
OUT 00, AL       ; Send data on port A.
```

We now see how different waveforms can be generated using this circuit.

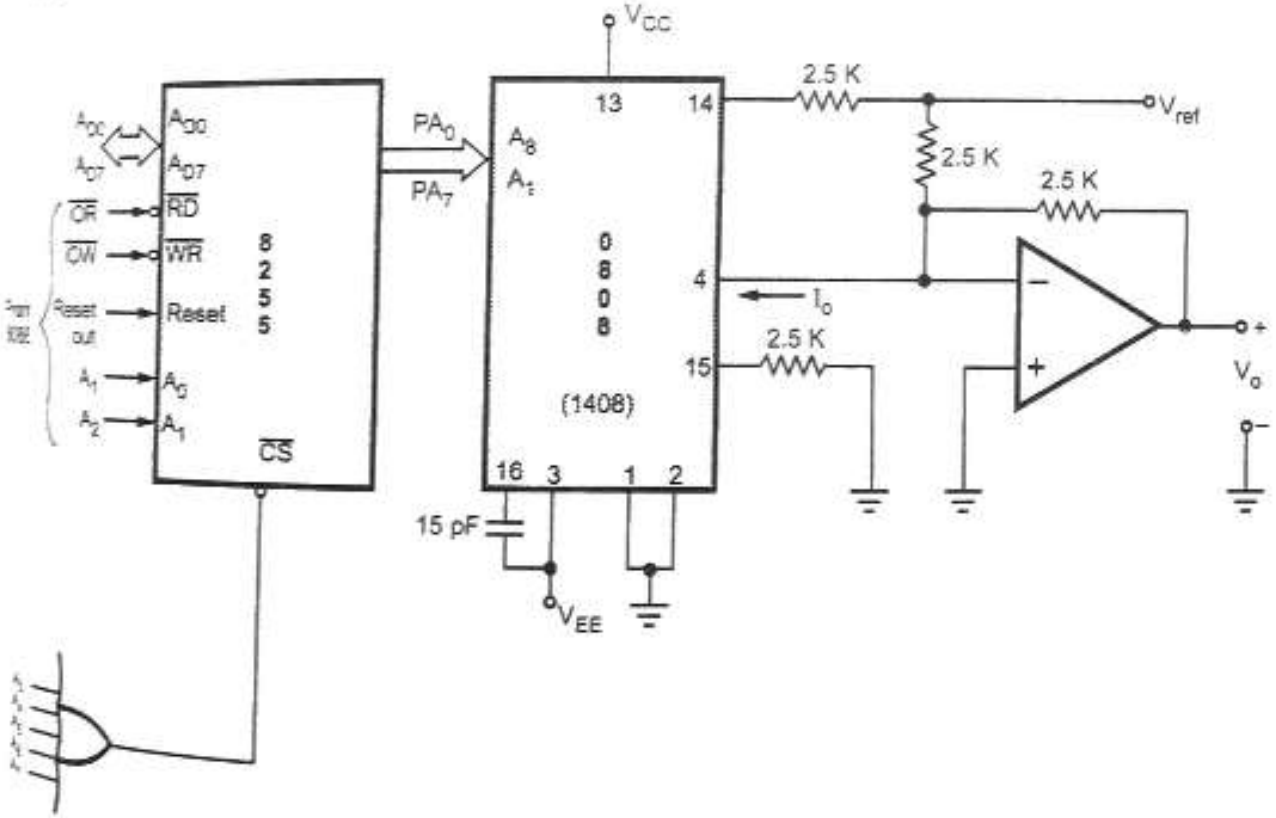


Fig. 3.5.1 Interfacing of 0808 with microprocessor

Example 3.5.1 *Generate square wave using DAC 0808.*

Solution : To generate square wave first we have to output FF and then 00 on Port A of 8255. The output of 8255 (Port A) connected to DAC 0808. According to frequency requirement delay is provided in between the two outputs.

Program

```
        MOV AL,80H      ; Initialise 8255 to configure all
        OUT 06, AL      ; ports as output ports
LOOP :  MOV AL, 00      ; Load A with digital data corresponds
                        ; to – 2.5 V output
        OUT 00, AL      ; Send digital data to the input of  DAC 0808
        CALL DELAY      ; Wait for specified time.
        MOV AL, FFH     ; Load A with digital data corresponds
                        ; to + 2.5 V output
        OUT 00, AL      ; Send digital data to the input of DAC 0808.
        CALL DELAY      ; Wait for specified time
        JMP LOOP        ; Repeat
```

Delay Program

```
        MOV BL, 08      ; Load delay count in register
BACK :  DEC B           ; Decrement count
        JNZ BACK        ; Check if count = 0
                        ; otherwise repeat
        RET
```

Example 3.5.2 *Generate triangular wave using DAC 0808.*

Solution : To generate triangular wave we have to output data from 00 initially, and it should be incremented upto FF. When it reaches FF it should be decremented upto 00.

