SEMESTER 2.

Micro processor

Syllabus

Unit 1: Introduction to Micro processor.
* 8085 programing model.
* Instruction formats, addressing models
* Microprocessor architecture and its application.
* 8085 mpu pinout and signal
* function block diagram.

Unit 2: 8085 instructions ~~set~~ and classification.
* Data transfer instructions.
* Arithmatic and logical instructions.
* Branching and machine control instruction programs technique
* Looping, counting and indexing.
* Writing assembly level program.

Unit 3: Stack (chapter 9 on text book)
* Push and pop ~~subroutine~~ subroutine
* Call and return * Restart, conditional call and return instruction.
* 8 bit, multibyte ~~and~~ BCD multiplication, Addition, Subtraction, Division
* ~~8 bit and BCD Division.~~

Unit 4: Conversions. (chapter 8 on text book)
* BCD to binary and Binary to BCD conversions.
* ASCII to BCD and BCD to ASCII conversion.
* Binary to ASCII & ASCII to binary conversion
* Counter and time delays

Unit 5 : 8085 ~~Instr~~ intrupts. (Chapter 12 on Text b

* EI, DI, TRAP, RST, SIM, RIM.
* DIRECT MEMORY ACCESS
* Memory interface, memory maped I/O

MICROPROCESSOR AND ITS APPLICATION.

1: Microprocessor, computer and Assembly language.

* The microprocenor is the programable integerated device that has computing and decision making capacity similar to that of the CPU of a computer.
* It is a multipurpose, programable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory.
* The usual procedure of microprocess ~~is called~~ can also perform as computer, that is it accepts the data and processes data accordg to the instructions ~~result~~ and provides results as output.

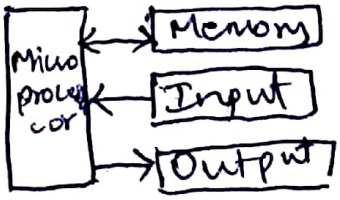
A programable machine can be represent in four components;

- microprocenor
- memory ~~input~~
- input
- output.

~~A plus~~ These ~~four~~ four components are connected to perform a task, thus forms a system.

- The physical entities of system is known as hardware
- A set of instructions written for microprocess to perform task known as programs, and the group of programs is called as software.

* Each microprocessor has a fixed set of instructions in the form of binary patterns called machine language.
* The binary instruction are given in abbreviated names called mnemonics. This forms the assembly language for the computer.

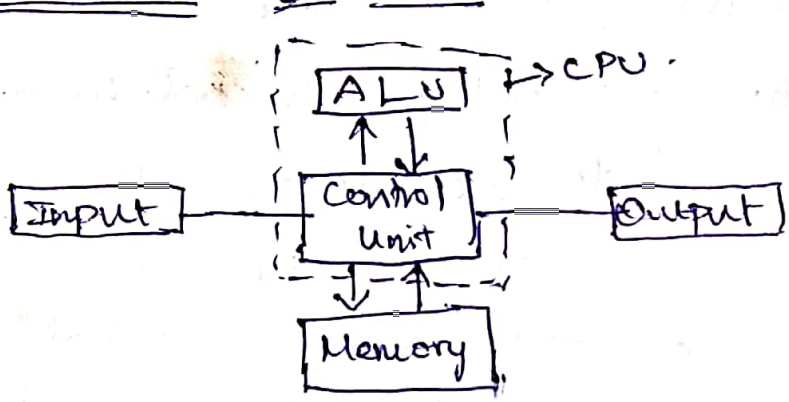


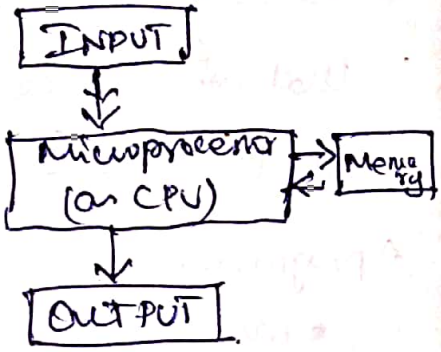
MICROCONTROLLER

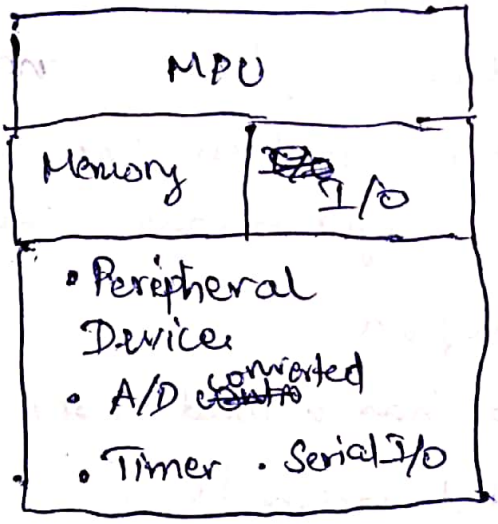
MEMORY:

Memory is like a pages of a ~~book~~ notebook with space for a fixed number of binary numbers on each line. Typically each line is a 8-bit register that can store eight binary bits, and several of these registers are arranged in the sequence called memory. These registers are grouped into power of two.

MICROPROCESSOR AS CPU:-







Micro controller

* CPU reads instructions from memory, and perform the input and output task specified.
* It communicates with I/o devices either to accept or execute data.
* However the timing of the communication process is controlled by group of circuits called control unit.
* As the CPU technology brought into the single chip called microprocessor, the replacement of CPU with microprocessor for a microprocessor unit (MPU) and the computer used is called as Microprocessor computer.
* It implies complete processing unit with necessary control signals.
* Some of the signals need to generate using discrete devices to make the complete functional unit of MPU, because of the limited number of pins in MP package
* As 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. This is known as microcontroller unit.

MICROPROCESS

MICROPROCESSOR:

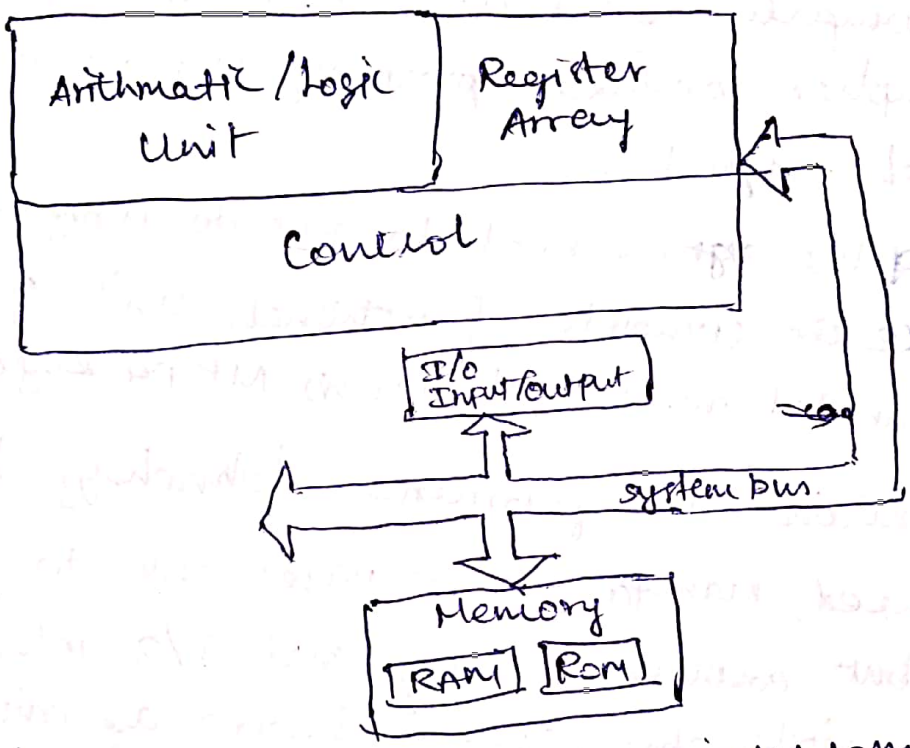
It is the clock ~~based~~ driven semiconductor device consisting of electronic ~~ckt~~ logic circuits manufactured by using either LSI and VLSI.

CPU

Implemented on one or more circuit board to perform computing functions

Microprocessor.

Includes all the circuits and control unit that are ~~now~~ brought to a single chip.

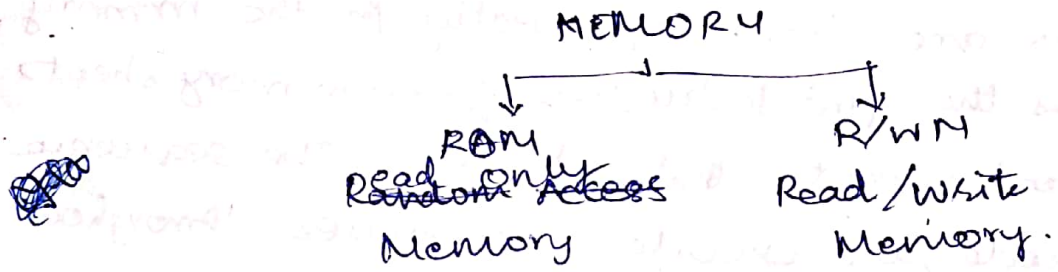


* Arithmetic / Logic Unit: This is where various computing functions are performed on data. ALU Performs arithmatic operations like addition, substraction and Logical operation like AND, OR and EX-OR.

* Register Array: It consists of various data storage registers that are named as letter as B, C, D, E, H, L. These Registers are primarily used to store data temporily and retrieves them.

*Control Unit: The control unit provides the necessary timing and control signals to all the operations in the microcomputer.

* Memory: Memory stores such binary information as instructions and data, and provides that information to the processor wherever necessary. To execute programmes, the MP reads instruction and data from memory and performs the computing operations in its ALU section. Results are stores either transfered to output section for display or store for later use.



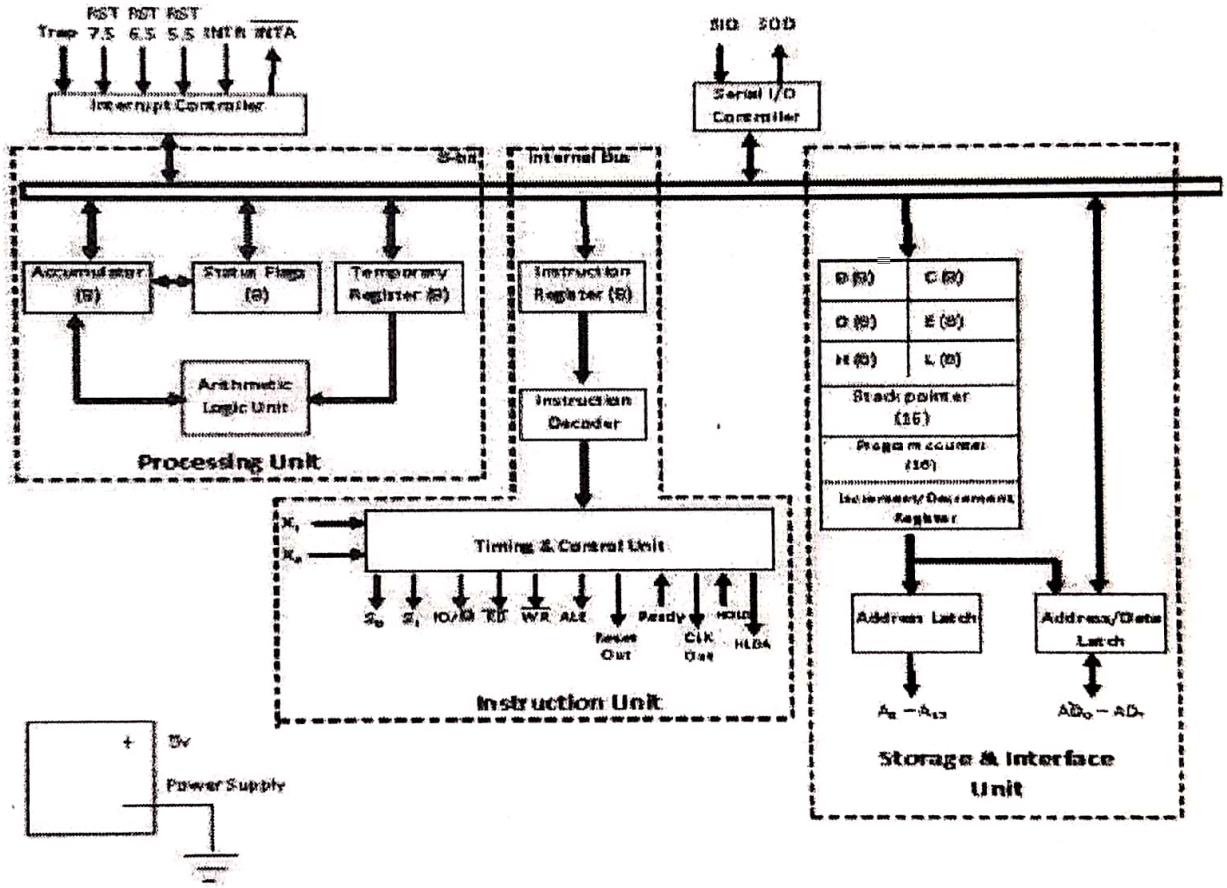
*I/O : The tertiary component of a MP-based system is Input / Output. Input devices such as switches, keyboard, A/D (analog to digital) converter transfers binary information from outside world to M.P and Output executes the processed information from M.P to outside world. These Input and output devices with other components are combinedly known as perifherals.

SYSTEM BUS: It is the connection path between the microprocessor and peripherals. It is nothing but the group of wires carrying bits. All peripherals share the same bus. The MP communicates with only one peripheral at the time. So timing is provided by the control unit of the Microprocessor.

How does it work?

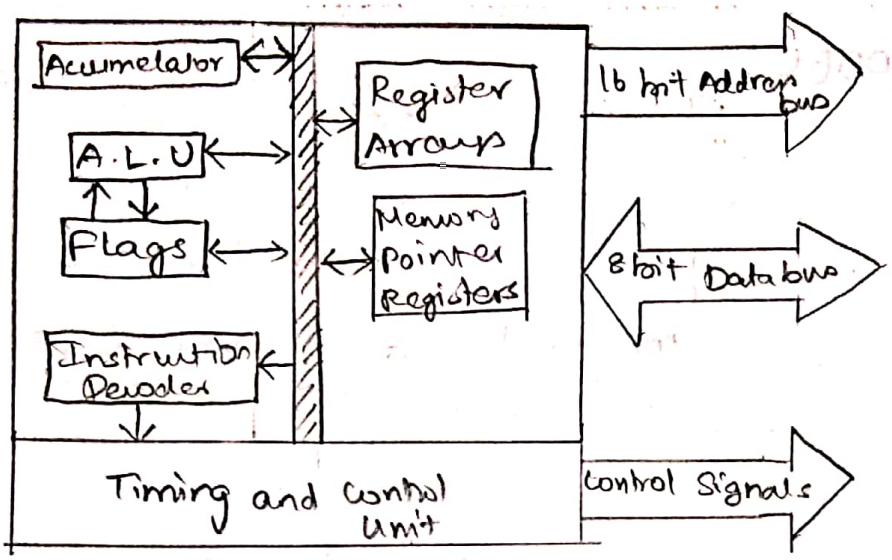
- When the microprocessor is given a command to execute the program, it reads and executes one instruction at a time and finally sends the result to the seven-segment LED's for display.
- The instructions are stored sequentially in the memory. The MP fetches the first instruction from memory sheet decoded it, and execute that instruction. The sequence of fetch, decode, and execute is continues throughout the MP untill it instruts to stop.
- During entire program, MP uses data buses to fetch the binary instructions and data from the memory.
- It uses registers to store the data and instructions temporarily and also it sends to ALU to perform a task assigned in ALU section. finally, the result is sends out by MP in form of binary, using bus lines to the seven-segment LED.

