19/7/23

(Architecture microprocessor)

Major Components of computer

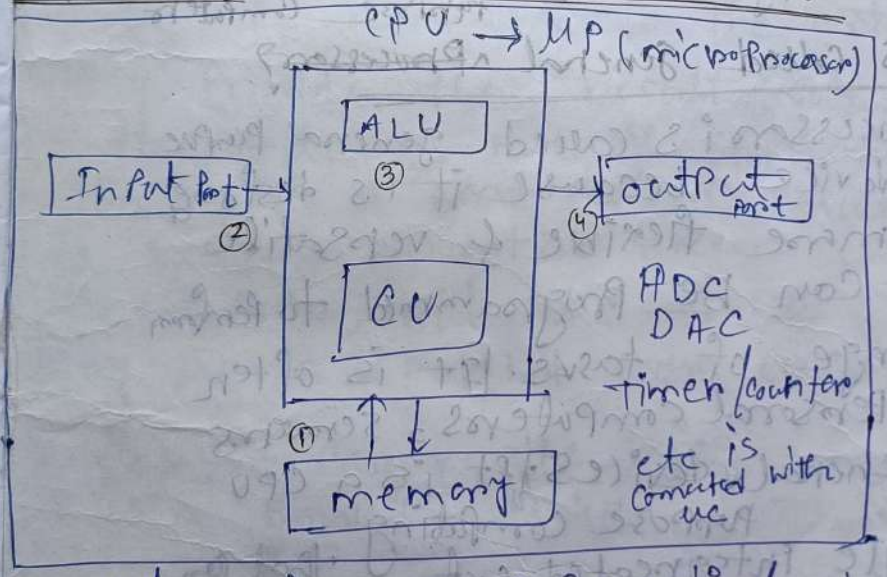
CPU
I & O
memory → RAM (Read-write memory)
→ ROM

→ physical component (hardware components)

1. Software instruction → Software component.

group of ins is called programme.
group of programmes is called S/W or S/W applications

Physical diagram of computer system (just basic diagram exclude MP, MC, i/o port)



MC (micro controller) 8051 4KB 128 byte

computer is a sequential device.
ALU & CU compred in a single chip, that is called microprocessor. (By integreated ckt technology)
Input Port is a input channel where input is given, samely output port.

Advantage of uc & MP. (Diff bw uc & MP) Imp(s)

microcontroller (diff comp) / microcontroller reduce cost & size of the system.

2) Usage of μc is simple, easy for troubleshoot & system maintaining
3) μc are cheap & very small in size, therefore they can be embedded on any device.

microprocessor - i) The μp is that, those are general purpose electronics processing devices which can be programmed to execute a no. of tasks. ii) compact size iii) high speed, iv) Low power consumption, v) It is very reliable vi) It is portable vii) less heat generation, viii) is very versatile.

Why μp is called general purpose Computer?

A microprocessor is called general purpose computer/device because it is designed to be a more flexible & versatile tool that can be programmed to perform a wide range of tasks. It is often used in personal computers, servers & other general purpose computing devices. It is a CPU on a single integrated ckt that can perform various operations based on the instructions it receives from s/w programs.

Embeeded system (Smart watch, fire etc...)

It is a computer system that is a combination of computer processor, computer memory & i/o peripheral devices (i.e. Software & hardware combination) that is designed for a specific function, with ~~major components of comp~~ larger system that has a dedicated function within larger system.

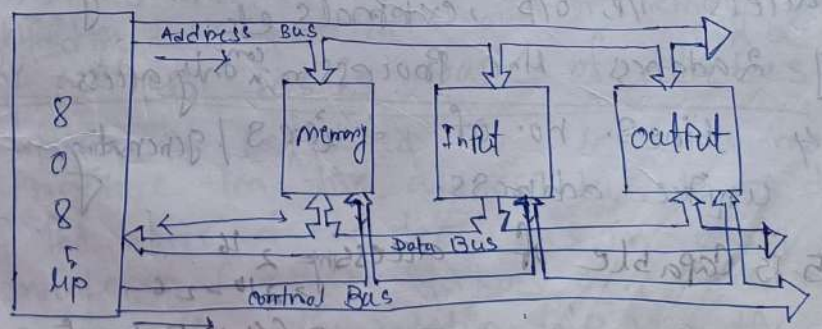
4040 → 4 bit microprocessor
8080 → 8 - - - - - (adv = advance version)
8085 → nmos technology (used for ~~student~~ to produce ICs)
8086
~~history of μP (just read)~~ & microchips

microprocessor - μp is a multipurpose, programmable, clock driven, register based digital device that can read the instructions of memory & access data from input & process it according to the instructions & finally provide the result as output. It can do simple as well as sophisticated task so there is a word called multipurpose. It is basically sequential device.
μc → μp, but μp → μc
Every μc is μp, but every μp isn't μc
→ Register carry the infos.
→ μp is a parallel device

CW

20/7/23

8085 μP Bus Structure



Features of 8085 µP

1) It is an 8 bit µp
2) It is capable of addressing 64 KB of memory.
3) The data bus is 8 bit wide
4) The data bus & least significant 8-bit of address, multiplexing are
5) The device has 40 Pins, requires a single 5V power supply.
6) It can operate with a 3 MHz clock and 2 MHz
7) It has 16 bit address bus.

→ At a time 8-bit data can be received by the Processor.

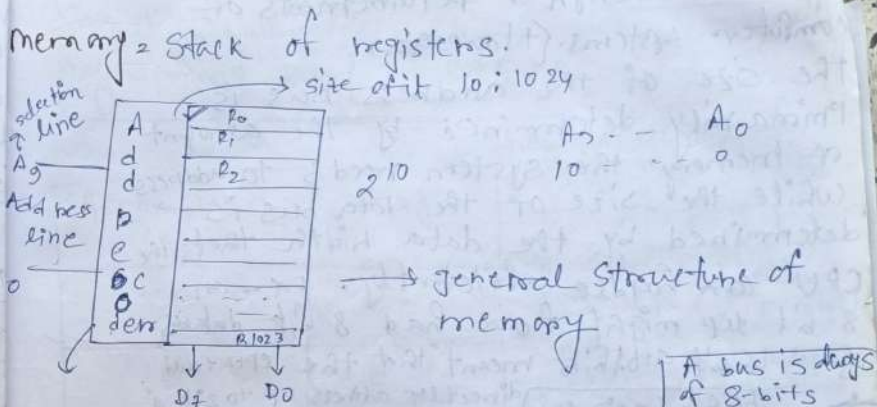
→ Address Data bus is bidirectional, this bus can only carry the info from externals (i/p, memory).

→ Address bus is unidirectional, it is used to identify or access i/p, o/p, externals etc.

→ By 2 address the Processor can only access $2^2 = 4$ lines. no. of devices / generating the unique address.

→ 8085 is capable of accessing $2^{16} = 2^{10} \times 2^6$ devices or 64 KB unique address [B = Byte] = 64 KB no. of

memory = Stack of registers.



→ general structure of memory

A bus is always of 8-bits

Decode the i/p values & enables one O/P.

→ Every register has enable lines.
→ At a time only one o/p is available.

Essential of address bus

i) It identifies the address of the location in cache or main memory that is to be read from or written to.
ii) It carries source or destination address of data, i.e where to store or from where to retrive the data.
iii) It is unidirectional.
iv) Width determines the amount of memory a system can address.

Why address bus is double of data bus

While it is used to be a common practice for the address bus to be double the size of the data bus in early µPs, modern computers systems can have various configuration & the relationship b/w the 2 buses isn't fixed. However, It all depends on the

specific design & requirements of computer systems. However, the size of the address bus is primarily determined by the amount of memory the system needs to address while the size of the data bus is determined by the data width that the CPU can handle efficiently. Ex - an 8-bit µp might have had 8-bit data bus & a 16-bit address bus basically this meant that the CPU could directly address up to 2^16 of memory.

Control bus controls the total structure of the processor. It can be RD, WR, ALE, ...

The C.S. definetly generates by executing the instruction by processor.

Why 8085 µp is 8-bit µp (as ALU structure of 8bit) ALU is of 8-bit thats why d.b is of 8-bits)

As 8085 processor has 8 bit ALU.. Its data bus is 8-bit wide & hence 8 bits of data can be transmitted in parallel form on to the µp.

26/9/23

Prog (software model)

Programming model of 8085 µP

Accumulative (A)	Flags
B	C
D	E
H	L
Stack Pointer	
Program counter	

Data bus ↕ Address bus

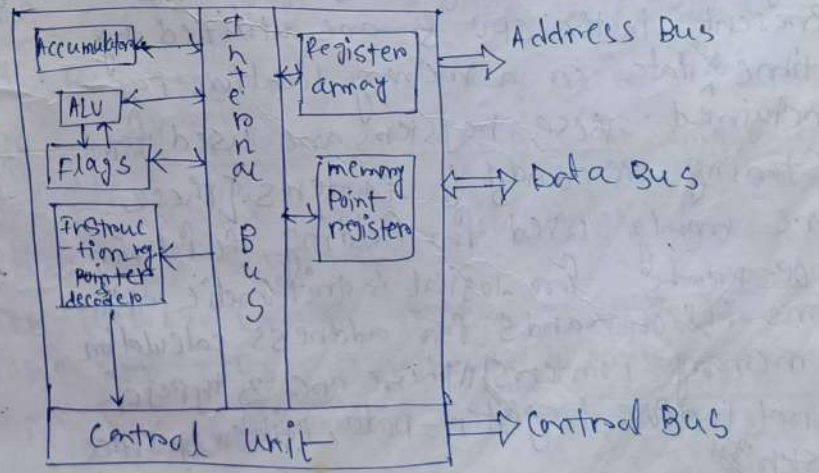
Acc = special type of register (8-bit size)

B, C, D, E, H, L = general purpose register (size - 8 bit) this 7 registers are used by users for programming purpose. So this 7 registers are called programmable registers]

[Reg. Pair B (B & C) D (D-E) H (H-L) Pairs — this always

Reg. Pair should be capable of storing address (16-bit) pairs with one another not with another reg]

Hardware model of 8085 µP

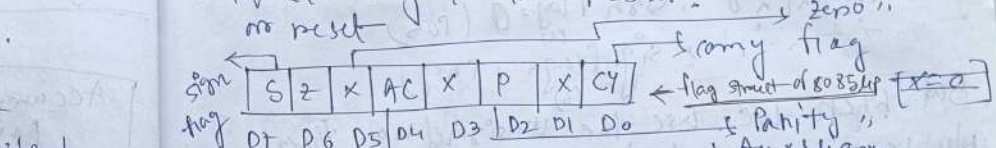


model - (conceptual representation of a real object. There are 2 types of model; hardware & software model.

Accumulator (and why it is special) - It is a type of register included in a CPU that It acts as a temporary storage location which holds an intermediate value in mathematical & logical calculations. It is used as a default destination for most arithmetic & logic instructions which simplifies programming, so it is called special purpose register. [It can be used as both source & destination of the result.] of the operand

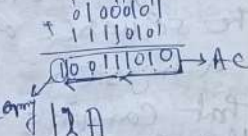
General Purpose register (only store the data) They are extra registers that are present in the CPU & are utilized any time after data or a memory location is required. these registers are used for storing operands & pointers. These are mainly used for holding the following i) operands for logical & arithmetic operations. ii) operands for address calculation iii) memory pointers. There are 3 types of general purpose register - Data, Pointer & Index register

Flag - group of 8 flipflops. this ff basically set according to the result.



In 8085 there are 5 no. of flags.

- logarithm term of binary value x.
- carry flag is set after 8 bit if the result is more than 8 bit (for add)



For this, CY = 1

- If the content of acc is even, then P = 1

If for this, P = 1 (If it is odd, P = 0)

- If there is a 'cy' after lower 4 bit that time 'AC' flag will be set

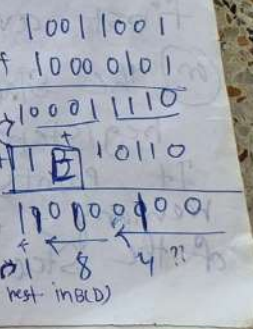
for this, AC = 0 (∵ 0 to 0) Set so, the 'AC' will be 0

At BCD the 'AC' basically used as internal.

- µP doesn't understand the BCD, it only understand the 'H'.

H to BCD conversion. $(11E)_H$ or $(184)_{BCD}$

- DAA easily convert binary value to 'BCD' value (Desimal Adjust Accumulator)
- If the AC (Accumulator) content is '0' then 'Z' will set, Here, Z = 0



• We use unsigned no. to increase the length of 'AC'
Last content = '0', sign flag = 0 (pos)
— — — '1', sign flag = 1 (neg)
(s) (s)

But when we use the unsigned no. this(1) will be neglected/eliminated.

(Addr func)
PC — 16 bit register. It holds the address of the instruction what/that is to be executed. It basically maintain the sequence of the execution. It is a register in a computer processor that contains the address (location) of the instruction being executed at the current time/immediately. It is a special purpose reg.