```
        MOV AL, 80H ; Initialise 8255 to configure all ports as output ports
        OUT 06, AL
        MOV AL, 00  ; Load accumulator with digital data
                    ; corresponds to – 2.5 V output
        OUT 00, AL  ; Send digital to the input of DAC 0808.
LOOP_1: INC AL      ; Increment digital data in the accumulator
        OUT 00, AL  ; Send digital data to the input of DAC 0808.
        CMP AL, FFH ; Check digital data for Peak output
        JNZ LOOP_1  ; If no repeat,
LOOP_2: DEC AL      ; Decrement digital data in the accumulator
        OUT 00, AL  ; Send digital data to the input of DAC 0808.
        JNZ LOOP_2
        JMP LOOP_1
```

Example 3.5.3 *Write a program to generate triangular waveform of period 10 ms. The CPU runs at 5 MHz clock frequency.*

Solution : The Fig. 3.5.2 shows waveform to be generated. As shown in the Fig. 3.5.2, the time period for half cycle is 5 ms. In half cycle we execute loop 256 times. Thus execution time for loop will be 5 ms/256

= 19.53 μs

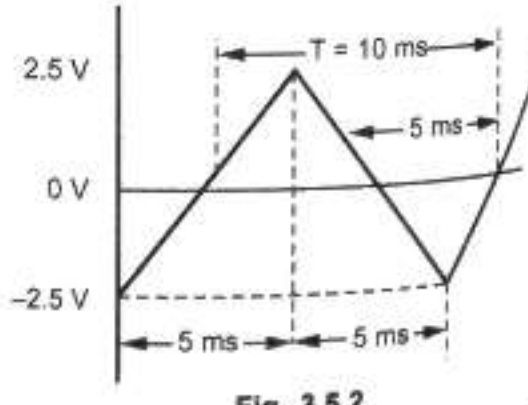


Fig. 3.5.2

Interface of Devices

The loop will be

LOOP 2 :	DEC AL	3	Clock cycles
	OUT 00, AL	10	Clock cycles
	JNZ LOOP 2	16	Clock cycles

Time for 1 cycle $= \frac{1}{5 \times 10^6} = 0.2\ \mu s$

Time for loop $= (3 + 10 + 16) \times 0.2 = 5.8\ \mu s$

Thus the delay time required in the inner loop will be

$19.53\ \mu s - 5.8\ \mu s = 13.73\ \mu s = 69$ Clock cycles

	MOV CL, 04	→	4	=	4
BACK :	DEC CL	→	3+3+3+3	=	12
	JNZ BACK	→	16+16+16+4	=	52
					69

The above three instructions in the loop will take 69 clock cycles, i.e. 13.73 µs to execute. Thus the final routine to generate desired waveform is as follows

```
        MOV   AL,   80H    ; Initialize 8255 to configure all
        OUT   06,   AL     ; Ports as output ports
        MOV   AL,   00     ; Load accumulator with digital data
        OUT   00,   AL     ; Corresponds to –2.5 V output
LOOP_1: INC   AL           ; Increment digital data in the accumulator
        OUT   00,   AL     ; Send digital data to the input of DAC 0808
BACK 1: MOV   CL,   04     ; Delay to consume 69 T cycles
        DEC   CL           ;
        JNZ   BACK1;
        CMP   AL,   FFH    ; Check digital data for peak output
        JNZ   LOOP_1       ; If no repeat
LOOP_2: MOV   CL,   04     ; Delay to consume 69 T cycles
BACK2 : DEC   CL
        JNZ   BACK2
        DEC   AL           ; Decrement digital data in the accumulator
        OUT   00,   AL     ; Send digital data to the input of DAC 0808
        JNZ   LOOP_2
        JMP   LOOP_1
```

Example 3.5.4 *Interface a typical 12-bit DAC with 8255 and write program to generate triangular waveform of period 10 ms. The CPU runs at 5 MHz clock frequency.*

Solution : In triangular waveform period for 1 ramp = 10 ms/2 = 5 ms and minimum time required to transfer data is about 40 µs.

