GOVERNMENT ARTS AND SCIENCE COLLEGE
Affiliated to Manonmaniam Sundaranar University,Tirunelveli)
PALKULAM, KANYAKUMARI-629 401.

STUDY MATERIAL FOR BCA

MICRO PROCESSOR

IV - SEMESTER

ACADEMIC YEAR 2022 - 2023

PREPARED BY

DEPARTMENT OF COMPUTER SCIENCE

STUDY MATERIAL FOR B.C.A
MICRO PROCESSORS
IV- SEMESTER, ACADEMIC YEAR 2022-2023

UNIT- I

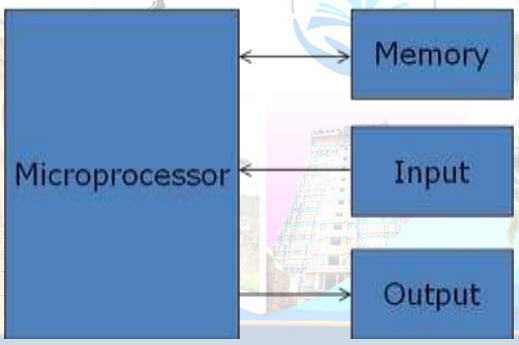
MICROPROCESSORS, MICROCOMPUTER AND ASSEMBLY LANGUAGE

Microprocessor

A Microprocessor is a multipurpose programmable, clock driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input, processes data according to those instructions and provide result s as output.

The microprocessor operates in binary 0 and 1 known as bits are represented in terms of electrical voltages in the machine that means 0 represents low voltage level and 1 represents high voltage level.

Each microprocessor recognizes and processes a group of bits called the word and microprocessors are classified according to their word length such as 8 bits microprocessor with 8 bit word and 32 bit microprocessor with 32 bit word etc.



The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a **result.**

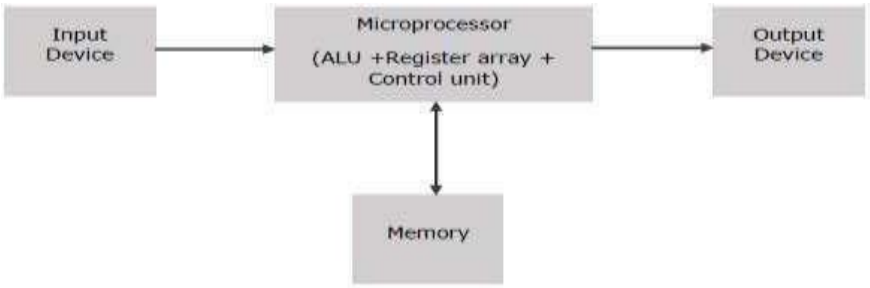
Terms used

- **CPU:** - Central processing unit which consists of ALU and control unit.
- **Microprocessor: -** Single chip containing all units of CPU.
- **Microcomputer: -** Computer having microprocessor as CPU.
- **Microcontroller:** single chip consisting of MPU, memory, I/O andinterfacing circuits.
- **MPU: -** Microprocessing unit – complete processing unit with thenecessary control signals.

A Microprocessor as a Programmable device:

The microprocessor can perform different sets of operations on the data it receives

depending on the sequence of instructions supplied in the given program. By changing the program, the microprocessor manipulates the data in different ways.

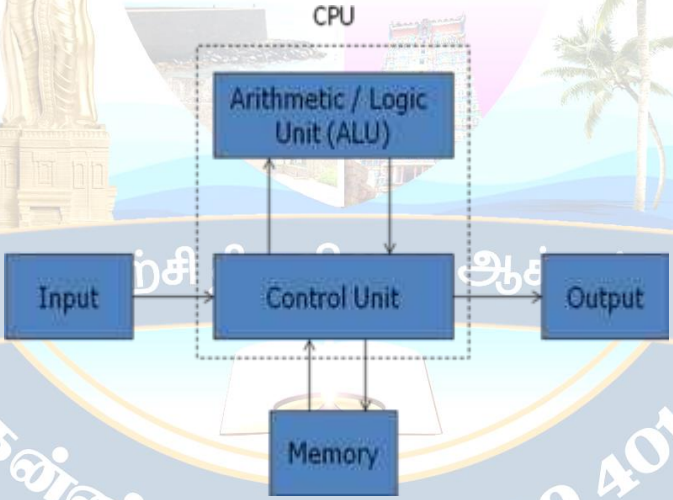


Instructions:

Each microprocessor is designed to execute a specific group of operations. This group of operations is called an instruction set. This instruction set defines what the microprocessor canand cannot.

A Microprocessor as a CPU:

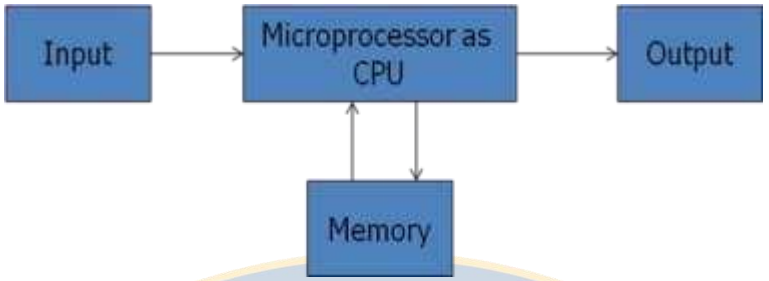
Traditionally, the computer is represented with four components such as memory, input, output and central processing unit (CPU) which consists of arithmetic logic unit (ALU) and control unit (CU).



The CPU contains various registers to store data, the ALU to perform arithmetic and logical operations, instruction decoders, counters and control lines.

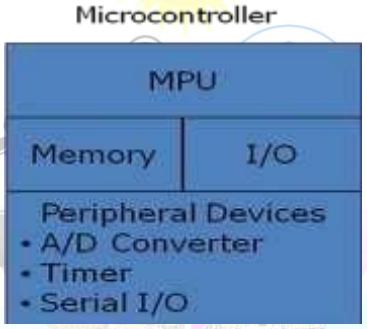
The CPU reads instructions from memory and performs the tasks specified. It communicates with input/output (I/O) devices either to accept or to send data, the I/O devices is known as peripherals.

Later on around late 1960's, traditional block diagram can be replaced with computer having microprocessor as CPU which is known as microcomputer. Here CPU was designed using integrated circuit technology (IC's) which provided the possibility to build the CPU on a single chip.



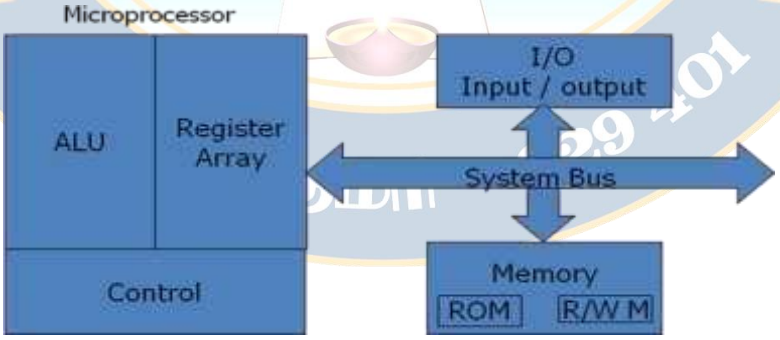
Block Diagram of a computer with the Microprocessor as CPU

Later on semiconductor fabrication technology became more advanced, manufacturers were able to place not only MPU but also memory and I/O interfacing circuits on a single chip known as microcontroller, which also includes additional devices such as A/D converter, serial I/O, timer etc.



Block Diagram of a MicrocontrollerOrganization of a microprocessor based system

Microprocessor based system includes there components microprocessor, input/output and memory (read only and read/write). These components are organized around a common communication path called a bus.



Microprocessor Based System with Bus Architecture

Microprocessor:

It is clock driven semiconductor device consisting of electronic logic circuits manufacturedby using either a large scale integration (LSI) or very large scale integration (VLSI) technique. It is capable of performing various computing functions and making decisions to change the sequence of program execution. It can be divided in to three segments.

* Arithmetic/Logic unit: It performs arithmetic operations as addition and subtraction and logic operations as AND, OR & XOR.
* Register Array: The registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through instruction. The registers can be identified by letters such as B, C, D, E, H and L.
* Control Unit: It provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor and memory & peripherals.

Memory:

Memory stores binary information such as instructions and data, and provides that information to the up whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU. Results are either transferred to the output section for display or stored in memory for later use. Memory has two sections.

A. Read only Memory (ROM): Used to store programs that do not need alterations and can only read.
B. Read/Write Memory (RAM): Also known as user memory which is used to store user programs and data. The information stored in this memory can be easily read and altered.

Input/Output:

* It communicates with the outside world using two devices input and output which are also Known as peripherals.
* The input device such as keyboard, switches, and analog to digital converter transfer binary information from outside world to the microprocessor.
* The output devices transfer data from the microprocessor to the outside world. They include the devices such as LED, CRT, digital to analog converter, printer etc.

System Bus:

It is a communication path between the microprocessor and peripherals; it is nothing but a group of wires to carry bits.

Microprocessor instruction set and computer languages

- Microprocessors recognize and operate in binary numbers.
- Each microprocessor has its own binary words, meanings, and language.
- The words are formed by combining a number of bits for a given machine.
  - The word (or word length) is defined as the number of bits the microprocessor recognizes and processes at a time.
  - The word length ranges from 4-bit to 64-bit.
- Another term commonly used to express word length is byte.
  - A byte is defined as a group of eight bits.
  - For example, a 16-bit microprocessor has a word length to two bytes.
- The term nibble stands for a group of four bits.
  - A byte has two nibbles.

8085 Machine Language

- Each machine has its own set of instructions based on the design of its CPU or of its microprocessor.
- To communicate with the computer, one must give
- The complete set of 8085 mnemonics is called the 8085 assembly language.
- A program written in these mnemonics is called an assembly language program.
- Machine language and assembly language are microprocessor-specific and are both considered low level languages.
- The machine language is in binary, and the assembly language is in English-like words; however, the microprocessor understands only the binary.

8085 Assembly languages

- Even though the instructions can be written in hexadecimal code, it is still difficult to understand a program written in hexadecimal numbers.
  - Therefore, each manufacturer of a MPU has devised a symbolic code for each instruction, called a mnemonic.
  - The mnemonic for a particular instruction consists of letters that suggest the operation to be performed by that instruction.
  - For example, 0011 1100_2 ($3C_h$) is represented by the mnemonic INR A.
- The mnemonics can be written by hand on paper and translated manually in hexadecimal code, called hand assembly.
- Or the mnemonics can be written on a computer using a program called an Editor in the ASCII code and translated into binary code by using the program called an assembler.

ASCII Code

ASCII—American Standard Code for Information Interchange. This is a 7-bit alphanumeric code with 128 combinations. Each combination is assigned to either a letter, decimal digit, a symbol, or a machine command.

Writing and Executing an Assembly Language Program

- To manually write and execute an assembly languag program on a single-board computer, with a Hex keyboard for input and LEDs for output, the following steps are necessary:
  - Write the instructions in mnemonics obtained from the instruction set supplied by the manufacturer.
  - Find the hexadecimal machine code for each instruction by searching through the set of instructions.
  - Enter (load) the program in the user memory in a sequential order by using the Hex keyboard as the input device.
  - Execute the program by pressing the Execute key. The answer will be displayed by the LEDs.

Assembler

- The hand assembly:
  - tedious and subject to errors;
  - suited for small programs.
- Alternative, use assembler:
  - The assembler is a program that translates the mnemonics entered by the ASCII keyboard into the corresponding binary machine codes of the microprocessor.
  - Each microprocessor has its own assembler because the mnemonics and machine codes are specific to the microprocessor being used, and each assembler has rules that must be followed by the programmer.

High-Level Languages

- Programming languages that are intended to be machine-independent are called high-level languages.
- These include such languages as BASIC, PASCAL, C, C++ and Java, all of which have certain sets of rules and draw on symbols and conventions from English.
- Instructions written in these languages are known as statements rather than mnemonics.

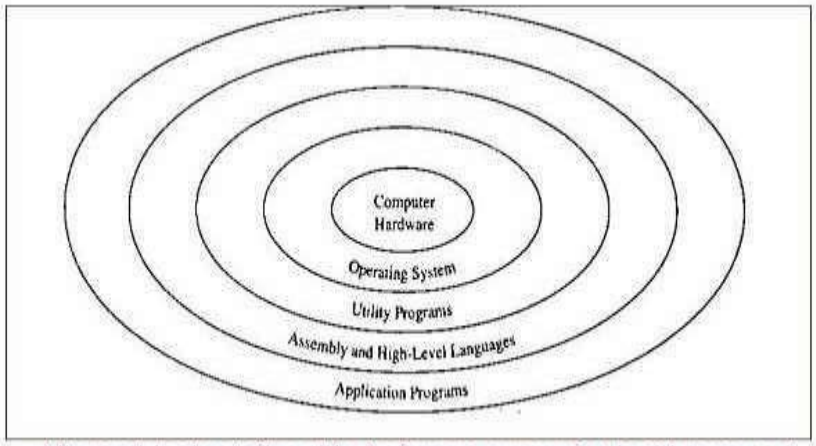
Operating Systems

- How are words in English converted into the binary languages of different microprocessors?
  - Through another program called either a compiler or an interpreter.
  - These programs accept English-like statements as their input, called the source code.
  - The compiler or interpreter then translates the source code into the machine language compatible (object code) with the microprocessor being used in the system.
  - Each microprocessor needs its own compiler or an interpreter for each high-level language.

- Compiler - a program that translates English-like words of a high-level language into the machine language of a computer.
  - A compiler reads a given program, called a source code, in its entirety and then translates the program into the machine language, which is called an object code.
- Interpreter - a program that translates the English-like statements of a high-level language into the machine language of a computer.
  - An interpreter translates one statement at a time from a source code to an object code.
- Assembler - a computer program that translates an assembly language program from mnemonics to the binary machine code of a computer.

- Operating system - a set of programs that manages interaction between hardware and software.
  - Responsible primarily for storing information on disks and for communication between microprocessor, memory, and peripherals.

From large computers to single chip microcontrollers:



Hierarchical relationship between computer hardware and software.

53

Different types of computers are designed to serve different purposes. Some are suitable for scientific calculations, while others are used simply for tuning appliances on and off.Until the 1970s computers were broadly classified in three categories: mainframe, mini, and microcomputers.

Large Computers

These are large, general purpose, multiuser, multitasking, computers designed to perform such data processing tasks as complex scientific and engineering calculations and handling of records for large corporations or government agencies. These computers can be broadly classified into mainframes and supercomputers, and mainframes are further classified according to their sizes. Typical examples are IBM System/390 series, Cray computers (Cray-2, Y-MP), the Fujitsu GS8800, and the Hitachi MP5800. Mainframes are high-speed computers,and their word length generally ranges from 32 to 64 bits. Supercomputers such as the Cray-2 and Y-MP are 64 bits high performance and high-speed computers.

Medium-size Computers

In the 1960s, these computers were designed to meet the instructional needs of small colleges, the manufacturing problems of small factories and the data processing tasks of medium-size business such as payroll and accounting. These are called mini-computers. These machines were slower and smaller in memory capacity than mainframes.

Microcomputers

The 4-bit and 8-bit microprocessors became available in the mid-1970s, and initial applications were primarily in the areas of machine control and instrumentation. As the price ofthe microprocessors and memory began to decline, the applications mushroomed in almost all areas, such as video games, word processing, and small-business applications. Early microcomputers were designed around 8 bit microprocessors. Since then 16, 32, 64 bits microprocessors, such as Intel 8086/88, Pentium I to IV, Motorola 68000, and the power PC series have been introduced, and recent micro computers are designed around these microprocessors. Present day micro computers can be classified in 4 groups: personal (or business) computers (PC), work stations, single board, and single chip microcomputers (microcontrollers).

Personal computers (PC)

These microcomputers are single user system and are used for a variety of purposes, such as payroll, business accounts, word processing, legal and medical record keeping, accessing Internet resources (email, Web search). They are also known as personal computers (PC). The HP Pavilion series, Apple Macintosh series, DELL series, etc. A typical configuration of the microcomputer spectrum includes a 32/64 bit microprocessor, 256MB-1GB of system memory (RAM), a monitor, a hard disk with storage capacity more than 80 GB, a CD-ROM drive.