(Addr func)
SP (Stack Pointer) — [16 bit register. ~~It hold the address of the instruction.~~ It basically hold the top location/address of the RAM (that is used for temporary data storage). It is a small reg. that stores the memory address of the last data element added to the stack or in some cases, the first available address in the stack.] //

(or) the Stack Pointer is also a 16-bit register used as a memory pointer. It points to a memory location (in R/W memory > called) of ~~the~~ top stack. the beginning of the Stack is defined by loading 16-bit address in the Stack Pointer. It is always incremented/decremented by 2 during push & pop operation.

• To end the prog. ~~&~~ we use HALT, RST 1 etc.
memory assigned with the hdd of the address
for this this 2 reg. are of 16-bit (SBPC)

Instruction register (IR) - It is a 8-bit reg.
It holds the operational codes for registers.
It is the part of a CPU's control unit that holds the instructions ~~to~~ are currently being executed or decoded.
[It is also a special purpose register.
It is n't accessible to the programmer.
i.e. there are no instructions by which the programmer can load it with values of his choice.] //

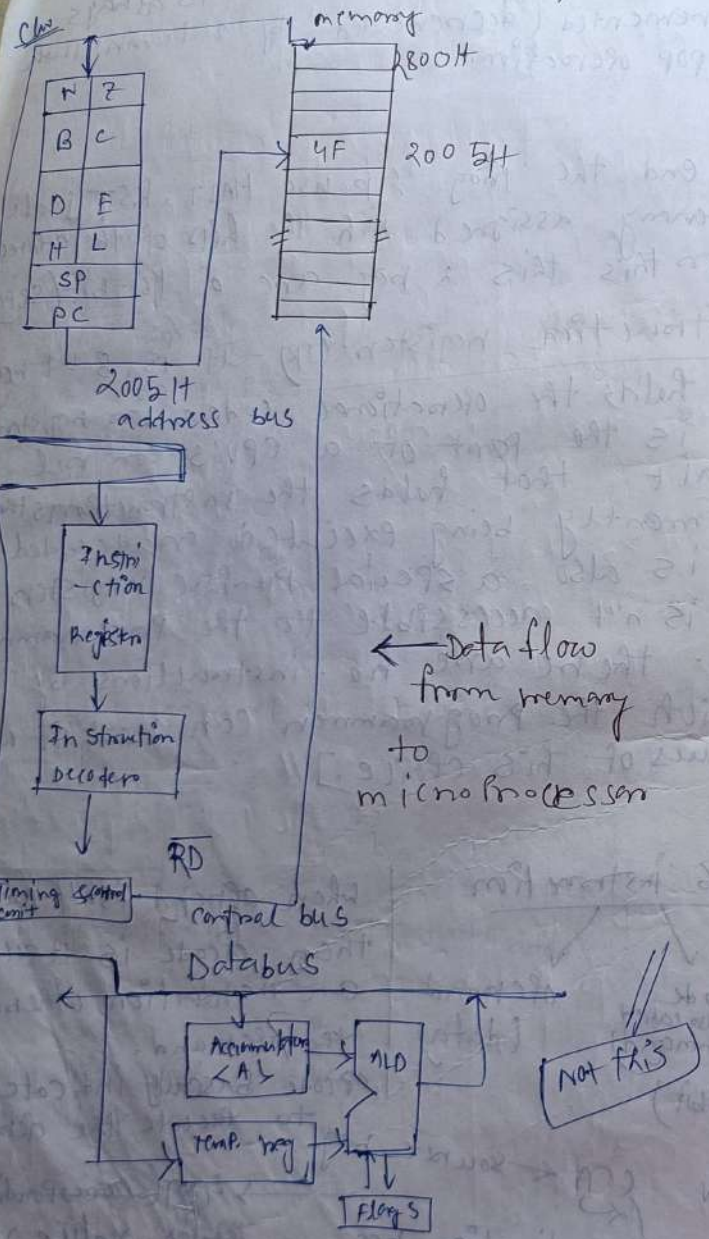
• 256 instruction
  - opcode (also called (command) (8bit))
  - operand (data)

when operand isn't present then opcode is it self an instruction, otherwise the operand.
opcode basically indicate how to execute the operation

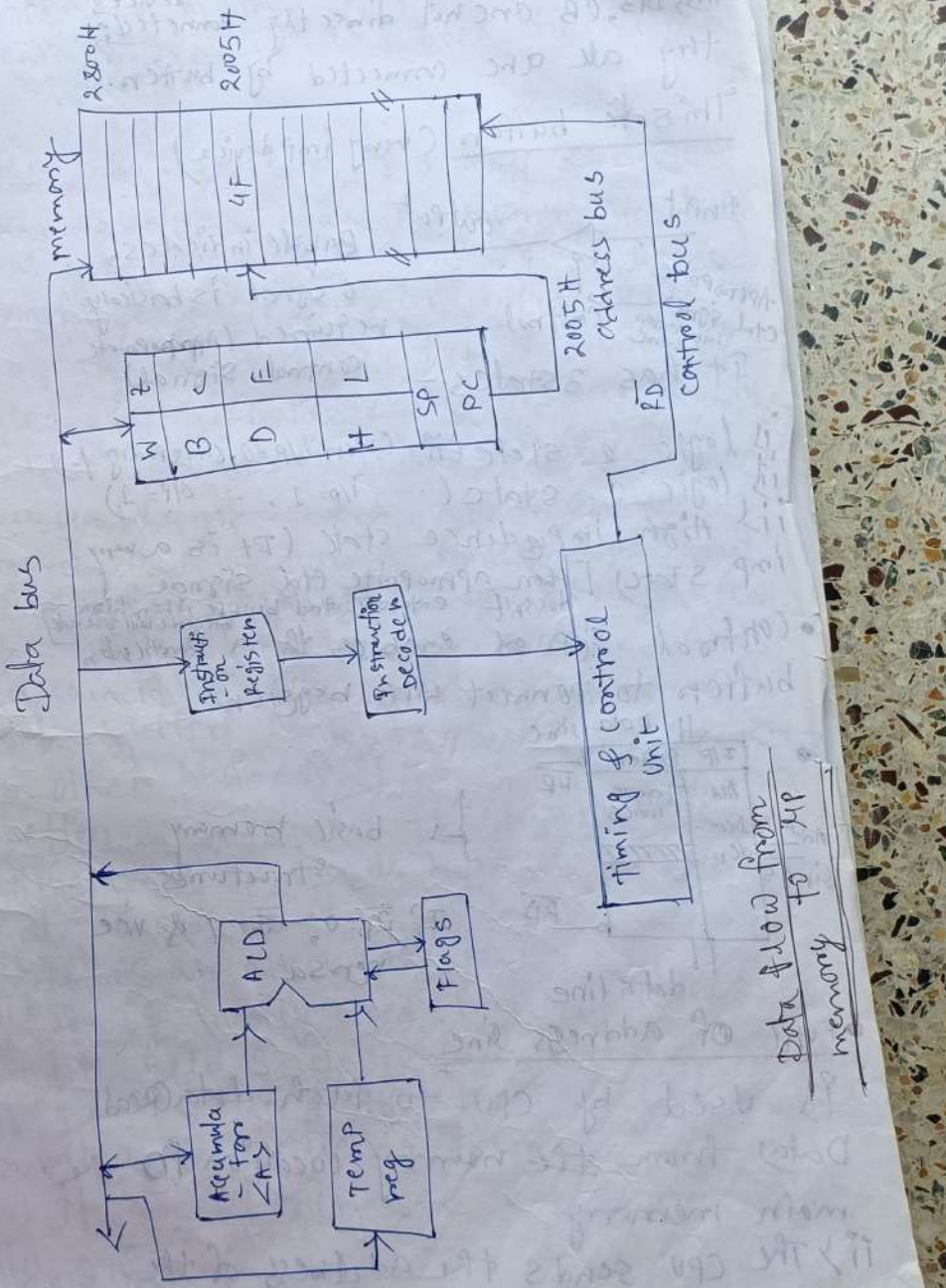
• mov (A ← source reg, destination reg.) → 4 bit (corresponding states value)

• ADI D0

27/7/23

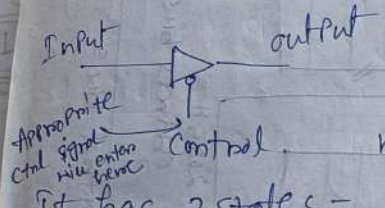

memory
8001H
4F
2005H
W Z
B C
D E
H L
SP
PC
2005H address bus
Instruction Register
Instruction Decoder
Timing & control unit
RD
control bus
Data bus
Accumulator <AL
ALU
Temp. reg
Flags
NOT FIG'S
Data flow from memory to microprocessor

PTO (→)


memory
2800H
2005H
4F
W Z
B C
D E
H L
SP
PC
2005H address bus
RD control bus
Data bus
Instruction Register
Instruction Decoder
Timing & control unit
ALU
Flags
Accumulator <A>
TEMP reg
Data flow from memory to MP

AB, DB, CB are not directly connected to the external devices they all are connected by buffers.

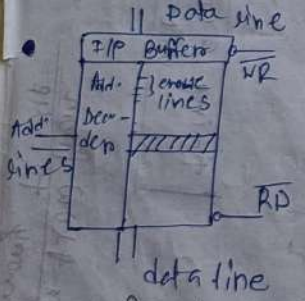
Tri state buffer (very imp device)



It has 3 states -

i) logic 0 state (when i/p = 0, corresponding o/p = 0)
ii) logic 1 state (--- i/p = 1, -- o/p = 1)
iii) high impedance state (It is a very imp state) [when appropriate ctrl signal doesn't enter control bubble, then high impedance will generate]

- Control signal enables a particular buffer to connect with register.



↳ basic memory structure

If $\overline{RD}$=0, $\overline{WR}$=1 & vice versa

- Func of address line

i) Used by CPU to fetch data (Read Data) from the memory location of the main memory.
ii) The CPU sends the address of the memory location, on the Address bus.
iii) The data in that memory location is sent on the data bus back to CPU. //

Buffer - Buffer is a temporary storage area, usually a blak in memory, in which items are placed, while waiting to be transferred from an input device to an output device. It is mostly used for i/o processes. As an example if you want to print a long document you would not want your CPU waiting around asking your printer "Are you ready for another paragraph?" Instead, CPU will fill a memory buffer with document's data, instruct printer to print buffer contents, and go back to its other business. //

function - i) It is used to compensate for differences in speed between two process that exchange or use data.
ii) It is an area of RAM
iii) It is mostly used for i/o processes
iv) It holds data for advance processing
v) It is used to store data temporarily before using them.
vi) It doesn't increase accessing time.
vii) Streaming music or streaming video is example of buffer

• At a time 1 device will be controlled by the control bus & the working will happen correspondingly.

8085 µP - the 8085 µP is an 8-bit µP that was developed by Intel in the mid-1970s. ~~It~~ the architecture of the 8085 µP consists of several key components, including the ac, registers, PC, SP, instruction register, flags register, data bus, address bus & control bus. The ac is an 8-bit register that is used to store arithmetic & logic results. It is the most commonly used register in the 8085 µP & used to perform arithmetic & logical operations such as add, subtract etc (bitwise operations also).

It ~~has~~ has 6 general purpose reg. including B, C, D, E, H, & L which can be combined to form 16-bit register pairs. The B & C registers can be combined to form BC register pair & likely all of remaining 4 registers can be combined into DE & HL. These ~~regis~~ register pairs are commonly used to store memory addresses & others data.

The PC is of 16-bit that contains the memory address of the next instruction to be executed. It is incremented after each instruction is executed.

The SP is a 16-bit register that is used to manage the stack. The SP is used to keep track of the top of the stack.

The instruction register is an 8-bit register that contains the current instruction being executed. It is used by µP to decode & execute instructions.

The flags register is an 8-bit register that contains status flags that indicate the result of an arithmetic or logical operation.

It has address, data & control bus.

~~(Analog to digital~~

ALU - It is used to perform mathematical operations like addition, multiplication, subtraction, division etc. Different operations are carried out in ALU: logical, Bit-shifting & Arithmetic operations.

Flag register - It is a 8-bit register that stores 0 or 1 depending upon which value is stored in the accumulator. Here 5-bits are imp & rest of 3-bits are 'dont' care conditions'. It is a dynamic register as it checks whether the

result is 0, positive or neg, whether there is any overflow occurred or not or for comparison of 2 8-bit numbers carray flag is checked.]
we can say that flag register is a status register & it is used to check the status of the current operation which is being carried out by ALU (i.e it is used to verify & check the results of the operations).
Diff. fields of flag reg. —

i) Carry flag — It is set when an arithmetic operation generates a carray.

ii) Zero flag — It is set when the result of an arithmetic or logical operation is '0'.

iii) Sign flag — The sign flag is set when the result of an arithmetic or logical operation is negative.

iv) Parity flag — It is set when the result of an arithmetic or logical operation has an even number of 1 bits.

v) Auxiliary Carry flag — It is also known as half carry flag. It indicates when a carry or borrow has been generated out of the least significant fours bits of the accumulator register following the execution of an arithmetic instruction. It is primarily used in decimal (BCD) arithmetic instructions.