Therefore, steps in one ramp $= \frac{5\,ms}{8\mu s} = 625$. So to reach from 0 – 4096 we have 625 steps

i.e. we have increment the digital count by 4096/625 = 6.55 ≈ 7

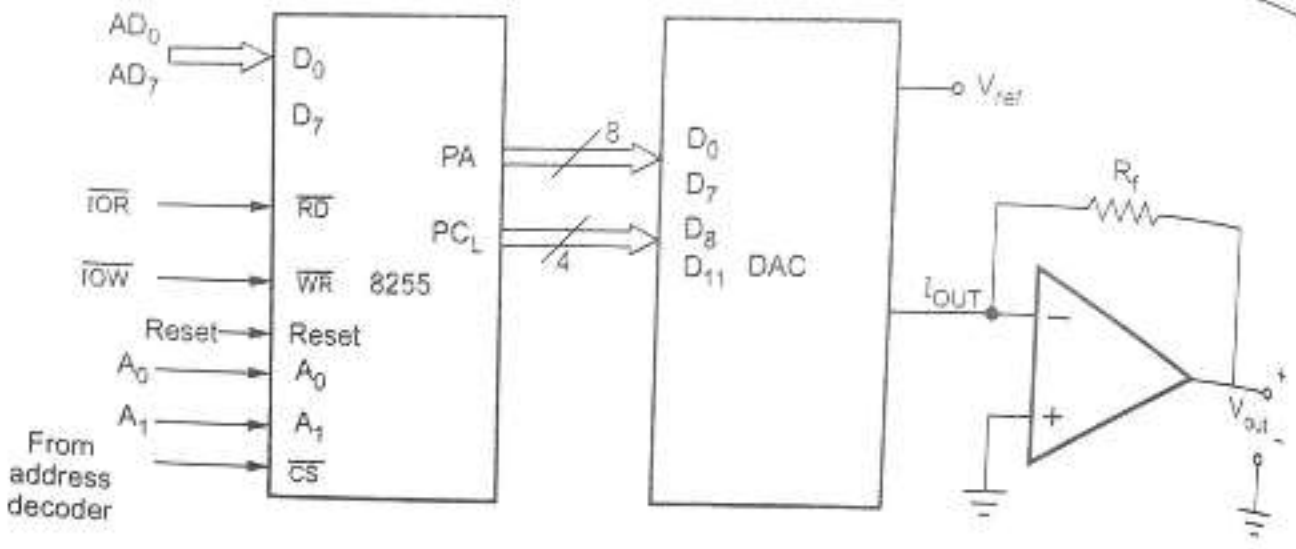


Fig. 3.5.3

With increment of 7 we get 585 steps and for each data transfer we get time $= \frac{5\,\text{ms}}{585} = 8.547\ \mu\text{s}$.

The clock frequency of 8086 is given 5 MHz,

$\therefore$ Time for one clock cycle $= \frac{1}{5\,\text{MHz}} = 0.2\ \mu\text{s}$

Therefore, clock cycles in one loop $= \frac{8.547\mu\text{s}}{0.2\mu\text{s}} = 43$

Program :

```
START :   MOV CX, 0FFFH
          MOV BX, 0000H
BACK :    MOV AL, BL      ...   Clock cycles
          OUT PA, AL            2
          MOV AL, BH            10
          OUT PC, AL            2
          INC BX                10
          NOP                   2
          INC DX                3     Delay Adjustment
          DEC CX                2     Delay Adjustment
          JNZ BACK              2
                                10
                                ---
                                43
          MOV CX, 0FFFH
BACK1 :   DEC BX
          MOV AL, BL
          OUT PA, AL
          MOV AL, BH
          OUT PC, AL
          NOP
```

```
INC DX
DEC CX
JNZ BACK1
JMP START
```

Example 3.5.5 *Generate sine wave using DAC 0808.*

Solution : To generate sine wave we have to output digital equivalent values which will represent sine wave as shown in the Fig. 3.5.4. Digital data 00H represents – 2.5 V, 7FH represents 0 V and FFH represents + 2.5 V. The digital equivalent for sine wave can be calculated as follows.

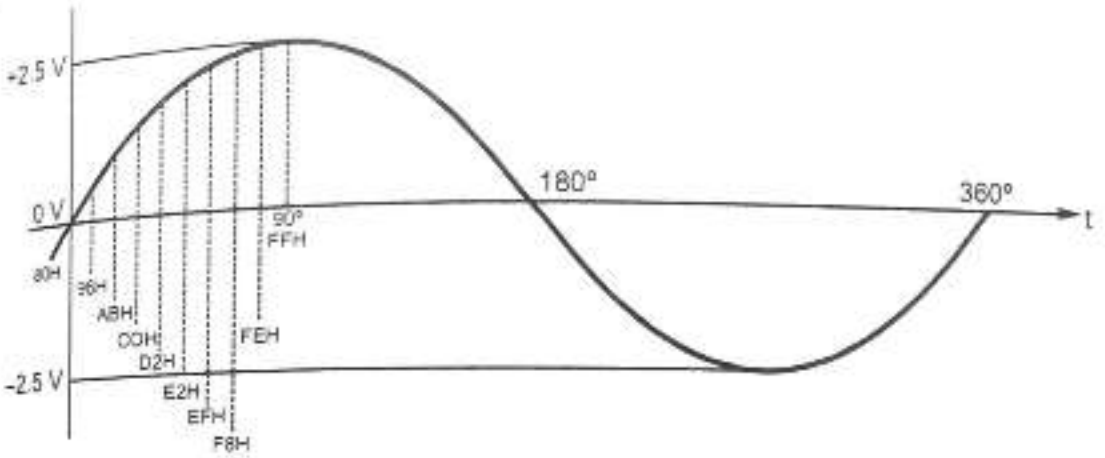


Fig. 3.5.4

We know that sin 0° = 0 and sin 90° = 1. The range sin 0° to sin 90° is distributed over digital range of 7FH to FFH i.e. (FFH – 7FH) 128 decimal steps. Therefore, taking 80H as a offset we can write,

Digital Equivalent Value (DEV) for sin x = (128 + 128 × sin x)

Where x is a angle in degrees and digital value is in decimal.