Single-Board Microcomputers

- Typically, these microcomputers include an 8- or 16-bit microprocessor, from 256 bytes to 8K bytes of user memory, a Hex keyboard, and seven-segment LEDs as display.
- The interaction between the microprocessor, memory, and I/Os in these small systems is managed by a monitor program, which is generally small in size, stored in less than 2K bytes of ROM.
- When a single-board microcomputer is turned on, the monitor program is in charge of the system;
  - it monitors the keyboard inputs, interprets those keys, stores progranis in memory, sends system displays to the LEDs, and enables the execution of the user programs.
- Monitor program - a program that interprets the input from a keyboard and converts the input into its binary equivalent.
  - The function of the monitor program in a small system is similar to that of the operating system in a large system.

.

Application: Microprocessorcontrolled Temperature System (Mcts)

This system is expected:

- to read the temperature in a room;
- display the temperature at a liquid crystal display (LCD) panel (described later);
- turn on a fan if the temperature is above a set point, and
- turn on a heater if the temperature is below a set point.

8085 PROGRAMMING MODEL:

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessordesigned by Intel in 1977 using NMOS technology.

It has the following configuration −

* 8-bit data bus
* 16-bit address bus, which can address upto 64KB

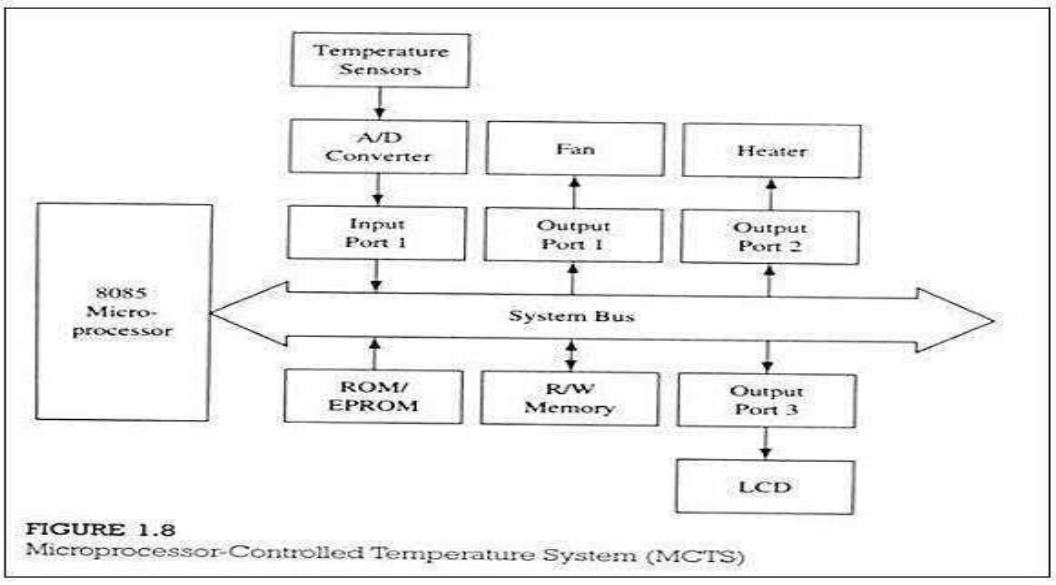
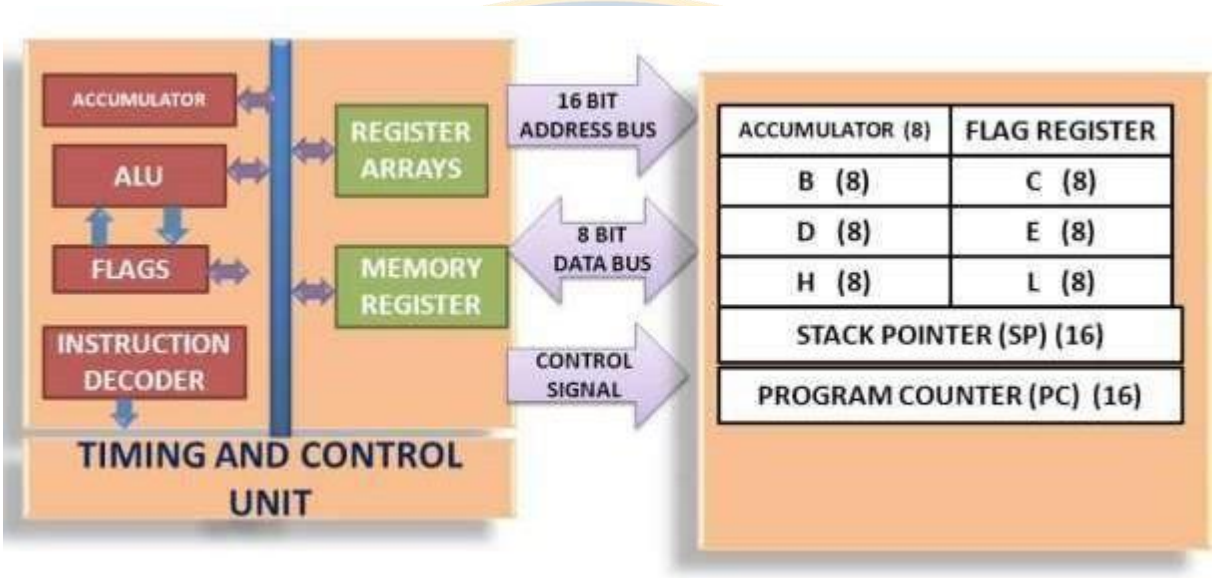


FIGURE 1.8
Microprocessor-Controlled Temperature System (MCTS)

58

* A 16-bit program counter
* A 16-bit stack pointer
* Six 8-bit registers arranged in pairs: BC, DE, HL
* Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.



8085 consists of the following functional units –

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register:

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is likeB-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that

is going to be executed.

Stack pointer: It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register: It is an 8-bit register, which holds the temporary data of arithmetic andlogical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending uponthe result stored in the accumulator.

These are the set of 5 flip-flops −

* Sign (S)
* Zero (Z)
* Auxiliary Carry (AC)
* Parity (P)
* Carry (C)

Its bit position is shown in the following table −

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

* **Sign Flag (S):** Sets or Resets based on the result stored in the accumulator. If the result stored is positive, the flag resets else if the result stored is negative the flag is set.
* **Zero Flag (Z):** Sets or Resets based on the result stored in the accumulator. If the result stored is zero the flag is set else it is reset.
* **Auxiliary Carry Flag (AC):** This flag is set if there is a carry from low nibble (lowest 4 bits) to high nibble (upper 4 bits) or a borrow from high nibble to low nibble, in the low order 8-bit portion of an addition or subtraction operation.

Instruction Classification

An **instruction** is a binary pattern designed inside a microprocessor to perform a specificfunction. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

Data Transfer (Copy) Operations

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. In technical manuals, the term *data transfer* is used for this copying function. The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
* Between Registers.	* Copy the contents of the register B into register D
* Specific data byte to a register or a memory location.	* Load register B with the data byte 32H.
* Between a memory location and a register.	* From a memory location 2000H to register B.
* Between an I/O device and the accumulator.	* From an input keyboard to the accumulator.

Arithmetic Operations

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition - Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

Subtraction - Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's compliment, and the results if negative, are expressed in 2's complement. No two other registers can be subtracted directly.

Increment/Decrement - The 8-bit contents of a register or a memory location can be incremented or decrement by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decrement by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

Logical Operations

These instructions perform various logical operations with the contents of the accumulator.

AND, OR Exclusive-OR - Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the

accumulator. The results are stored in the accumulator.

Rotate- Each bit in the accumulator can be shifted either left or right to the next position.

Compare- Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.

Complement - The contents of the accumulator can be complemented. All 0s are replaced by 1sand all 1s are replaced by 0s.

Branching Operations - This group of instructions alters the sequence of program execution either conditionally or unconditionally.

Jump - Conditional jumps are an important aspect of the decision-making process in the programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called unconditional jump.

Call, Return, and Restart - These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

Machine Control Operations

These instructions control machine functions such as Halt, Interrupt, or do nothing.

Instruction, data format and storage

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction Word Size

The 8085 instruction set is of three groups according to word size:

* One-word or one-byte instructions.
* Two-word or two-byte instructions.
* Three-word or three-byte instructions.

In the 8085 microprocessor, byte and words are synonymous because it is an 8-bit microprocessor. But, instructions are commonly referred to in terms of bytes rather thanwords.

One-byte instructions

A one-byte instruction includes a opcode and a operand in the same byte.

Task	Opcode	Operand	Binary Code	Hex code
Copy the content of accumulator inthe register C.	MOV	C, A	0100 1111	4FH
Add the contents of register B tothe contents of the accumulator.	ADD	B	1000 0000	80H
Invert each bit in the accumulator.	CMA	None	0010	2FH

In the first instruction, operand and registers are specified. In the second instruction, the operand B is specific and the accumulator is not there. Similarly, in the third instruction, the accumulator is assume to be the implicit operand.

Two-byte instructions

In a two-byte instruction, the first byte specifies the operation code and second byte specifies the operand.

Task	Opcode	Operand	Binary code	Hex code	
Load an 8-bit data byte in the accumulator.	MVI	A 35H	0011 1110 0011 0101	3EH 35H	First byte Second byte

Three-byte instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit operand. The second byte is the low-order operand and the third byte is the high-order operand. If a 16-bit numeral is present in the instruction then that instruction will be of three-byte. For example, in LXI H,3500H and STA 2500H, etc.

Task	Opcode	Operand	Binary code	Hex code	
Transfer the program sequence to the memory location 2085h	JMP	2550H	1100 0011 0101 0000 0010 0101	C3 50 25	First byte Second byte Third byte

Opcode Format

In the case of microprocessor, the instruction or operation are specified by using

Registers	Code
B	0 0 0
C	0 0 1
D	0 1 0
E	0 1 1
H	1 0 0
L	1 0 1
M (Memory)	1 1 0
A	1 1 1

Register Pairs	Code
BC	0 0
DE	0 1
HL	1 0
AF or SP	1 1

specificbit pattern unique for each instruction. These bit patterns contain all the information about operation, register used, memory to. The register and register pair are specified by using certain combination of bits. These combinations of bits are in table below.

Data Format

In 8-bit processor systems, commonly used codes and data formats are,

* ASCII Code- This is a 7-bit alphanumeric code that represents decimal numbers, English alphabets and non-printable characters such as carriage return.
* BCD Code- The term BCD stands for binary-coded decimal; it is used for decimal numbers. The decimal numbering system has ten digits.
* Signed Integer- A signed integer is either a positive number or a negative number. In a 8-bit processor, the most significant digit, d7, is used for the sign; 0 represents the positive sign and 1 represents the negative sign.
* Unsigned Integer- A integer without a sign can be represented by all the 8 bits in a Microprocessor register.

Data Storage: Memory

Memory is a storage of binary bits. Memory chips used in most systems are nothing but8-bit registers stacked one above the other.

How to write, assemble and execute a simple program

A program is a sequence of instruction written to tell a computer to perform a specificfunction.

Illustrative Program: Adding Two Hexadecimal Numbers

Problem – Write instructions to load 2 hexa-decimal numbers 32H and 48H in registers A and B,respectively. Add the numbers and display the sum at LED output port PORT1.

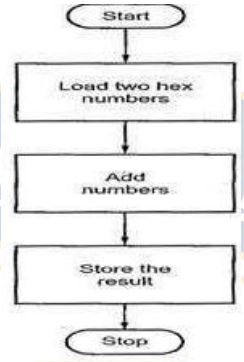
Problem analysis

* Load the numbers in the registers.
* Add the numbers.

* Display the sum at the output port PORT1.

Flowchart

The steps listed in the problem analysis and the sequence can be represented in a blockdiagram is called a flowchart.



Assembly Language	**Hex Code**
MVI A,32H	3E
32	
MVI B,48H	06
48	
ADD B	80
OUT 01H	D3
01	
HLT	76

Storing Hex Code in the Memory:

Once the hex code is ready, it has to be loaded in the memory of specially designed microprocessor system (Microprocessor training kit) for execution. To perform this task we should know the address range of read/write memory in the system.

Let us assume that the read/write memory ranges from 2000H to 22FFH. The microprocessor training kit has keypad to enter the hex code, in the memory. It provides a special routine (monitor program) to enter a hex code byte by byte and execute the program. Typical steps for storing hex code in the memory from address from address 2000H are as follows:

* Reset the microprocessor system by pressing the RESET key.
* Enter into store mode by pressing SET key.

* Enter the address of the memory 2000H, where the first hex code (starting address of theprogram) is to be stored using hex keys.
* Enter the hex code using hex keys.
* Increment the memory address by 1 using INC key.
* Repeat steps 4 and 5 until the last hex code.

Executing the Program:

The microprocessor training kit provides a procedure to execute the program. To activate the procedure we have to enter the starting address of the program (2000H in our example). To enter this address we have to go into execute mode by pressing GO key and enter the starting address using hex keys.

Once the starting address is entered, the program can be executed by pressing EXECUTEkey. The EXECUTE key procedure loads the starting address of our program, 2000H into the program counter and program control is transferred from monitor program to our program.

After this microprocessor reads one hex code at a time, and when it fetches the complete instruction, it executes that instruction. Then it fetches the next instruction and this process continues until the last instruction in the program is executed.

Overview of the 8085 instruction set

The 8085 microprocessor instruction set has 74 operation codes that result in 246instructions.

Data transfer instructions in 8085 microprocessor

Data transfer instructions are the instructions which transfers data in the microprocessor.

They are also called copy instructions.

Following is the table showing the list of logical instructions:

OPCODE	OPERAND	EXPLANATION	EXAMPLE
MOV	Rd, Rs	Rd = Rs	MOV A, B
MOV	Rd, M	Rd = Mc	MOV A, 2050
MOV	M, Rs	M = Rs	MOV 2050, A
MVI	Rd, 8-bit data	Rd = 8-bit data	MVI A, 50
MVI	M, 8-bit data	M = 8-bit data	MVI 2050, 50
LDA	16-bit address	A = contents at address	LDA 2050
STA	16-bit address	contents at address = A	STA 2050
LXI	r.p., 16-bit data	loads the specified register pair with data	LXI H, 3050
LDAX	r.p.	indirectly loads at the accumulator A	LDAX H
STAX	16-bit address	indirectly stores from the accumulator A	STAX 2050
XCHG	none	exchanges H with D, and L with E	XCHG
PUSH	r.p.	pushes r.p. to the stack	PUSH H
POP	r.p.	pops the stack to r.p.	POP H

OPCODE	OPERAND	EXPLANATION	EXAMPLE
IN	8-bit port address	inputs contents of the specified port to A	IN 15
OUT	8-bit port address	outputs contents of A to the specified port	OUT 15

Arithmetic instructions in 8085 microprocessor

Arithmetic Instructions are the instructions which perform basic arithmetic operations such as addition, subtraction and a few more. In 8085 microprocessor, the destination operand is generally the accumulator. In 8085 microprocessor, the destination operand is generally the accumulator.

Following is the table showing the list of arithmetic instructions:

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	R	A = A + R	ADD B
ADD	M	A = A + Mc	ADD 2050
ADI	8-bit data	A = A + 8-bit data	ADD 50
ADC	R	A = A + R + prev. carry	ADC B
ADC	M	A = A + Mc + prev. carry	ADC 2050
ACI	8-bit data	A = A + 8-bit data + prev. carry	ACI 50
SUB	R	A = A – R	SUB B
SUB	M	A = A – Mc	SUB 2050
SUI	8-bit data	A = A – 8-bit data	SUI 50
SBB	R	A = A – R – prev. carry	SBB B
SBB	M	A = A – Mc -prev. carry	SBB 2050
SBI	8-bit data	A = A – 8-bit data – prev. carry	SBI 50
INR	R	R = R + 1	INR B
INR	M	M = Mc + 1	INR 2050
INX	r.p.	r.p. = r.p. + 1	INX H
DCR	R	R = R – 1	DCR B
DCR	M	M = Mc – 1	DCR 2050
DCX	r.p.	r.p. = r.p. – 1	DCX H
DAD	r.p.	HL = HL + r.p.	DAD H

In the table,

R stands for register M stands for memory

Mc stands for memory contents

r.p. stands for register pair

Logical instructions in 8085 microprocessor

Logical instructions are the instructions which perform basic logical operations such as AND,OR, etc. In 8085 microprocessor, the destination operand is always the accumulator. Here logical operation works on a bitwise level.