Block Diagram of 8085



Gursharan Singh Tatla
professorgstatla@gmail.com

2. 8085 ASSAMBLY LANGUAGE & PROGRAMING

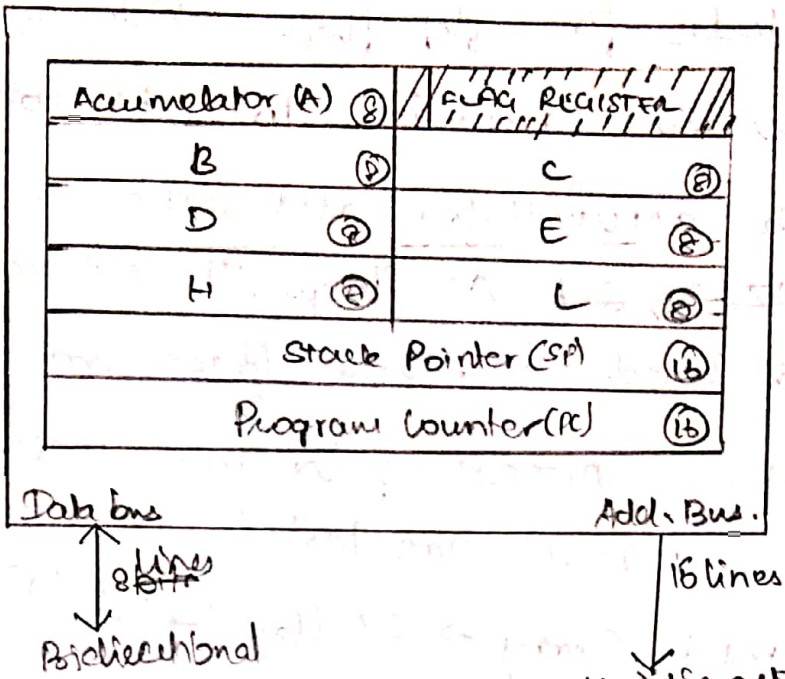


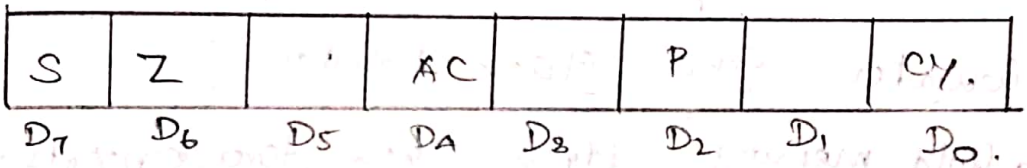
8085 HARDWARE MODEL

Hardware Model:

* Hardware shows two segment. One has 8 bit register called accumelator, ALU, flags and instruction decoder. While other has register arrays, Memory pointer registers that carries 8 bit and 16 bits respectively. Both section are connected to the common interconnection called internal bus.
* Arithmatic and logical processes are done in ALU and are stored in accumelator and flipflops called flag. This flags are set or reset to reflect the results.
* There are three external buses; 16 bit addren bus, 8 bit Data bus and control bus.
  - → Memories 16 bit add. bus to send out memory address.
  - → 8 bit data is used to send data provided to external.
  - → Control signals for timing signals.

8085 Programing model :





S - sign
CY - Carry
Z - Zero
AC - auxiliary Carry
P - Parity

Programing model consists of some segments of the ALU and registers. This model does not reflect the physical structure of the 8085 but includes the information that is critical in writing assembly language programs.

Registers : 8085 programming model uses 6 registers to store bits of memory. Each register consists of 8-bit and are labled as B, C, D, E, H, L. They can also combined as BC, DE, HL to perform as 16 bit registers. The programmer can use this to store or copy the data into it.

Accumelator : Accumelator is a part of ALU which is a 8bit register. This register is used to store 8 bit data and to perform arithmatic and logic operations. The results are also stored here.

Flags:

All 8085 includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are Z, CY, S, P, AC flags. The MP uses these flags to test data conditions. These flags are critical importance in dicision making process of the microprocessor. The conditions of the flags (set/reset) are tested through software instructions

Ex., JC → Jump to Carry → CY flag is set (1)

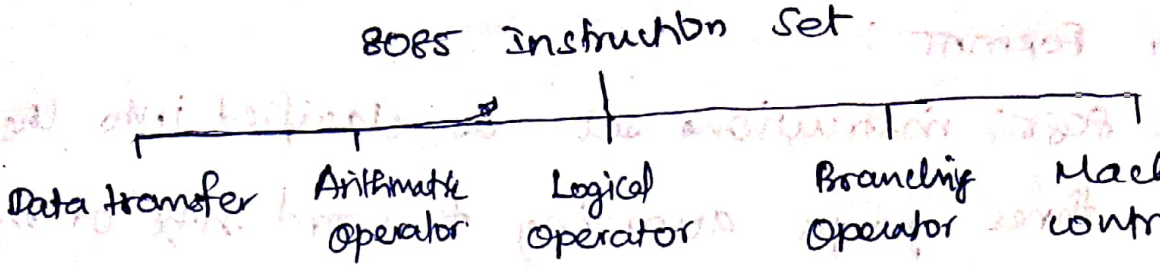
Z - Zero CY - Carry S - Sign P - Parity AC - Auxiliary carry.

Program Counter and Stack Pointer:

To hold memory address, these two concepts were brought

* Program counter is used by M-P to sequence the execution of the instructions. It has 16-bits of memory
* When a byte (machine code) is fetched, the program counter is increased by one to point to the next memory location.
* Stack pointer is used as the memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16 bit address in the stack pointer.

INSTRUCTION CLASSIFICATION.



Data transfer: Transfers (copies) data from source location to its distination.

Type: • Between Registers.
• Specific data to registers.
• Between Memory location and a register.
• Between an I/O device and accumulator.

Arithmatic operators:

Type: • Addition • Subtraction • Increment/decrement

Logical operators:

Type: • AND, OR, EX-OR. (Results are stored in accumulator)
• Rotate (shift left/right to the next position)
• Compare (memory location is compared for equality/non-equality)
• Complement (all 0's replaced as 1 and vice versa)

Branching operators:

Type: • Jump • Call • Return • Restart.

Machine control operators:

Type: • Halt • Interept • Do nothing.

INSTRUCTION, DATA FORMAT AND STORAGE:

INSTRUCTION FORMAT:

The 8085 instructions set is classified into the following three groups according to word size or byte size.

1) 1 byte instructions. 2) 2 byte instructions.

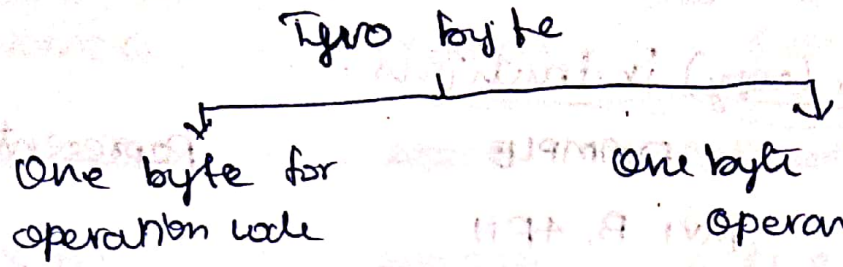
3) 3 byte instructions.

One byte instructions:

A 1 byte instruction includes the opcode and the operand in the same byte.

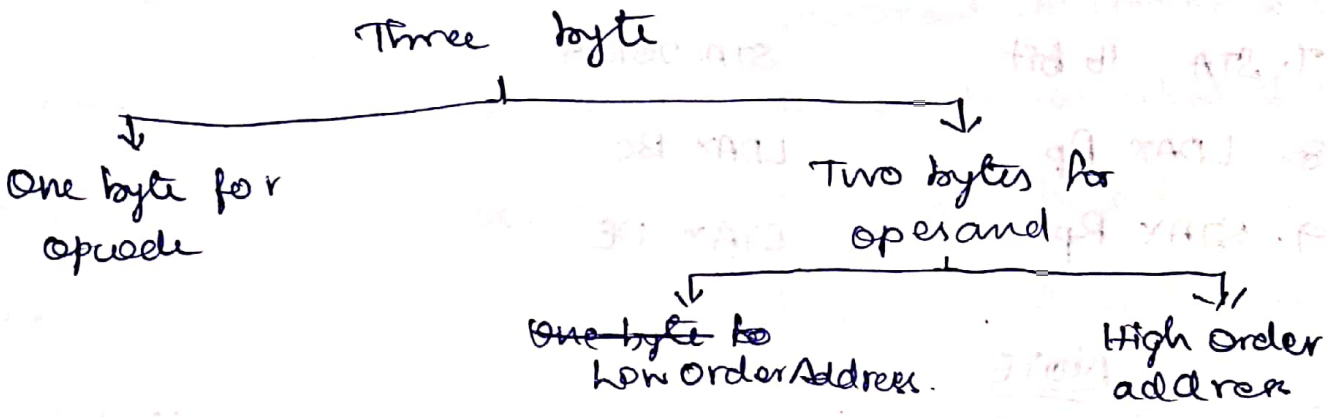
Opcode	Operand	Hex code	Task.
MOV	C, A	4FH	Copies the content of the accumulator (A) in register C.
ADD	B	80H	Adds the content in B to the contents in Accumulator.
CMA		2FH	Invert each bit in accumulator

Two byte Instruction:



Opcode	Operand	Hex code	Task
MVI	A, 32H	3E (I byte) 32 (II byte)	Loads an 8 bit data to accumulator.
MVI	B, F2H	06 (I byte) F2 (II byte)	Loads an 8 bit data from H to B.

Three byte Instruction



Opcode	Operand	Hex code	Task
LDA	2050 H (L H)	3A (I) 20 (II)(L) 50 (III)(H)	Load contents of memory 2050 H to A
JMP	2085 H	C3 (I) 85 (II)(L) 20 (III)(H)	Transfers the program sequence to Memory Location 2085 H.

8085 INSTRUCTION SET

A - Data transfer (copy) instructions.

	MNEMONICS	EXAMPLE	Representation
1.	MVI R, 8bit	MVI B, 4FH	
2.	MOV Rd, RS*	MVI C, B	
3.	LXI Rp, 16 bit	LXI BC, 2050H	
4.	OUT 8bit	OUT 01 H	
5.	IN 8bit	IN 07H	
6.	LDA 16bit	LDA 2050H	
7.	STA 16 bit	STA 2070 H	
8.	LDAX Rp	LDAX BC	
9.	STAX Rp.	STAX DE	

NOTE

R -> 8085 8bit register (A, B, C, D, E, H, L)

M -> Memory ~~register address~~ address

Rs -> Register Source. Rd - Register Destination

Rp -> Register destination.

( ) -> contents of

10.	MOV R, M	MOV B, M
11.	MOV M, R	MOV M, B

B. Arithmatic instructions.

	MNEMONICS	EXAMPLE	REPRESENTATION.
1.	ADD R.	ADD B	Adds contents in B with accumulator
2.	ADI 8bit	ADD 37H	Adds a 8bit data to Accumulator
3.	ADD M	ADD M	Adds a memory (usually HL) with A
4.	SUB R.	SUB C	Subtracts C with accumulator
5.	SUI 8bit	SUI 7AH	Subtracts A with a 8bit data
6.	SUB M	SUB M	Subtracts A with memory.
7.	INR R.	INR D	Increment the contents of D
8.	INR M	INR M	Increments the content of Mem
9.	DCR R	DCR E	decrement the contents of E
10.	DCR M	DCR M	decrement the contents of Mem
11.	INX Rp	INX H	Increment the contents of register pair H.
12.	DCX Rp.	DCX B.	Decrement the contents of register pair of B.

(C) Logic and Bit manipulation instructions.

• AND • OR • X-OR • compare • Rotate bits.

	MNEMONICS	EXAMPLE	Representation.
1.	ANA R	ANA B	
2.	ANI 8bit	ANI 2FH	
3.	ANA M	ANA M	

	MNEMONICS	EXAMPLE	REPRESENTATION
4	ORA R.	ORAE	
5.	ORA 8bit	ORA 2FH	
6.	ORA M	ORA M	
7.	XRA R	XRA B	
8.	XRI 8bit	XRI 40H	
9.	XRA M	XRA M	
10.	CMP R.	CMP B	
11.	CPI 8bit	CPI 4FH	

D. Branch instructions:

	MNEMONICS	EXAMPLE	REPRESENTATION
1.	JMP 16 bit add.	JMP 2050H	Change the program sequence to 2050H Memory address.
2.	JZ 16 bit add.	JZ 2080H	Change the program sequence to address of zero at 2080H which is set.
3.	JNZ 16 bit add.	JNZ 2070H	Change the program sequence to work at Add. 2070H which is set
4.	JC 16 bit add	JC 2025 1H	Jump the sequence to 2025H adds. at where carry is set
5.	JNC 16bit add.	JNC 2030H	Jump the sequence to 2030H adds. at where carry is reset
6.	CALL 16 bit add.	CALL 2075H	Change the program sequence to the subroutine of program.
7.	RET	RET	Return the sequence from subroutine to the main program (after completing the task assigned by subroutine).

F. Machine Control Instructions:

1. HLT	HLT	Halt
2. NOP	NOP	Blocks the operation process (Do not perform any operation).

8085 ASSEMBLY LANGUAGE:

* An instruction (based on machine language) is an binary format that senses from input to memory at which the memory reads the binary pattern and make to process.
* However these instructions are in need to execute in hexadecimal code, it is still difficult to understand a written program in these format, for the programmers.
* Thus the manufacturer of MP has derived a symbolic code for each instruction, called mnemonic. It consists of letters that suggests the operations to be performed by that instruction.
* These elements were studied under assembly language through 8085 microprocessor.
* The Machine level and Assembly level languages were microprocessor specific and also considered as low-level language. But the difference and a better point of suggestion is that machine level is binary based and assembly uses mnemonics

~~3. Microprocessor Architecture and Microcomputer system~~

which translates into binary pattern accurately so that the process becomes correct and expected output can come.