Lookup table shows the digital equivalent values for sine wave.

Degrees	Equation	Digital equivalent in decimal	Digital equivalent in hex
0°	(128 + 128 × sin 0)	128	80H
10°	(128 + 128 × sin 10)	150	96H
20°	(128 + 128 × sin 20)	171	ABH
30°	(128 + 128 × sin 30)	192	C0H
40°	(128 + 128 × sin 40)	156	D2H
50°	(128 + 128 × sin 50)	226	E2H
60°	(128 + 128 × sin 60)	239	EFH
70°	(128 + 128 × sin 70)	248	F8H
80°	(128 + 128 × sin 80)	254	FEH

90 °	(128 + 128 × sin 90)	256 → 255	FFH
100 °	(128 + 128 × sin 100)	254	FEH
110 °	(128 + 128 × sin 110)	248	F8H
120 °	(128 + 128 × sin 120)	239	EFH
130 °	(128 + 128 × sin 130)	226	E2H
140 °	(128 + 128 × sin 140)	156	D2H
150 °	(128 + 128 × sin 150)	192	C0H
160 °	(128 + 128 × sin 160)	171	ABH
170 °	(128 + 128 × sin 170)	150	96H
180 °	(128 + 128 × sin 180)	128	80H
190 °	(128 + 128 × sin 190)	106	6AH
200 °	(128 + 128 × sin 200)	84	54H
210 °	(128 + 128 × sin 210)	64	40H
220 °	(128 + 128 × sin 220)	46	2EH
230 °	(128 + 128 × sin 230)	30	1EH
240 °	(128 + 128 × sin 240)	17	11H
250 °	(128 + 128 × sin 250)	08	08H
260 °	(128 + 128 × sin 260)	02	02H
270 °	(128 + 128 × sin 270)	00	00H
280 °	(128 + 128 × sin 280)	02	02H
290 °	(128 + 128 × sin 290)	08	08H
300 °	(128 + 128 × sin 300)	17	11H
310 °	(128 + 128 × sin 310)	30	1EH
320 °	(128 + 128 × sin 320)	46	2EH
330 °	(128 + 128 × sin 330)	64	40H
340 °	(128 + 128 × sin 340)	84	54H
350 °	(128 + 128 × sin 350)	106	6AH
360 °	(128 + 128 × sin 360)	128	80H

Program :

```
        MOV AL,      80H        ; Initialize 8255 to configure all
        OUT 06H,     AL         ; Ports as output ports
START:  MOV CL,      25H        ; Initialize counter
        MOV BX, OFFSET TABLE    ; Initialize  pointer to point to lookup table
BACK:   MOV AL, [BX]            ; Get the digital equivalent data
                                ; from lookup table
        OUT 00,      AL         ; Send digital data to DAC
        INC BX                  ; Increment lookup table pointer
        DEC CL                  ; Decrement counter
        JNZ BACK                ; If not zero, go to BACK
        JMP START               ; Repeat
TABLE DB  80, 96H, ABH ..... 54H, 6AH, 80H
```

Review Questions

1. *Interface a typical 8-bit DAC with 8255.*
2. *Draw and explain the typical interfacing circuit for DAC 0808.*
3. *Draw and explain the typical interfacing circuit for DAC 0808 in the bipolar range*
4. *Explain the procedure of interfacing D/A and A/D converter circuit.*

3.6 Analog to Digital Interface

- The ADC 0808 and ADC 0809 are monolithic CMOS devices with an 8-channel multiplexer. These devices are also designed to operate from common microprocessor control buses, with tri-state output latches driving the data bus.

3.6.1 Features

- 8-bit successive approximation ADC.
- 8-channel multiplexer with address logic.
- Conversion time 100 μs.
- It eliminates the need for external zero and full-scale adjustments.
- Easy to interface to all microprocessors.
- It operates on single 5 V power supply.
- Output meet TTL voltage level specifications.

3.6.2 Pin Diagram

- Fig. 3.6.1 shows pin diagram of 0808/0809 ADC.