(iv) flag structure of 8085 µP

Temporary Register — A temp reg is the only register that can be read & written more than once in a single instruction. Each temp register has single-write & triple-read access. [Therefore, an instruction can have as many as three temporary registers in its set of input source operands.] ||

ALD (Analog to digital converter) — [It stands for automatic atomic layer deposition is a technique capable of depositing a variety of thin film materials from the vapor phase].

Interrupt Control — It is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. [An interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current working process.] ||

Timing & control unit - It is also used to control the operations performed by the microprocessor & the devices connected to it. There are certain timing & control signals like control signals, DMA etc.

Address Bus - It is a group of conducting wires which carries address only. Address bus is unidirectional because data flow is one directional, from MP to memory or ~~vice versa~~ ~~the~~ from MP to I/O devices. It is of 16 bits.

Data Bus - Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information results of arithmetic ~~ope~~ etc between memory & the MP. Data bus is bidirectional in nature. The data bus width of 8085 MP is 8-bit

~~Control Bus~~ - The control bus is bidirectional & assists the CPU in synchronizing control signals. [to internal devices & external components. It is comprised of interrupt lines, byte enable lines, r/w signals & status lines.

Memory - It is the electronic holding place for the instructions & datas, ~~ted~~ a computer needs to reach quickly. It's where, information is stored for immediate use. the memory is one of the basic functions of a computer, because without it, a computer would not be able to function properly.

CPU - A central processing unit, also called a central processor, main processor or just processor, is the electronic circuitry device that executes instructions comprising a computer program. the CPU ~~bas~~ performs basic arithmetic logic, controlling & i/o operations specified by the instruction in the program.

ALU - It stands for arithmetic logic unit. It is a key component of a computer's central processor unit. [The ALU performs all arithmetic & logic operations that must be performed on instruction words. the ALU is split into 2 parts in some MP architectures: the AU & LU.]

CU – It stands for control unit. It is a circuitry within a computer's processor that directs the operations. It instructs the memory, logic unit, and both output & input devices of the computer on how to respond to the program's instructions.

Address decoder – It is commonly used component in microelectronics that is used to select memory cells in randomly addressable memory (RAM) devices. Such a memory cell consists of a fixed numbers of memory elements or bits. The address decoder is connected to an address bus & reads the address created there.

Instruction register decoder – It decodes the information present in the instruction register. It identifies the instructions. It takes the infos from instruction register & decodes the instruction to be performed.

Memory pointer register – It is also known as memory address register (MAR). It holds the address of the location to be accessed from memory. It stores the address / location of the prog / instruction that is getting to be fetched.

Register array – It consists of registers identified by letters like B, C, D, E, H, L & accumulator.

Internal Bus – It is also known as internal data bus, memory bus, system bus or front-side bus, connects all the internal components of a computer, such as CPU and memory, to the motherboard. This are also referred to as local buses, as they are connected to local devices intended to

OPCode – It is a collection of bits that represents the basic operations including add, subtract, mul, complement etc.

Operand – It is an object that is operated on by an operator. They can be decimal, single & double in data types.

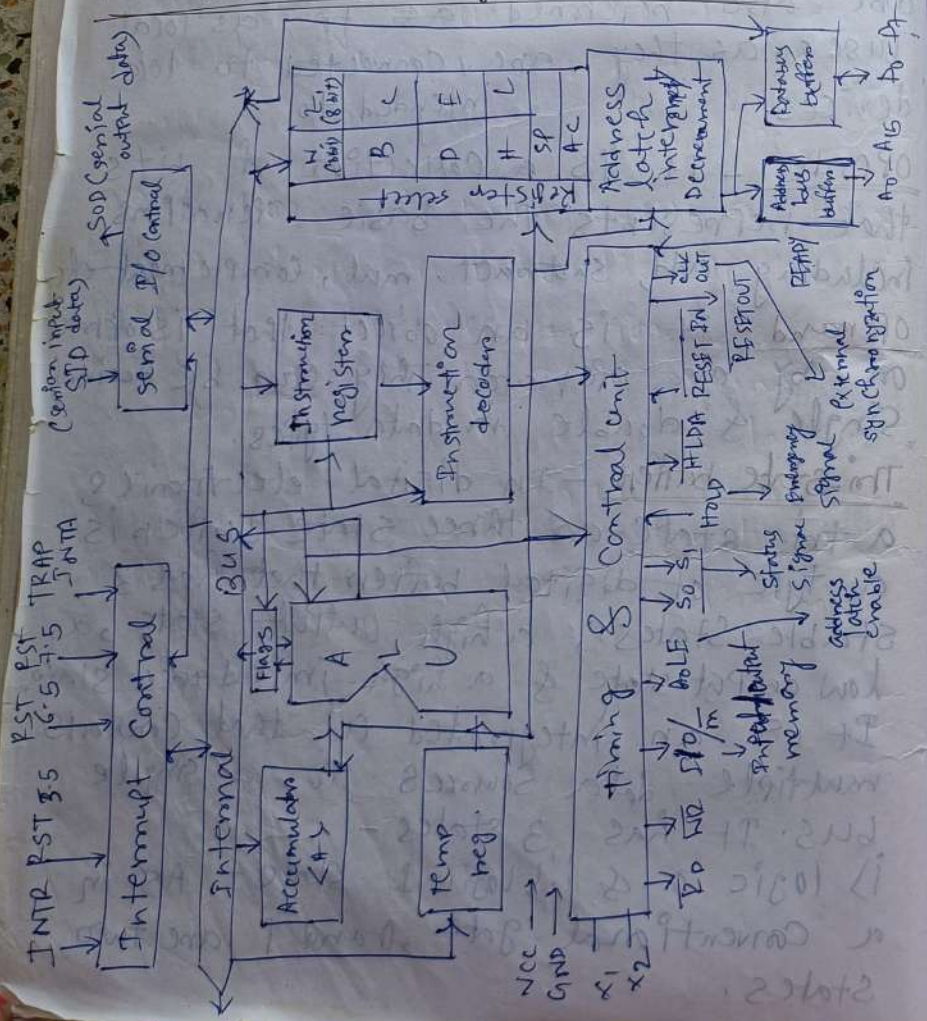
Tri-state buffer – In digital electronics, a tri-state or three state buffer is a type of digital buffer that has 3 stable states; a high output state, a low output state & a high-impedance state. It is an integrated ckt that connects multiple data sources to a single bus. It has 3 states –
i) logic 0 & ii) logic 1 state – As in a conventional gate, 0 and 1 are two states.

iii) high-impedance state - It behaves like an open ckt. If o/p is not connected, then there is no logical significance. When the control i/p is 0, the o/p is disabled & the gate will be in a high impedance state.

(1)

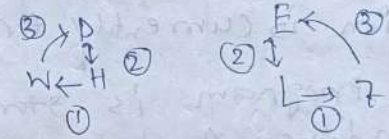
2/8/23

Functional block diagram of 8085/4P



Function of SP, PC - done

- W & Z are called temporary registers bcoz because it is used temporarily.
-



Instruction - It contains opcode & operand

- opcode (8-bit)
- operand (8-bit) } 16 bit address of data

- LDA 9000

↓

not the data, it is the address of the data

2nd Ref

The function of SP -

i) It points to the location of the top of stack.

ii) It is used to keep track of the current location in the stack, & it is updated as data is pushed onto or popped off of the stack.

iii) It is used by CPU to know where to return to after a function call so that the program can continue executing where it left off. (thats' why

Stack is used for recursion).

Function of PC

i) It keeps track of the address of which instruction is currently executing

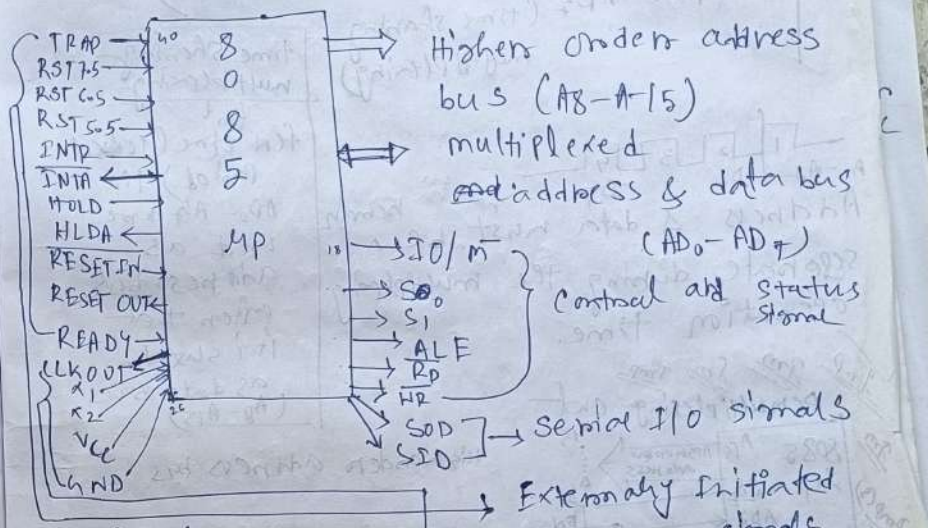
ii) The machine program is stored in memory at address 0. //

iii) When a sequence of instructions are being executed the program counter will be incremented to point at the next instruction.

Instruction - An instruction is a binary pattern designed inside the µP to perform a specific function. In other words it is actually a command to the µP to perform a given task on specified data. The entire group of instructions ~~are~~ is called instruction set. Instruction contains opcode & operand.

9/8/23

Pinout & signals of 8085 µP



signals

1 → address bus
2 → Data bus
3 → Control & Status signals
4 → Power supply & frequency signal
5 → Externally initiated signal
6 → Serial I/O signals

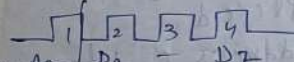
- [INTA, HLDA, RESET OUT → Acknowledgement signals]
- total pin = 40

Address bus - 16 bit (unidirectional)

- → Higher order address bus [A8-A15]
- → lower order address bus [A0-A7]

Data bus - 8 bit (it multiplex with lowers order address)

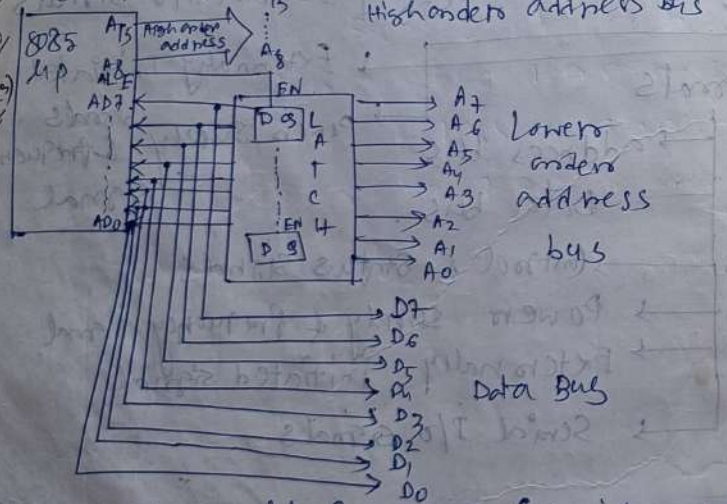
AD_0 - AD_7 (time sharing multiplexing)



Address & data must be seperate during the multiplexing operation time.

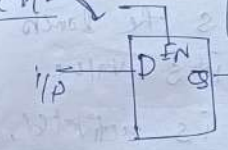
time sharing multiplexing
for
few time (1 clock Period) the AD_0 - AD_7 are used as address bus after that it is used as data bus (A_8 - A15)

Demultiplexing ckt



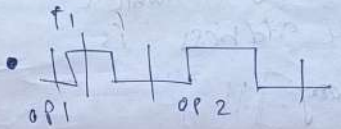
ALE = Address latch enable (positive going pulse that generates every time when µp start it's operation) (µp starts its operation when it's tasks)

• Latch → [EN - enable signal]



(O/p remain same until the new en. stored enter to latch)

(the prev one is the 8-bit D-latch)



let say after t1 op has no chance so o/p → (i.e. o/p also will not receive any changes) So it will disappear (O/p will disappear, but it will not be deleted, it will be hold by memory, if the o/p change the o/p will not be deleted (it will for life time), it will be overwritten by new memory).

Function of the ALE - [It stands for address latch enable. It is used to enable or disable the address bus, the address bus will be enabled during the 1st clock cycle as the ALE pin goes high that is logic '1' during the 1st half cycle.

It Demultiplexed the multiplexed Address/data, Address/status & $\overline{BHE}$ / S7 bus as a output signal.]