Addressing Modes

1. Immediate Addressing mode: (Pass the butter)

In this, the source operand is always a data.

If 8 bit → 2 byte instruction. Example: MVI B 45

16 bit → 3 byte instruction

45 → direct data
B → register

2. Register Addressing mode: (Pass the bowl)

The data to be operated is always inside the registers and register is operant. operation is performed from registers to registers.

Example: mov A, B A & B are registers.

3. Direct Addressing mode (combination number 17 on the menu)

Here, the data to be operated is available inside a memory location which is directly treated as operand.

Example: LDA 2050

2050 → memory location at which the input (data) stores in

4. Register Indirect Addressing mode: (I will have what Sujeeha has)

Here, the data which is inside the memory location is indirectly specified to a register pair.

Example:

LXI H 9570

(Load immediate the HL pair with address of location 9570)

5. Implicit Addressing mode:

Here, the operand is hidden and the data to be operated is available in instruction itself.

Example: CMA, RRC, RLC

CMA → complement
RRC, RLC → Rotates the data of A by 1 bit.

LOOPING, COUNTING & INDEXING

Looping:

The programing technique that used to repeat the tasks is called looping. Loop is set up by instructing the MP to change the sequence of program execution and perform the task again. This is done mostly by using Jump instruction.

Loop ← Conditional
Loop ← Unconditional (Continuous)

(i) Conditional loop

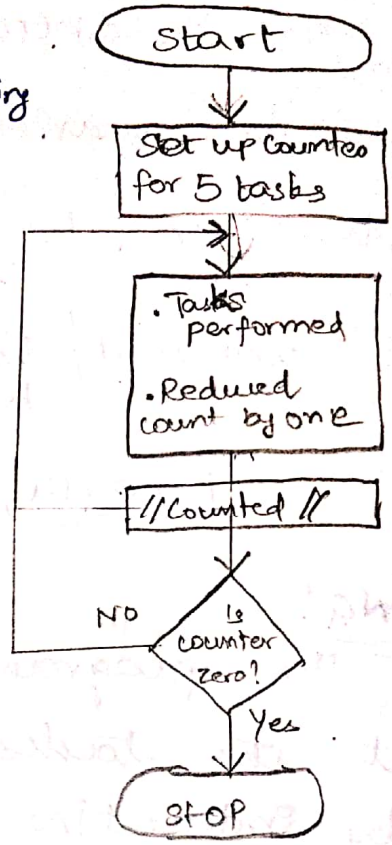
It is set up by using conditional Jump statements, which carries with flags such as Zero, Carry, etc; and repeats the task if the conditions are satisfied. It includes counting and Indexing.

* Counting has the specific application. It is needed for the M.P to repeat times is done. When it is completed, it needs a flag.

Functions (General) of Looping and counting:

* Counter is set up by loading an appropreate count in a register
* Counting is performed by either increasing or decresing the counter.
* Loop is set up by a conditional jump instruction
* End of counting is indicated by a flag.

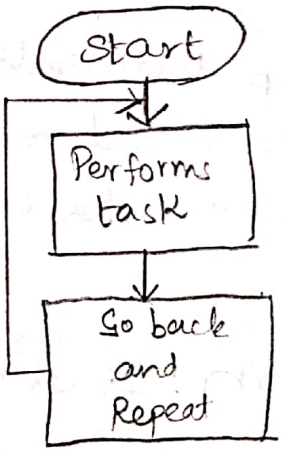
FLOW CHART



Counting and indexing:

- Pointing of reference object with sequential numbers. This is called indexing
- Data types are stored in memory location and those data bytes are refered by their memory locations.

(ii) Continuous loop:



A continuous loop is set up by using the unconditional loop Jump instruction (JMP instruction only) shown in this flowchart

~~The program with~~

However, the program with this cannot stoped and repeated again untill ~~re~~ ~~restart~~ the system is reset.

BACK TO UNIT I

Micro processor Architecture and its Operation.

The microprocessor is a programable digital device designed with registers, flip flops and timing elements. 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 various functions performed by microprocessor are classified into 3 general categories.

* Initiated operations.
* External operations
* Peripheral operations (or external operations).

I. Initiated Operation:

The MPU performs primarily four basic operations. They are,

- Memory Read : Reads data from memory
- Memory write : Writes data from memory
- I/O Read : Accepts data from input devices
- I/O write : Sends data to output devices.

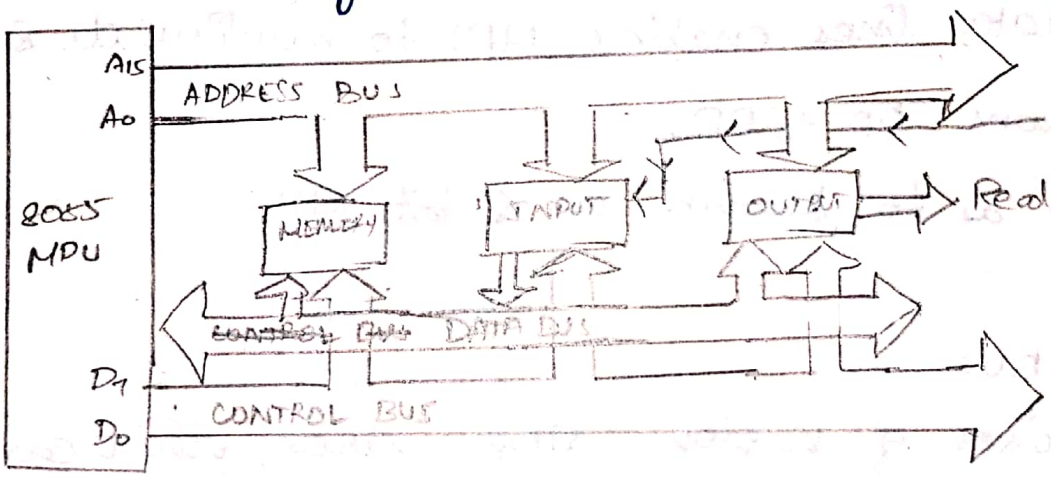
All these operations are part of communication process between the MPU and peripheral devices. To communicate with a peripheral devices, the MPU needs to perform the following steps:

step 1: Identify the peripheral or memory location.

step 2: Transfers binary information.

step 3: Provide timing ~~or~~ synchronising signals.

The 8085's MP uses three busses as described in the below diagram:



*ADDRESS BUS:

The address bus is a group of 16 bits identified as A0 to A15. It is unidirectional which the group of bits flows from MPU to peripheral devices. ~~The~~ MPU ~~uses the address bus to perform one~~

Performance:

. Identifies the (binary) peripheral or a memory location

In computer devices, the peripheral location is identified by memory location as; similar to postal address ~~of~~ a house.

Most of the 8 bit microprocessor has 16 bit address lines.

DATA BUS:

* It is a group of 8 bit lines.
* Bidirectional
* flows from MPU to peripheral devices and memory
* Performance: Transfers the binary information (collection of data)
* Eight data lines enables MPU to manipulate 8 bit data from 00 to FF.
* 8085 so as to known as 8 bit MP.

CONTROL BUS

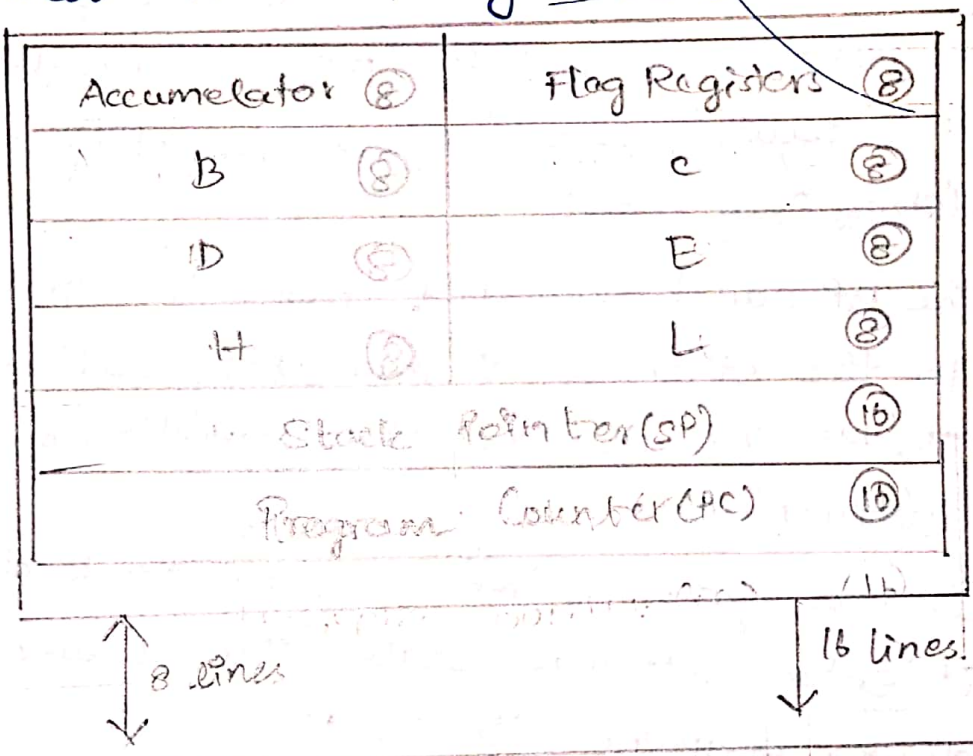
* It comprises of various single lines that carry synchronous signals.
* Performance: To provide synchronous signals to peripheral devices.
* These are individual lines to provide a pulse to indicated MPU operation.
* MPU generates specific control signals for every operation it performs.
* Those signals are use to identify the device type which MPU intends to communicates.

II. Internal Data operations and the 8085 registers:

It determines how and what operations can be performed with the data.

Operations:

* Stores 8 bit data
* Performs arithmatic and logical operations.
* Tests for condition.
* Sequence the execution of instructions.
* Store data temporarily during execution in the defined R/W memory location called the stack.



. Example:

```
LDA   2000H
MVI   B,76H
MVI   A,F12H
ADD   B
HLT
```

Mechanism with Example:

- First loads an input with addren 2000H in register.
- The value 76H is inserted to Register B.
- Then the value F2H is stored in accumelator.
- The value on B register is added to the data in accumelator and stores result in the same register (A).
- Finally halts the instructions.

III Peripheral (External) Initiated operation:

External devices or signals can initialise the following operations, for which individual pins on MP chip are assigned:

- Reset
- Interept
- Ready
- Hold

* **Reset**: When this pin is activated, all the internal operations are suspended and PC (program counter) is cleared. The execution can boring again at the zero memory address (say 0000H).

* **Interept**: The MP can be Intereped from the normal execution to the other set of instruction called service routine. The MP then resumes after completion of such set of instruction.

* **Ready**: If this signal is high, then MP starts of initiate, else it enters into wait state. This is used to synchronise slower peripheral with the M.P.

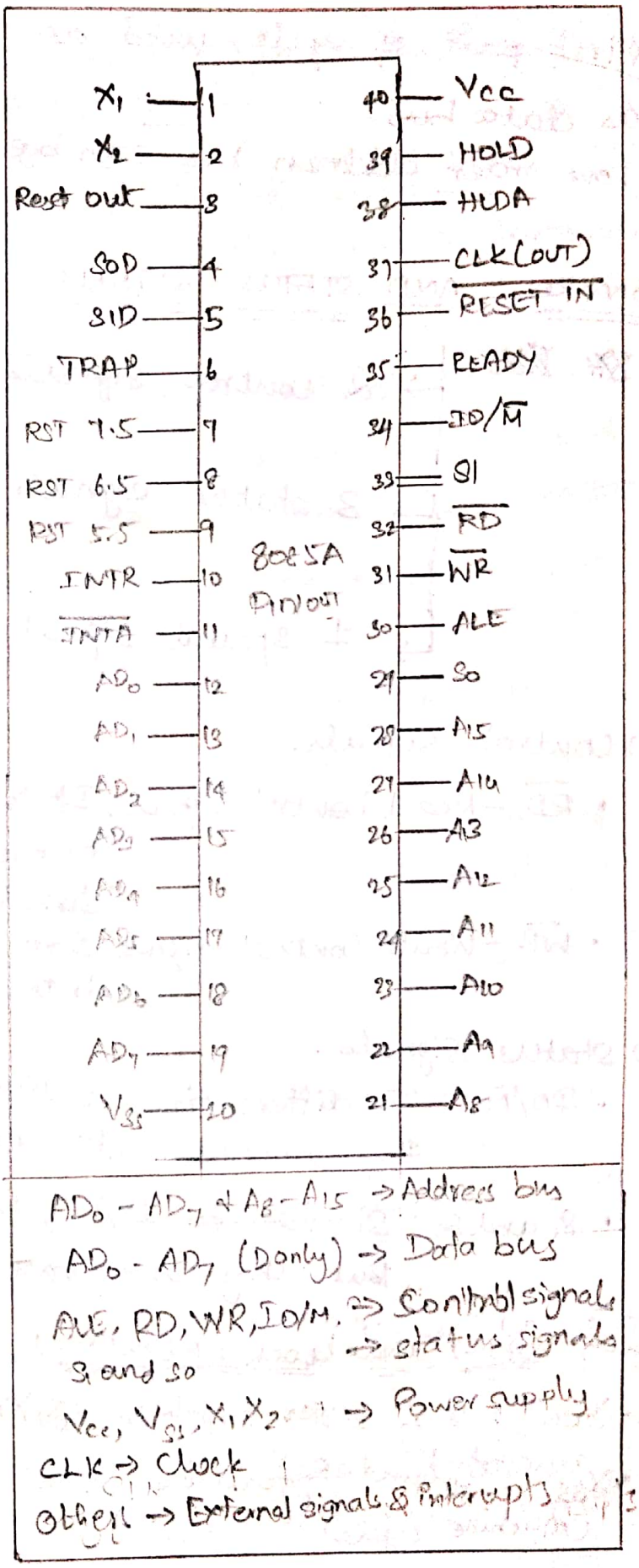
* **Hold**: When Hold pin is activated (by external signal), it relinquished control of bus and allows the external peripheral to use them.

The 8085 MPU Pinout & Signals

The 8085 A is an 8 bit general purpose microprocessor capable of addressing 64k of memory. The device has 40 pins, which operates on a 3-MHz on single phase clock, while 8085 A 2 Version operates on 5 MHz.

They are classified into six groups.

(i) Address bus (ii) Data bus (iii) control and status signals (iv) power supply and frequency signals (v) externally initiated signals (vi) serial I/O ports.



$AD_0 - AD_7$ & $A_8 - A_{15}$ → Address bus

$AD_0 - AD_7$ (Donly) → Data bus

ALE, RD, WR, IO/M. ⇒ Control signals → status signals

S and SO

V_{cc}, V_{SS}, X_1, X_2 → Power supply

CLK → Clock

Others → External signals & Interrupts

ADDRESS BUS:

* It has 16 signal lines used as address bus.
* These lines were split into two segments..