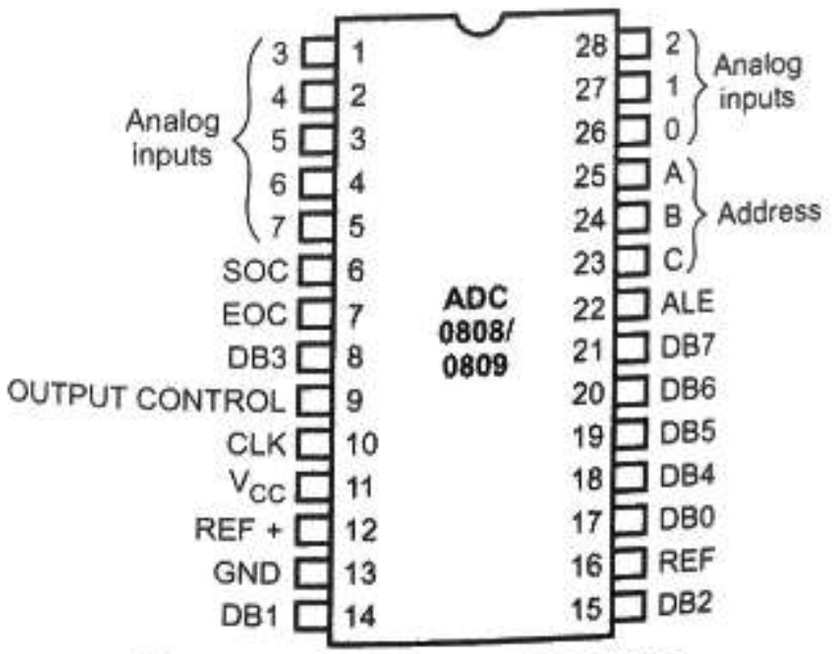


Fig. 3.6.1 Pin diagram of 0808/0809

3.6.3 Operation

- ADC 0808/0809 has eight input channels, so to select desired input channel, it is necessary to send 3-bit address on A, B and C inputs.
- The address of the desired channel is sent to the multiplexer address input through port pins. After at least 50 ns, this address must be latched. This can be achieved by sending ALE signal.
- After another 2.5 µs, the Start Of Conversion (SOC) signal must be sent high and then low to start the conversion process.
- To indicate end of conversion ADC 0808/0809 activates EOC signal.
- The microprocessor system can read converted digital word through data bus by enabling the output enable signal after EOC is activated. This is illustrated in Fig. 3.6.2.

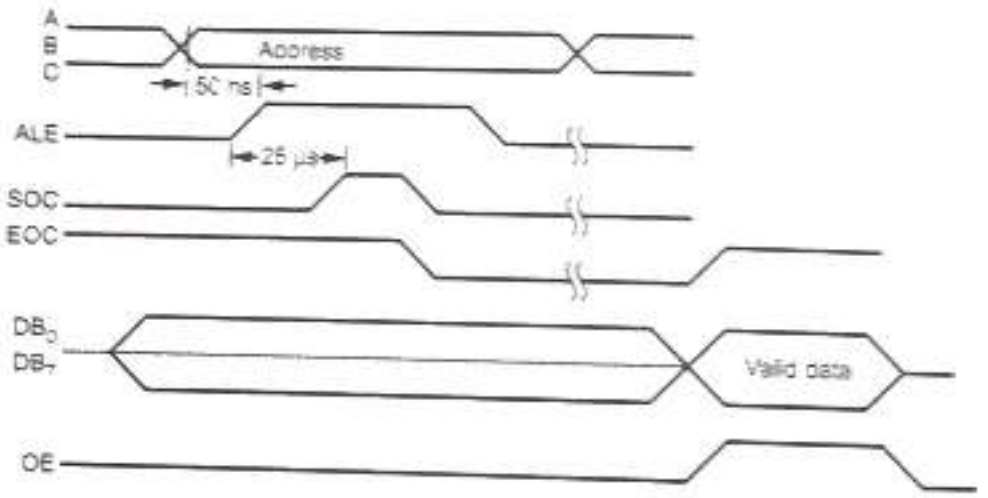


Fig. 3.6.2 Timing waveforms for ADC 0808

3.6.4 Interfacing

- Fig. 3.6.3 shows typical interfacing circuit for ADC 0808 with microprocessor system.
- The zener diode and LM 308 amplifier circuitry is used to produce a V_{CC} as $+V_{REF}$ of 5.12 V for the A/D converter. With this reference voltage the A/D converter will have 256 steps of 20 mV each.