Following is the table showing the list of logical instructions:

OPCODE	OPERAND	DESTINATION	EXAMPLE
ANA	R	A = A AND R	ANA B

OPCODE	OPERAND	DESTINATION	EXAMPLE
ANA	M	A = A AND Mc	ANA 2050
ANI	8-bit data	A = A AND 8-bit data	ANI 50
ORA	R	A = A OR R	ORA B
ORA	M	A = A OR Mc	ORA 2050
ORI	8-bit data	A = A OR 8-bit data	ORI 50
XRA	R	A = A XOR R	XRA B
XRA	M	A = A XOR Mc	XRA 2050
XRI	8-bit data	A = A XOR 8-bit data	XRI 50
CMA	none	A = 1's compliment of A	CMA
CMP	R	Compares R with A and triggers the flag register	CMP B
CMP	M	Compares Mc with A and triggers the flag register	CMP 2050
CPI	8-bit data	Compares 8-bit data with A and triggers the flag register	CPI 50

RRC	none	Rotate accumulator right without carry	RRC
RLC	none	Rotate accumulator left without carry	RLC
RAR	none	Rotate accumulator right with carry	RAR
RAL	none	Rotate accumulator left with carry	RAR

OPCODE	OPERAND	DESTINATION	EXAMPLE
CMC	none	Compliments the carry flag	CMC
STC	none	Sets the carry flag	STC

In the table,

R stands for register M stands for memory

Mc stands for memory contents

Branching instructions in 8085 microprocessor

Branching instructions refer to the act of switching execution to a different instructionsequence as a result of executing a branch instruction.

The three types of branching instructions are:

* Jump (unconditional and conditional)
* Call (unconditional and conditional)
* Return (unconditional and conditional)

1. **Jump Instructions –** The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JMP	address	Jumps to the address	JMP 2050

* **Conditional Jump Instructions:** Transfers the program sequence to the described memoryaddress only if the condition in satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JC	address	Jumps to the address if carry flag is 1	JC 2050

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

2. **Call Instructions –** The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALLis pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

* **Unconditional Call Instructions:** It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

* **Conditional Call Instructions:** Only if the condition is satisfied, the instructions executes.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050

CNC	address	Call if carry flag is 0	CNC 2050

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if parity flag is 1	CPE 2050
CPO	address	Calls if parity flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RET	none	Return from the subroutine unconditionally	RET

(b) Conditional Return Instruction: The program sequence is transferred unconditionally fromthe subroutine to the calling program only is the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RNZ	none	Return from the subroutine if zero flag is 0	RNZ

RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

The Machine Control instructions are as follows:

HLT – Halt NOP - No Operation

UNIT II

MICROPROCESSOR ARCHITECTURE AND MICROCOMPUTER SYSTEMS

Microprocessor Architecture and its operations

The microprocessor is a programmable logic device, designed with registers, flip-flops. The microprocessor has a set of instructions designed internally, to manipulate data and communicate with peripherals. This process of data manipulation and communication is determined by the logic design of the microprocessor, called the architecture.

The microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set. These instructions are given to the microprocessor by writing them into its memory. Writing (Or entering) instruction and data is done through an input device such as a keyboard. The result can be stored in memory or scent to such output devices as LEDs or a CRT terminal. All the various functions performed by the microprocessor can be classified in three general categories:

* Microprocessor-initiated operations
* Internal data operations
* Peripheral (or externally) initiated operations.

To perform these functions, the microprocessor requires a group of logic circuits and a set of signals called control signals. However, early processors do not have the necessary circuitry on one chip; the complete units are made up more than one chip. Therefore, the term Micro Processing Unit (MPU) is defined here as a group of devices that can perform these functions with the necessary set of control signals. This term is similar to the term Central Processing Unit (CPU).

Microprocessor-Initiated Operations and 8085/8080A Bus Organization

The MPU performs primarily four operations:

* Memory Read: Reads data from memory.
* Memory Write: Writes data into memory.
* I/O read: Accepts data from input devices.
* I/O Write: Sends data to output devices.

All these operations are part of the communication process between the MPU and peripheral devices (including memory). To communicate with a peripheral (or a memory location), the MPU needs to perform the following steps:

Step 1: Identify the peripheral or the memory location (with its address).

Step 2: Transfer data

Step 3: Provide timing or synchronization signals.

The 8085/8080A MPU performs these functions using three sets of communication lines called

buses: The address bus, the data bus, and the control bus.

Internal Data Operations and the 8085/8080A Registers

The internal architecture of the 8085/8080A microprocessor determines how and what operations can be performed with the data. These operations are,

* Store 8-bits data.
* Perform arithmetic and logical operations.
* Test for conditions.
* Sequence the execution of instructions.
* Store data temporarily during execution in the defined R/W memory locations called thestack.

To perform these operations, the microprocessor requires registers, an arithmetic logic unit(ALU) and control logic, and internal buses (path for information flow).

Externally Initiated Operations in Microprocessor

8085 microprocessor support some **Externally initiated operations**, which are also known as **Peripheral operations**. Different external input/output devices or signals can initiate these type operations. Following are the some externally initiated operations:

1. **RESET–**

   This RESET key is used to clear the program counter and update with 0000H memory location. When this RESET pin is activated by any external key, then all the internal operations are suspended for that time. After that the execution of the program can begin at the zero memory address.

2. **Interrupt–**

   8085 microprocessor chip have some pins for interrupt like TRAP, RST 5.5, RST 6.5 and RST

   7.5. The microprocessor can be interrupted from the normal instructions and asked to perform some other emergency operations, which are also known as Service routine. The microprocessor resumes its operation after the completion Service routine.

3. **READY–**

   The 8085 microprocessor has a pin called READY. If the signal at this READY pin is in low state then the microprocessor enters into the Wait state. This signal is used mainly to synchronize slower external devices with the microprocessor.

4. **HOLD–**

   When the HOLD pin is activated by an external signal, the microprocessor relinquishes control buses and allows the external peripheral to use them. For example, the HOLD signal is used direct memory access (DMA) data transfer.

MEMORY

Memory is an essential component of a microcomputer system. It stores binary instructions and data for the microprocessor. There are two types of memory: Read/Write Memory(R/WM) and Read-Only Memory (ROM).

Memory Organization (R/W Memory): To communicate with memory, the MPU shouldbe able to:

* select the chip
* Identify the register
* Read from or Write into the register

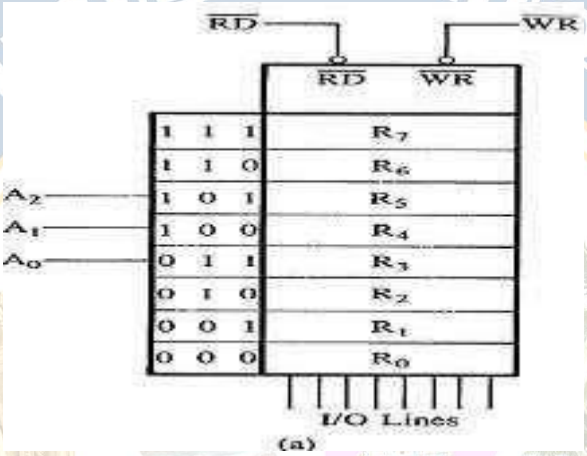


Figure (a) shows memory chip of eight register with three address lines, one chip select (CS) line, one read write (R/W) line, and eight I/O lines. The MPU uses the CS line to select the chip, and the R/W line to control data flow.

Memory Map

Memory map is defined as the assignment of addresses to memory registers in various memory chips in a system. In a system based on the 8085/8080A microprocessor, the entire memory map can range from 0000h to FFFFh (2^{16}= 65.536).

Example-1- Illustrate the memory map of the chip with 256 bytes of memory and hardware of the chip select (CS) line in fig 2.

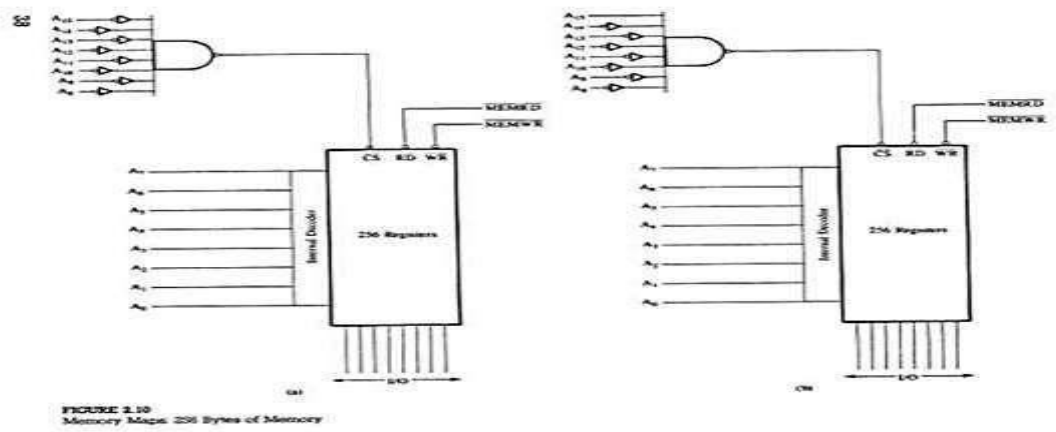


Fig 2:-Memory map: 256 bytes of memory

Memory classification

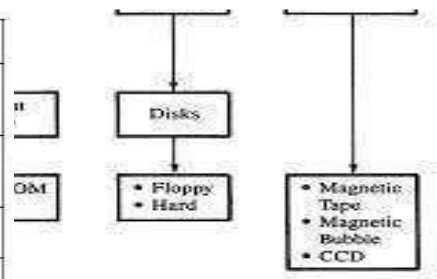
Memory is the most essential element of a computing system because without it computer can't perform simple tasks. Computer memory is of two basic type – Primary memory (RAM and ROM) and Secondary memory (hard drive,CD,etc.). Random Access Memory (RAM) is primary- volatile memory and Read Only Memory (ROM) is primary-non-volatile memory. Memory can be classified into two groups as shown in fig.

1. Random Access Memory (RAM) –

* It is also called as read write memory or the main memory or the primary memory.
* The programs and data that the CPU requires during execution of a program are stored in thismemory.
* It is a volatile memory as the data loses when the power is turned off.
* RAM is further classified into two types- SRAM (Static Random AccessMemory) and DRAM (Dynamic Random Access Memory).

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM



1. Read Only Memory (ROM)

- Stores crucial information essential to operate the system, like the program essential to bootthe computer.
- It is not volatile.
- Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into 4 types- *ROM, PROM, EPROM,*and *EEPROM*.

Types of Read Only Memory (ROM) –

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

- PROM (Programmable read-only memory) – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.
- EPROM (Erasable Programmable read only memory) – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.
- EEPROM **(Electrically erasable programmable read only memory)** – The data can be erasedby applying electric field, no need of ultra violet light. We can erase only portions of the chip.

Input and output (I/O) devices

The MPU accepts the binary data from input devices such as keyboard and analog/digital converters and sends data to output devices such as printers and LEDs. For performing this task, MPU first need to identify the input/output devices.

There are two different methods by which I/O devices can be identified: Using 8-bit address, and Using 16-bit address. These methods are described briefly in the following sections:

I/Oswith8-bitaddresses–

This is also known as **peripheral-mapped I/O or I/O-mapped-I/O**. In this type of I/O, MPU uses eight address lines to identify an input or an output devices. This is an 8-bit numbering system for I/Os used in conjunction with the input and output instructions. This is also known as I/O space that is separate from the memory space which is 16-bit numbering

system. The eight address lines have 2^8 combinations which is total 256 addresses; therefore MPU can identify 256 input devices and 256 output devices with addresses ranging from 00H to FFH.

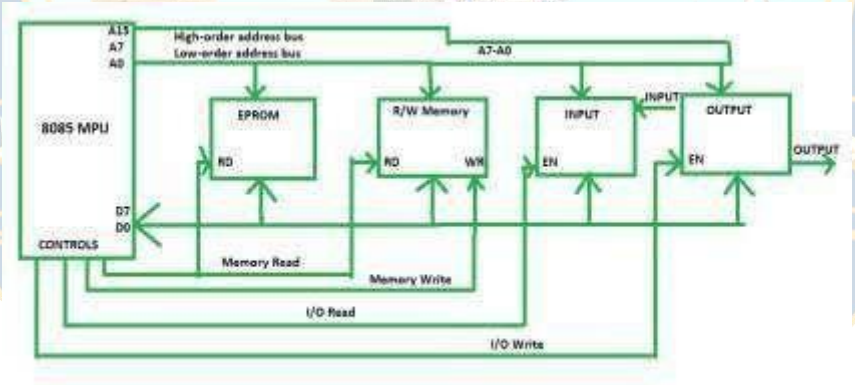
The input and output devices can be differentiated by using the control lines I/O Read and I/O Write. MPU uses I/O Read control signal for input devices and I/O Write control signal for output devices.

I/Oswith16-bitaddresses–

This is also known as **Memory-mapped I/O**. In this type of I/O, MPU uses sixteen address lines to identify an input or an output devices; an I/O is connected as if it is a memory register. The MPU uses same control signal (Memory Read and Memory Write) and instructions as those of memory. In some microprocessors, such as the Motorola 6800, all I/Os have 16-bit addresses; I/Os and memory share the same memory map (64K). The steps are summarized below:

* The MPU places an 8-bit address (or 16-bit address) on the address bus, which is decodedby external decode logic.
* The MPU sends a control signal (I/O Read or I/O Write) and enables the I/O device.
* Data are transferred using the data bus.

Example of microcomputer system



A Microcomputer System

The 8085 microprocessor is an example of Microcomputer System. A microprocessor system contains : two types of memory that are EPROM and R/WM, Input and Output devices and the buses that are used to link all the peripherals (memory and I/Os) to the MPU.In 8085, we 16 address lines ranging from A0 to A15 that are used to address memory. The lower order **address bus A0-A7** is used in the identification of the input and output devices.

This microcomputer system has 8 **data lines D0-D7** which are bidirectional and common to all the devices.It generates four **control signals**: Memory Read, Memory Write, I/O Read and I/O Write and they are connected to different peripheral devices. The MPU communicates with only one peripheral at a time by enabling that peripheral through its control signal.

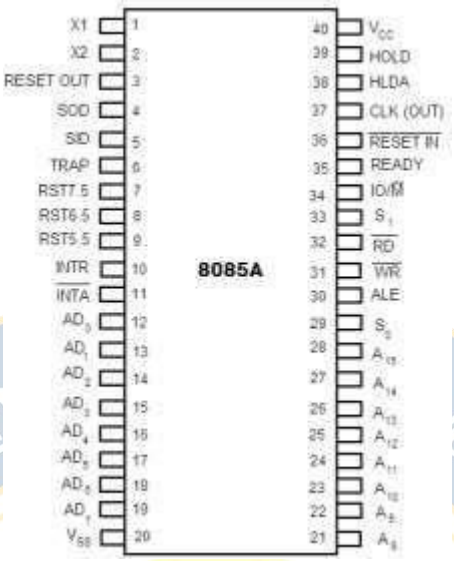
For example, sending a data to the output device, the MPU places the device address (or output port number) on the address bus, data on the data bus and enables the output device by using its control signal I/O Write. After that the output device display the result.

8085 Microprocessor architecture and memory interfacingThe 8085 MPU

The term Micro Processing Unit (MPU) is similar to the term Central Processing Unit (CPU) used in traditional computers. We define the MPU as a device or a group of devices (as a unit) that can communicate with peripherals, provide timing signals, direct data flow, and perform computing tasks as specified by the instructions in memory. The unit will have the necessary lines for the address bus, the data bus, and the control signals, and would require only a power supply and a crystal (or equivalent frequency - determining components) to be completely functional.

Using this description, the 8085 microprocessor can almost qualify as an MPU, but with thefollowing two limitations.

* The low-order address bus of the 8085 microprocessor is multiplexed (time shared) with thedata bus. The buses need to be de multiplexed.
* Appropriate control signals need to be generated to interface memory and I/O with the 8085. (Intel has some specialized memory and I/O devices that do not require such control signals).



The 8085 Microprocessor

The 8085 is an 8-bit general purpose microprocessor capable of addressing 64K of memory. The device has forty pins, requires a +5 V single power supply, and can operate with a 3-MHz single- phase clock. The 8085 is an enhanced version of predecessor, the 8080A; its instruction set is upward-compatible with that of the 8080A, meaning that the 8085 instruction set includes all the 8080A instructions plus some additional ones. Programs written for the 8080A will be executed by the 8085, but the 8085 and the 8080A are not pin compatible.