ALE - It is an address latch enable signal. It goes high during first T state of (clock cycle)

a machine cycle & enables the lower 8-bits of the address, if it's value is 1 otherwise data bus is activated.
[IO/$\bar{m}$ - It is a status signal which determines whether the address is for input-output or memory.] ||

[moreover, it is an address latch enable signal. It goes high during first T state of a machine cycle & enables the lower 8-bits of address, if its value is 1 otherwise data bus is activated.] ||

[IO/$\bar{m}$, if the sig. = 0, the address of address bus is used for memory & when sig. = 1, the address of address bus (IO/$\bar{m}$) is used for input output.]

S_1	S_0	operation
0	0	HOLD operation
0	1	write "
1	0	Read "
1	1	Featch "

- [When IO/$\bar{m}$ = 0, & S_1S_0 = 00 → ", IO/$\bar{m}$ is S_1S_0 = 01 → write for memory
S_1S_0 = 10 → "
S_1S_0 = 11 → "
~~When IO/$\bar{m}$ = 1, & S_1S_0~~
Read from the memory → IO/$\bar{m}$ = 0, RD = 0] ||
[IO/$\bar{m}$ will be always = 0]

~~next~~ the function of Pin out signals
the applied time of IO/$\bar{m}$ to SID (when they are applied) - Address & Data Bus done

i) control & status signal
IO/$\bar{m}$ = In intel 8085 up I/O & memory operations are differentiated by IO/$\bar{m}$ status signal. IO/$\bar{m}$ stands for 'input-output/memory'. when IO/$\bar{m}$ is logic 0, it means that the address sent out by the processor is for addressing a memory location. when IO/$\bar{m}$ is logic 1, it means that the address sent out by the processor is for addressing an I/O input port. (ALE - done)

ii) S_0, S_1 - These are status signals. They distinguished the various types of operations such as halt, reading, instruction fetching or writing. ⊛ (RPM)

iii) $\overline{RD}$ - It is a signal to control READ operation - when it is low(0) the selected memory or input-output device is read.

iv) $\overline{WR}$ - It is a signal to control WRITE operation. when it goes low, the data on the data bus is written into the selected memory or I/O location.

v) READY - It senses whether a peripheral is ready to transfer data or not. If READY is high(1) the peripheral is ready - If it is low(0) the up waits till it goes high. slow It is useful for interfacing low speed devices.

power supply & clock frequency

⊛ ~~RD (# page)~~ in loose page ⊛

Ideal square wave signal when on time = off time. //

$T_c = t_c + t_c$

$\Rightarrow T_c = 2t$

$\Rightarrow 3 \quad 2t = \frac{2}{f_c}$

$T = t_c + t_c$

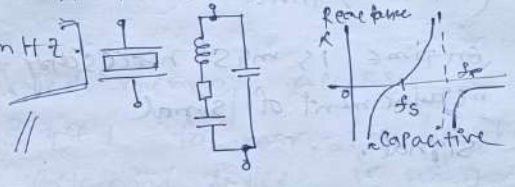
$\Rightarrow \frac{1}{f} = 2t_c = \frac{2}{f_c}$

$\frac{1}{f} = \frac{2}{f_c}$

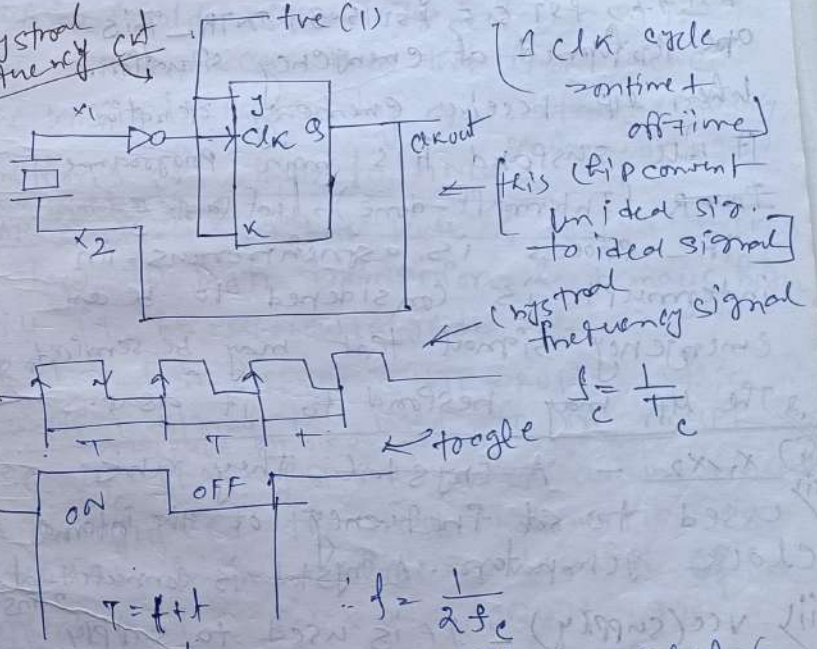
$f = \frac{f_c}{2}$

$f_c = 2f$

Crystal frequency – Internally 8085A works with a frequency of 3MHz internally with clock frequency (this is CF) Hence a crystal of frequency of 6MHz crystal gets connected between X1 & X2. Every operation in the entire 8085 system occurs with the given synchronization process with the clock. It a these are available with a range of standard resonant frequencies from about 10KHz to 100 MHz.



• crystal frequency CKT



PTA

(1) A µP basically operated at 3MHz clk signal - what will be the crystal frequency?

Ans (as, $f = \frac{1}{2} f_c$)

$f = 3$

$\therefore f_c = 2f$

$= 2 \times 2 = 6$ MHz

Dc = $\frac{\text{On time}}{(\text{on+off}) \text{ time}}$

50% on time is must necessary for an requirement of signal

ideal signal.

• Interrupt (emergency operation) - TRAP, RST 7.5, RST 6.5, RST 5.5, INTR. This ope performed at emergency situation. When up receive emergency operation it will suspend it's main programme. INTR (Interrupt-done) - that part of the process is asynchronous. An interrupt is considered to be an emergency signal that may be serviced. The up may respond to it asap.

x) $X_1 X_2$ - A crystal they are used to set frequency of the internal clock generators. A crystal is connected at this 2 pins

xi) Vcc (supply) - It is used to supply Power / voltage.

xii) GND - It is ground reference It is connected to ground - 0V. Power supply.

• INTR is a connector interrupt interrupt

• INTA is the acknowledgement signal of INTR.

• DMA : Direct memory access.

• HOLD is used to make the up know that another external device is using the up for DMA in up address & data bus.

• HLDA (Hold acknowledgement) - By it the up can know that from where the DMA is performing ops performed already.

Functions of TRAP to GND -

External initiated signal

i) TRAP - It is a both level & posi it makes a low to high transition & remains high utill it is acknowledged.

ii) RST - RST : RESET INTERRUPT. this function signal is used to interrupt the up. RST 7.5, RST 6.5 - RST 5.5 are under this. [Ex - RST 6.5, when an interrupt is recognized the next instruction is executed from a fixed location of the memory i.e 6.5 x 8 = 0034 H.]

iii) INTR - The Interrupt Request (INTR) pin is used to request the up to stop its current operation & service an interrupt request from an external device.

iii) INTA – It is an interrupt acknowledgment signal. A MP interrupt acknowledgement is sent after INTR is received.

HOLD – It indicates that another device is requesting the use of the address & data bus. It determines whether new graphics objects are added to the graph or replace objects in the graph.

HLDA – It is HOLD acknowledge. It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clk cycle.

RESET IN – When the signal on this pin is low (0), the prog. counter (PC) is set to '0', the buses are tristated & the MP unit is reset.

RESET OUT – It is used to reset all the connected devices when the MP is reset.

READY – It is used to synchronize the operation of the MP with the external memory or input/output devices.

CLK OUT – This signal is used as the system clock for devices connected with the MP.

P10

10/8/23

Instruction of 8085 MP

The instruction that break the ins. (not consecutive ins.) are called branching ins.

- Instruction
  - Data transfer instruction
  - Arithmetic & logical instruction
  - Branching instruction
  - machine control instruction (NOP)

This ins. basically control the machine & control that operation for which the operations will not delay.

- Instruction (according to length)
  - 1 byte ins. (8-bit ins.) (length of ins.) [Ex – MOV A, B] [At least 1 reg. is required]
  - 2 byte ins. (16-bit ins.)
  - 3 byte ins. (24-bit ins.) (length) [At least 2 consecutive register are required] [Ex – MVI A, 00 (8 bit, 8 bit)]

(3 consecutive reg. are required for storing ins. in memory) [Ex: LDA 9000 (8 bit, 16 bit)]

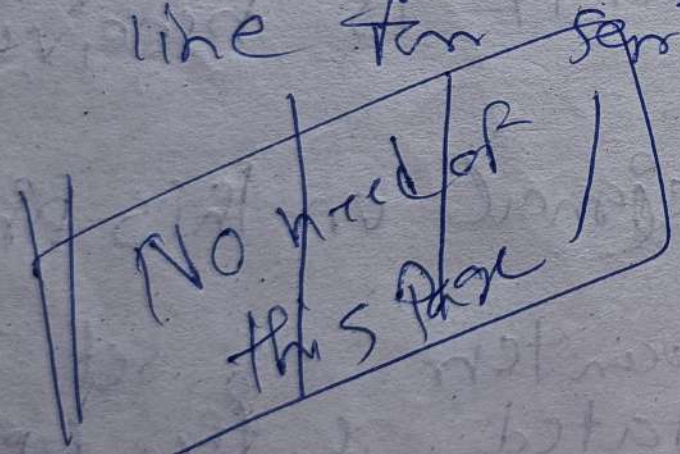
- We always store the ins. at RAM & ROM.

e signal on this
og. Counter (PC)
ISR ... goes now
et
th
her
o
th
ut
ma
r

- Instruction (according to
  - 1 byte ...

Serial I/O Ports

i) SIP & SOP - SIP is a data line for serial input where as SOP is a data line for serial output.



onnected

- we always store the ins

acknowledge.
has receiv
d it wher
t clk
al
nte
se
ese
the
on
the
devices.
is used as
devices connected

for which the operation
delay.

(according to length)

1) INTR - It is an interrupt request signal.
2) INTA - It is an interrupt acknowledgement
sent by the µp after INTR is received.

Reset signals -

1) RESET IN - when the signal on this pin is low(0), the program counter is set to 0, the buses are tristated & the µp unit is

• we always store the ins. of RA

Function of RAM – i) It is used to store the temporary information for limited (events for current ins. processing) time.
ii) It can be directly accessed by the processor.
iii) It is volatile in nature as it automatically erased when Computer shutdown.
iv) It stores data in MBS.
v) It's data can be read, erased or modified.

Function of Rom
i) It stores data permanently.
ii) Here data remains even after Power supply isn't present. (Non volatile)
iii) Rom data is read only.
iv) It is used to store data that is needed to bootstrap the computer.
v) It stores data in MBS.

Machine code – (Uncountable noun) It is a way of expressing instructions and information in the form of numbers which can be understood by a Computer or microchip. It is also known as machine language.

- 64 KB
  - Rom 32 KB: 00000 H – FFFF H
  - RAM 32 KB: 8000 H – FFFF H

- memory
  R0
  R1
  MVI A 00
  LDA 00 00
  Rn

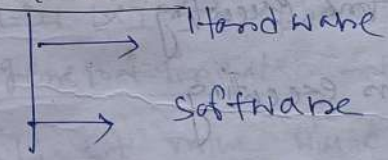
Memories → Assembler → **machine code**

Memories: users basically comfortable at that language that is consist of alphas characters.

machine code: The µp basically comfortable at that language that is the combination of 1, 0.

- B, C, D, E, H, L, A, M = memory
- HL value is basically used for the address of memory registers (By default).
- Before using the M, 1st initialize the value of HL (Why?)

Assembler



Assemblers – It is a software that converts an assembly language code to machine code. It takes basic commands & conve

-rts them into binary code that CPU can use to perform basic operations. It is also named as compiler of assembly language.

Mnemonics - A mnemonic is a term, symbol or name used to define or specify a computing function. [mnemonics are used in computing to provide users with a means to quickly access a function, service or process by passing the optional name instead of lengthy method used to perform or achieve it.]

Data transfer instruction

① mov → copy from source to destination

general format (when a no. that is variable (not fixed) move to the destination)

mov R_d, R_s → Source register, destination } they are any one of memory or programmable register.

Ex-

① mov A, B

A = 35H

B = 90H } Before executing

A = 90H } After executing

B = 90H

Size of this instruction = 1 byte

Here operand is = B

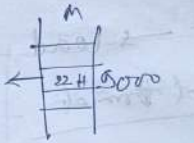
Here values, only consists of alphabets then ins. size = 1 byte]

② mov A, M

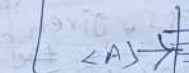
A = 35 (before execution)

A = 22 (After execution)

HL	
90	00
	A = 35



[if, mov M, A



]

② MVI → move immediate 8 bit data. (then a fixed no. move to the destination)

general format - [MVI R_d, 8 bits]

this is 2 byte ins. as itself 8-bit is a 1 byte & R_d = 1 byte (or MVI??)