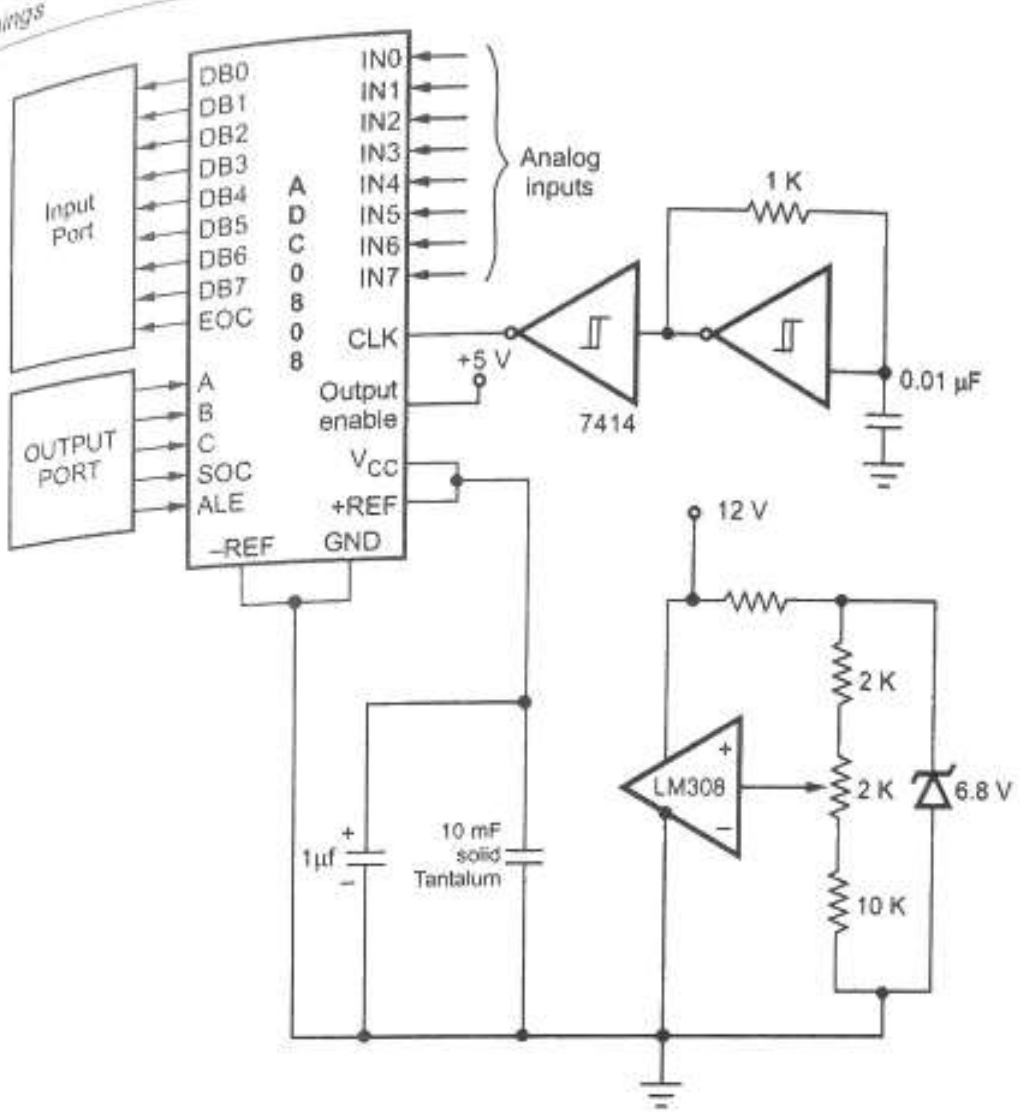


Fig. 3.6.3 Typical interface for 0808/0809

Example 3.6.1 *An 8-bit digital ramp ADC with a 40 mV resolution uses a clock frequency of 2.5 MHz a comparator with V_T = 1 mV. Determine the following values :*

i) The digital output for V_{in} = 6.000 V.

ii) The digital output for V_{in} = 6.035 V.

iii) The maximum and average conversion times for this ADC.

Solution : The digital output is given by,

$$\text{Count} = \frac{V_{in}}{\text{Resolution}}$$

i)

$$V_{in} = 6.000 \text{ V}$$

$$\text{Count} = \frac{6.000}{40 \times 10^{-3}} = 150$$

ii)

$$V_{in} = 6.035 \text{ V}$$

$$\therefore \quad \text{Count} = \frac{6.035}{40\times10^{-3}} \cong 150$$

iii) $(T_C)_{max}$ = (Maximum count) × T

$$= (2^n - 1) \times \frac{1}{f} = \frac{(255)}{2.5\times10^6}$$

$$= 102\ \mu s$$

$$(T_C)_{average} = \frac{(T_C)_{max}}{2}$$

$$= 51\ \mu s$$

Example 3.6.2 *Use successive approximation type ADC to convert the analog voltage 7.28 V to 8-bit binary. If V_R = 10 V, determine the final binary answer and the error present. Show the timing waveforms that would occur in the successive approximation ADC process.*

Note : V_R = Full scale input voltage to the DAC.

Solution : The reference voltage V_R is full scale input voltage to the DAC.

Hence the 10 V reference corresponds to 2^n levels i.e. 2^8 = 256 levels.