Multiplexed Address / Data Bus

The signal lines AD7 to AD0 are bidirectional, they serve a dual purpose. They are used as the low-order address bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the low-order address bus. During the later art of the cycle, these lines are used as the data bus. (This is also known as multiplexing the bus). However, the low-order address bus can be separated from these signals by using a latch.

Control and Status Signals

This group of signals includes two control signals (RD and WR), three status signals (IO/M, S1 andS0) to identify the nature of the operation, and one special signal (ALE) to indicate the beginning of the operation. These signals are as follows:

* **ALU - Address Latch Enable:** This is a positive going pulse generated every time the 8085, begins an operation (machine cycle); it indicates that the bits on AD7 - AD0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A7 to A0.

* **RD - Read:** This is a Read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

* **WR - Write:** This is a Write control signal (active low). This signal indicates that the data on the bus are to be written into a selected memory or I/O location.

* **IO/M:** This is a status signal to differentiate between I/O and memory operations. When it is high, it indicates and I/O Operation; when it is low, it indicates a memory operation. This signal is combined with RD (Read) and WR (Write) to generate I/O and memory control signals.
* **S1 and S0:** There status signals, similar to IO/M, can identify various operations; but they are rarely used in small systems.

Power Supply and Clock Frequency

The power supply and frequency signals are as follows:

* VCC: + 5 volt power supply.
* VSS: Ground Reference.
* X1, X2: A crystal (or RC, LC network) is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3 MHz, the crystal should have frequency of 6 MHz.
* CLK (OUT) - Clock Output: This signal can be used as the system clock for other devices.

Interrupts and Externally InitiatedSignals

* **HOLD (INPUT**): HOLD indicates that another device is requesting for the use of the address and data bus.
* HLDA (OUTPUT): HLDA is a signal for HOLD acknowledgement which indicates that the HOLD request has been received. After the removal of this request the HLDA goes low.
* INTR (Input): INTR is
* an Interrupt Request Signal. Among interrupts it has the lowest priority. The INTR is enabled or disabled by software.
* INTA (Output): INTA is an interrupt acknowledgement sent by the microprocessor after INTR is received.
* RST 5.5, 6.5, 7.5 **and TRAP (Inputs):** These **all are interrupts**. When any interrupt is recognized the next instruction is executed from a fixed location in the memory as given below:

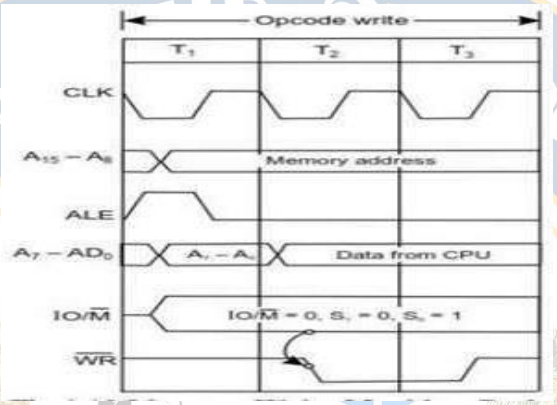
Line	Location from which next instruction is picked up
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

RST 7.5, RST 6.5 and RST 5.5 are the restart interrupts which cause an internal restart to be

automatically inserted.

The TRAP has the highest priority among interrupts. The order of priority of interrupts is as follows:

- TRAP (Highest priority)
- RST 7.5
- RST 6.5
- RST 5.5
- INTR (Lowest priority).



Reset Signals

- **RESET IN (Input):** It resets the program counter (PC) to 0. It also resets interrupt enableand HLDA flip-flops. The CPU is held in reset condition till RESET is not applied.
- **RESET OUT (Output):** RESET OUT indicates that the CPU is being reset.

Clock Signals

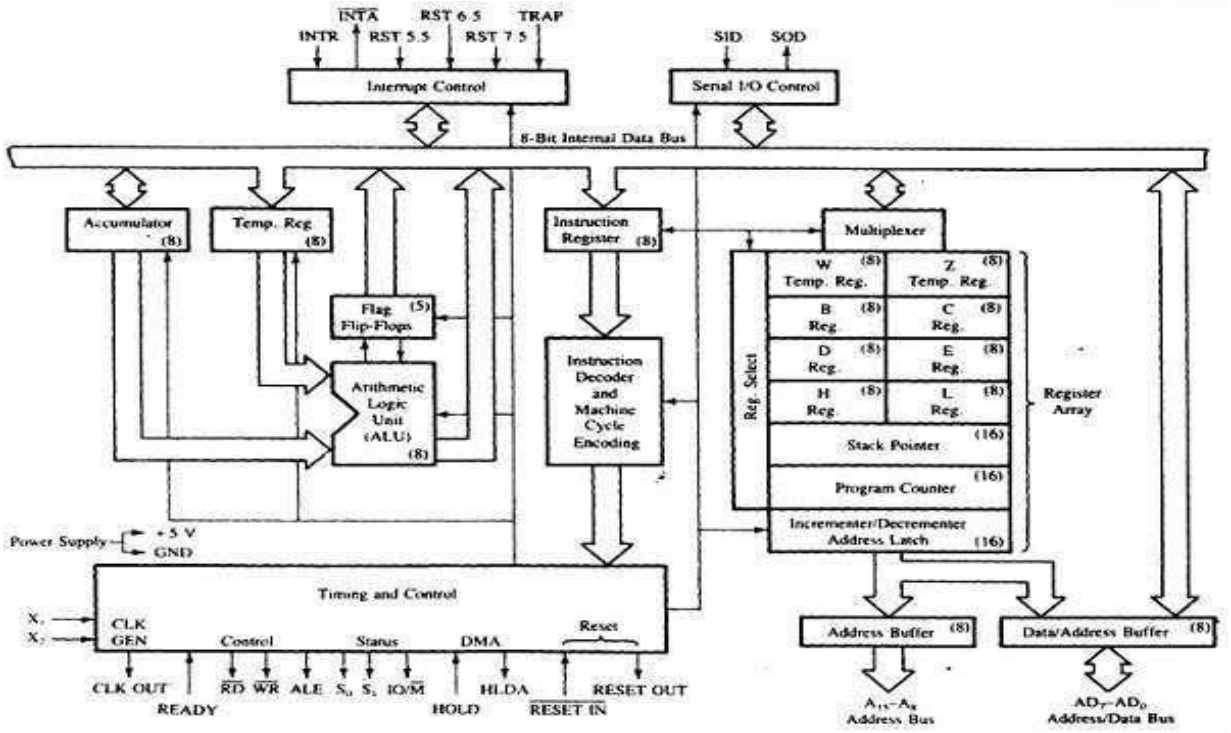
- **X1, X2 (Input):** X1 and X2 are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor. It is used to produce a suitable clockfor the operation of microprocessor.
- **CLK (Output):** CLK is a **clock output** for user, which can be used for other digital ICs. Itsfrequency is same at which processor operates.

Serial I/O Signals

- **SID (Input):** SID is data line for **serial input**. The data on this line is loaded into the seventhbit of the accumulator when RIM instruction is executed.
- **SOD (Output):** SOD is a data line for **serial output**. The seventh bit of the accumulator isoutput on SOD line when SIM instruction is executed.

A Detailed Look at the 8085 MPU and its Architecture

It includes the ALU, Timing and Control Unit, Instruction Register and Decoder, Register Array, Interrupt Control and Serial I/O Control.



1. ALU

- The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’ etc.
- Uses data from memory and from Accumulator to perform arithmetic operation and alwaysstores result of operation in Accumulator.
- The ALU consists of accumulator, flag register and temporary register.

a) Accumulator :-

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. Theresult of an operation is stored in the accumulator. The accumulator is also identified asregister A.

b) Flag Register:-

8085 has 8-bit flag register. Flags are flip-flops which are used to indicate the status of the accumulator and other register after the completion of operation. There are only 5active flags.

S	Z		AC		P		CY

i. Sign flag(S):

Sign flag indicates whether the result of a mathematical or logical operation is negative or positive. If the result is negative, this flag will be set (i.e. S=1) and if the result is positive, the flag will be reset (i.e. S=0).

ii. Zero flag (Z):

Zero flag indicates whether the result of a mathematical or logical operation is zero or not. If the result of current operation is zero, the flag will be set (i.e. Z=1) otherwise the flag will be reset (Z=0). This flag will be modified by the result in the accumulator as well as in the other register.

iii. Auxiliary carry flag (AC):

In operation when a carry is generated by bit D3 and passes on to bit D4, the AC flag will be set otherwise AC flag will be reset. This flag is used only internally for BCD operation and is not available for the programmer to change the sequence of program with the jump instruction.

iv. Parity flag (P):

This flag indicates whether the current result is of even parity (no. of 1's is even) or odd parity(no. of 1's is odd). If even parity, P flag will be set otherwise reset.

v. Carry flag (CY):

This flag indicates whether during an addition or subtraction operation carry or borrow is generated or not. If carry or borrow is generated, the flag will be set otherwise reset.

a) Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

2. Timing and control unit

* This unit produces all the timing and control signal for all the operation.
* This unit synchronizes all the MP operations with the clock and generates the controlsignals necessary for communication between the MP and peripherals.

3. Instruction register and decoder

* The instruction register and decoder are part of ALU. When an instruction is fetched frommemory, it is loaded in the instruction register.
* The decoder decodes the instruction and establishes the sequence of events to follow.
* The IR is not programmable and cannot be accessed through any instruction.

4. Register array

The register unit of 8085 consists of

- Six general-purpose data registers B,C,D,E,H,L
- Two internal registers W and Z
- Two 16-bit address registers PC (program counter) and SP (stack pointer)
- One increment/decrement counter register
- One multiplexer (MUX)
- The six general-purpose registers are used to store 8-bit data. They can be combined asregister pairs BC, DE, and HL to perform some 16-bit operations.
- The two internal registers W and Z are used to hold 8-bit data during the execution of someinstructions, CALL and XCHG instructions.
- SP is 16-bit registers used to point the address of data stored in the stack memory. It alwaysindicates the top of the stack.
- PC is 16-bit register used to point the address of the next instruction to be fetched andexecuted stored in the memory.

Example of an 8085-based microcomputer

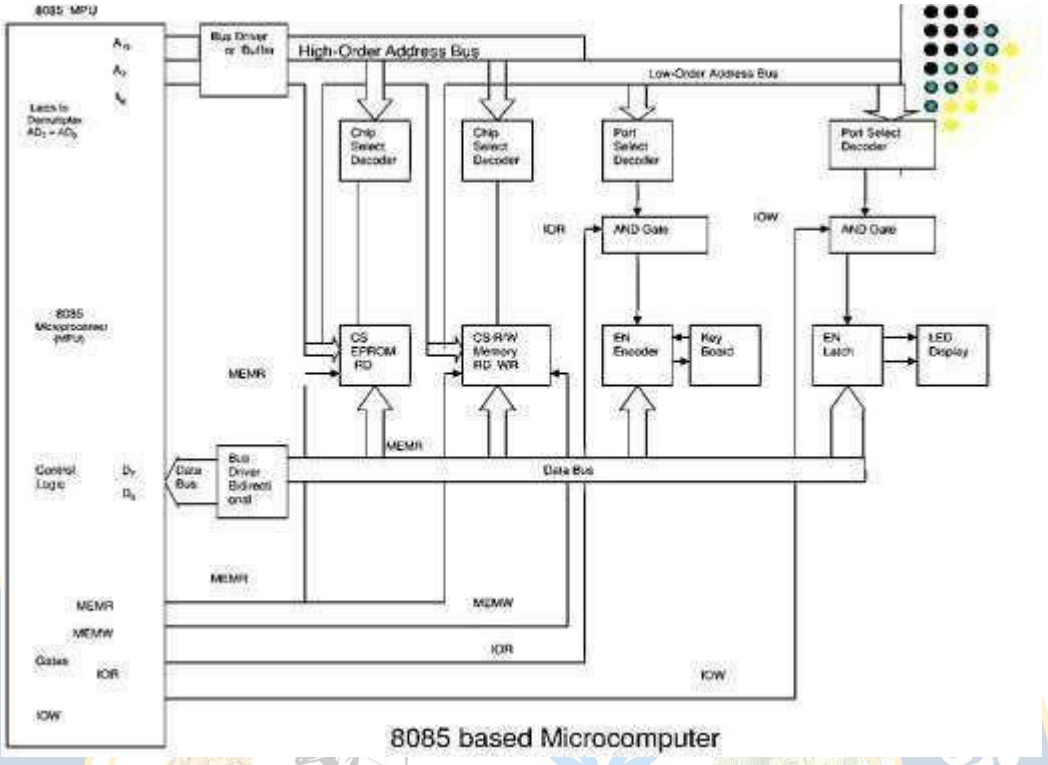
The 8085 MPU module includes devices such as the 8085 microprocessor, an octal latch and logic gates. The octal latch demultiplexes the bus AD7-AD0 using the signal ALE and the logic gates generate the necessary control signals.

THE 8085 MACHINE CYCLES

The 8085 executes several types of instructions with each requiring a different number of operations of different types. However, the operations can be grouped into a small set. The three main types are:

- Memory Read and Write.
- I/O Read and Write.
- Request Acknowledge.

These can be further divided into various smaller operations (machine cycles).



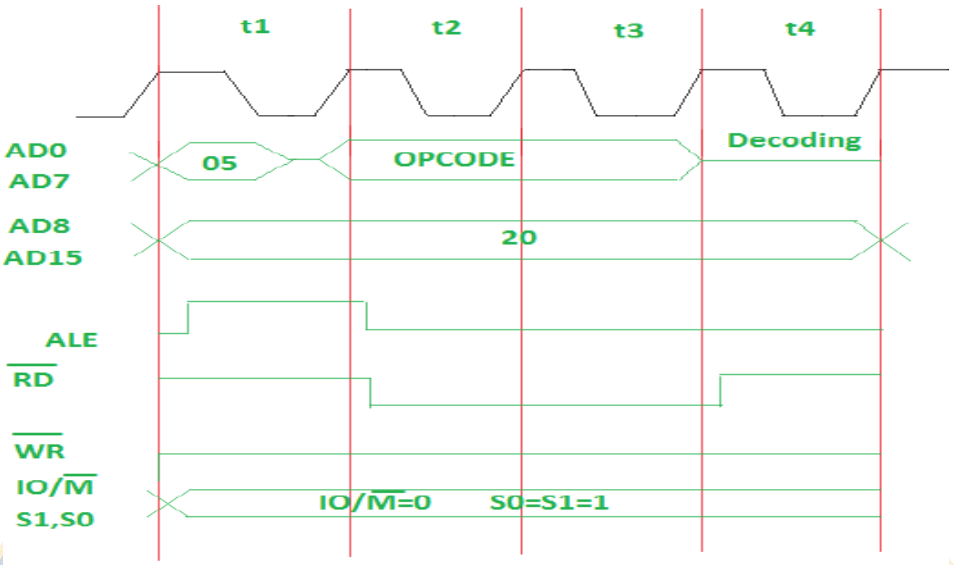
8085 based Microcomputer

OPCODE FETCH MACHINE CYCLE

- The first step of executing any instruction is the Opcode fetch cycle.
- In this cycle, the microprocessor brings in the instruction's Opcode from memory.
- To differentiate this machine cycle from the very similar "memory read" cycle, the control &status signals are set as follows:
- IO/M=0, s0 and s1 are both 1.
- This machine cycle has four T-states.
- The 8085 uses the first 3 T-states to fetch the opcode.
- T4 is used to decode and execute it.
- It is also possible for an instruction to have 6 T-states in an opcode fetch machine cycle.

Memory read machine cycle

The memory read machine cycle is exactly the same as the opcode fetch except:It only has 3 T-states. The s0 signal is set to 0 instead.



Timing diagram for opcode fetch

Above diagram represents:

- **05 –** lower bit of address where opcode is stored. Multiplexed address and data bus AD0- AD7 are used.
- **20 –** higher bit of address where opcode is stored. Multiplexed address and data bus AD8- AD15 are used.
- **ALE –** Provides signal for multiplexed address and data bus. If signal is high or 1, multiplexed address and data bus will be used as address bus. To fetch lower bit of address, signal is 1 so that multiplexed bus can act as address bus. If signal is low or 0, multiplexed bus will be used as data bus. When lower bit of address is fetched then it will act as data bus as the signal is low.
- **RD (low active) –** If signal is high or 1, no data is read by microprocessor. If signal is low or 0, data is read by microprocessor.
- **WR (low active) –** If signal is high or 1, no data is written by microprocessor. If signal is low or 0, data is written by microprocessor.
- **IO/M (low active) and S1, S0 –** If signal is high or 1, operation is performing on input output. If signal is low or 0, operation is performing on memory.