Eg- MVI A, 90

(A) = 90

MVI M 90

giving address/value (content to HL → MVI HL 90, MVI HL 00

(9000 is the address of the memory, if nothing is defined at HL, then it will automatically take the garbage values, so we have to initialize / define it is)

③ ~~LDI~~ LXI → Load immediate 16 bit value

general format - ~~ADI~~ [LXI Register Pair, 16-bit value]

(16-bit value will directly transfer to the destination). It is used to initialize a memory pointer.

Eg- LXI H/B/D, 9000 ∴ HL ⇒ 90 00 → (lower content)

(Higher content)

① LDA → load accumulator direct

general format – | LDA 16 bit address |

It loads the content of the memory whose address given in the instruction from the accumulation. Address of the memory reg is directly given in the ins.

Eg- LDA 5000H <A> ← [5000]

② STA → store accumulator direct

general format – | STA 16 bit address |

It stores the address of the memory. The storing address is given directly to the instruction

E.g - STA 5000H [5000] → <A>

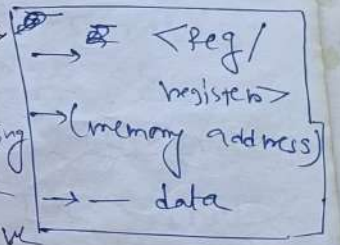
① write a program to store 90H in acc. Then transfer it's content to C & to memory location 8500H using direct addressing. Also transfer the content of reg C to memory location 8600H using indirect approach.

mov M, C

memory address	mnemonics	Remarks
5000H	MVI A, 90H (2 byte ins)	<A> ← 90H
5002H	mov C, A (1 byte ins)	<C> ← <A>
5003H	STA 8500H (3 byte ins)	<A> → (8500)H ← address
5006H	LXI H, 8600H (as there is no ins so we have to use H)	<HL> ← (8600)H ← this isn't address H's value
5009H	mov M, C	(<HL>) ← <C>
500AH	RST 1/HALT	END

As the value 8600 is address to store

Remarks format (general)



we are doing this step as for C we can't use STA, we have use 2 byte ins. But by doing this there may be a chance that we are sending them at to incorrect location rathers than 8600. So

Ex-② (from net)

No need

1st before that we have to lode that location 8600 at the ~~memory~~, ie we. As we know that HL value is basically use as address of memory register by default. & as we have to do it by indirect approach we are using LXI & after only taking reg pair, 16 bit address

At next line we move C's content to memory

As the memory address is by default it's address now.

Higher order address bus - The 8085 µP has 8 pins dedicated to higher order address A15-A8. These pins are used to connect higher order address bus. [These pins generate output signals i.e. the bus is unidirectional.]

Multiplexed address & data bus - The multiplexed address & data bus is the bus configuration that's address pins are shared with Dg signal. [By using the shared pins, total pin count is reduced compared to conventional products that use a seperate address and data bus configuration.]

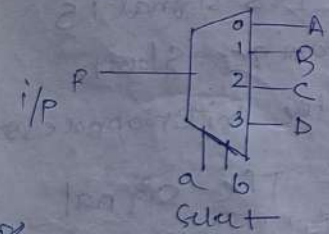
Externally initiated signals - This signal is used mainly to synchronize slower external devices with the microprocessor.

Lower order address bus - The signal lines AD7 to AD0 are bidirectional & they serves as 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.

Time sharing multiplexing - It is a sharing

technique. Specifically, it breaks up the time you have available into a stream of fixed-sized slots, and distributes those slots among the various activities that need to be accomplished.

Demultiplexing ckt - The demultiplexer is a combinational logic circuit designed to switch one common input line to one of several seperate output lines. [The data distributor, known more commonly as the demultiplexer or 'Demux' for short, is the exact opposite of the multiplexer.]



Latch - [A latch is an electronic device that changes its output immediately on the basis of the applied input.] We can use it to store either 0 or 1 at a specified time. A latch contains two inputs - SET & RESET, and it has also 2 o/ps.] // It is a flipflop but it hasn't any clk signal. This is the diff. between flipflop & latch. [It has enable line | signal that is a positive going pulse generated when a new operation is started by MP.] //

Crystal frequency ckt - A crystal oscillator is an electronic oscillator type ckt that uses a piezoelectric resonator, a crystal, as its frequency-determining element.

Toggle - A toggle, in general computing is a switch between one setting and another. The term implies that it is a switch that has only two outcomes: A or B or on or off.

Data transfer instruction - The data transfer instructions moves data between memory and the general-purpose & segment registers & perform operations such as conditional moves, stack access & data conversion

Arithmetic & logical instruction - The arithmetic instructions include addition, subtraction, multiplication, division, comparison, negation, increment & decrement. The logical instructions include AND, OR, Exclusive-OR, NOT, shifts etc.

Branching instruction – These refers to the act of switching the execution to a different instruction sequence as a result of executing a branch instruction. The three types of branching instructions are: Jump (conditional & unconditional) Call (unconditional & conditional) These is used to implement control the flow of programs loops & conditionals i.e. executing a particular sequence of instructions only if certain conditions are satisfied) //

Machine control instruction (MOP) – They have specific features that affect the µp's operations. They control the bus usage and execution machine control instructions in µp //

Opcode	Operand	Explanation of instruction
NOP	none	No operation
HLT	none	Halt & enter wait state
DI	none	Disable interrupts
EI	none	Enable interrupts

1-byte instruction – A one-byte instruction includes a opcode and a operand in the same byte. (Operands) are internal registers and are in the instruction in form of codes. If there is no numeral present in the instruction then that instruction will be of 1-byte, for ex- mov, C, A RAL, ADD etc.

2-byte instruction – It is the type of instruction in which the 1st 8-bits indica-tes the opcode and the next 8 bits indicates the operand [Ex- 1: Task - Load the hexadecimal data 32H in the Ac] It 2-byte ins. the 1-byte is used to specify the opcode and 2nd byte is used for an operand which can be DATA or an 8-bit address Ex- MVI B, 05. machine code = 06, 05

3-byte instruction – It is the type of instruction in which the first 8 bits indicates the opcode & the next two bytes specify the 16-bit address the low-order address is represented in second byte & the high order address

is represented in the 3rd byte. Ex- jmp, LDA etc.

16/8/23

Q) Data transfer ins?

1. LDAx : Load accumulator indirect
2. STAx : Store accumulator indirect
3. LHLD : Load HL reg pair direct
4. SHLD : Store HL register pair direct
5. XCHG : Exchange the HL content with DE

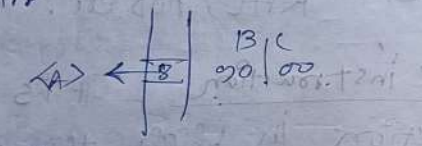
1) LDAx reg pair

LDAx B
LDAx D
LDAx H ✗

same meaning

mov A, M

the m address will be given in HL



• Why LDAx H is invalid instructions?

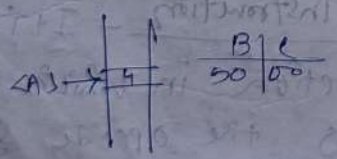
Note that LDAx H is not Provided in 8085 instruction set. This is because, LDAx H is the same as mov A, m in its function [LDAx B & LDAx D are the instructions instruc available but ins. like LDBx rp, LDCx rp etc] //

2) STAx reg pair

STAx B
STAx D
STAx H ✗

Same instruction

mov M, A



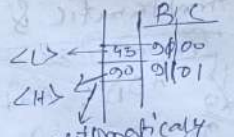
m means HL

[m ≃ HL]

3) LHLD 16 bit memory address

LHLD 9100

Here the "HL" is only valid.



automatically load 90 in <H>

LXI • HL, 9500 H = 95, L = 00

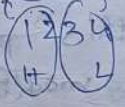
↑ this is for LXI only not for LHLD. There 45 will load to L & then automatically 90 load to the H. (the content of 9500 = 45, 9101 = 90)

4) SHLD 9100

5) XCHG XCHG

• 9500 } CD 3E
9501 } AB 3D

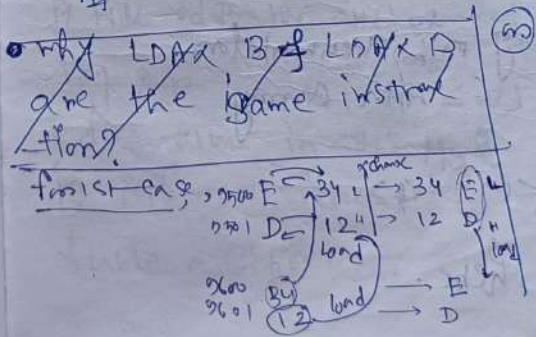
9600 } 34
9601 } 12



→ for this the programme is—

LHLD 9500
XCHG
LHLD 9600
RST 1
LHLD 9500
mov E, L
mov D, H
LHLD 9600
RST 1

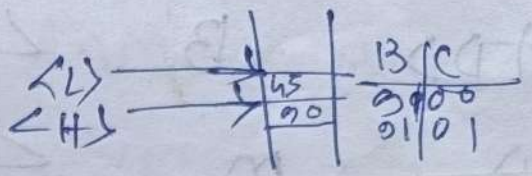
• why LDAx B & LDAx D are the same instructions?



(55)

: meaning
: general format
E.g. Explanation with diagram

SHLD 9100 ⇒



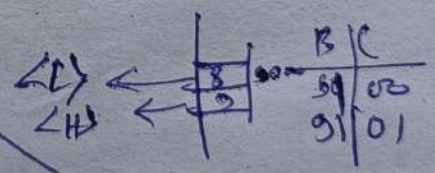
LHLD 16 bit memory address → It will load the content of 16 bit add. in lower order address & then

LHLD

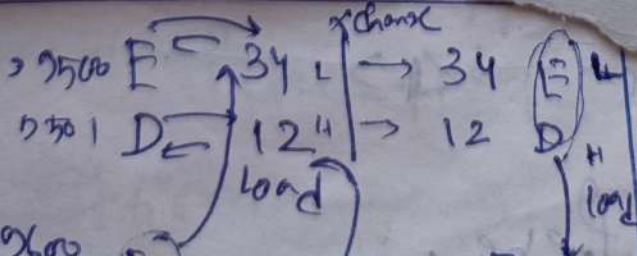
It next 16 bit memory address of above one

automatically it will load the content of next 16 bit address in higher order address.

Eg - LHLD 9500



for 1st case;



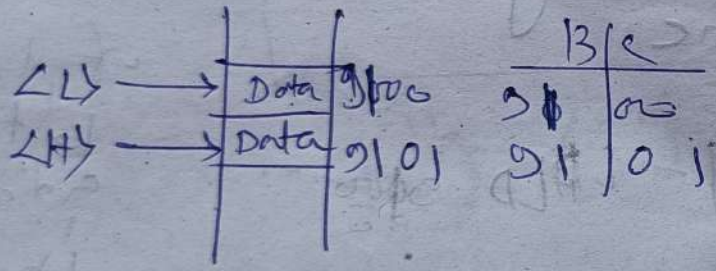
12 ST

this is for LXI only not for

load to L & then automatically

SHLD 16 bit memory address → same as LHLD, but it will store the L5 content in this 16 bit mem. add. & then automatically store the content of <H> in the next 16 bit mem. add.

SHLD 9000
B C



B	C
90	00
90	01

LHLD 9600
RST 1
LHLD 9500
mov E, L
mov D, H

9601

LDAX B

Arithmetic & logical instruction

(Arithmetic ins)

1) ADD : Add reg/memory to accumulator

General format →

ADD reg/m

↓

Content of reg/memory add with the content of Ac & result will place to the Ac.

After executing the instruction the flag will be set/modified

E.g, ADD B <A> + <B> → <A>

ADD M <A> + <M> → <A>

2) ADI : Add immidiate 8-bit data

General format → ADI 8-bit data

ADI 90 <A> + 90 → <A>

After execution of this programme all flags will be modified.

3) ADC Add reg/m to accumulator with carry

ADC B

<A> + <B> + <CY> → <A>

↳ Same thing will be here.

4) SUB D <A> − <D> → <A>

5) SUI 90 <A> − 90 → <A>

6) SBB D <A> − <D> − <CY> → <A>

7) SBI 85 <A> − 85 − <CY> → <A>

8) ACI : add 8-bit immediate data with carry

ACI 90 [I = required the data]

<A> + <90 + <C> → <A>

9) DAD

Double ↳ Addition

<A> AB CD → AC

78 95 → B

D

ADCD

[ADD — B

2 → 8 bit addition

1 → 8-bit addition is simple & another is with carry — Double addition]

DAD regpair

DAD B <HL> + <BC> → <HL>

It will add the <HL> content to the regpair content of <HL> & store the value in <HL>. As 2 addition is performed here it is called double addition.