$A_{15} - A_8$: Address bus used for MSD, so called as high order address.

$AD_7 - AD_0$: multiplexed data bus.

MULTIPLEXED DATA BUS :

* $AD_7 - AD_0$ are bidirectional
* Used as lower order address bus as well as data bus
* First part of cycle, used as address bus later it is used as data bus.
* Low order address bus can be seperate signals using latch.

CONTROL AND STATUS SIGNALS :

- 2 control signals → $\overline{RD}$ (Read), $\overline{WR}$ (Write)
- 3 status signals → $IO/\overline{M}$ (Input/output/memory), S_1, S_0
- 1 special signal → ALE (Address Latch Enable)

(i) Control Signals.

- $\overline{RD}$ - Read control signal: It is to be read the Input/output or memory and datas are available on data bus.
- $\overline{WR}$ - Write control signal: Data on data bus were written into selected memory or I/O location.

(ii) Status Signals.

- $IO/\overline{M}$: To differentiate IO (when high) or memory (when low). they combined with $\overline{RD}$ and $\overline{WR}$.
- S_1 and S_0 : Similar to $IO/\overline{M}$, to identify various operations. But they are rarely used in small systems.

(iii) Special Signals :

- ALE - Address Latch Enable: This is a positive going pulse generated everytime the 8085 begins an operation. (machine cycle)

• This is used to latch the low-order address from the multiplexed bus and generate a seperate set of eight address lines, A_7-A_0.

* This is because the machine cycle indicates that the bits on AD_7-AD_0 are address bus.

EXTERNALLY INITIATED SIGNALS, INCLUDING INTERRUPTS:

The 8085 has five interept signals. Some of interepts were considered with externally initiated signals. The purpose is to Interupt the sequence of instructions.

* INTR: (Interupt Request): This is used as the general purpose of interupt, which is similar to INR in 8080A.
* INTA: (Interupt acknowledge): It is to acknowledge the interupt.
* RST: (Restart) These are vectored interupt that transfers program control to specific memory location. There are high priorities than INTR. Three of them, priority of order is 7.5, 6.5 and 5.5.
* TRAP: This is a nonmarkable interupt and has the high priority.
* HOLD: this signal indicates that a peripheral such as DMA (Direct Memory Access) controller is requesting the use of address and data buses.
* HLDA: To acknowledge HOLD Request.
* READY: It is used to delay the MP read/write cycles until a slow-responding peripheral is ready to send or accept data. When low, the MP waits for integeral number of clock cycle till it goes high.

The 8085 MPU Pin Out & Signals.

* $\overline{\text{RESET}}$ IN : When this signal is low,
  - program counter becomes zero
  - Buses are tri-stated.
  - MPU is reset.

* $\overline{\text{RESET}}$ OUT : This indicates the MPU is being reset. It is used to reset other outer devices.

SERIAL I/O PORTS

→ The 8085 has 2 signals to implement the serial transmission:

* SID (serial input data) * SOD (serial Output Data)

→ In this serial transmission, data bits are sent over a single line, one line at the time.

POWER SUPPLY AND CLOCK FREQUENCY

- V_{CC} : +5V power supply.
- V_{SS} : Ground referance.
- X_1, X_2 : The crystal is connected to these two pins. Frequency is divided into 2 segments. each that operates 3MHz, so the crystal should have 6 MHz
- CLK (OUT) : It is used as the system clock to other devices.

- X_1, X_2: Crystal is connected to 2 pins. Frequency is divided into 2 parts: each operates on 3 MHz so the crystal should have 6 MHz.
- CLK (clock) (OUT): This signal is used as a system clock for other devices.

Writing Assembly language Program

Writing a program is equivalant to giving specific commands to the microprocessor in a sequence to perform a task. There are some clues to write a programs.

* Perform a task: What is the task you are asking it todo?
* Sequence: What is the sequence you want to follow?
* commands: What are the commands (instruction set) it can understand?

GETTING STARTED:

Here are some steps that are followed by the about given clues.

Step 1: Read the program Carefully.
Step 2: Break it down into small steps
step 3: Represent these small steps in a possible sequence with flowchart - a plan of attack.
step 4: Translate each block of a flowchart - a plan of attack into appropriate mnemonic instructions.
Step 5: Translate mnemonics into machine code (assembler)
Step 6: Enter the (program) machine code in memory and execute. Only on rare occations is a program successfully executed on the first attempts.
step 7: Start trouble shooting (or debugging).

* A programer that translates the mnemonics into machine code is called as assemblers.

Documentation:

A program is similar to circuit diagram. Its ~~purpose~~ is to communicate to others what the program does and how it does it. Appropriate comments are critical for converting the logic behind a program.

* The comments should explain what is intended and should not explain the mnemonics.
  Example: ~~First comment~~.
* Some comments are omitted if it does not say anything more than repeat a mnemonics.
* Some tables can be used whenever we need to start or ~~stop~~ shift the program ~~execution~~ sequence.

Translation from Assembly language to machine code.

It is nothing but translating a program into machine code, together ~~which~~ with the corresponding suggested memory address. The machine code is of hexadecimal format.

Program Execution:

Those translated codes are loaded into ~~I/O~~ R/W memory, starting memory address specified. The execution will be of two ways.

Program Execution

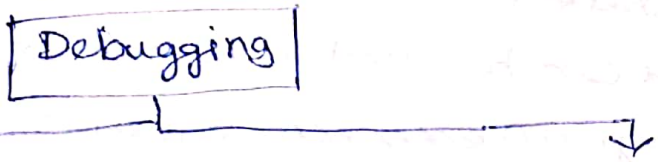
- Execution of entire ~~code by program~~ pressing execution key
- Each instruction code is executed by using single step key.

Debugging :

* An Error in a computer program or system is called bug. The process of locating and removing errors from a computer program or system is called debugging.

* Debugging these program is similar to troubleshooting hardware, but it is more difficult and cumbersome.

* Debugging process can be classified into two parts : static debugging and dynamic debugging.



Static debugging
similar to visual inspection of circuit board. It is done by simple check on flow chart and machine code.

Dynamic Debugging
Involves observation of output, register contents, following the execution of each instructions (single step key method) or group of instruction (breakpoint technique)

*

*

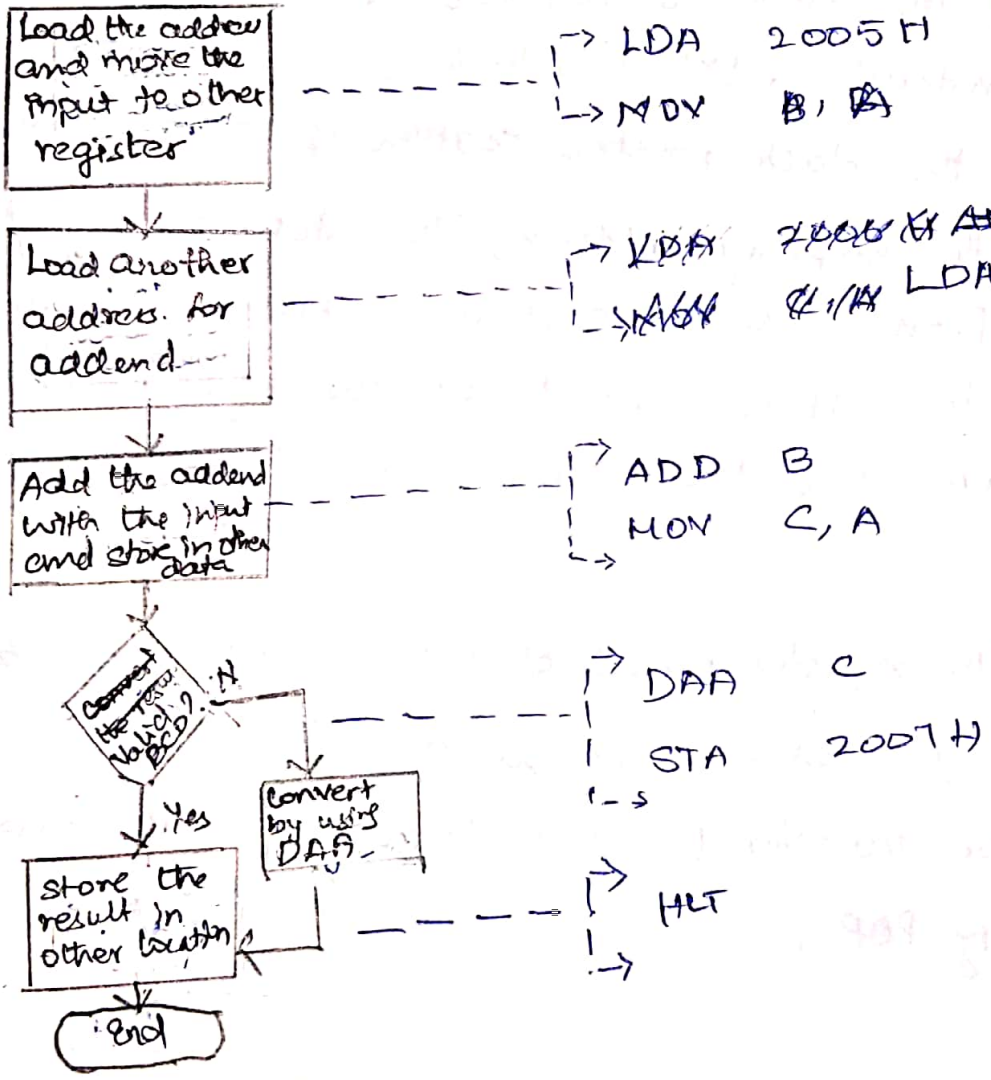
ILLUSTRATION OF WRITING ASSAMBLY LANGUAGE PROGRAM

Addition of 2 BCD number and store results in ~~other~~ ~~memory~~ memory of specified location.

Algorithm :

- Take addunt in accumelator from memory.
- Transfer the addelend to B register.
- Take the augend in accumelator from memory.
- Add two numbers.
- Convert the result into BCD form using DAA set
- Store the sum in memory.
- End.

Flow chart



Program

```
LDA   2005H
MOV   B, A
LDA   2006H
ADD   B
MOV   C, A
DAA
STA   2007H
HLT
```

STACK AND SUBROUTINE

The stack is a group of memory location in the R/W memory that is used for temporary storage of binary information during the execution of a program.

A subroutine is a group of instruction that performs a subtask of repeated occurrence. A large software project is divide into subtask called modules.

-STACK-

~~STACK:~~

* In an 8085 microprocessor system, It can be described as a set of memory location in the R/W memory, specified by a programmer in a main program.
* The begining of the stack is defined in the program by using the instruction LXI SP, which loads a 16 bit address ~~to the~~ from the stack pointer register of the microprocessor.
* Once defined the memory address, the storage of bytes will initiate from the ~~pre~~ address of preceeding one location. Example. If we point out the location hosend data at 2099H, then the storage starts (initiates) from 2098H.
* Data bytes in the register pairs of microprocessor can be stored in stack in reverse order using PUSH
* Data bytes can be transfered from the stack to respective registers by using POP.

INSTRUCTIONS AND PROCEDURES:

I. LOAD [LXI SP, 16 bit]:

① First think to do is to load a stack from address to the stack pointer.

II. PUSH : Store register pair on stack. And It is 1 byte instruction.

Example:

```
PUSH  Rp
PUSH  B
PUSH  D
PUSH  H
PUSH  PSW
```

~~NOTE~~:

~~For PUSH B~~ ~~higher order address and lower order address~~.

Procedure:

① It copies the contents of the specified register pair on the stack as described.

② The stack pointer is decremented and the content of the higher order register are copied to the location stored as per pointer.

③ The stack pointer is decremented again and the content of lower order register is stored in the specified location.

④ -

Note: PSW is an operand that meaning is content of accumulator and flag.

* From Examples, B, D, H represents the register pair. Example PUSH B equivalent to PUSH BC. Here
* The described register is higher order register and hidden register is lower order register

~~H Pop~~ :

III. POP

Examples:

```
Pop  Rp
Pop  B
Pop  D
Pop  H
Pop  PSW
```

Retrives Register pair from stack. This is a 1 byte instruction.

Procedure:

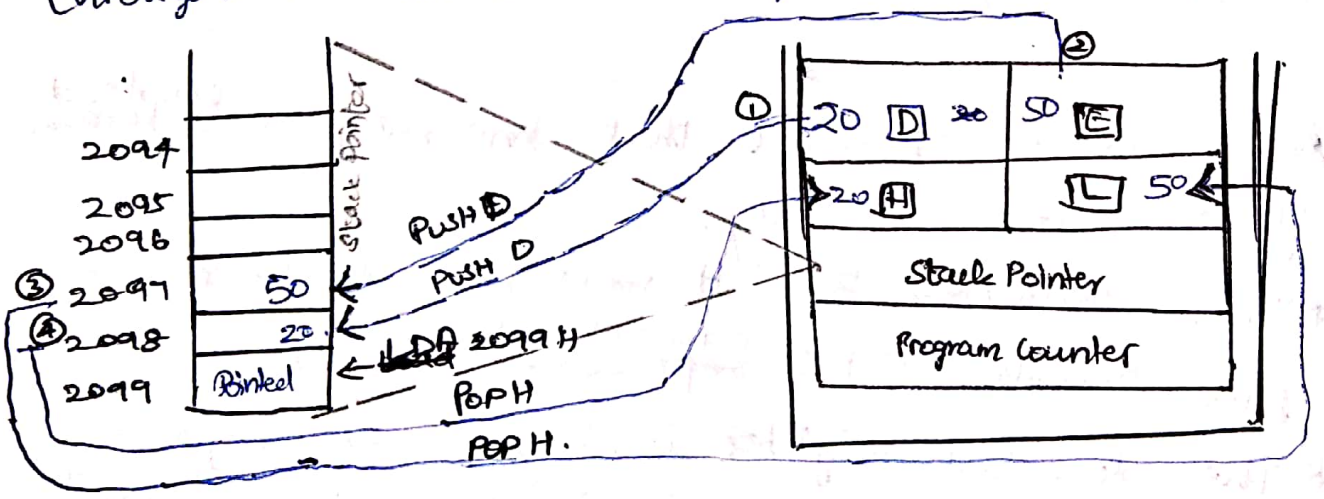
① Copies the content of the top two memory location pointed by stack pointer register and are copied to the lower order register.

② Then incremented the pointer by 1.

③ The contents of the next memory location is copied into higher order register. And again incremented by 1.

Diagramatic Illustration of Push and Pop instructions.

To shift the data from one register pair to HL pair through stack, an R/W memory.



— SUBROUTINE —

Subroutine :

A subroutine is a group of instructions written seperately from the main program to perform a function that occurs repeatedly in the main program.

8085 Instructions for subroutine

- CALL
  - Calls the subroutine
  - Declared at begining of subroutine.
- RETURN
  - Returns from subroutine
  - Declare at the end of subroutine.