Memory interfacing

Memory is an integral part of a microprocessor system, and in this section, we will discuss how to interface a memory device with the microprocessor. The Memory Interfacing in 8085 is used to access memory quite frequently to read instruction codes and data stored in memory. This read/write operations are monitored by control signals. The microprocessor activates these signals when it wants to read from and write into memory. In the last section we have already seen the memory read and memory write machine cycles, and status of the RD, WR and IO/M status signals for read/write operation. In the following section we will see

memory structure and its requirements, concepts in Memory Interfacing in 8085 and interfacing examples.

Memory Structure and its Requirements

As mentioned earlier, read/write memories consist of an array of registers, in which each register has unique address. The size of the memory is Nx M as shown in Fig. 1 where N is the number of registers and M is the word length, in numberof bits.

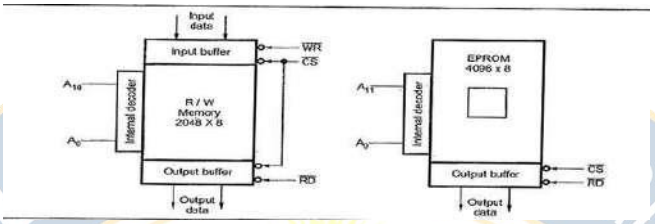
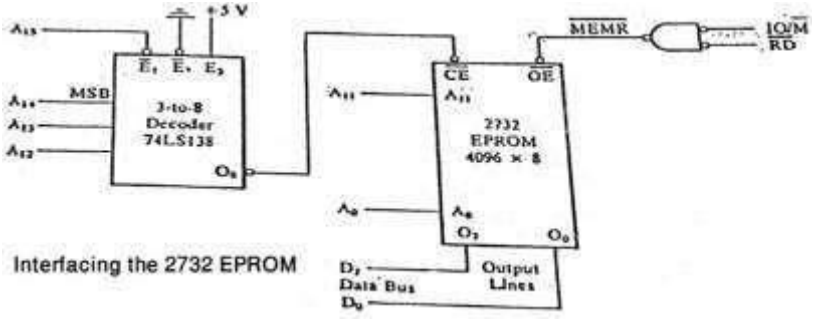


Fig 1

Basic Concepts in Memory Interfacing:

For Memory Interfacing in 8085, following important points are to be kept in mind. Microprocessor 8085 can access 64Kbytes memory since address bus is 16-bit. But it is not always necessary to use full 64Kbytes address space. The total memory size depends upon the application.

1. Generally EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 64Kbytes is shared by them.
2. The capacity of program memory and data memory depends on the application.
3. It is not always necessary to select 1 EPROM and 1 RAM. We can have multiple EPROMs and multiple RAMs as per the requirement of application.
4. We can place EPROM/RAM anywhere in full 64 Kbytes address space. But program memory (EPROM) should be located from address 0000H since reset address of 8085 microprocessor is 0000H.
5. It is not always necessary to locate EPROM and RAM in consecutive memory For example: If the mapping of EPROM is from 0000H to OFFFH, it is not must to locate RAM from 1000H. We can locate it anywhere between 1000H and FFFFH. Where to locate memory component totally depends on the application.

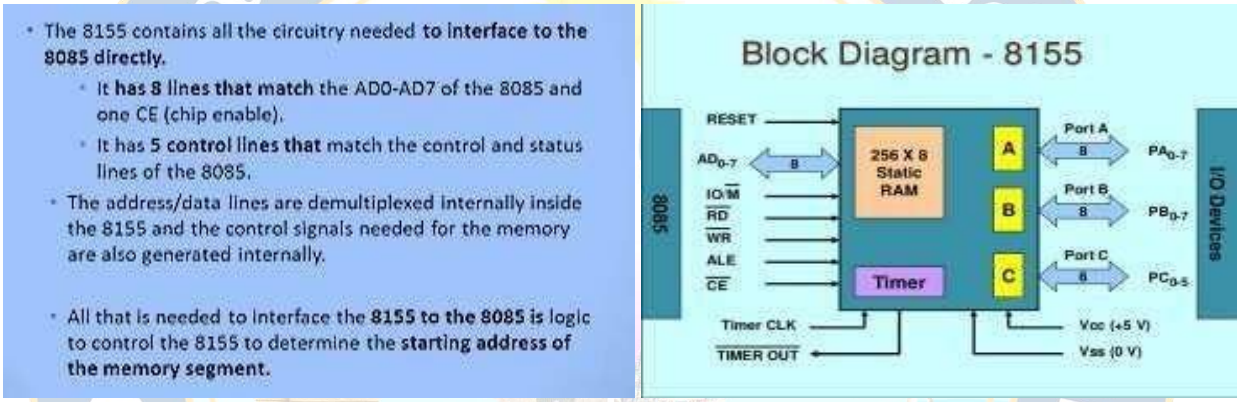


Interfacing the 2732 EPROM

The memory interfacing requires to:

* **Select the chip**
* **Identify the register**
* **Enable the appropriate buffer.**

Microprocessor system includes memory devices and I/O devices. It is important to note that microprocessor can communicate (read/write) with only one device at a time, since the data, address and control buses are common for all the devices. In order to communicate with memoryor I/O devices, it is necessary to decode the address from the microprocessor. Due to this each device (memory or I/O) can be accessed independently. The following section describes common address decoding techniques.



INTERFACING THE 8155 MEMORY SEGMENT

UNIT III

DATA TRANSFER OPERATION

Data transfer (Copy) operations

It is the longest group of instructions in 8085. This group of instruction copy data from a source location to destination location without modifying the contents of the source. The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator. None of these instructions changes the flag. The instructions of this group are:

1) MOV Rd, RsRdRs (move register instruction)

–1 byte instruction

–Copies data from source register to destination register.

–Rd & Rs may be A, B, C, D, E, H &L

–E.g. MOV A, B

2) MVI R, 8 bit data (move immediate instruction)

–2 byte instruction
–Loads the second byte (8 bit immediate data) into the register specified.
–R may be A, B, C, D, E, H & L
– E.g. MVI C, 53H53H

3) MOV M, R (Move to memory from register)

–Copy the contents of the specified register to memory. Here memory is the location specified by contents of the HL register pair.
–E.g. MOV M, B

4) MOV R, M (move to register from memory)

–Copy the contents of memory location specified by HL pair to specified register.

–E. g. MOV B, M

Write a program to load memory locations 7090 H and 7080 H with data 40H and 50H andthen swap these data.

Soln : MVI H, 70H MVI L, 90H

MVI A, 40H MOV M, A MOV C, M MVI L, 80H MVI B, 50H MOV M, B MOV D, M MOV M, C MVI L, 90HMOV M, D HLT

5) LXI RP, 2 bytes data (load register pair)

–3-byte instruction

–Load immediate data to register pair

–Register pair may be BC, DE, HL & SP(Stack pointer)

–1st byte- Op-code

–2nd byte – lower order data

–3rd byte- higher order data

– E.g. L X I B, 4532H; B ← 45, C ← 32H

6) MVI M, data (load memory immediate)

–2 byte instruction.

–Loads the 8-bit data to the memory location whose address is specified by the contents of HL pair.

7) LDA 4035H (Load accumulator direct)

–3-byte instruction

–Loads the accumulator with the contents of memory location whose address is specified by 16 bit address.-A ← [4035H]

8) IN 8-bit address

2-byte instruction

Read data from the input port address specified in the second byte and loads data into the accumulator.

9) OUT 8-bit address

2-byte instruction

Copies the contents of the accumulator to the output port address specified in the 2nd byte. That means accumulator to output port: P A ←

Addressing modes:

Instructions are command to perform a certain task in microprocessor. The instruction

consists of op-code and data called operand. The operand may be the source only, destination only or both of them. In these instructions, the source can be a register, a memory or an input port. Similarly, destination can be a register, a memory location, or an output port. The various format (way) of specifying the operands are called addressing mode. So addressing mode specifies where the operands are located rather than their nature. The 8085 has 5 addressing mode:

1. **Direct Addressing Mode:**

   The instruction using this mode specifies the effective address as part of instruction. The instruction size either 2-bytes or 3-bytes with first byte op-code followed by 1 or 2 bytes of address of data.

2. **Register Direct Addressing Mode:**

   This mode specifies the register or register pair that contains the data. E g. MOV A, B

   Here register B contains data rather than address of the data. Other examples are: ADD, XCHGetc.

3. **Register Indirect Addressing Mode:**

   In this mode the address part of the instruction specifies the memory whose contents are the address of the operand. So in this type of addressing mode, it is the address of the address rather than address itself. (One operand is register)

4. **Immediate Addressing Mode:**

   In this mode, the operand position is the immediate data. For 8-bit data, instructionsize is 2 bytes and for 16 bit data, instruction size is 3 bytes.

5. **Implied or Inherent Addressing Mode:**

   The instructions of this mode do not have operands.

   E.g.

   NOP: No operation HLT: Halt

   EI: Enable interrupt DI: Disable interrupt

Arithmetic Operations:

The 8085 microprocessor performs various arithmetic operations such as addition, subtraction, increment and decrement. These arithmetic operations have the following mnemonics.

1. **ADD R/M**

   Byte add instruction, Adds the contents of register/memory to the contents of the accumulatorand stores the result in accumulator⇐ E. g. Add B; A [A] + [B]

2. ADI 8 bit data

byte add immediate instruction, Adds the 8 bit data with the contents of accumulatorand storesresult in accumulator.– E g. ADI 9BH ; A A+9BH

3. SUB R/M

One byte subtract instruction, subtracts the contents of specified register / m with the contents ofaccumulator and stores the result in accumulator. E. g. SUB D ; A ←A-D

4. SUI 8 bit data

Two byte subtract immediate instruction, Subtracts the 8 bit data from the contents of accumulatorstores result in accumulator.– E. g. SUI D3H; A ←A-D3H

5. INR R/M, DCR R/M

One byte increment and decrement instructions, Increase and decrease the contents of R(register) or M(memory) by 1 respectively.

Addition operation in 8085:

8085 performs addition with 8-bit binary numbers and stores the result in accumulator. Ifthe sum is greater than 8-bits (FFH), it sets the carry flag.

E.g. MVI A, 93H	1 0	1 1 0 1 1 1		B7
MVI C, B7H	+ 1 0	0 1 0 0 1 1		+ 93
ADD C	**1 0 1** [redacted]	**1 0**		**1** 4A
	CY			**CY**

Subtraction operation in 8085:

8085 performs subtraction operation by using 2's complement and the steps used are:

- Converts the subtrahend (the number to be subtracted) into its 1's complement.
- Adds 1 to 1's complement to obtain 2's complement of the subtrahend.
- Adds 2's complement to the minuend (the contents of the accumulator).

* Complements the carry flag.

Logic Operations:

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hardwired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, XOR and NOT (Complement):

The following features hold true for all logic instructions:

* The instructions implicitly assume that the accumulator is one of the operands.
* All instructions reset (clear) carry flag except for complement where flag remain unchanged.
* They modify Z, P & S flags according to the data conditions of the result.
* Place the result in the accumulator.
* They do not affect the contents of the operand register.

The logical operations have the following instructions.

1. **ANA R/M** (the contents of register/memory)

   – Logically AND the contents of register/memory with the contents of accumulator, 1byte instruction and CY flag is reset and AC is set.

2. **ANI 8 bit data**

   – Logically AND 8 bit immediate data with the contents of accumulator, 2 byte instruction andCY flag is reset and AC is set.

3. **ORA R/M**

   – Logically OR the contents of register/memory with the contents of accumulator, 1 byteinstruction and CY and AC is reset and other as per result.

4. **ORI 8 bit data**

   – Logically OR 8 bit immediate data with the contents of the accumulator,2 byte instructionand CY and AC is reset and other as per result.

5. **XRA R/M**

   – Logically exclusive OR the contents of register memory with the contents of accumulator, 1byte instruction and CY and AC is reset and other as per result.

6. **CMA (Complement accumulator)**

   – One byte instruction, Complements the contents of the accumulator and No flags are affected.

Branching Operations:

The microprocessor is a sequential machine; it executes machine codes from one memory location to the next. The branching instructions instruct the microprocessor to go to a different memory location and the microprocessor continues executing machine codes from that new location.

The branching instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions. The branching instruction code categorized in following three groups:

* Jump instructions
* Call and return instruction
* Restart instruction

Jump Instructions:

The jump instructions specify the memory location explicitly. They are 3 byte instructions, one byte for the operation code followed by a 16 bit (2 byte) memory address. Jump instructions can be categorized into unconditional and conditional jump.

Unconditional Jump

8085 includes unconditional jump instruction to enable the programmer to set up continuous loops without depending only type of conditions. E.g. JMP 16 bit address: loads the program counter by 16 bit address and jumps to specified memory location.

E.g. JMP 4000H

Here, 40H is higher order address and 00H is lower order address. The lower order byte.

Conditional Jump

The conditional jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags. After logic and arithmetic operations, flags are ser or reset to reflect the condition of data. These instructions check the flag conditions and make decisions to change or not to change the sequence of program. The four flags namely carry, zero, sign and parity used by the jump instruction.

Mnemonics	Description
[redacted]	[redacted]
JC 16 bit	Jump on carry (if CY=1)
JNC 16 bit	Jump on if no carry (if CY=0)
JZ 16bit	Jump on zero (if Z=1)

JNZ 16bit	jump on if no zero (if Z=0)
JP 16bit	jump on positive (if S=0)
JM 16bit	jump on negative (if S=1)
JPE 16bit	Jump on parity even (if P=1)
JPO 16bit	Jump on parity odd (if P=0)

Call and return instructions: (Subroutine)

Call and return instructions are associated with subroutine technique. A subroutine is a group of instructions that perform a subtask. A subroutine is written as a separate unit apart from the main program and the microprocessor transfers the program execution sequence from main program to subroutine whenever it is called to perform a task. After the completion of subroutine task microprocessor returns to main program. The subroutine technique eliminates the need to write a subtask repeatedly, thus it uses memory efficiently. Before implementing the subroutine, the stack must be defined; the stack is used to store the memory address of the instruction in the main program that follows the subroutines call.

To implement subroutine there are two instructions CALL and RET.

1. CALL 16 bit memory

* Call subroutine unconditionally.
* byte instruction.
* Saves the contents of program counter on the stack pointer. Loads the PC by jump address (16 bit memory) and executes the subroutine.

2. RET

* Returns from the subroutine unconditionally.
* 1 byte instruction
* Inserts the contents of stack pointer to program counter.

E.g. **Write an ALP to add two numbers using subroutines.**

2000	MVI B, 4AH	3000	MOV A, B
2002	MVI C, A0H	3001	ADD C
2004	CALL 3000H	3002	RET
2007	MOV B, A		

SP = 2007 (i.e. PC) PC = 3000 (i.e. 16bit)

Writing a Assembly Language ProgramSteps to write a program

1 Analyze the problem

2 Develop program Logic

3 Write an Algorithm

4 Make a Flowchart

5 Write program Instructions using Assembly language of 8085

Program 8085 in Assembly language to add two 8-bit numbers and store 8-bit result in register

Analyze the problem

Addition of two 8-bit numbers to be done

Program Logic

* Add two numbers
* Store result in register C

Algorithm

1. Get two numbers
2. Add them
3. Store result
4. Stop

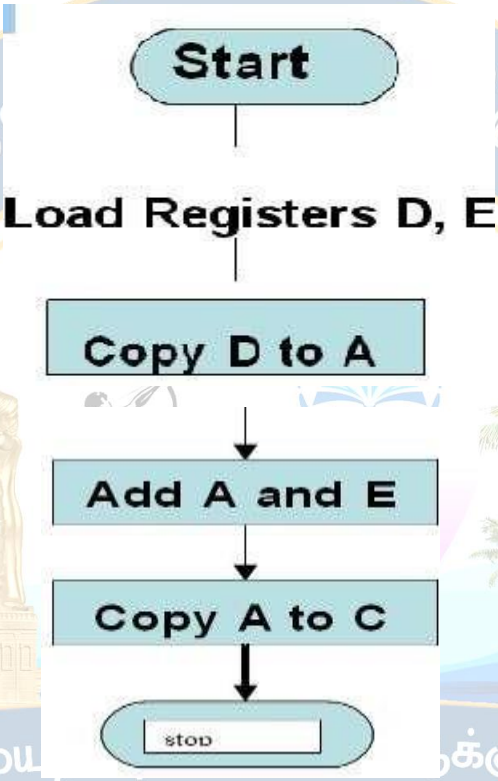
Translation to 8085 operations

* Load 1st no. in register D
* Load 2nd no. in register E
* Copy register D to A

* Add register E to A
* Copy A to register C

MVI D, 2HMVI E, 3HMOV A, DADD E MOV C, A HLT

FLOWCHART



DEBUGGING A PROGRAM

Debugging is the process of identifying and removing bug from software or program. It refers toidentification of errors in the program logic, machine codes, and execution. It gives step by step information about the execution ofcode to identify the fault in the program.