* only carry flag will be modified here, other will not be modified (only clarify the flag if the ins > 16-bit)

10) INR (IN → increment, R → Reg) : Increment the content of reg or memory by '1'. the result will be store in the same register.

11) INX
12) DCR
13) DCX

INR reg/m ← general format

INR B <B> + 01 → <B>

ADI 01 <A> + 01 → <A>

Diff. between INR & ADI.

INR	ADI
Here all flags will not be modified. Only carry flag will be modified	Here all flags will be modified

11) INX, no flags will be modified here.

General format → INX regpair

INX H <HL> + 01 → <HL>

increment the content of HL pair by 1.

12) DCR, Decrement the content of the reg/m DCR reg/m

DCR m

<m> − 01 → <m>

13) DCX, Decremented by 1. reg pair will be

DCX H

<HL> − 01 → <HL>

No flags will be modified here.

Q) Write a prog to add a no. which is store in memory location 8200 & another no. that is stored in 8300. Store the result from 8500. memory loc.

```
LDA   8200
MOV   B, A
LDA   8300
ADD   B
STA   8500
MVI   A, 00   (<A> ⇐ 00)
ADC   A       (Carry is care to <A>)
STA   8501    (<A> + <A> + <CY> → <A>
RST 1          00 + 00 + <CY> → <A>)
```

(the ins. for store the carry

(store the result at next location)

We can't use SUB (500) as if there will be a carry it will eliminate that. we have to do the comm by the same 3 commands.

(2) WAP for the addition of 10 no. store in successive mem loc starting from 8100. Store the result in 8200 & 8201

A = 00

Input
8100 = 11H +
8101 = 22H + A
8102 = 33H + A
8103 = 44H
8104 = 55H
8105 = 66H
8106 = 77H
8107 = 88H
8108 = 99H
8109 = AAH

B = 00 01 02

ADD reg/m
<A> + <B> → <A>
<A> + <m> → <A>
↑
<HL>

AB
+ 23
(1) 144 → <A>
. FF
2 43

Start
Initialize a counter <C> = 0A
" " Accumulator = 00
" " memory address =
" " carry reg. B = 00

(3) WAP to subtract a no. which is stored in memory location 8200 from a no. which is stored in a mem loc 8300. Store the diff in 8500 & the B. in 8501. use SUB ins (1) & 2's complement (2) method.

(1) Using SUB,

```
LDA 8200
MOV B, A
LDA 8300
SUB B
STA 8500
MVI A, 00
ADC A
STA 8501
RST 1
```

<A> - <B> → <A>

→ for this if B generates we have to add this to the result.

(2)
8300 = 95H
8200 = 80H
→ (Content in in 2's comp. & then ADD so the SUB will be performed)

```
LDA 8200
CMA
ADI 01
MOV B, A
LDA 8300
ADD B
STA 8500
CMC
MVI A, 00
ADC A
STA 8501
RST 1
```

2's complement of content of 8200

[1's complement = the difference between 1 & that no.
9's complement = The difference between 9 & that no.]

[If there will be any carry that will be complemented by CMC]

17/8/23

CW (Logical ins.)

1) DAA: Decimal adjust accumulator
2) CMP
3) CPI
4) ANA
5) ANI
6) XRA
7) XRI
8) ORA
9) ORI
10) CMA
11) CMC

general format :-

DAA (Convert a binary value into a BCD automatically)
↳ DAA reg/m

+ 85 BCD
99 BCD
184 BCD

5+9 = 14 → E
8+9 = 17 → 11
∴ 11E (184 BCD)

A → D

(BCD to Hex (no need))

(184) BCD ?

(0001|1000|0100)

2|184
2|92 - 0
2|46 - 0
2|23 - 0
2|16 - 1
2|8 - 0
2|4 - 0
2|2 - 0
1 - 0

∴ (10000|000)

cy

1000 0101
1001 1001 ← BCD conversion

(1)0 001 1110
0110

10000 100
1 8 4

(We have to add no. is in Hex as MP only understand H)

0001
+ 0111
0001
+ 0111 = 0

2) CMP: Compare reg/mem with accumulator.

CMP B

<A> - <B> After execution of the ins. the res will be shown in flags. After subtraction the original value of <A> & <B> doesn't change.

The diff between original & virtual subtraction (compare)

Items	Subtraction / SUB ins. (original subtraction)	Compare / CMP or CPI ins (virtual subtraction)
1) Purpose	1) It is used to find the diff. between 2 values.	1) It is used to determine the relationship between 2 values without actually performing the subtraction
2) Operator type	2) Arithmetic	2) Logical
3) Result	3) We will get an actual result.	3) Here the MP sets or clear flags (also known as condition codes) in flag registers based on the relationship between the 2 operands

for CMP, CMI <A> value / <B>	CY	Z
P > Q (P: let <A>, Q: let <B>)	0	0
P < Q	1	0
P = Q	0	1

→ <A> - <B>
P - Q ∴ P - Q < 1
∴ there will be a Borrow / carry
[The B is one type of carry]

3) CPI: Compare immediate 8-bit data

CPI 90
<A> - 90

10) CMA: Complement accumulator
that is 1's complement.

11) CMC: Complement Carry " "

4) ANA: logical AND reg/memory with accumulator.

ANA B
<A> . <B> → <A>
CY flag = 0 Auxiliary CY flag = 1

5) ANI: logical AND of 8-bit data immediate with accumulator.

ANI 00 <A> . 00 → <A>
the result of the <A> = 00 (As any content of <A> mull with 00, means 00 (AND gate)).

6) XRA: Logical XOR reg/mem with accumulator.

XRA <A> ⊕ <B> → <A> [<A> res = 00]
[∵ XOR gate]

7) XRI: Logical XOR reg/mem with 8-bit data accumulator.

8) ORA: logical OR reg/mem with ac.

9) ORI: logical OR 8bit data with accumulator.

	SUB	CMP/CMPI
4) use cases-	4) It is used when we need to find the actual numerical diff. between 2 vals.	4) It is used when we want to make decision based on the relationship between 2 vals.
5) Instruction type -	5) It is typically involves Instructions like 'SUB' or 'SUBTRACT' that perform the actual arithmetic operation.	5) It is often performed using instructions like 'CMP' or 'TEST' that set flags based on the result of a comparison without storing the result itself.
6) Flag usage	6) This ins. may also set flags, but the result is stored in a register or memory. Flags might be used for overflow, carry, or sign information.	6) This instructions specifically set flags (such as zero, carry, sign and overflow flags) based on the outcome of the comparison. These flags are then checked in conditional branches.

(3)

23/8/23

Branching instructions

- Unconditional → (JMP 16 bit address)
- Conditional → (JNC, JC, JNZ, JZ, JM, JP, JPE, JPO)

Jump (1 byte)

Jmp 9000	8000	Ins1
	8001	Ins2
SP PC	flags	
	8005	Jmp 9000
	8000	Ins

This can't be used by users

JNC — 16 bit address (jump not carry)

JC — (") 9000 Jump on carry

JNZ (") Jump not zero

JZ — (") 9100 Jump on zero

JM — (") Jump on minus

JP — (") } Specified in sign flag

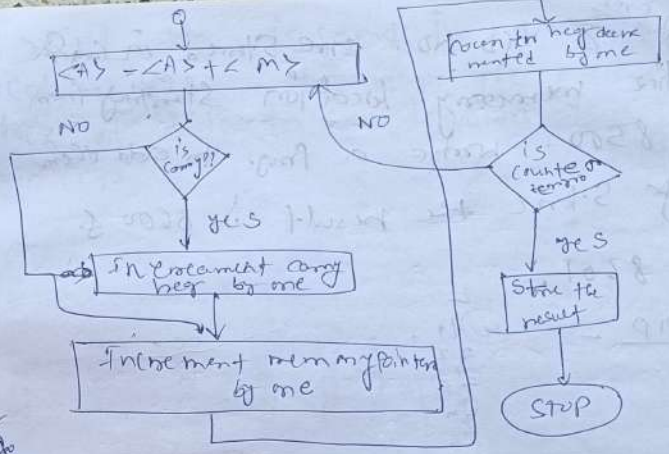
JPE — (") Jump on positive

JPO — (") Jump on Parative even

Conditional Jump Ins:- This transfers the Prog sequence to the described memory address only if the condition is satisfied.

Unconditional Jump Ins — Transfers the Prog sequence to the described memory address.

(2)



(Memory address to store result)

8000	MVI	C, 0A	
8002	SUB	A	
8003	LXI	H, 8100	
8006	MOV	B, A	
8007	ADD	M	<A> ← <A> + <M>
8008	JNC	16 bit address	800C
800B	INR	B	<B> ← <B> + <01>
800C	INX	H	
800D	DCR	C	
800E	JNZ	8007	

(Here the last used hex 80 • 10)

8011	STA	8201
8014	MOV	A, B
8015	STA	8201
8018	RST 1	

taking random address

(2)

Prog

10 no and there so 0A

8000	MVI C, 0A	→ ct = 0A	
8002	SUB A	→ A = 0	<A> - <A> → <A> = 00
8003	LXI H, 8100	→ m = initialize, <HL> ← 8100	
8006	MOV B, A	→ If there is any value of A by any chance so it will give wrong res. so we are moving 0 to B	
8007	ADD M	→ <A> + <M> ↓ <HL> → <A>	
8008	JNC 800C	→ CY not 0 so Jump not to any right address acc chart	CY=0? no / yes
800B	INR B	→ ++ CY reg	CY = 0 ?? no (B)
800C	INX H	→ ++ mem. pointer	
800D	DCR C	→ -- ct	
800E	JNZ 8007	→ ct = 0? → NO, i.e <A> + <M> repeat	ct=0? no / yes
8011	STA 8200	→ Store, 8200 [As this JNZ is executing for 10 times the last used add 8010]	yes
8014	MOV A, B	→ as B = ct = 0 at end so store 0 at A (B = carry content)	
8015	~~ADC~~ STA 8201	→ Store carry at 8201	
8018	RST 1		

8000
8002
8003
8006
8007
8008
800B
800C
800D
800E
8011
8014
8015
8018

(3) 8 BCD NO. are stored in consecutive memory location starting from 8500. Write a prog. to add them & store the result in 8600 & 8601.

MP -

2
11
22
33
44
55
66
77
88

306

264 ← H

36
16 | 36 | 2
32
6

36
+2
38
-32
6

8000	MVI	C, 08
8002	SUB	A
8003	LXI	H, 8500
8006	MOV	B, A
8007	ADD	M
8008	DAA	
8009	JNC	~~800E~~ 800D (Address will change from here)
800C	INR	B
800D	INX	H
800E	DCR	C
8011	JNZ	8007
8014	STA	8600
8015	MOV	A, B
8018	STA	8601
	RST	1

(4) 6 > 8 bit data is stored in memory location 8050, transfer the entire block of data to new memory location starting from C070.

C050 =	AA	AA =	C070
C051 =	BB	BB =	C071
C052 =	CC	CC =	C072
C053 =	DD	DD =	C073
C054 =	EE	EE =	C074
C055 =	FF	FF =	C075

```
~~LDA 8500~~ LXI B, C050
LXI D, C070
MVI H, 06    (this will work like a PC)
LDAX B
STAX D
~~DCR~~
INX B
INX D
DCX H        (Program counter value decremented by 1 & it will run untill <PC> = 00)
JNZ
RST 1
```

H/W (5) 8bit data is stored from memory location 8050, transfer from C070 in reverse order.