I. CALL INSTRUCTION :-

- It is the 3 byte instruction.
- It calls subroutine unconditionally.
- Declared at beginig of subroutine.

PROCEDURES OF CALL INSTRUCTION:-

① Save the content of the program counter on the stack.

② Decrements the stack pointer register by two.

③ Jumps uncondicionally to memory location spesified by 2nd and 3rd bytes.

④ This instruction is accomplished by return instructions

CALL	16 bit memory address

RETURN INSTRUCTION:

- Returns from subroutine unconditionally.
- It is 1 byte instruction.
- ~~Inserts the 2 bytes from the top of the stack to the program counter~~

PROCEDURE OF RETURN INSTRUCTION :-

① Inserts the 2 bytes from the top of the stack to the program counter and

② Increments the stack pointer register by two.

③ Unconditionally returns from the subroutine.

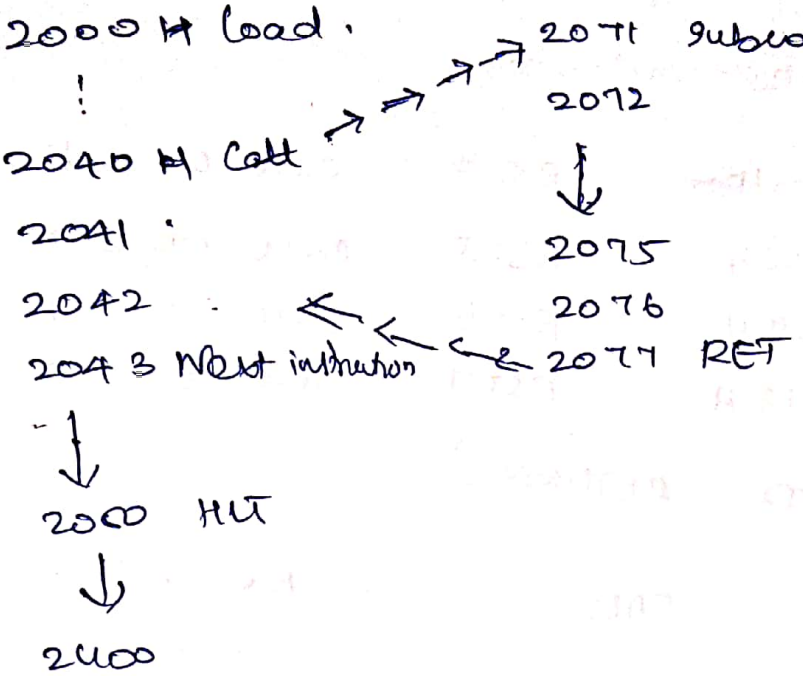
RET	16 bit memory address

EXAMPLE:

Let us illustrate the exchange of information between the stack and the program counter for the following program if the available user memory range from 2000H to 23FFH.

2000H	LXI SP, 2042H	2071H	SUBROUTINE
:		2072H	
2042H	CALL SUBROUTINE	2073H	
2043H		:	
2044H		2075H	
2045H	NEXT INSTRUCTION	2076H	
:		2077H	RET
2050H	HLT		
↓	↓		
23FEH	STACK POINTED		
23FFH	↑ memory free		
2400H	STACK POINTED		

Program Execution



RESTART:

* 1 byte Instruction.
* Transfers program execution to specific location. (from 00H)
* there are 8 RST instructions.

RST 0	Call	0000H	RST 4	call	00 20H
RST 1	call	0008H	RST 5	call	00 28H
RST 2	call	0010H	RST 6	call	00 30H
RST 3	call	0018H	RST 7	call	00 38H

CONDITION CALL AND RETURN:

CONDITION		CALL	RETURN
- NONE -	—	CALL	RET
Carry	(set)	CC	RC
No Carry	(Reset)	CNC	RNC
Zero	(set)	CZ	RZ
No Zero	(Reset)	CNZ	RNZ
Parity	Even	CPE	RPE
Parity	odd	CPO	RPO
Signed:	+ve	CP	RP
	-ve	CM	RM

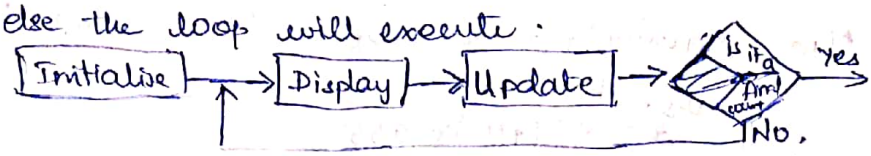
Microprocessor

Counter and Time delay

Counter:

A counter is designed simply by loading an appropriate number into one of the register and using the INR or DCR instructions.

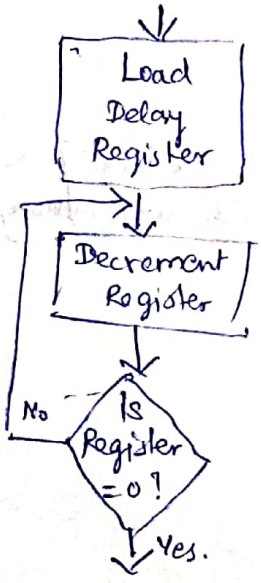
A loop is established to update the count, and each count is checked to determine whether it has reached that number; else the loop will execute.



Time Delay:

* The procedure used to design a specific delay is similar to that used to set up a counter.
* A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with a condition loop instruction.

Time delay Using One Register:



Let us take an example instruction:

			T state
	MVI	C, FFH	7
Loop:	DCR	C	4
	JNZ	LOOP	10/7.

Usually, An 8085 based computer with 2 MHz clock frequency will execute the instructions MVI, as follows.

Clock frequency of a system (8085) f = 2 MHz.

Clock period $T = 1/f = \frac{1}{2} \times 10^{-6} = 0.5\ \mu s$

Time to execute MVI instruction

$T = 7 \times 0.5$

$= 3.5\ \mu s.$

If clock frequency has 1 MHz in system, and the ... May require 7 μs to execute the same instruction.

Time delay depends on T-states and no. of times the instruction is executed.

Time delay in the loop with 2 MHz clock frequency is Calculated as,

$T_L = T \times \text{loop T states} \times N_{10}$ (b)

$T_L \rightarrow$ Time delay $T \rightarrow$ System clock period

$N_{10} \rightarrow$ Equivalent number of hexa. count loaded

$T_L = 0.5\times10^{-6} \times (4+10) \times 255$

$= 0.5\times10^{-6} \times 14 \times 255$

$= 1785\ \mu s = 1.8 ms.$

$\boxed{T_L = 1.8 ms}$

A time delay to execute the loop,

$T_{LA} = T_L - (3\ T\text{states} \times \text{clock period})$

~~1.8ms~~

$= 1785\ \mu s - (1.5\ \mu s) = 1783.5 \mu s.$

~~3 T-states~~,

The total time delay,

$T_D = T_0 + T_{LA}$ [$T_0 \rightarrow$ Time delay outside the loop, $T_{LA} \rightarrow$ Time delay inside loop instruction]

$= (7 \times 0.5) + (1783.5)$

$= 3.5 + 1783.5$

$= 1787 \mu s \approx 1.8 ms.$

$\boxed{T_D = 1.8 ms}$

Time Delay Using a Register Pair.

Let us consider this example,

	Mnemonic		T state.
	LXI	B, 2384H	10
Loop:	DCX	B	6
	MOV	A, C	4
	ORA	B	4
	JNZ	Loop	10/7.

NOTE: In LXI, 23H stores in B and 84H stores in C.

Decrement is 2384 - 1 ⇒ 2383H.

~~let us take~~

We load 2384H into BC Register pair, such that

B ⇒ 23H C ⇒ 84H.

As it is hexadecimal, let us convert into decimals,

$2384 = 2\times16^3 + 3\times16^2 + 8\times16^1 + 4\times16^0$

$= 9092_{10}$.

$T = 0.5\mu s$.

$\therefore T_L = 0.5 \times 24 \times 9092_{10}\ \mu sec$

$= 109$ msec (without last cycle).

Total time delay,

$T_D = T_L + T_0$

~~= 109t~~

= 109 ms (The instruction LXI adds only 5 μs.)

③. Time delay using a loop within a loop technique.

Example :

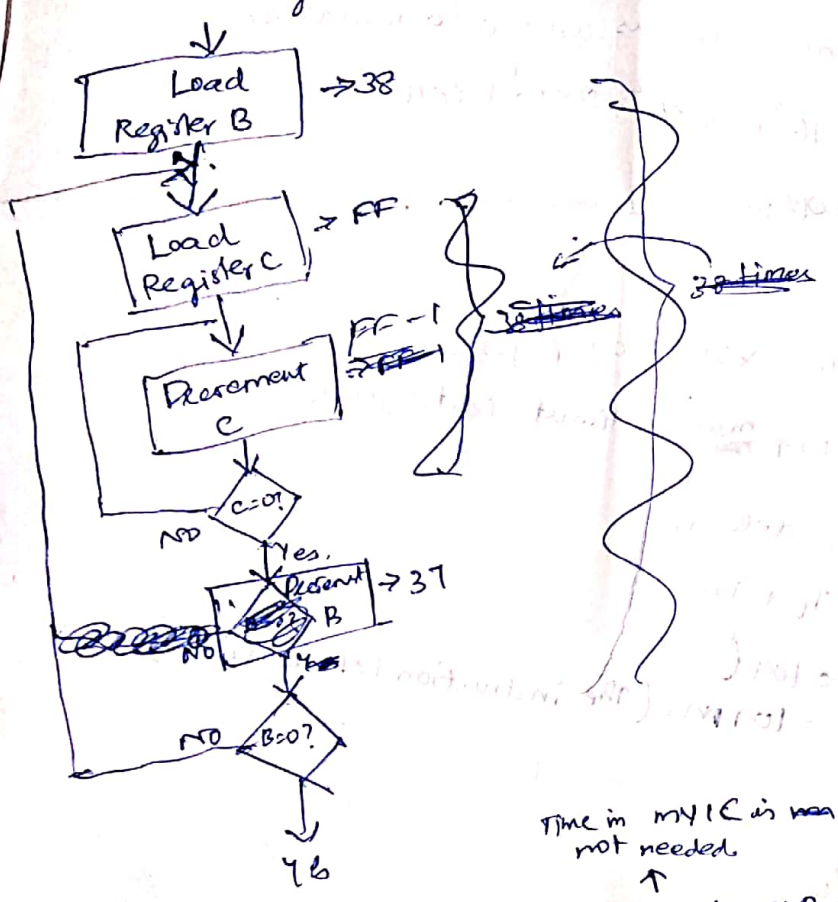
```
        MVI  B, 38H   (7T)
Loop 2: MVI  C, FFH   (7T)
Loop 1: DCR  C        (4T)
        JNZ  Loop 1   (10/7T)
        DCR  B        (4T)
        JNZ  Loop 2   (10/7T)
```

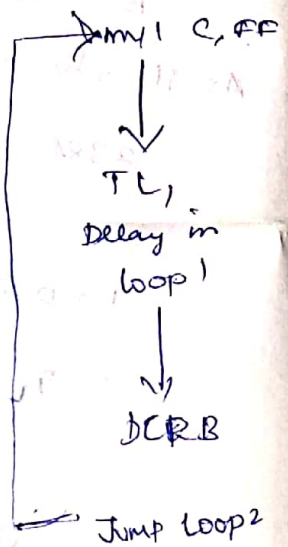
38
8 × 16 = 128
3 × 16 = 48
56

Delay Calculations

Delay in loop 1 is T_{L1} = 1785 μs (as in ① type)

This loop executes 56 times because of the count 38 H in Register B. [$38_H = 56_2$]





Time in MVI C is not needed.

T_{L2} = 56 (T_{L1} + 21 T states × 0.5 μs)

= 56 (1785.5 μs + 10.5

= 100.46 ms.

ASSAMBLY LANGUAGE PROGRAM.

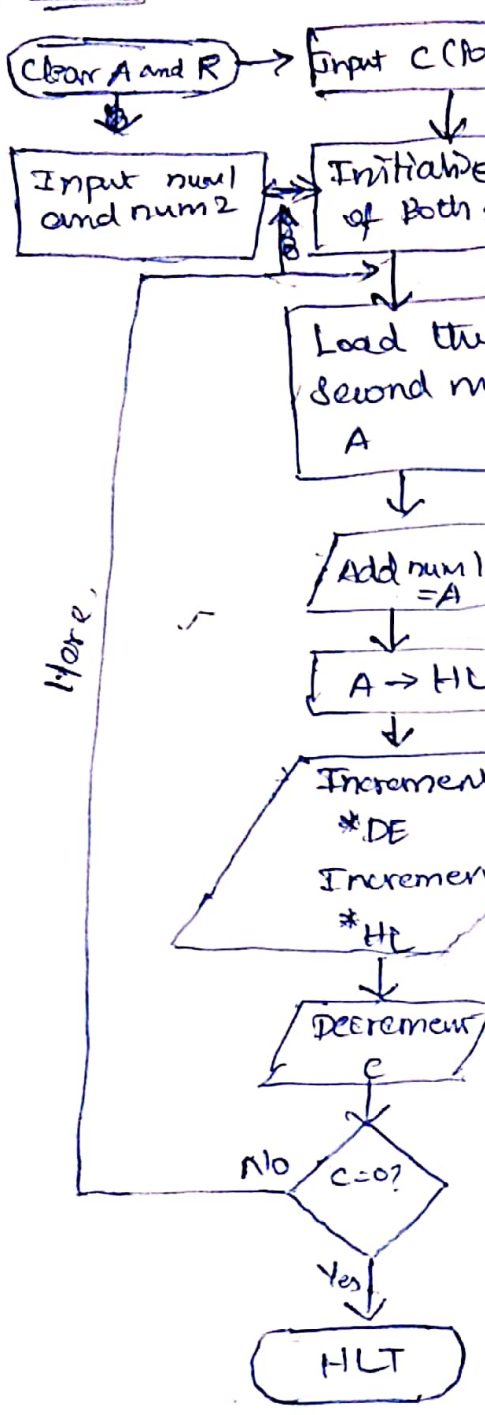
1. MULTIBYTE ADDITION
2. MULTIBYTE SUBTRACTION
3. BCD ~~MULTIBYT~~ ADDITION
4. BCD SUBTRACTION
5. BCD MULTIPLICATION
6. BCD DIVISION
7. BINARY TO BCD
8. BCD TO BINARY
9. BINARY TO ASCII
10. ASCII TO BINARY.
11. ~~8 bit ADDITION~~ BCD TO ASCII
12. ~~8 bit SUBTRACTION~~ ASCII TO BCD

- ASSAMBLY LANGUAGE -

- PROGRAMS -

1. MULTIBYTE ADDITION

Flow chart:

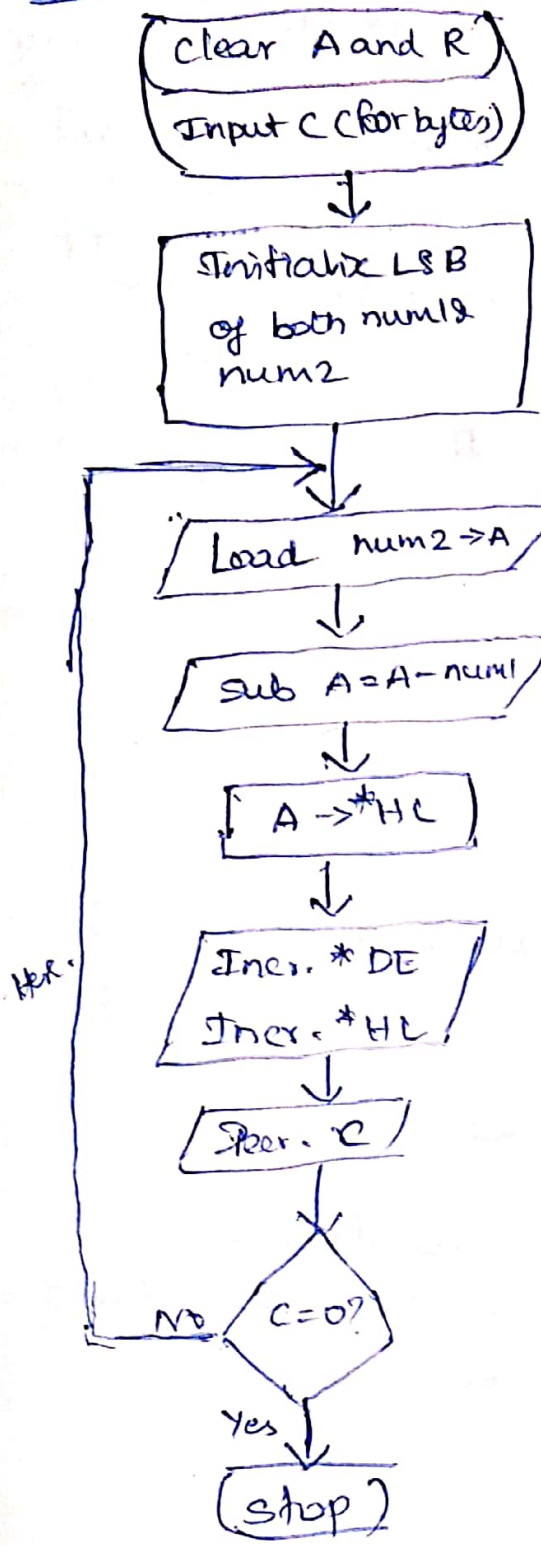


PROGRAM

```
        XRA   A
        MVI   C, 08H    ; counter
        LXI   H, 2050H
        LXI   D, 2060H
        LDAX  D         ; HERE is the table
        ADD   M
        MOV   M, A
        INX   H
        INX   D
        DCR   C
        JNZ   HERE
        HLT.
```

2. MULTIBYTE SUBTRACTION

Flow chart



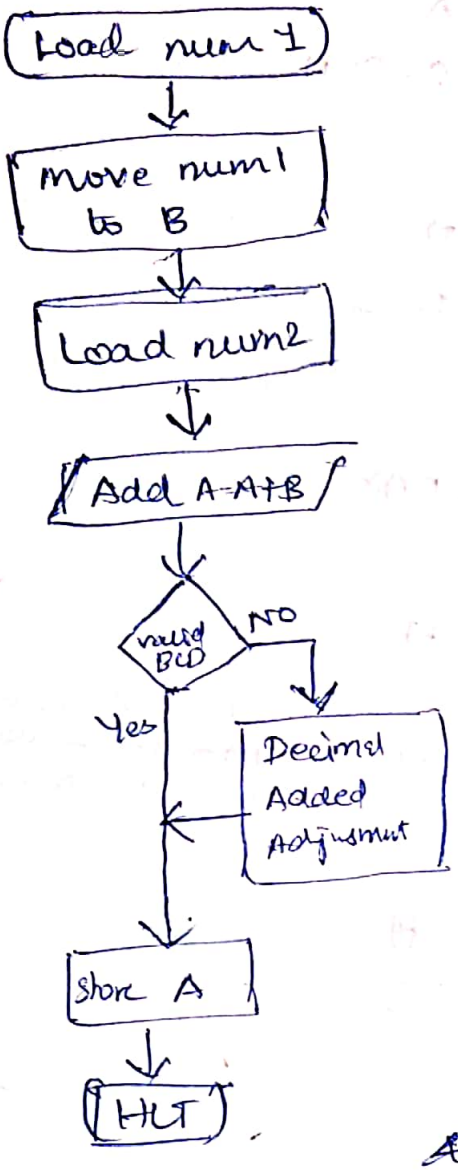
Program

```
        XRA   A
        MVI   C, 08H    ; counter.
        LXI   H, 2050H
        LXI   D, 2060H
HERE:   LDAX  D
        SBC   H
        MOV   M, A
        INX   H
        INX   D
        DCR   C
        JNE   HERE
        HLT
```

M → Memory
→ after 2050H for each counter.

3. BCD ADDITION

Flow Chart



LDA 2050 H

MOV B, A

LDA 2051 H

ADD B

DAA

STA 2052 H.

HLT.

~~A~~ BCD SUBTRACTION

NOTE: SAME AS (Prog-3), difference is instead of Add, we use Sub.

LDA 2060 H

MOV B, A

LDA 2061 H

SUB B

DAD

STA

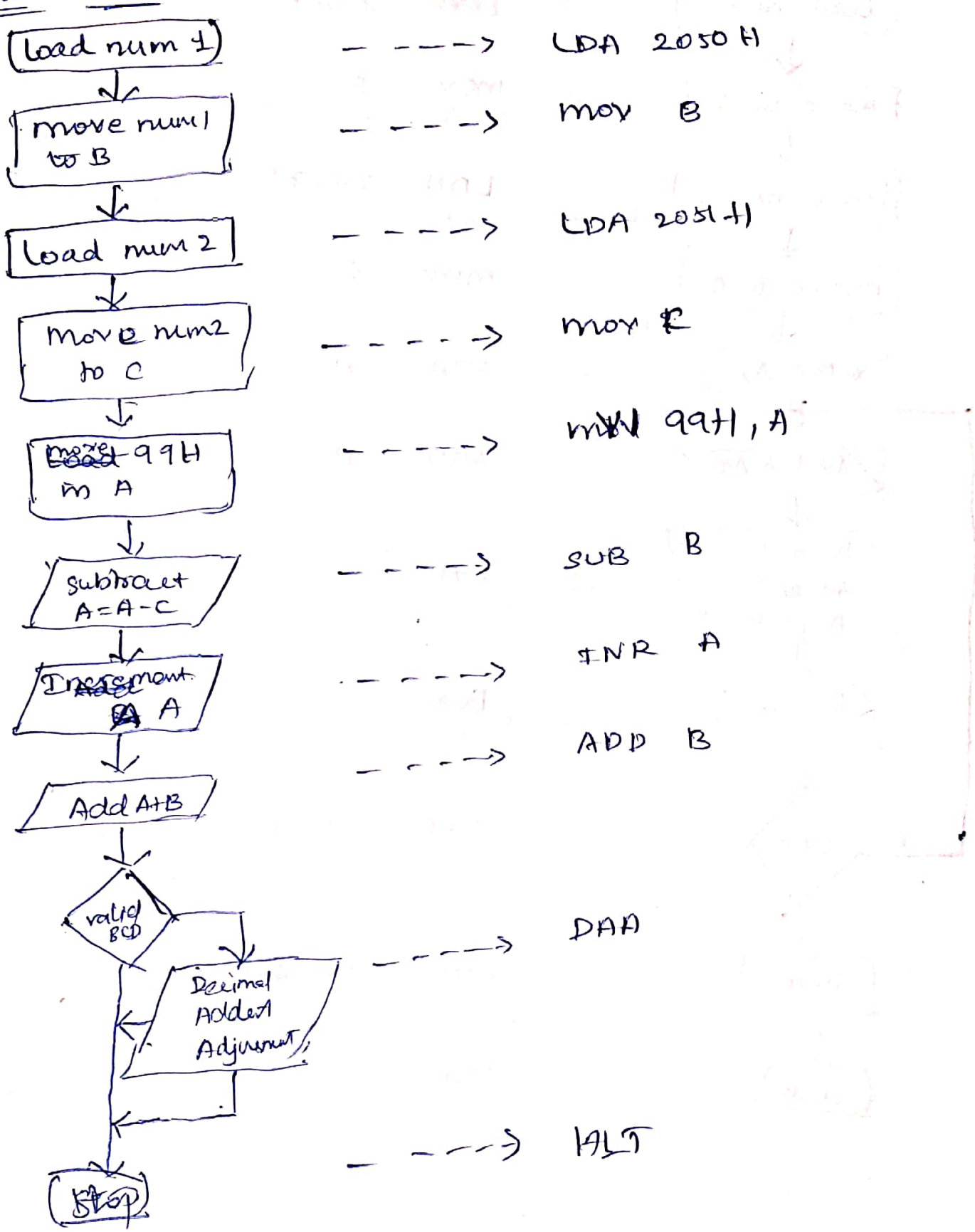
HLT

~~4. BCD MULTIPLICATION SUBTRACTION~~

4. BCD SUBTRACTION

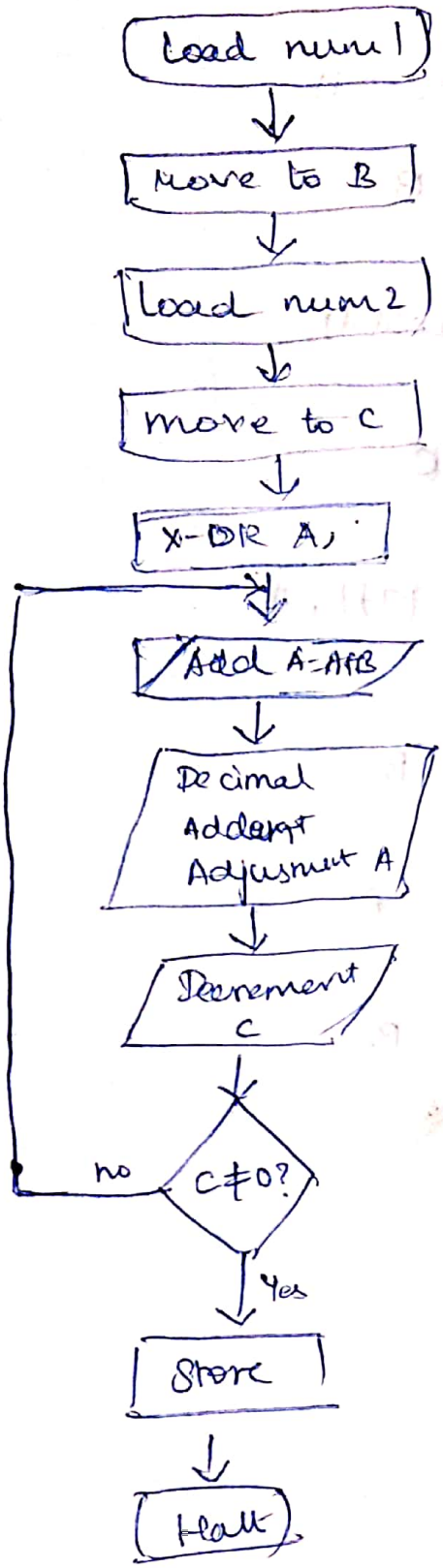
[Here, 10's complement based subtraction is used]

Flow chart:



5. BCD MULTIPLICATION

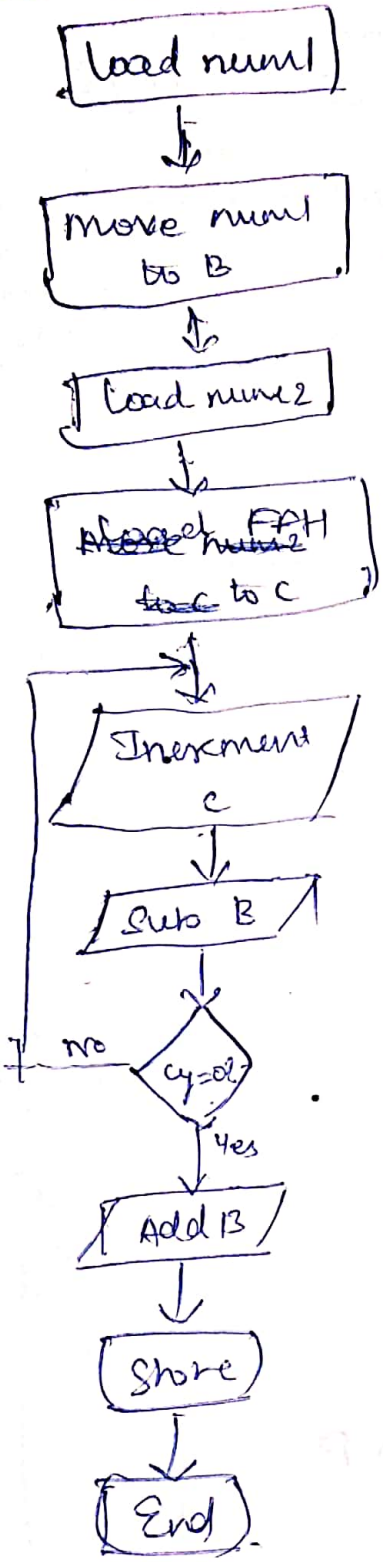
Flow Chart



:	LDA	2051 H
:	MOV	B
:	LDA	2052H
:	MOV	C
:	XRA	A
:	ADD	B
:	DAA	
:	DCR	C
:	JNC	2009
:	STA	2053H
:	HLT	

6. BCD DIVISION

Flow Chart



: LDA 2051H

: MOV B,A

: LDA 2052H

: MVI C,FFH

: INR C

: SUB B

: JNC 2009

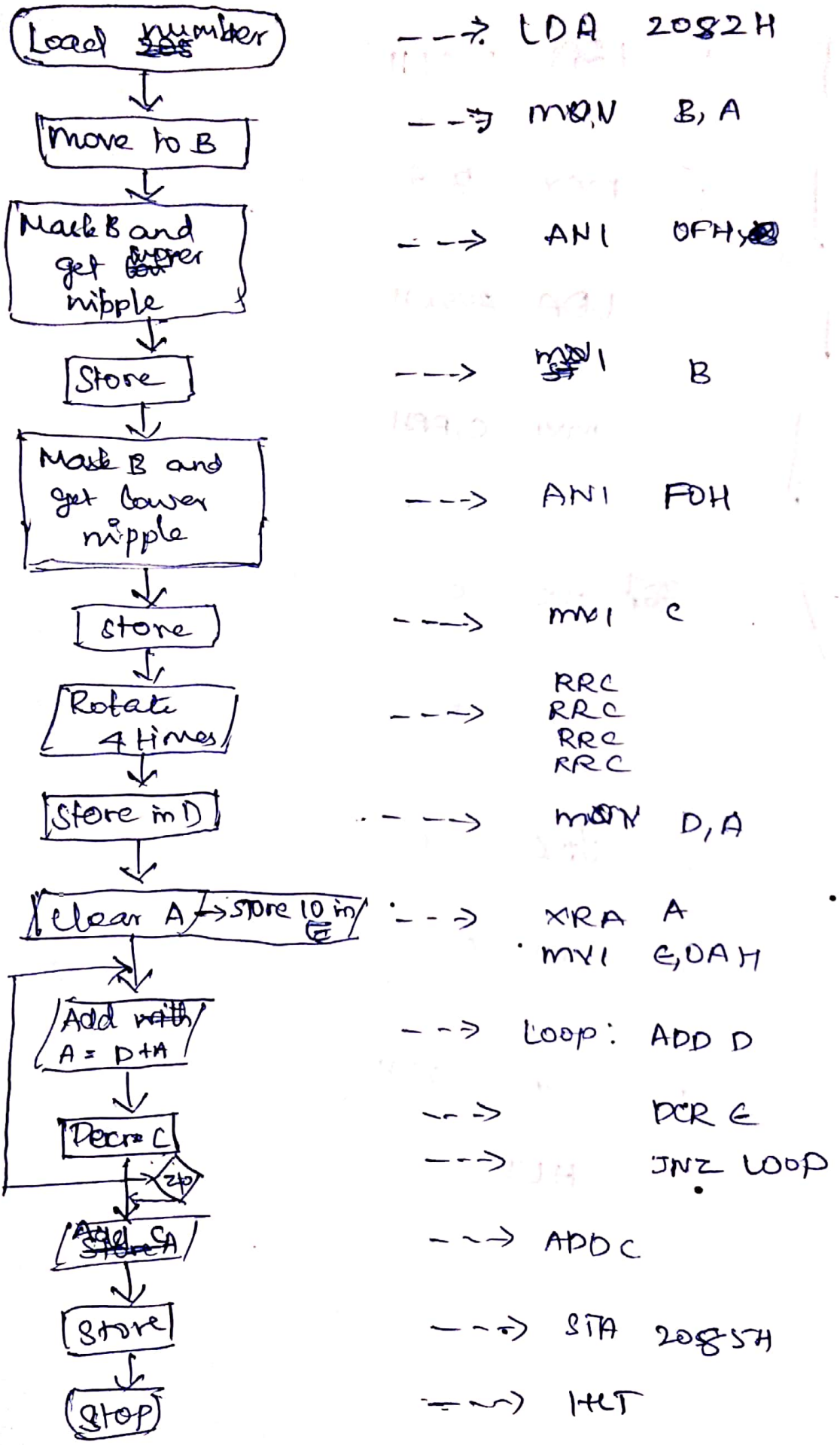
: ADD B

: STA 2053H

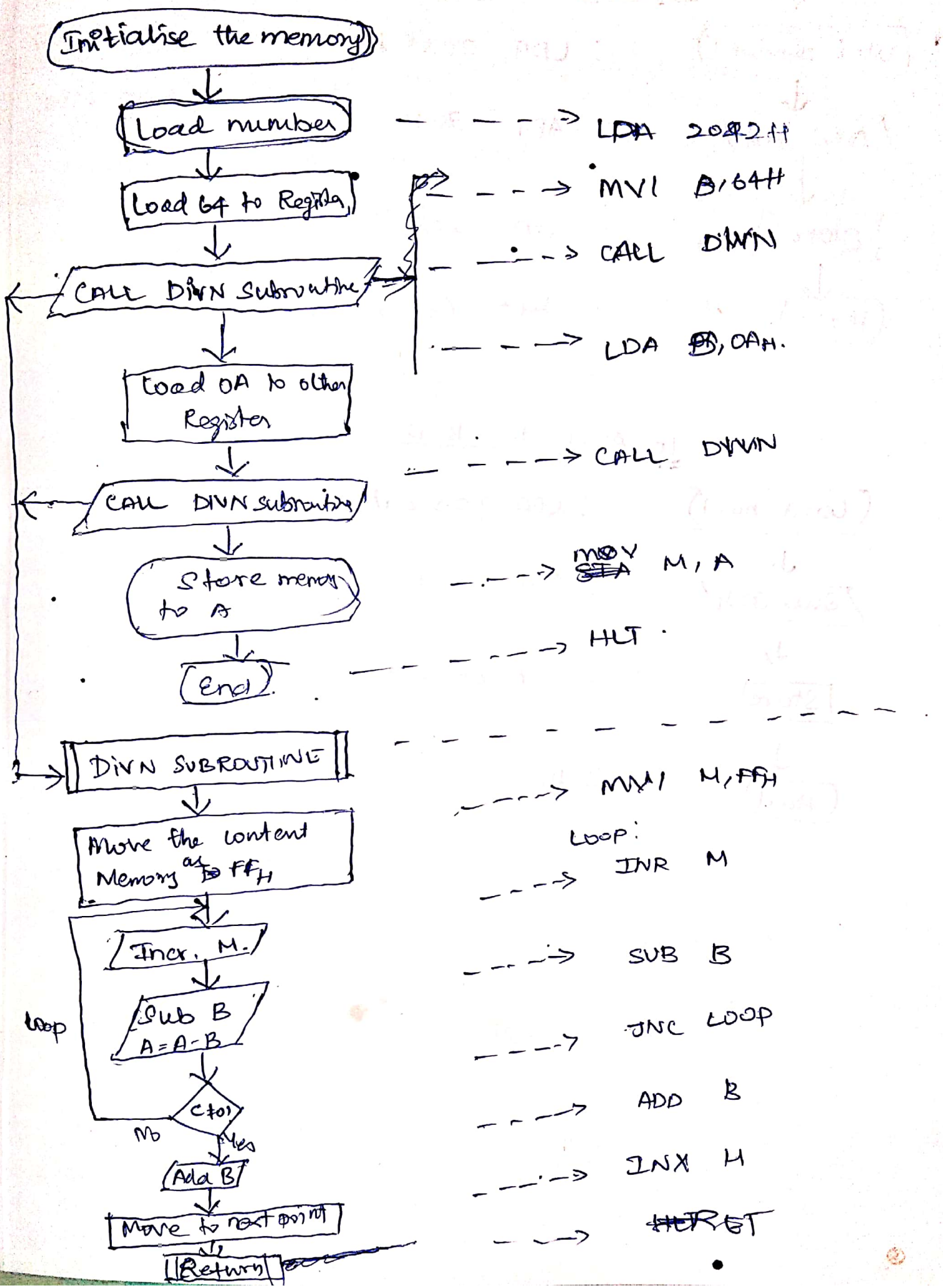
: HLT

CONVERSIONS:

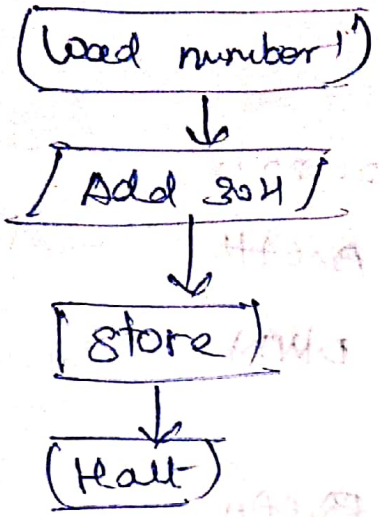
1. BCD to Binary



8. BINARY BCD TO ~~BCD~~ ~~BINARY~~.

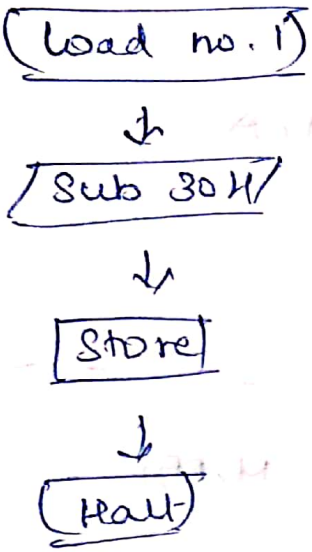


9. BCD TO ASCII



: LDA 2050 H
: ADI 30 H
: STA 2051 H
: HLT

10. ASCII TO BCD



: LDA 2052 H
: SUB 30H
: STA 2053 H
: HLT

~~11. ASCII TO BINARY.~~

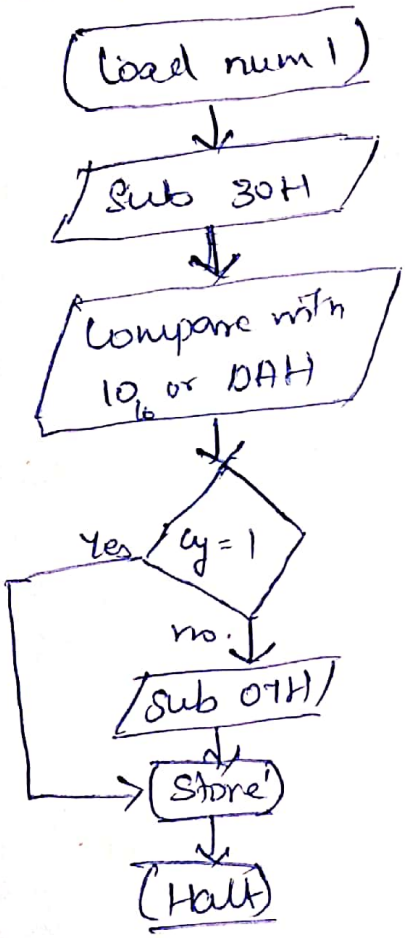
~~Flow chart~~ :

COMPARISON STATEMENTS:

Memory location - Accumulator	Cy	Z
Greater (M>A)	0	0
Lesser (M<A)	1	0
Equal to (M=A)	0	1

11. ASCII TO BINARY

FLOW CHART:



: LDA 2051H

: SUB 30H

: CPI 0AH

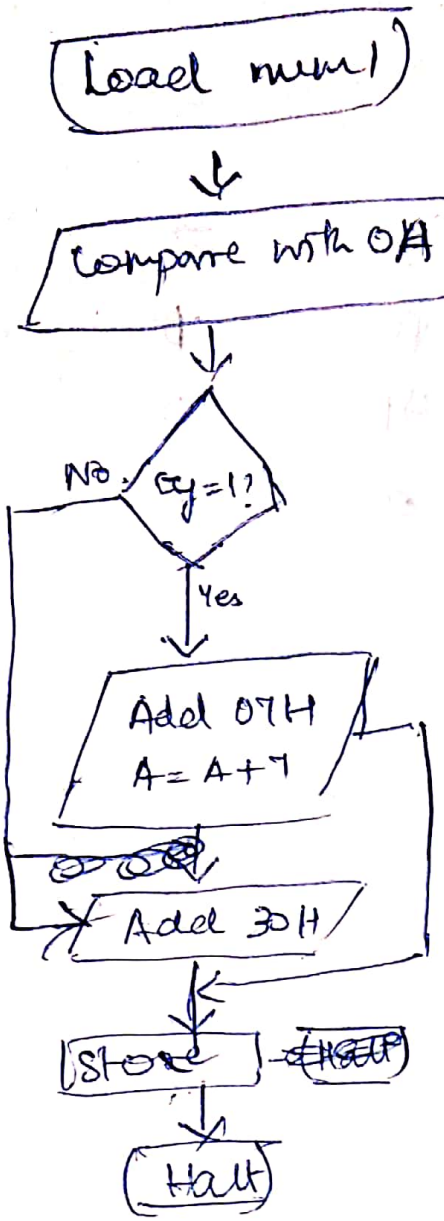
: JC Skip 1

: SUB 07H

: Skip 1: STA 2052H

: HLT.

12. BINARY TO ASCII




```
     LDA   2051H
     CPA   0AH
     JC    SKIP7
     ADD   07H
SKIP7: ADD 30H
     STA   2052H
     HLT
```

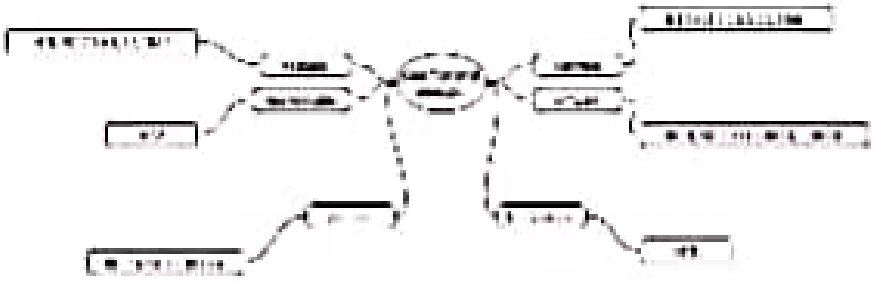
COMPARE

UNIT 5

Interrupts in 8085 microprocessor

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub routine by generating CALL signal and after executing sub routine by generating RET signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.



Interrupts can be classified into various categories based on different parameters:

Hardware and Software Interrupts

- When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as Hardware Interrupts. There are 5 Hardware Interrupts in 8085 microprocessor. They are – INTR, RST 7.5, RST 6.5, RST 5.5, TRAP
- Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

Vectored and Non-Vectored Interrupts

- Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.
- Non Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub routine for these Interrupts. INTR is the only

non-vectored interrupt in 8085 microprocessor.

Maskable and Non Maskable Interrupts

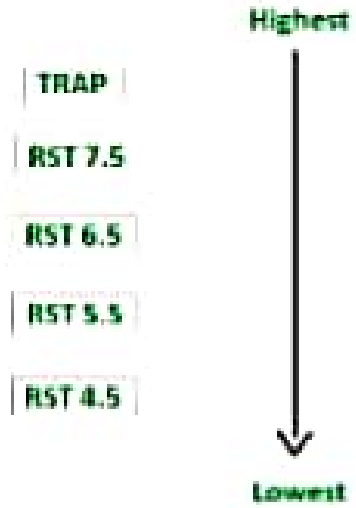
- Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge triggered or level triggered, so they can be disabled. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor.
- Non Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

TRAP Interrupt

It is the only non maskable interrupt which cannot be needed to enabled as it cannot be disabled. It has the highest priority among all the interrupts. *It is edge and level sensitive it needed to be hight and stay high when recognised.* Once recognized, it cannot be recognized again until it goes low and high again Trap is used for power failure and emergency autoff.