The debugging process is divided into two parts:

Static Debugging: It is similar to visual inspection of circuit board, it is done by a paper and pencil to check the flowchart and machine codes. It is used to the understanding of code logic and structure of program.

Dynamic Debugging: It involves observing the contents of register or output after execution ofeach instruction (in single step technique) or a group of instructions (in breakpoint technique).

Common sources of error:

Selecting a wrong code

Forgetting second or third byte of instructionSpecifying wrong jump locations

Not reversing the order of high and low bytes in a Jump instructionwriting memory addresses in decimal instead of hexadecimal.

PROGRAMMING TECHNIQUES WITH ADDITIONAL INSTRUCTIONS PROGRAMMING TECHNIQUES: LOOPING, COUNTING AND INDEXING

The Programming Technique used to instruct the microprocessor to repeat task is called looping. This process is accomplished by using jump instructions. A loop can be classified into two groups:

Continuous loop- repeats a task continuously

Conditional loop-repeats a task until certain data condition are met

Continuous loop

A continuous loop is set up by using the unconditional jump Instruction shown in the flowchart. A program with Continuous loop does not stop repeating the tasks until the system is reset.

Conditional Loop

A Conditional loop is setup by the conditional jump instructions. These instructions Check flags (zero, carry, etc.) and repeat the specified task if the conditions are satisfied. These loops usually include counting and indexing. Conditional loop is shown by the Flowchart as follow.

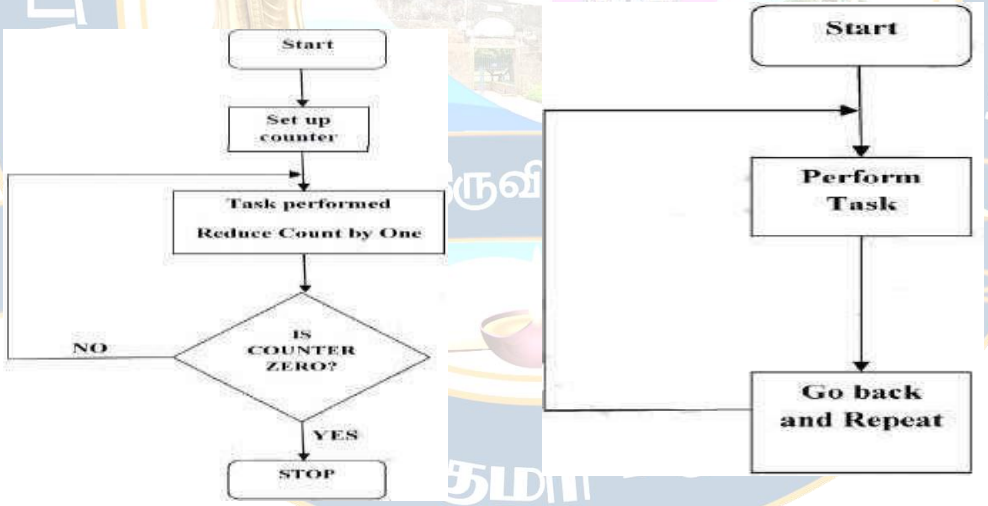


Fig: - Conditional Loop AND Continuous Loop

The Flowchart is translated into the program as follows:

* Counter is setup by loading an appropriate count in a register.
* Counting is performed by either incrementing or decrementing the counter.
* Loop is set up by a conditional jump instruction.

* End of counting is indicated by a flag

Additional data transfer and 16 bit arithmetic instructions

1. 16 BIT DATA TRANSFER TO REGISTER PAIR (LXI)LXI Reg. pair, 16-bit data.

-The instruction loads 16-bit data in the register pair designated in the operand.

Example: LXI H, 2034H or LXI H, XYZ

2. DATATRANSFER FROM MEMORY TO MICROPROCESSORMOV R,M

-R, M copies data byte from Memory to Register. Memory location, its location is specified by the contents of the HL registers.

Example**:** MOV B, M

Load accumulator indirectLDAX B/D Reg. pair

- The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered.

Example: LDAX B

Load accumulator DirectLDA 16-bit address

-The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator.

Example: LDA 2034H

3. DATATRANSFER FROM MICROPROCESSOR TO MEMORY OR DIRECTLY INTO MEMORYMOVM,R

-This instruction copies the contents of the source.

-The source register are not altered. As one of the operands is a memory location, its location is specified by the contents of the HL registers.

Example: MOV M, B

STA 16-bit address

-The contents of the accumulator are copied into the memory locationspecified by theoperand.

-This is a 3-byte instruction, the second byte specifies the low-order address andthe third byte specifies the high-order address.

Example: STA 4350H

Store accumulator indirect

STAX Reg. pair

-The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

Example: STAX B

Increment / Decrement register pair by 1

INX R

-The contents of the designated register pair are incremented by 1 and the result is stored in the same place.

Example: INX H

DCX R

-The contents of the designated register pair are decremented by 1 and the result is stored in the same place.

Example: DCX H

Arithmetic operations related to memoryAdd memory

- **ADD M:** The contents of the operand (memory) are added to the contents of the accumulator and the result is stored in the accumulator.
- The operand is a memory location, its location is specified by the contents of the HLregisters.
- All flags are modified to reflect the result of the addition.

Subtract memory

- **SUB M:** The contents of the operand (memory) are subtracted to the **contents of the accumulator and the result is stored in the accumulator.**
- **The operand is a memory location, its location is specified by the contents of the HL registers.**
- **All flags** are modified to reflect the result of the Subtraction.

Increment memory by 1/Decrement memory by 1

- **INR M/DCR M:** The contents of the memory are incremented by 1 using INR anddecremented by 1 using DCR and the result is stored in the same place.
- The operand is a memory location, its location is specified by the contents of the HL registers.

Logic operations: Rotate

ROTATE is a logical operation of 8085 microprocessor. It is a 1 byte instruction. This instruction does not require any operand after the opcode. It operates the content of accumulator and the result is also stored in the accumulator. The Rotate instruction is used to rotating the bits of accumulator.

Types of ROTATE Instruction:

There are 4 categories of the ROTATE instruction: Rotate accumulator left (RLC), rotate accumulator left through carry (RAL), Rotate accumulator right (RRC), Rotate accumulator right through carry (RAR). Among these four instructions; two are for rotating left and two are for rotating right. All of them are explain briefly in the following sections:

1. **Rotate accumulator left (RLC) –**

   In this instruction, each bit is shifted to the adjacent left position. Bit D7 becomes D0. Carry flagCY is modified according to the bit D7.

2. **Rotate accumulator left through carry (RAL) –**

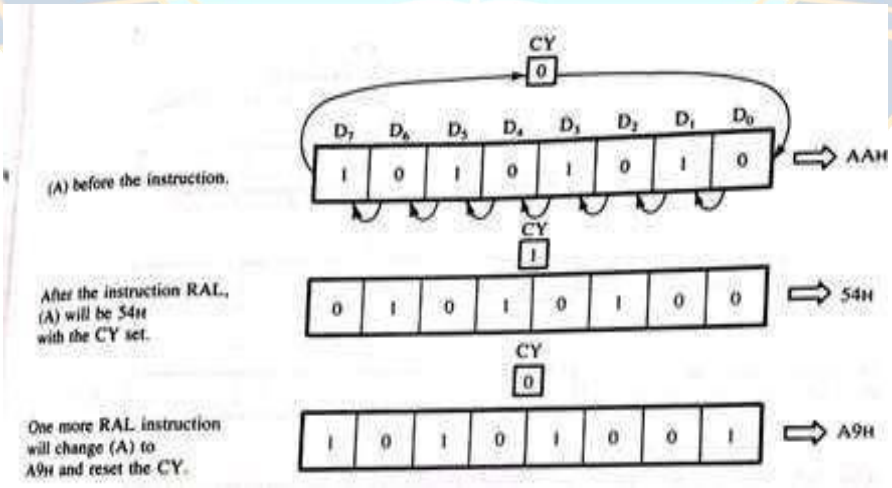
   In this instruction, each bit is shifted to the adjacent left position. Bit D7 becomes the carry bitand the carry bit is shifted into D0. Carry flag CY is modified according to the bit D7.

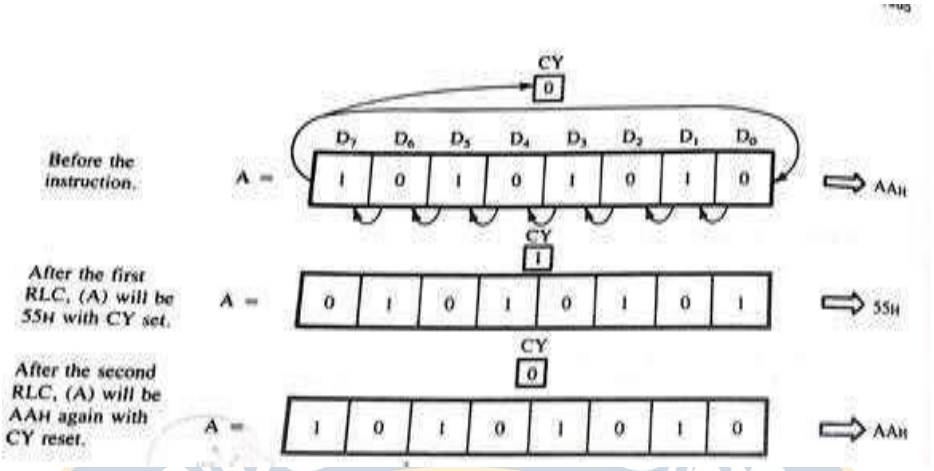
3. **Rotate accumulator right (RRC) –**

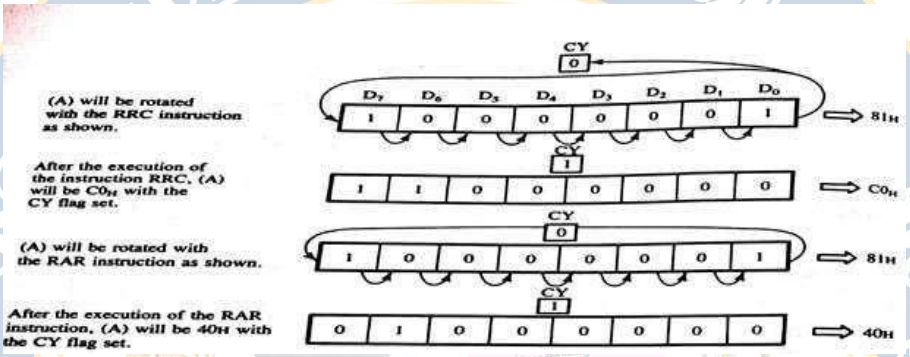
   In this instruction, each bit is shifted to the adjacent right position. Bit D7 becomes D0. Carryflag CY is modified according to the bit D0.

4. **Rotate accumulator right through carry (RAR) –**

   In this instruction, each bit is shifted to the adjacent right position. Bit D0 becomes the carry bitand the carry bit is shifted into D7. Carry flag CY is modified according to the bit D0.







Applications of ROTATE Instructions:

The ROTATE instructions are primarily used in arithmetic multiply and divide operationsand for serial data transfer. For example: If A is 0000 1000 = 08H

* By rotating 08H right: A = 0000 0100 = 04Hthis is equivalent to dividing by *2*.
* By rotating 08H left: A = 0001 0000 = 10Hthis is equivalent to multiplying *by 2*.

However, these procedures are invalid when logic 1 is rotated left from D7 to D0 or vice versa.For example, if 80H is rotated left it becomes 01H.

DYNAMIC DEBUGGING

Checking for logical and syntax errors in the program is called as debugging.

TYPES:

* STATIC
* DYNAMIC

STATIC DEBUGGING:

Checking for errors in the program manually using paper and pencil is known as static debugging.

DYNAMIC DEBUGGING:

Checking for errors in the program by observing the execution of instructions is called as DYNAMIC DEBUGGING.

TOOLS FOR DYNAMIC DEBUGGING:

* SINGLE STEP
* REGISTER EXAMINE3.BREAK POINT

SINGLE STEP:

* Allows you to execute one instruction at a time and observe the results of each instruction.
* A Single step facility is built with a hard wired logic circuit.
* By pressing Single-step key in the microprocessor, able to observe addresses and codes as they are executed.
* Single-step technique will be able to spot incorrect address
* Incorrect jump locations for loops incorrect data or missing codes.
* To effectively use this technique reduce the number of loops and delay counts to minimum number.
* Better to use this technique for short programs.
* This technique helps to infer the flag status by observing the execution of jump instructions.

REGISTER EXAMINE:

* Register Examine Key in the microprocessor allows to examine the contents of the microprocessor register.
* By pressing appropriate keys, the monitor program displays the contents of the registers.
* This technique is used in conjunction with either the single step or break point facilities.
* After executing a block of instructions, we can examine the register contents of the program at a critical juncture and compare the contents with the expected outcomes.

BREAK POINT:

* Breakpoint facility is a software routine that allows to execute a program in sections.
* The break point in a program can be set in a program using RST instructions.
* When we push the EXECUTE key, program will be executed until the breakpoint, using

monitor we can check the registers for expected results.

* If the segment of the program found satisfactory, a second breakpoint can be set at a subsequent memory address to debug the next segment of the program.
* With Breakpoint facility we can isolate the segment of the program with errors.
* Then that segment of the program can be debugged with the single step facility.
* The break point technique can be used to check out the timing loop, I/O section and Interrupts.

Common Source of Errors:

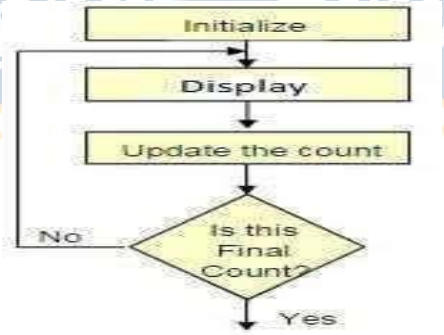
* Failure to clear the Accumulator when its used to add numbers.
* Failure to clear the carry registers or keep track of a carry.
* Failure to update a memory pointer or a counter.
* Failure to set a flag before using a conditional jump instruction.
* Use of an improper combination of Rotate instructions
* Specification of a wrong memory address for a jump instruction.
* without the knowledge of clearing the flag before using a jump instruction

UNIT IV

COUNTERS AND TIME DELAYS

Counter and time delays

A counter is designed simply by loading appropriate number into one of the registers andusing INR or DNR instructions. Loop is established to update the count. Each count is checked to determine whether it has reached final number; if not, the loop is repeated.



TIME DELAY

Procedure used to design a specific delay. A register is loaded with a number, depending on the time delay required and then the register is decremented until itreaches zero by setting up a loop with conditional jump instruction.