C070 + (n - 1) → 1st address to transfer

(4) ... data is stored in memory

(3)

Random add.	Prog	
8000	MVI C, 08	→ ct = 08 (As 8 BCD no. are there ct = 8)
8002	SUB A	→ <A> - <A> → <A> = 0
8003	LXI H, 8500	→ <HL> ← 8500 m = initialize
8006	MOV B, A	→ cy = B = 0
8007	ADD M	→ <A> + <m> → <A> (binary)
8008	DAA	→ binary → BCD
8009	JNC 800D	→ cy = 0 ? → no then jump not to carry
800C	INR B	→ ++ cy reg. It will go to ++ mem. pointer
800D	INX H	→ ++ mem. pointer
800E	DCR C	→ -- ct
800F	JNZ 8007	→ ct = 0 ? → no, then go <A> + <m> → <A>
8011	STA 8600	→ Store res. at 8600 loc
8014	MOV A, B	→ Store cy reg. content
8015	STA 8601	→ Store cy reg. at ...
8018	RST 1	→ end

... transfer ... from C070 in reverse order

C070 + (n - 1) → 1st address to transfer

reg pair → Address

④ Prog | meaning

8000	LXI	B, C050	→	<BC> ← C050
8003	LXI	D, C07D	→	<DE> ← C07D
8006	MVI	H, 06	→	<HL> ← 00 (it will act like a ct)
8009	LDAX	B	→	<A> ← <BC>
800A	STAX	D	→	<A> → <DE>
800B	INX	B	→	++ BC (reg pair)
800C	INX	D	→	++ DE
800D	DCX	H	→	-- HL (ct--)
800E	JNZ 8009		→	if ct = 0? → no then again load & store it <D> in
8011	RST	1	→	end

[8040 + 3 (next 8043)]

⑤ mine logic

Prog | meaning

LXI	B, C050	→	Initialize the memory pointer 1
LXI	D, C070+(n-1)	→	Initialize the memory pointer 2
MVI	H, 00 n	→	Initialize the counter
LDAX	B	→	load the val of <BC> reg pair in <A>
STAX	D	→	Store the val of <A> in <DE> reg pair
INX	B	→	++ mem ptr 1
DCX	D	→	++ mem ptr 2
DCX	H	→	-- ct
JNZ	8009	→	ptr to next

from C070 in reverse order

C070 + (n - 1) → last address to transfer

③ Let n = 6

C050 = AA
C051 = BB
C052 = CC
C053 = DD
C054 = EE
C055 = FF

C070 = FF
C071 = EE
C072 = DD
C073 = CC
C074 = BB
C075 = AA

tip - C070 + (n-1)

AA = (C070 + (6-1)) = C075
BB = C074
CC = C073
DD = C0
EE =
FF =

```
      LXI  B, C050
      LXI  D, C051
      MVI  H, 00   → (this will work like a pointer) PC
 →    LDAX B
      LDAX D
      INX  B
      INX  D       → not this
      DCR  H
      JNZ
      RST  1
```

24/8/23

① Find the sum of 2 - 16 bit binary no.'s store in mem loc 8100 & 8200 respectively. Save the result the mem loc 9500.
A) Using DAD ins.
B) without using DAD ins.

Im prog Imp

② WAP to compare 2 string assume that the 1st byte of both string contain a no. of bytes in that strings (counts how many strs to compare) the starting address of the 2 string is 9000 & 9100 respectively. If both strings are found equal place 11 in mem loc 9500 and else place 22.

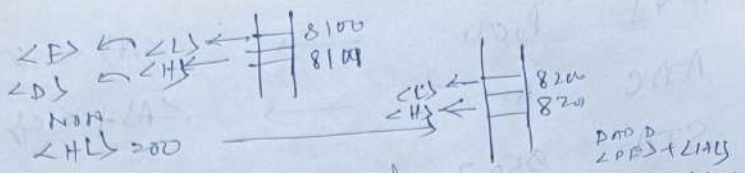
① Input
8100 = 34H
8101 = 12H → 01 contains the higher values
8200 = CDH
8201 = AB H

A)

[DAD B
<HL> + <BC> → <HL>]

  1234
+ ABCD
  BE01 → 9500

```
LHLD 8100   (8100 content loads to L reg & the 8101 content automatically loads to H)
MOV E, L    | XCHG replace of this
MOV D, H    |
LHLD 8200
DAD D       → can take any one of D or E as they are reg pair.
SHLD 9500
MVI A, 00
ADC A
STA 9502    → receives the value from memory
RST 1/HALT  → stores the value from memory
```



(2) 1)

A) Prog

meaning

LHLD 8100 → as in ins, the given 8100 & 8200 memory loc & as DAD ins will be used here so, we have to use <HL> So, LHLD is used otherwise we can use (LDA) or LXI.

XCHG ⎫ MOV E, L
MOV D, H ⎭

LHLD 8200

DAD D

SHLD 9500

MVI A, 00

ADC A

STA 9502

Store carry at next loc of 9501, i.e 9502

RST 1 or HALT → end

B) Prog

meaning

LHLD 8100 → H 8100 → <L>, H 8101 → <H>

XCHG → <L> ⇄ <E>, <H> ⇄ <D>

LHLD 8200 → <L> ← H 8200, <H> ← H 8201

MOV A, E → <A> ← <E>

ADD L → <A> + <E> → <A>

STA 9500

MOV A, D → <A> ← <D>

ADD H → <A> + <D> → <A>

STA 9501

[As 12-bit binary no" is mentioned here so we must have to use LHLD & SHLD]

```
MVI   A, 00      ⟶   <A> ← 00
ADC   A          ⟶   <A> + <C> + <A> ⟶ <A>
                                        ⟶ 00
STA   9502
RST   1
```

② mine logic

// Prog — meaning

```
LXI   B, 9000    ⟶   <B> ← #9000
LXI   D, 9100    ⟶   <D> ← #(9100)
MVI   H, 00      ⟶   Act as ct
LDAX  B          ⟶   <A> ← <B>
CMP   D          ⟶   <A> - <D>
JNZ   ____       ⟶   initially do Prog for not equal
 equal? yes
MVI   M, 22      ⟶   Place 22
JMP   ____       ⟶   Stop it
INR   B          ⟶   ++ B
INR   D          ⟶   ++ D
DCX   H          ⟶   -- H (ct--)
JNZ   ____       ⟶   do it again until ct=0
 ct=0? yes
MVI   M, 11      ⟶   Place 22
STA   9500
RST   1
```

(B) LHLD 8100

XCHG → ≈ [mov E, L / mov D, H] ← Add this two lst

LHLD 8200

MOV A, E

ADD L

STA 9500

MOV A, D

ADC H

STA 9501

MVI A, 00

ADC A

STA 9502

RST 1

(2) CMP B

	A=B	CY	Z
	A>B	0	0
	A<B	1	0

String 1

9000 = 05 ← A

9001 = 11

9002 = 15

9003 = 20

9004 = 25

String 2 ← C

9100 = 05

9101 = 11

9102 = 15

9103 = 20

9104 = 25

[Assume it contains 5 no.

If 1st no. of 1st string = 1st no. of the 2nd string then Proceed]

If equal 9500 = 4

If not = 22

[If 11, then equal

If 22, not equal]

LXI B, 9000

LXI D, 9100

MOV D, M

LDAX B

STAX B

CMP M

JNZ

MVI A, 22

JMP

INX B

INX H

DCR D

JNZ

MVI A, 11

STA 9500

RST 1 ← not that (PTO)

[1st no. of 1st string indicates how much compare has to be executed]

[the ~~2nd~~ of no of 2nd string also do that same above (1)]

[If Z = 1 then don't ~~check~~ place it in 22 ← Otherwise check]

Cos this will be the case when 1st string = 2nd string

(1) H W

2 no-s P & Q are store in mem loc 9500 & 9600. ADD for the null of 2 no.s store the result from the memory loc 9700. (Repeated add)

(2) 2 no. - - - - - - - - - - (same same)

- - - - - Find P/Q.

Store the ~~coefficient~~ quotient 8500 & rem 8600. (Repeated Subtraction)

P Q

4 3

[0 4 0 4 = 0B

0 3 0 3 0 3 = 0B

↓4

[1st check P=0 or not Q=0 or not then Proceed (Checking is essential for the Prog)]

[3 - 2 = 1] [1 - 2 = CY]
(1) inr add ref

[If p > q then subtract]

(1) LDA 9500
MOV B, A
LDA 9600
MUL B
STA 9700
RST 1

LXI H, 9500
MOV B, M
INX H
MOV C, A
ADD M
HLT

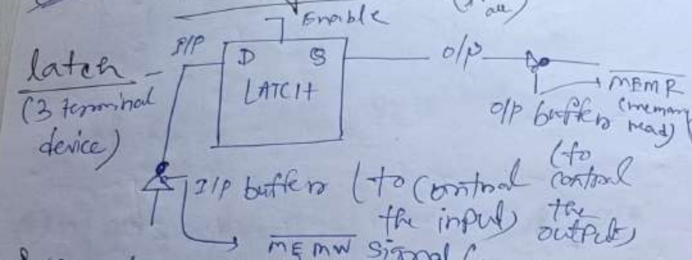
(2) LDA 9500
MOV B, A
LDA 9600
DIV B
STA 8500

LXI H, 9600
MOV B, M
MVI C, 00
MOV A, M

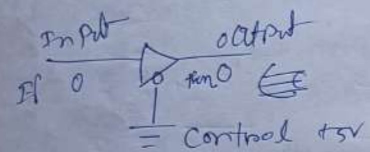
31/8/23

memory interface (Imp all)

latch - (3 terminal device)



Imp buffers (3-state)



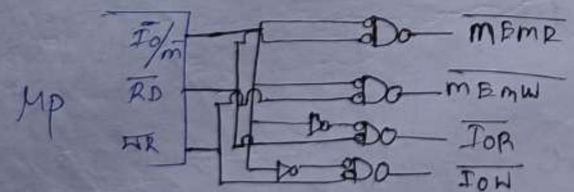
logic 0 (If i/p=0, o/p=0)
logic 1 (If i/p=0, o/p=1 or vice versa)
high impedance state (If control has +5V)

IOR - Input read. It can separate the I/O operation & memory operation

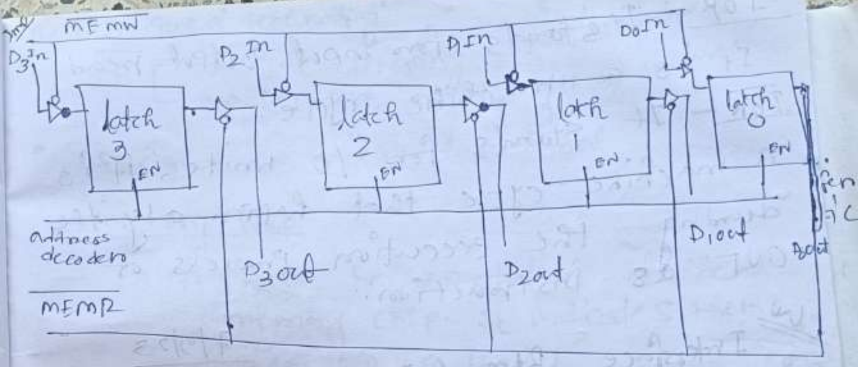
IOW - Input write.

- µP generates this signals

Imp
Imp



memory & I/O select signal



4-bit memory registers

- latch is the basic element of the memory.

Address decoders - In every memory there is a address decoder.

- Why latch is the basic element of the memory?
Latches are called the basic element of the memory for their storing capacity, they are sometimes referred to as bistable memory devices.

MEMR - The IO/M & RD can be combined to generate MEMR (memory read) control signal that can be used to enable the output buffer [by connecting to the memory signal RD]

MEMW - - - - & WR
MEMW (memory write) .. WR]

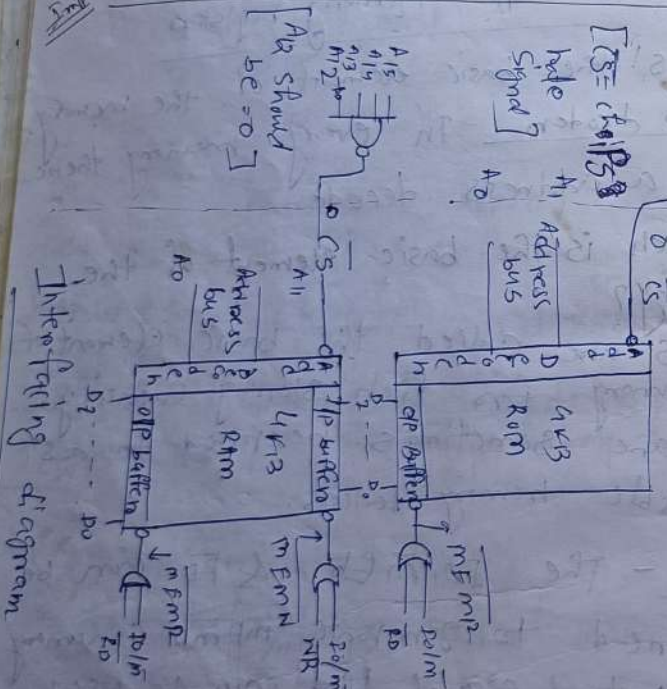
I/P (-

IOR – It stands for input output read. It is a machine cycle.

IOW – It stands for i/o write, it is a machine cycle that happen only the during the execution process of OUT 8s instruction.

7/9/23

Interface RAM & ROM with 8085MP.



4KB = $2^2 \times 2^{10}$ Byte

= 2^{12} Byte → no. address in the address decoders

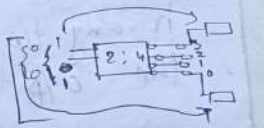
12:4 × 1024 → size of the decoders (it is inbuild size)

The technique of generating $\overline{CS}$
- Gate addressing (8mp) → 10 mnemonics
- Decoders addressing (8mp) → 10 mnemonics

- $\overline{CS}$ signal is active low signal

memory map

→ Graphical representation of the memory chip. It indicates where the chip is located.