Priority of Interrupts –

When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

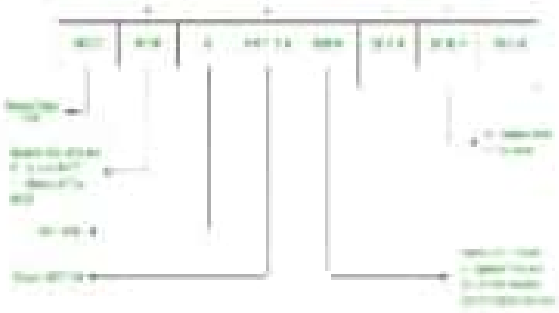


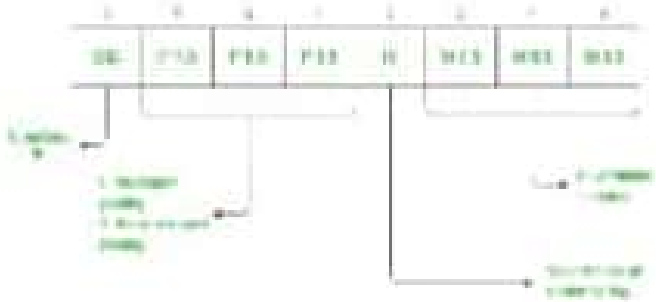
Instruction for Interrupts –

Enable Interrupt (EI) – The interrupt enable flip flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).

Disable Interrupt (DI) – This instruction is used to reset the value of enable flip flop hence disabling all the interrupts. No flags are affected by this instruction.

Set Interrupt Mask (SIM) – It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.





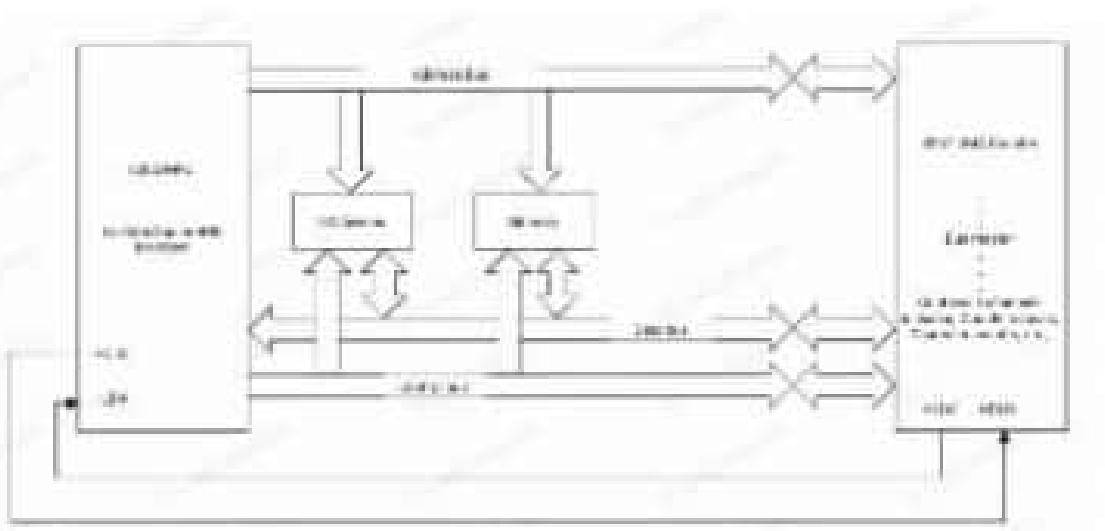
Read Interrupt Mask (RIM) – This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.

Direct Memory Access (DMA)

The Direct Memory access is the process of communication or data transfer controlled by an peripheral. The microprocessor something become slow in control of data transfer, In such situation DMA is used.

The 8085 microprocessor has 2 important signals for DMA. They are HOLD and HLDA (Hold acknowledgement).

- HOLD – This is an active high input signal to the 8085 from other master memory questioning the use of the address and data buses. After receiving the HOLD request, the MPU relinquishes the buses in the following machine cycle. All buses are tri-states and a Hold acknowledge signal is sent out. The MPU regains the control buses after HOLD goes low
- HLDA –Hold acknowledge is an active high output signal indication that the MPU is relinquishing the control of the buses.



FUNCTIONS:

1. The external DMA controller sends the high signal to the hold signal.
2. The processor completes the machine cycle, floats the address, data and control lines and sends the hold acknowledgement signal to the controller.
3. DMA takes the control of the buses and transfers the data from the source to destination.
4. At the end of the data transfer, the DMA controller sends the low signal to the processor and finally, the processor regains the control of the buses.

Memory Interfacing

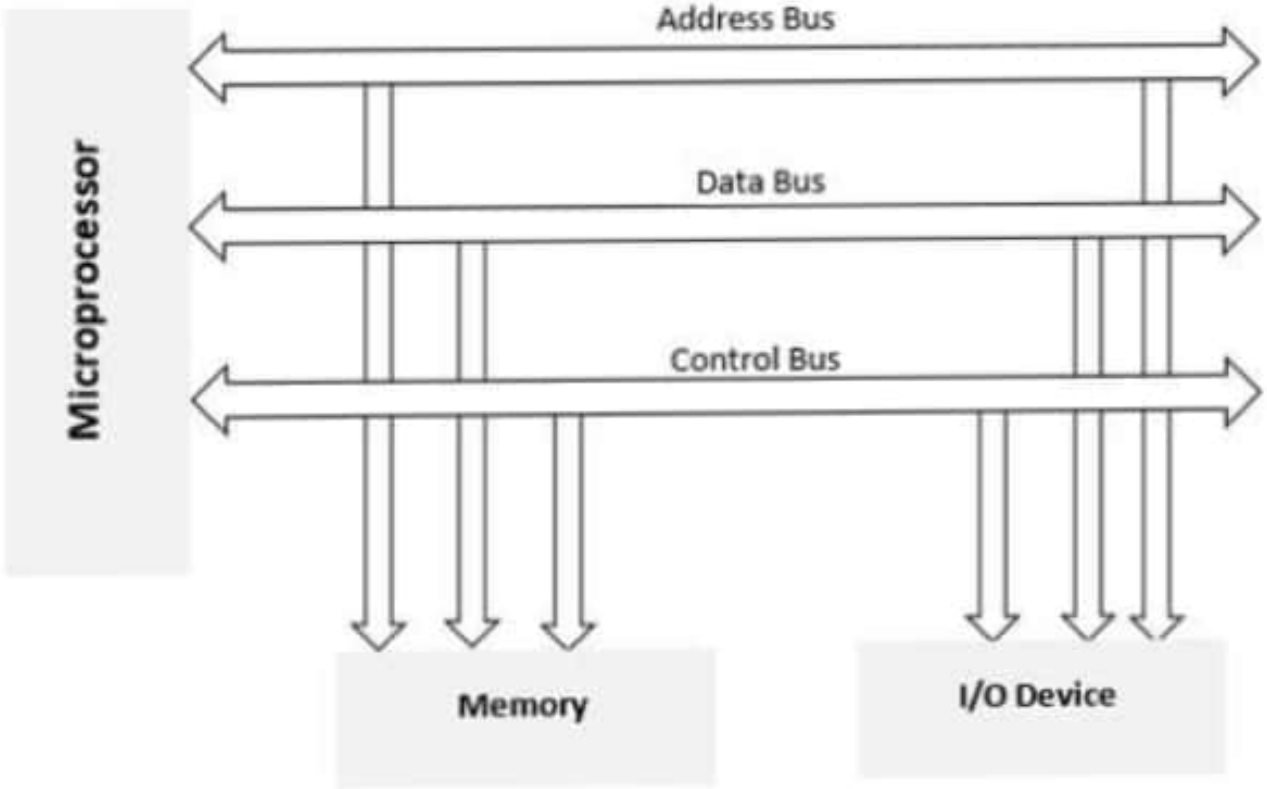
When we are executing any instruction, we need the microprocessor to access the memory for reading instruction codes and the data stored in the memory. For this, both the memory and the microprocessor requires some signals to read from and write to registers.

The interfacing process includes some key factors to match with the memory requirements and microprocessor signals. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

IO Interfacing

There are various communication devices like the keyboard, mouse, printer, etc. So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers. This type of interfacing is known as I/O interfacing.

Block Diagram of Memory and I/O Interfacing



8085 Interfacing Pins

Following is the list of 8085 pins used for interfacing with other devices –

- A_{15} - A_8 (Higher Address Bus)
- AD_7 - AD_0(Lower Address/Data Bus)
- ALE
- RD
- WR
- READY

Ways of Communication – Microprocessor with the Outside World?

There are two ways of communication in which the microprocessor can connect with the outside world.

- Serial Communication Interface
- Parallel Communication interface

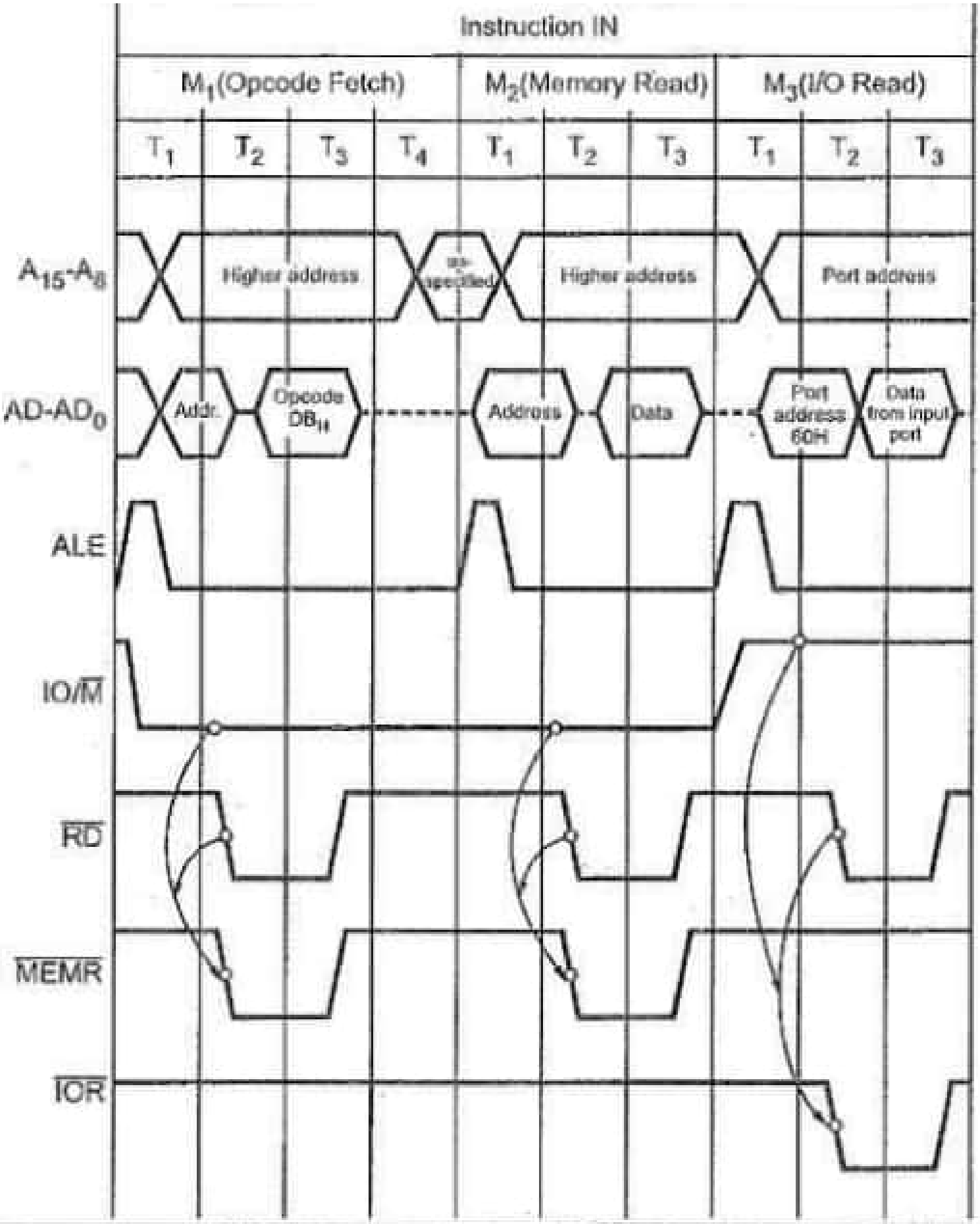


Fig. 4.30 Timing diagram for IN instruction

Memory mapped I/O and Isolated I/O

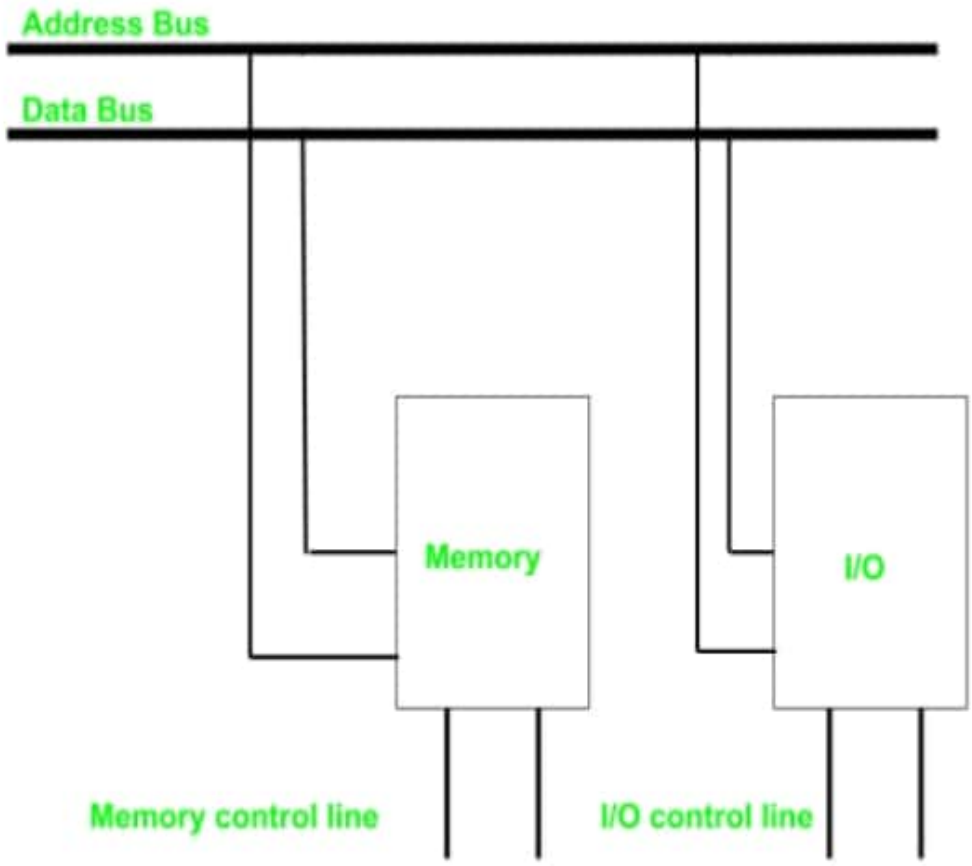
As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them :

1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address, and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but require more buses.

Isolated I/O –

Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory.



Memory Mapped I/O –

In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.