LABEL	OPCODE	OPERAND	COMMENTS	T STATES
	MVI	C,FFH	Load register C	7
LOOP:	DCR	C	Decrement C	4
	JNZ	LOOP	Jump back to	10/7

Clock frequency of the system = 2 MHz Clock period=1/T= 0.5 μs

Time to execute MVI = 7 T states * 0.5= 3.5 μs

Time Delay in Loop TL= T*Loop T states * N10

= 0.5 * 14* 255= 1785 μs = 1.8 ms

N10 = Equivalent decimal number of hexadecimal count loaded in the delay register

TLA= Time to execute loop instruction=TL –(3T states* clock period)=1785-1.5=1783.5 μs

TIME DELAY USING A REGISTER PAIR

LXI	B,2384H	Load BC with 16-bit count	10
LOOP: DCX	B	Decrement BC by 1	**6**
MOV	A,C	Place contents of C in A	**4**
ORA	B	OR B with C to set Zero flag	**4**
JNZ	LOOP	if result not equal to 0 ,	**10/7**
		jump back to loop	

Time Delay in Loop TL= T*Loop T states * N10

= 0.5 * 24* 9092

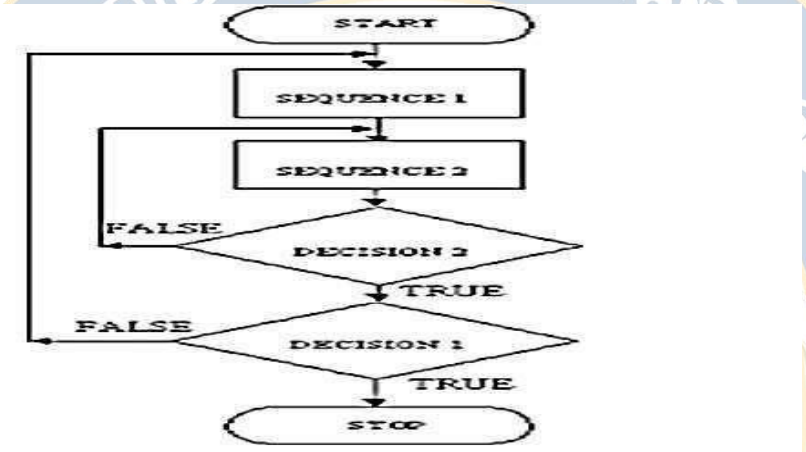
= 109 ms

Time Delay using a LOOP within a LOOP

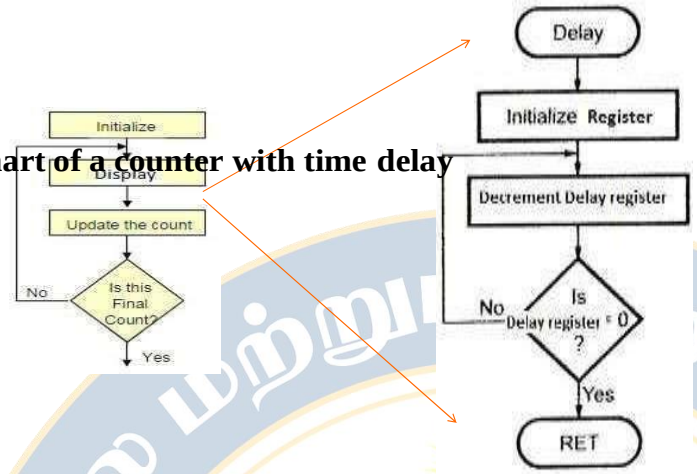
```
        MVI     B,38H     7T
LOOP2:  MVI     C,FFH     7T
LOOP1:  DCR     C         4T
        JNZ     LOOP1     10/7 T
        DCR     B         4T
        JNZ     LOOP 2    10/7T
```

Delay in Loop TL1=1783.5 µs

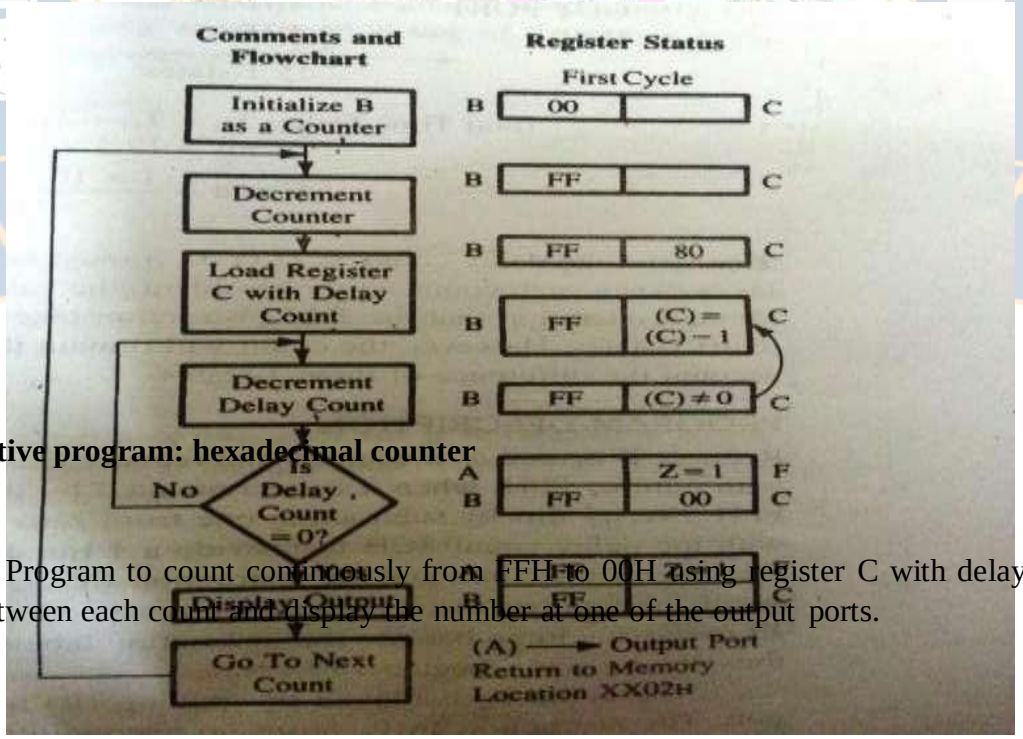
Delay in Loop TL2= (0.5*21+TL1)*56
=100.46ms



Flowchart for time delay with two loops



Flowchart of a counter with time delay



Illustrative program: hexadecimal counter

Write a Program to count continuously from FFH to 00H using register C with delaycount 8CH between each count and display the number at one of the output ports.

```
        MVIB,00H
NEXT:   DCR B
        MVI C,8CH
DELAY:  DCR C
        JNZ DELAY
        MOV A,B
        OUT PORT#
        JMP NEXT
```

Generates a continuous square wave with the period of 500 Micro Sec. Assume the system clock period is 325ns, and use bit D0 output the square wave

```
        MVI D, AAH
X:      MOV A, D
        RLC
        MOV D, A
        ANI 01H
        OUT PORT1
        MVI B, COUNT
Y:      DCR B
        JNZ Y
        JMP X
```

Delay outside loop: T0=46

Tstates*325=14.95 micro sec.

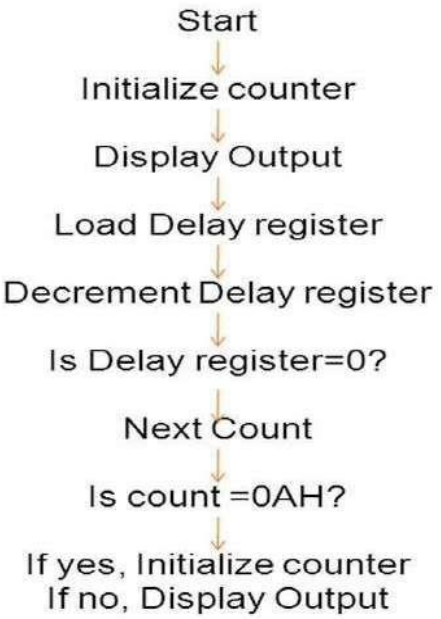
Loop delay: TL=4.5 micro sec

Total Td=To+T_L

Count=34 H

(MODULO TEN) COUNTER

```
START:  MVI B,00H
        MOV A,B
DSPLAY: OUT PORT #
        LXI H,16-bit
LOOP:   DCX H
        MOV A,L
        ORA H
        JNZ LOOP
        INR B
        MOV A,B
        CPI 0AH
        JNZ DSPLAY
        JZ START
```



Debugging counter and time delay programs

It is designed to count from 100(base 10) to 0 in Hex continuously with a 1 second delay between each count. The delay is set up using two loops. The inner loop is executed to provide approximately 100ms delay and is repeated 10 times, using outer loop to provide a total delay of 1 second. The clock period of system is 330ns.

```
MVI A, 64H      7
X: OUT PORT1    10
Y:MVI B, 10H    7
Z:LXI D, X      10
DCX D           6
NOP             4
NOP             4
MOV A, D        4
ORA E           4
JNZ Z           10/7
DCR B           4
JZ Y            10/7
DCR A           4
CPI 00H         7
JNZ X           10/7
```

Delay in loop1=32T X count x $330x10^{-9}$
100ms =32T X count x $330x10^{-9}$
Count=9470

Stack and Subroutines:Stack

The stack is defined as a set of memory location in R/W memory, specified by a programmer in a main memory. These memory locations are used to store binary information temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP, 16 bit address. Once the stack location is defined, it loads 16 bit address in the stack pointer register. Storing of data bytes for this operation takes place at the memory location that is one less than the address e.g. LXI SP, 2099H

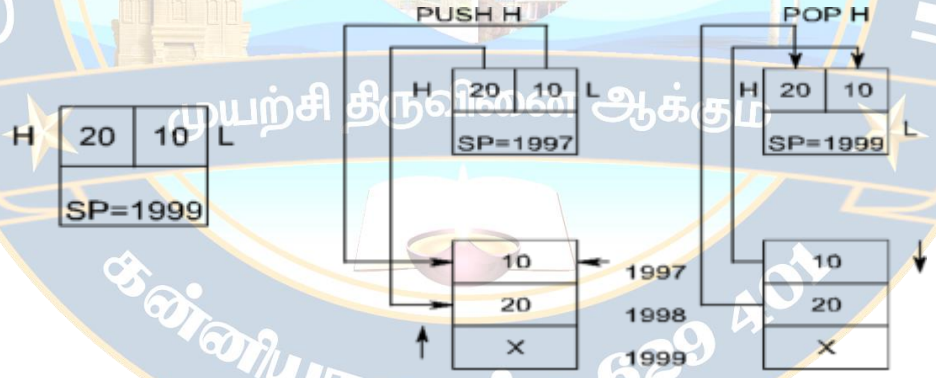
Here the storing of data bytes begins at 2098H and continuous in reverse order i.e 2097H. Therefore, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The stack instructions are:

PUSH Rp/PSW (Store register pair on stack)

One byte instruction. Copies the contents of specified register pair or program status word (accumulator and flag) on the stack. Stack pointer is decremented and content of high order register is copied. Then it is again decremented and content of low order register is copied.

POP Rp/PSW (retrieve register pair from stack)

One byte instruction. Copies the contents of the top two memory locations of the stack into specified register pair or program status word. A content of memory location indicated by SP is copied into low order register and SP is incremented by 1. Then the content of next memory location is copied into high order register and SP is incremented by 1.



XTHL – exchanges top of stack (TOS) with HLSPHL – move HL to SP

PCHL – move HL to PCDI – disable interrupt

EI – Enable interrupt SIM – set interrupt mask

RIM – read interrupt maskE.g.

LXI SP, 1FFFHLXI H, 9320H LXI B, 4732H LXI D, ABCDHMVI A, 34H PUSH H PUSH B PUSH D PUSH PSW POP H

POP B POP D POP PSWHLT

Subroutines

➢ **A subroutine is a group of instructions that will be used repeatedly in different locationsof the program.**

- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

➢ In Assembly language, a subroutine can exist anywhere in the code.

- However, it is customary to place subroutines separately from the main program.

➢ The 8085 has two instructions for dealing with subroutines.

- The CALL instruction is used to redirect program execution to the subroutine.
- The RET instruction is used to return the execution to the calling routine.

The CALL Instruction

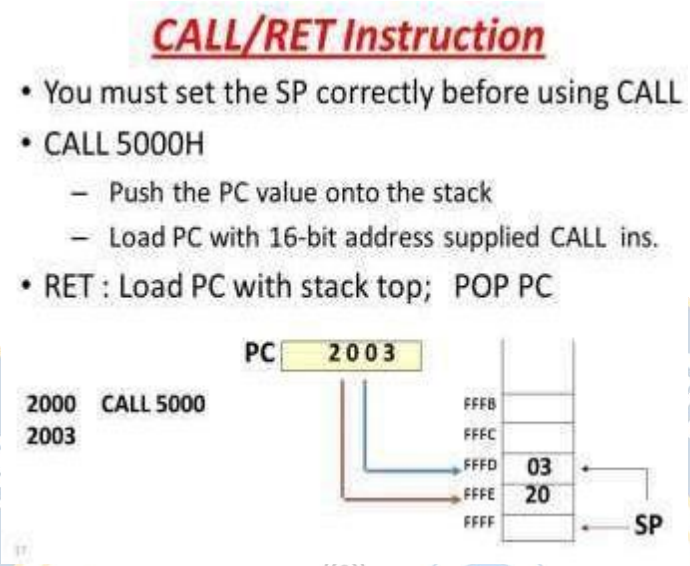
➢ **CALL 4000H (3 byte instruction)**

- When CALL instruction is fetched, the MP knows that the next two Memorylocation contains 16bit subroutine address in the memory.

The RET Instruction

➢ **RET (1 byte instruction)**

- Retrieve the return address from the top of the stack
- Load the program counter with the return address.



The CALL instruction places the return address at the two memory locations immediatelybefore where the Stack Pointer is pointing.

Set the SP correctly BEFORE using the CALL instruction. The RET instruction takes the contentsof the two memory locations at the top of the stack and uses these as the return address. Do not modify the stack pointer in a subroutine.

Number of PUSH and POP instruction used in the subroutine must be same, otherwise, RETinstruction will pick wrong value of the return address from the stack and program will fail.

Passing Data to a Subroutine

- Data is passed to a subroutine through registers.
- Call by Reference:

➢ The data is stored in one of the registers by the calling program and the subroutine uses the value from the register. The register values get modified within the subroutine. Then these modifications will be transferred back to the calling program upon returning from a subroutine

- Call by Value:

➢ The data is stored in one of the registers, but the subroutine first PUSHES register values in the stack and after using the registers, it POPS the previous values of the registers from the stack while exiting the subroutine.

i.e. the original values are restored before execution returns to the calling program.

The other possibility is to use agreed upon memory locations.

– The calling program stores the data in the memory location and the subroutine retrievesthe data from the location and uses it.

PUSH and POP should be used in opposite order.

- There has to be as many POP's as there are PUSH's.
  - If not, the RET statement will pick up the wrong information from the top of the stackand the program will fail.
- It is not advisable to place PUSH or POP inside a loop.

Conditional CALL and RTE Instructions

- The 8085 supports conditional CALL and conditional RTEinstructions.
  - The same conditions used with conditional JUMP instructions can be used
  - CC, call subroutine if Carry flag is set.
  - CNC, call subroutine if Carry flag is not set
  - RC, return from subroutine if Carry flag is set
  - RNC, return from subroutine if Carry flag is not set
- According to Software Engineering practices, a proper subroutine:
  - Is only entered with a CALL and exited with an RTE
  - Has a single entry point
- Do not use a CALL statement to jump into differentpoints of the same subroutine.
  - Has a single exit point
- There should be one return statement from any subroutine.

Write a Program that will display FF and 11 repeatedly on the sevensegment display. Write a 'delay' subroutine and Call it as necessary.

C000: LXISP FFFFC003: MVIA FF C005: OUT 00

C007: CALL 14 20

C00A: MVIA 11

C00C: OUT 00

C00E: CALL 14 20 C011: JMP 03 C0

DELAY: C014: MVIB FFC016: MVIC FF

C018: DCR C C019: JNZ 18 C0C01C: DCR B C01D: JNZ 16 C0C020: RET

UNIT V

CONVERSIONS

BCD to Binary Conversion

The conversion of a BCD number into its binary equivalent employs the principle of positional weighting in a given number.

Problem – Write an assembly language program for converting a 2 digit BCD number to its binary equivalent using 8085 microprocessor.

Examples:

Input: 72H $(0111\ 0010)_2$

Output: 48H (in hexadecimal) $(0011\ 0000)_2$ ((4x16) + (8x1)) =72

Algorithm:

Load the BCD number in the accumulator

Unpack the 2 digit BCD number into two separate digits. Let the left digit be BCD_1 and the rightone BCD_2

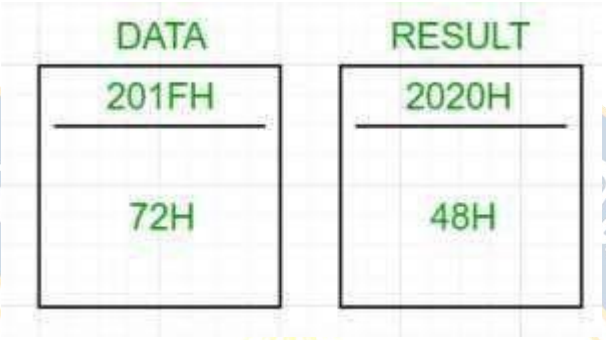
Multiply BCD_1 by 10 and add BCD_2 to it

If the 2 digit BCD number is 72, then its binary equivalent will be 7 x OAH + 2 = 46H + 2 = 48H

Steps:

- Load the BCD number from the memory location (201FH, arbitrary choice) into the accumulator.
- Temporarily store the accumulator's value in B.
- Obtain BCD2 by ANDing the accumulator with 0FH and store it in C.
- Restore the original value of the accumulator by moving the value in B to A. AND the accumulator with F0H.