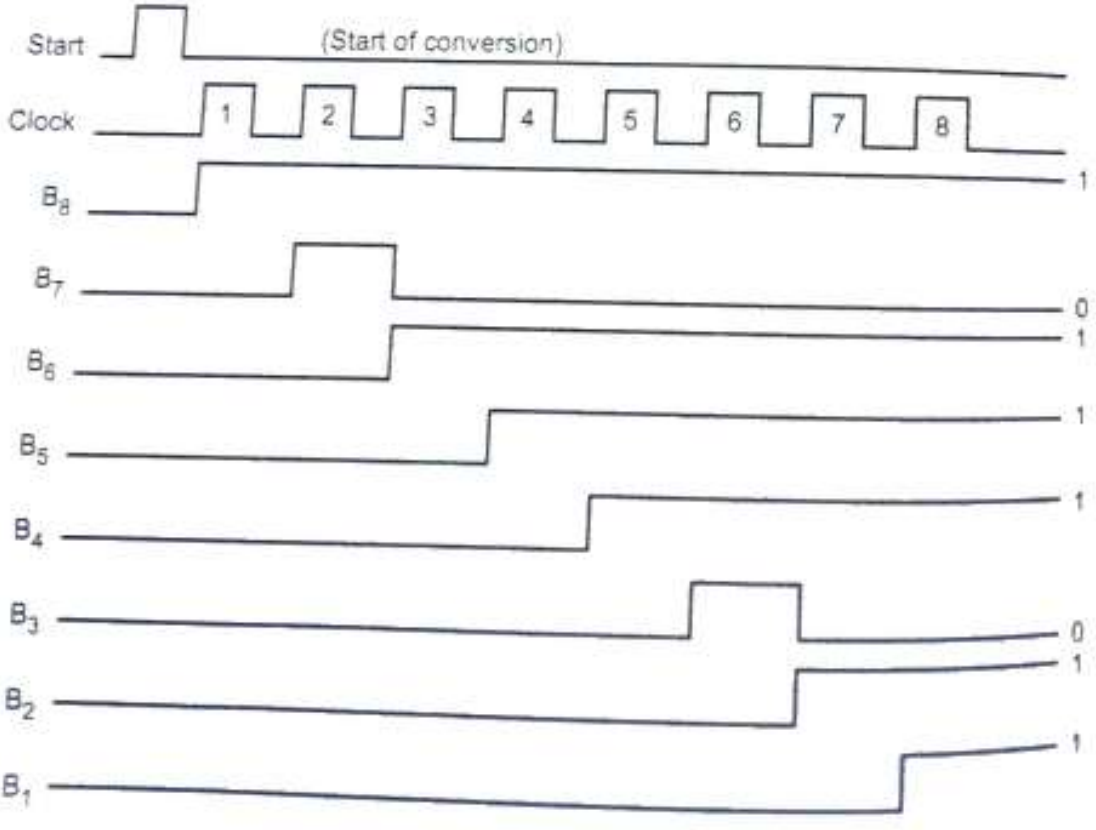


Fig. 3.6.4

Hence one level corresponds to,

$$= \frac{10}{256} = 0.03906\ V$$

This is nothing but the resolution of the ADC.

The timing waveforms are shown in the Fig. 3.6.4

Hence the final binary number is,

$$= (1\ 0\ 1\ 1\ 1\ 0\ 1\ 1)_2$$

Hence the total output count is,

$$= (1 \times 128) + (0 \times 64) + (1 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)$$

$$= 187$$

$$\therefore \text{Actual output} = \text{Count} \times \text{Resolution} = 187 \times 0.03906$$

$$= 7.3046 \text{ V}$$

$$= 7.28 \text{ V}$$

$$\text{Exact value} = 7.28 \text{ V}$$

$$\therefore \text{Error} = 7.3046 - 7.28$$

$$= 0.024 \text{ V}$$

Review Questions

1. *List the features of ADC 0808.*
2. *Explain the interfacing of ADC 0808 to the microprocessor.*
3. *Show a typical 8-bit ADC interface with 8086. Explain functionality of each signal used.*
4. *Explain the procedure of interfacing D/A and A/D converter circuit.*

3.7 Stepper Motor Interfacing

- A stepper motor is a digital motor. It can be driven by digital signal.
- Fig. 3.7.1 shows the typical 2 phase motor interfaced using 8255. Motor shown in the circuit has two phases, with center-tap winding. The center taps of these windings are connected to the 12 V supply. Due to this, motor can be excited by grounding four terminals of the two windings.
- Motor can be rotated in steps by giving proper excitation sequence to these windings. The PA0 through PA3 pins of 8255 are used to generate excitation signals in the proper sequence.