- If the value in the accumulator equals 0, then BCD_2 is the final answer and store it in the memory location, 2020H (arbitrary).
- Else, shift the accumulator to right 4 times to obtain BCD_1. Next step is to multiply BCD_1 by 0A.
- Multiplication: Move BCD_1 to D and initialise E with 0AH as the counter. Clear the accumulator to 0 and add D to it E number of times.
- Finally, add C to the accumulator and store the result in 2020H



ADDRESS	LABEL	MNEMONIC
2000H		LDA 201FH
2001H		
2002H		
2003H		MOV B, A
2004H		ANI 0FH
2005H		
2006H		MOV C, A
2007H		MOV A, B
2008H		ANI F0H
2009H		
200AH		JZ SKIPMULTIPLY
200BH		
200CH		
200DH		RRC

200EH		RRC
200FH		RRC
2010H		RRC
2011H		MOV D, A
2012H		XRA A
2013H		MVI E, 0AH
2014H		
2015H	SUM	ADD D
2016H		DCR E
2017H		JNZ SUM
2018H		
2019H		
201AH	SKIPMULTIPLY	ADD C
201BH		STA 2020H
201CH		
201DH		
201EH		HLT

Binary-to-BCD conversion

The microprocessor processes data in binary form but data that is displayed is in BCD form. Hence, we require to convert the binary data to BCD.

Here we are taking a number from the memory, and initializing it as a counter. Now

In most microprocessor-based products, numbers are displayed in decimal. However, if data processing inside the microprocessor is performed in binary, it is necessary to convert the binary results into their equivalent BCD numbers just before they are displayed. Results are quite often stored in R/W memory locations called the **Output Buffer.**

The conversion of binary to BCD is performed by dividing the number by the powers of ten; the division is performed by the subtraction method.

For example, assume the binary number is

$$1\ 1\ 1\ 1\ \ 1\ 1\ 1\ 1_2\ (FFH) = 255_{10}$$

To represent this number in BCD requires twelve bits or three BCD digits, labeled here as BCD_3 (MSB), BCD_2, and BCD_1 (LSB).

$$= 0\ 0\ 1\ 0\ \ 0\ 1\ 0\ 1\ \ 0\ 1\ 0\ 1$$
$$BCD_3 \quad BCD_2 \quad BCD_1$$

The conversion can be performed as follows:

Step 1: If the number is less than 100, go to Step 2; otherwise, divide by 100 or subtract 100 repeatedly until the remainder is less than 100. The quotient is the most significant BCD digit, BCD_3.

Step 2: If the number is less than 10, go to Step 3; otherwise divide by 10 repeatedly until the remainder is less than 10. The quotient is BCD_2.

Step 3: The remainder from Step 2 is BCD_1.

Example			Quotient
255			
−100	=	155	1
−100	=	55	1
		BCD_3 =	2
55			
−10	=	45	1
−10	=	35	1
−10	=	25	1
−10	=	15	1
−10	=	05	1
		BCD_2 =	5
		BCD_1 =	5

These steps can be converted into a program as illustrated next.

Problem Statement

A binary number is store dat location 800H. Convert the number into its BCD equivalent and store it to the memory location 8050H.

Discussion

in each step of this counter we are incrementing the number by 1, and adjust the decimal value. By this process we are finding the BCD value of binary number or hexadecimal number. We can use INR instruction to increment the counter in this case but this instruction will not affect carryflag, so for that reason we have used ADI 10H.

Program

Address	HEX Codes	Labels	Mnemonics	Comments
F000	21, 00, 80		LXI H,8000H	Initialize memory pointer
F003	16, 00		MVI D,00H	Clear D- reg for Most significant Byte
F005	AF		XRA A	Clear Accumulator
F006	4E		MOV C, M	Get HEX data
F007	C6, 01	LOOP	ADI 01H	Count the number one by one
F009	27		DAA	Adjust for BCD count
F00A	D2, 0E, F0		JNC SKIP	Jump to SKIP
F00D	14		INR D	Increase D
F00E	0D	SKIP	DCR C	Decrease C register
F00F	C2, 07, F0		JNZ LOOP	Jump to LOOP
F012	6F		MOV L, A	Load the Least Significant Byte
F013	62		MOV H, D	Load the Most Significant Byte
F014	22, 50, 80		SHLD 8050H	Store the BCD
F017	76		HLT	Terminate the program

BCD-TO-SEVEN-SEGMENT-LED CODE CONVERSION

Many times 7-segment LED display is used to display the results or parameters in the microprocessor system. In such cases we have to convert the result or parameter in 7-segment code. This conversion can be done using look-up technique. In the look-up table the codes of the digits (0-9) to be displayed are stored sequentially in the memory. The conversion program locates the code of a digit based on its BCD digit. Let us see the program for BCD to common cathode 7-segment code conversion.

Find the 7-segment codes for given numbers.

Statement : Find the 7-segment codes for given 5 numbers from memory location 6000H and store the result from memory location 7000H.

Source Program

```
        LXI H, 6200H ; Initialize lookup table pointer
        LXI D, 6000H ; Initialize source memory pointer
        LXI B, 7000H ; Initialize destination memory pointer
BACK:   LDAX D       ; Get the number
        MOV L, A     ; A point to the 7-segment code
        MOV A, M     ; Get the 7-segment code
        STAX B       ; Store the result at destination memory
                     ; location
        INX D        ; Increment source memory pointer
        INX B        ; Increment destination memory pointer
        MOV A, C
        CPI 05H      ; Check for last number
        JNZ BACK     ; If not repeat
        HLT          ; End of program
```


BCD ADDITION

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Sum equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

```
   6     0 1 1 0    ← BCD for 6
+  3     0 0 1 1    ← BCD for 3
-------  ---------
   9     1 0 0 1    ← BCD for 9
```

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

```
    6     0 1 1 0     ← BCD for 6
+   8     1 0 0 0     ← BCD for 8
-------   ---------
   14     1 1 1 0     ← Invalid BCD number     (1110) > 9
```

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

```
    6     0 1 1 0     ← BCD for 6
+   8     1 0 0 0     ← BCD for 8
-------   ---------
   14     1 1 1 0     ← Invalid BCD number
+         0 1 1 0     ← Add 6 for correction
-------------------------
  0 0 0 1   0 1 0 0   ← BCD for 14
    1         4
```

After addition of 6 carry is produced into the second decimal position.

Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD

```
    8          1 0 0 0      ← BCD for 8
+   9          1 0 0 1      ← BCD for 9
-------   -----------------
   17     0 0 0 1 0 0 0 1    ← Incorrect BCD result
```

In this case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown.

```
    8          1 0 0 0      ← BCD for 8
+   9          1 0 0 1      ← BCD for 9
-------   -----------------
   17     0 0 0 1 0 0 0 1    ← Incorrect BCD result
      +   0 0 0 0 0 1 1 0    ← Add 6 for correction
-------------------------
          0 0 0 1 0 1 1 1    ← BCD for 17
```

Going through these three cases of BCD addition we can summarise the BCD addition procedure as follows :

1. Add two BCD numbers using ordinary binary addition.
2. If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

3. If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

The 8085 supports DAA (Decimal Adjust Accumulator) instruction for adjusting the result of addition to the BCD number. (See chapter 2 for DAA instruction).

Add two 2-digit BCD numbers.

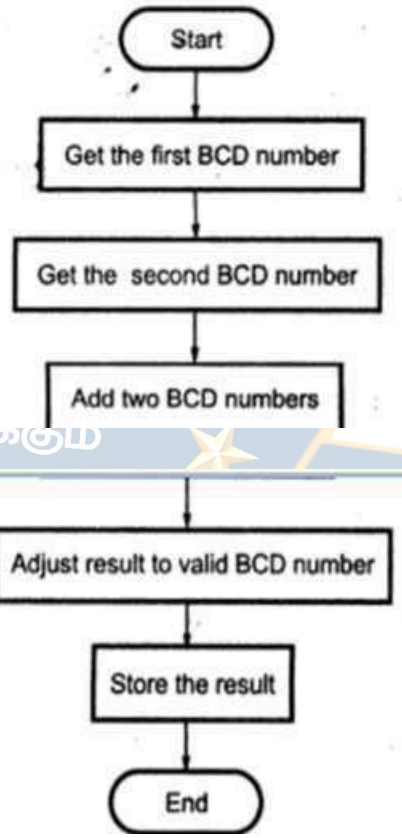
Statement : Add two 2 digit BCD numbers in memory location 2200H and 2201H and store the result in memory location 2300H.

Sample problem

```
(2200H)  = 39
(2201H)  = 45
Result   = (2300H) = 39 + 45
         = 7E + 6 = 84
```

(lower nibble is greater than 9 so add 6)

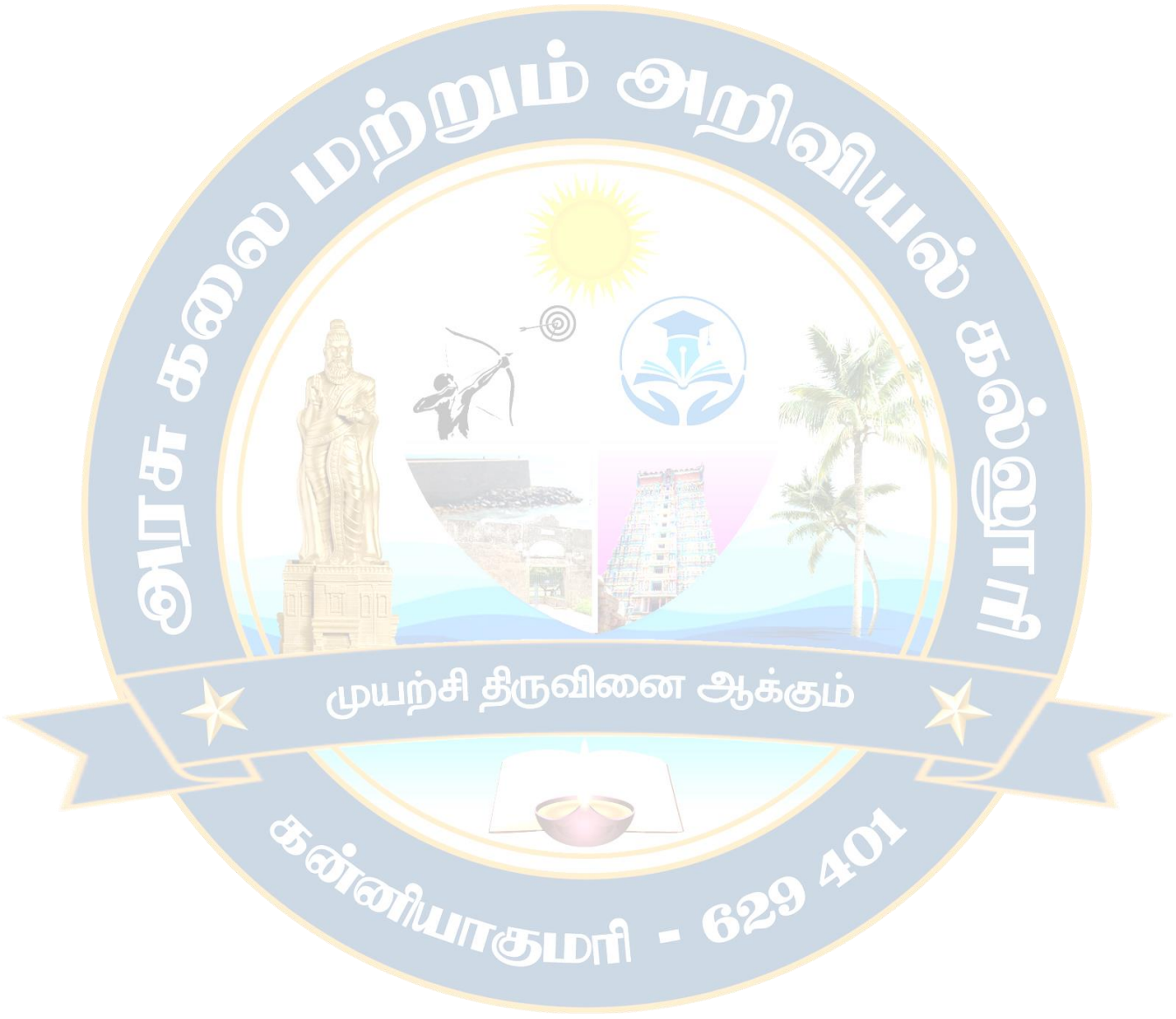
Flowchart



Source program

```
LXI  H, 2200H ; Initialize pointer
MOV  A, M     ; Get the first number
INX  H        ; Increment the pointer
ADD  M        ; Add two numbers
DAA           ; Convert HEX to valid BCD
STA  2300H    ; Store the result
HLT           ; Terminate program
                execution
```

Add two 4-digit BCD numbers.


அரசு கலை மற்றும் அறிவியல் கல்லூரி
முயற்சி திருவினை ஆக்கும்
கன்னியாகுமரி - 629 401

BCD SUBTRACTION

Statement : Subtract the BCD number stored in E register from the number stored in the D register.

Source Program

```
MVI A,99H
SUB E          ; Find the 99's complement of subtrahend
INR A          ; Find 100's complement of subtrahend
ADD D          ; Add minuend to 100's complement of subtrahend
DAA            ; Adjust for BCD
HLT            ; Terminate program execution
```

MULTIPLICATION

Problem – Multiply two 8 bit numbers stored at address 2050 and 2051. Result is stored at address 3050 and 3051. Starting address of program is taken as 2000.

Example –

Input Data ⇨	07	43
Memory Address ⇨	2051	2050

Output Data ⇨	01	D5
Memory Address ⇨	3051	3050

Algorithm –

1. We are taking adding the number 43 seven(7) times in this example.
2. **As the multiplication of two 8 bit numbers can be maximum of 16 bits** so we needregister pair to store the result.

Program –

MEMORY ADDRESS	MNEMONICS	COMMENT
2000	LHLD 2050	H←2051, L←2050
2003	XCHG	H↔D, L↔E
2004	MOV C, D	C←D

MEMORY ADDRESS	MNEMONICS	COMMENT
2005	MVI D 00	D←00
2007	LXI H 0000	H←00, L←00
200A	DAD D	HL←HL+DE
200B	DCR C	C←C-1
200C	JNZ 200A	If Zero Flag=0, goto 200A
200F	SHLD 3050	H→3051, L→3050
2012	HLT	

Subtraction with carry

Write an assembly language program in 8085 microprocessor to subtract two 16 bit numbers.

Assumption –Starting address of program: 2000Input memory location: 2050, 2051, 2052, And 2053

Output memory location: 2054, 2055Example –

INPUT: (2050H) = 19H(2051H) = 6AH

(2052H) = 15H(2053H) = 5CH

OUTPUT:

(2054H) = 04H

(2055H) = OEH

Algorithm –

* Get the LSB in L register and MSB in H register of 16 Bit number.
* Exchange the content of HL register with DE register.
* Again Get the LSB in L register and MSB in H register of 16 Bit number.
* Subtract the content of L register from the content of E register.
* Subtract the content of H register from the content of D register and borrow from
* Previous step.

* Store the result in memory location.

Program –

MEMORYADDRESS	MNEMONICS	COMMENTS
2000	LHLD 2050	Load H-L pair with address 2050
2003	XCHG	EXCHANGE H-L PAIR WITH D-EPAIR
2004	LHLD 2052	Load H-L pair with address 2052
2007	MVI C, 00	C<-00H
2009	MOV A, E	A<-E
200A	SUB L	A<-A-L
200B	STA 2054	2054<-A
200E	MOV A, D	A<-D
200F	SBB H	SUBTRACT WITH BORROW
2010	STA 2055	2055<-A
2013	HLT	TERMINATES THE PROGRAM