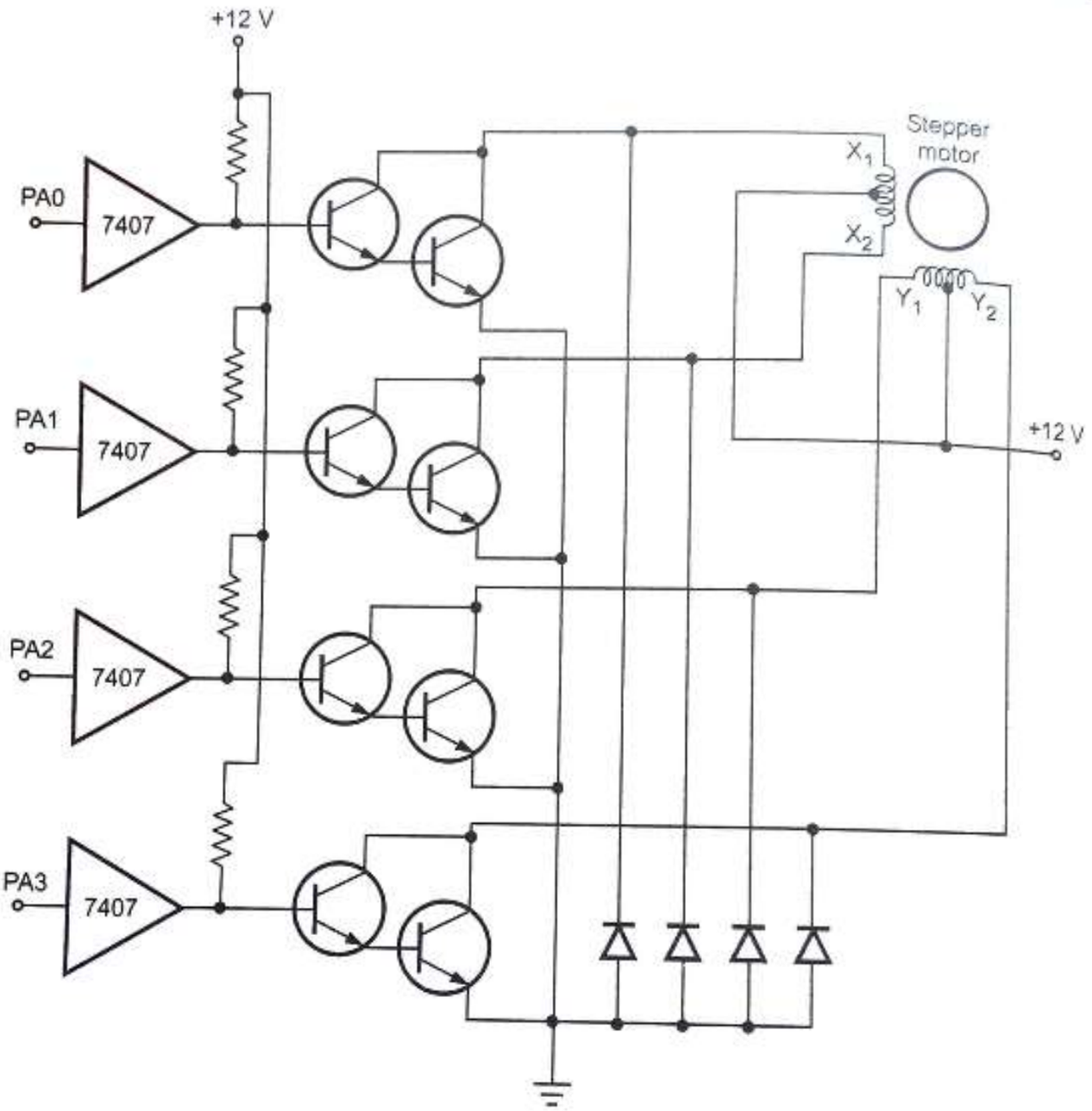


Fig. 3.7.1 Stepper motor interface

- Table 3.7.1 shows typical excitation sequence. The given excitation sequence rotates the motor in clockwise direction. To rotate motor in anticlockwise direction we have to excite motor in a reverse sequence.
- The excitation sequence for stepper motor may change due to change in winding connections. However, it is not desirable to excite both the ends of the same winding simultaneously. This cancels the flux and motor winding may damage. To avoid this, digital locking system must be designed.

Step	X_1	X_2	Y_1	Y_2
1	0	1	0	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0
1	0	1	0	1

Table 3.7.1 Full step exciation sequence

- Fig. 3.7.2 shows a simple digital locking system. Only one output is activated (made low) when properly excited ; otherwise output is disabled (made high).

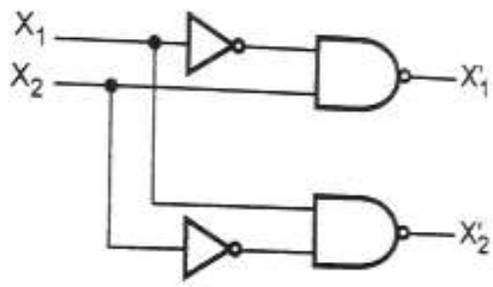


Fig. 3.7.2 Digital locking system

- The excitation sequence given in Table 3.7.1 is called **full step sequence.** In which excitation ends of the phase are changed in one step and motor is rotated by 1.8°. The excitation sequence given in Table 3.7.2 takes two steps to change the excitation ends of the phase. Such a sequence is called **half step sequence** and in each step the motor is rotated by 0.9°.

Step	X_1	X_2	Y_1	Y_2
1	0	1	0	1
2	0	0	0	1
3	1	0	0	1
4	1	0	0	0
5	1	0	1	0
6	0	0	1	0
7	0	1	1	0
8	0	1	0	0
1	0	1	0	1

Table 3.7.2 Half step excitation sequence

Review Questions

1. *Draw and explain the schematic for interfacing a stepper motor using 8255.*
2. *What is full step and half step sequence ?*