Imp

Gate addressing method

1st select the $\overline{CS}$ signal

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0000 (from 0 1st reg)
F				0				0								
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF (from last reg)
F				F				F								
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000 (1st)
E				0				0								
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF (last)
3				F				F								

Diff between last & 1st add.

for 1st Part(←), FFFF − E000 = 1FFF

for 2nd Part(←), EFFF − E000 = 1FFF ↕ Same

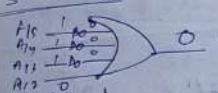
(for Decoders o/p)

for OFFF select $\overline{CS}$

then change (Pmr (←)),

$\overline{CS}$ for Rom

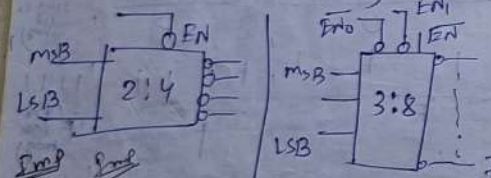
$\overline{CS}$ for RAM



→ for gate add. change these Pmrl with that Pmrl of I.d

if the i/p are 1110 then only the chip will be selected, otherwise not, as if any i/p values are changed then the o/p of the ckt will not be 0.

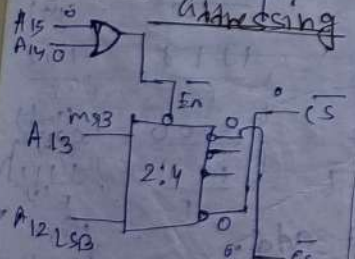
- OOOO to 1FFF 32×13 range, 8000 to FFFF 32×13 range.
- 2 types of decoders



Imp Imp

Decoders

Addressing



(memory map of this is in Pmr memory map of gate cam)

the best of the Pmrl ~~of~~ in Pmr diagram (mentioned by I)

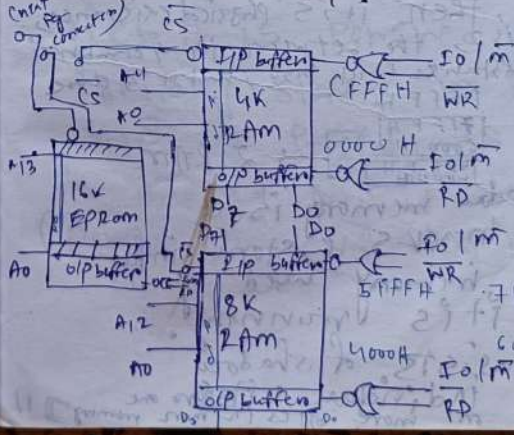
Chip's data signal (CS) - The numbers of chips (bits) in the spreading signal is significantly greater than the data bits. // Chip's rate is the number of pulses per second of a signal (chip's per second) at which the signal is transmitted or received.

Memory map - [memory interfacing/mapping is used to provide more memory space to accomodate complex programs for more complicated systems.] - memory mapping is a mechanism that maps a portion of a file, or an entire file, on disk to a range of addresses within an application's address space. //

14/9/23

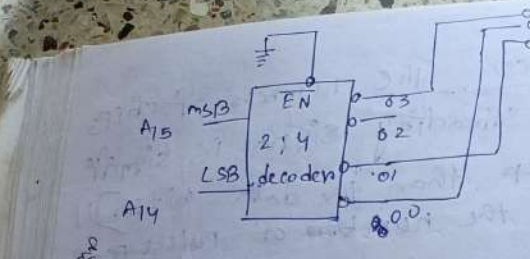
Imp Imp Imp Imp Imp Imp Imp

① Interface 4K RAM, 8K RAM & 16K EPROM with 8085 µp.



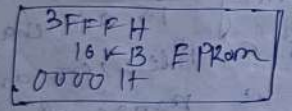
4K = $2^{12} \times 2^{10}$

$= 2^{(12)}$ → Address line



A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H (first reg)
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH (last reg)

FFFFH



If, x = 0, then 4000H

If, x = 1, then 5FFFH

[as x can be any one of 0 or 1]

(if don't care line is 1, then it's physical existance, but if don't care line is 2, the between the 4 combination 1 is physical existance (1st ins), rest of 3 are shadow of the 1st.)

→ 2 RAM

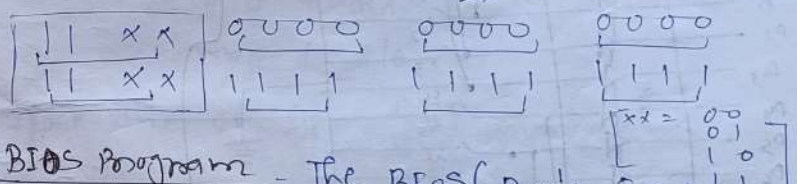
Shadow memory - Shadow memory is a technique that tracks & store info about the computer memory used by a program while it is running. Shadow memory consists of shadow bytes that map to individual bits (zero or more bytes in main memory)

Partial & absolute decoding technique

Partial decoding is converted into the absolute decoding.

Partial decoding is a technique used in MPS when the CPU supports a large address space but the system uses a small amount of memory. In partial decoding only some of the address lines are decoded. This uses less memory as the encoded data isn't retained.

Absolute decoding is a technique used in microprocessors to select a memory chip. In this technique, all the higher address lines are decoded to select the memory chip.



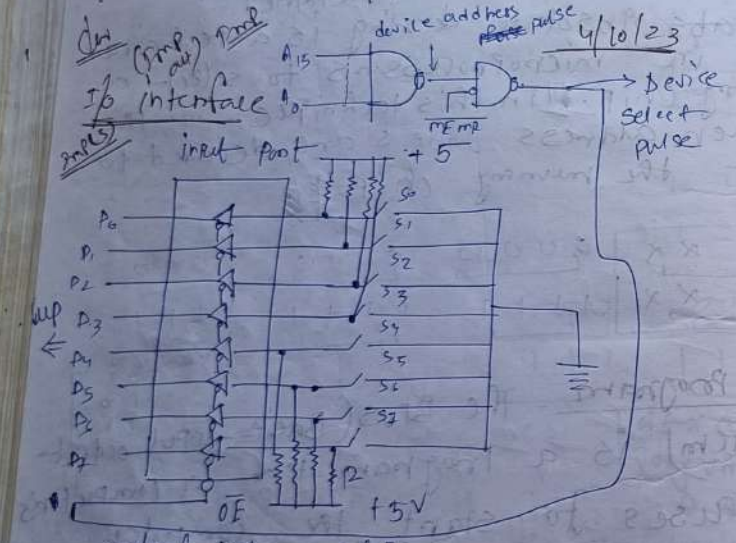
BIOS Program - The BIOS (Basic input output system) is a program that a computer's MP uses to start the computer after it is powered on. The BIOS is typically stored in EPROM. It performs start-up procedures when the computer is turned on. It also performs POST (power on self test)

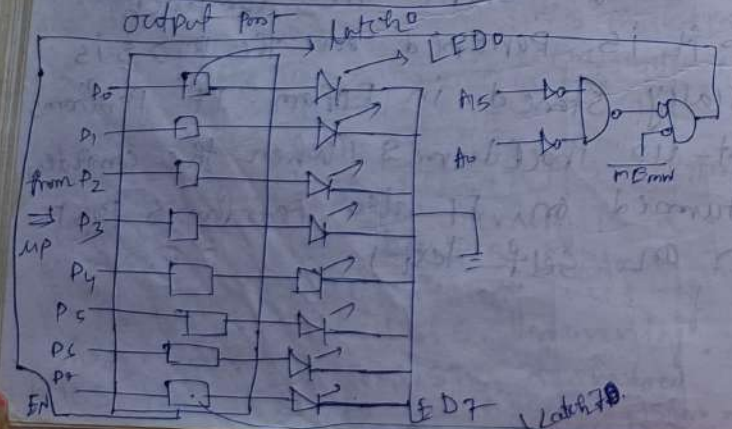
Conv qn, if absolute decoding, must have to draw all decoders & have to draw all address lines. If said 2 RAM, 2 ROM like this then ~~we~~ must 1 ROM must be start from 00 address, rest of files/decoders we can do ~~everything~~ anything)

In Practical aspect, Rom must be 0000.

1st address of device address

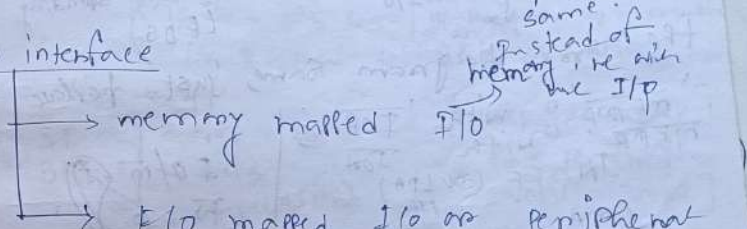
4/10/23





Why i/p port is consist by buffer & o/p port is consist by latch?

latch is necessary to hold the output data for display. the input data byte is obtained by enabling a tri-state buffer and placed in the accumulator.



All are same. Instead of memory we write I/p

I/p range of I/p = $FF to 00

Diff between memory & I/O mapped.

Characteristics	m m I/O	I/p m I/O or peripheral mapped I/O
i) Device address	16 bit	8 bit
ii) control signal for I/O input output or iii) Instruction available	MEMR, MEMW LDA, STA, LDAX, STAX	IOR, IOW IN, OUT
iv) max no. I/O possible	memory map (64K is shared between I/O and system memory)	256 input devices & 256 output device can be taken
v) Execution Speed-	13T (LDA & STA)	10T states

$A_0 - A_{15}$ → all are 1 ∴ LDA FFFF
after that
<A> = FF

LEDs are controlled by the output of input port.

$A_0 - A_{15}$ → memory mapped I/O

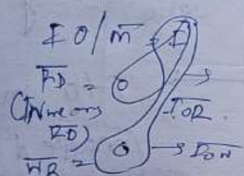
FO ← LDA FFFF
OF ← (MA
(will STA 000 glow & off the corresponding LEDs)

$A_0 - A_7$ → I/O mapped I/O

the prev diagram same, just replace

'MEMR' with 'IOR' &
'MEMW' with 'IOW'

IN FF (≈ LDA FFFF)
CMA
OUT 00 (≈ STA 000)



(imp) Timining diagram (Gomarkis) from here (3)

Graphical represen of instruction execution -tation

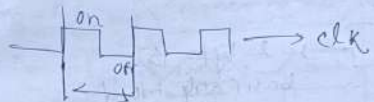
(Q) there are 3 types of instruction—

i) 1 byte ins. (fetch)
ii) 2 byte ins. (opcode fetch + MR)
iii) 3 byte ins (fetch + MR + MR + operation)

Instruction
- opcode (8 bit)
- operand
  1) data
  2) registers
  3) 16 bit memory address

1-byte: MOV A,B → operand (when operand is reg)
2-byte: MVI A,50 → operand (when operand is a value)
3-byte: LDA 9000 (when operand is address)
<A> ← [ ] 9000

Fetch operation – when μP reads the opcode that is called fetch.



1 clock cycle = T state

2 mHz
$T = \frac{1}{F} = \frac{1}{2\times10^{6}} = 0.5\times10^{-6} = 0.5\ \mu s$

machine cycle – It consists of many T states.

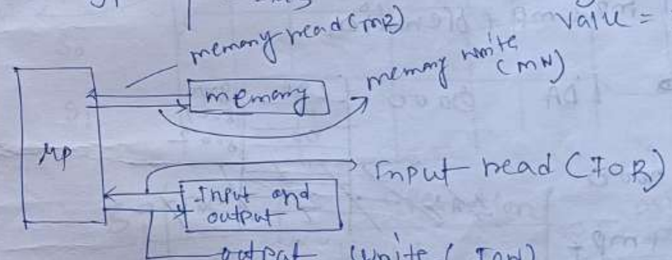
Instruction cycle – it consists of many machine cycles.

T-State – This is the unit that is used to calculate the time taken for execution of an ins.

If the freq is 2 mHz the fetch operation need time = 4T
$= 4\times0.5\ \mu s$
$= 2\ \mu s$

MR = 3T
MW = 3T

[μP generally has the freq value = 3mHz]



• Every ins has a 1-byte opcode, hence every ins starts from opcode fetching.

1-byte Operation

Operation may be present may not be, may be a combination of operation is present.

MOV A, B
F

MOV A, M
F + MR

MOV M, A
F + MW

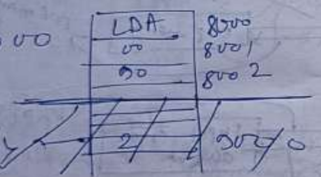
2-byte
operation may be or may not be Present

→ 2 byte is basically must to read 2-byte ins.

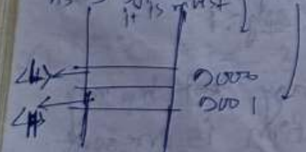
Here F + mR + operation
↓
this must be Present

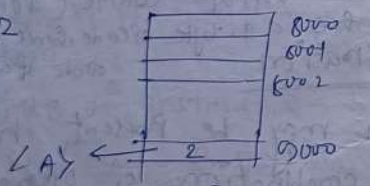
3-byte
F + mR + mR + operation

LDA 8000



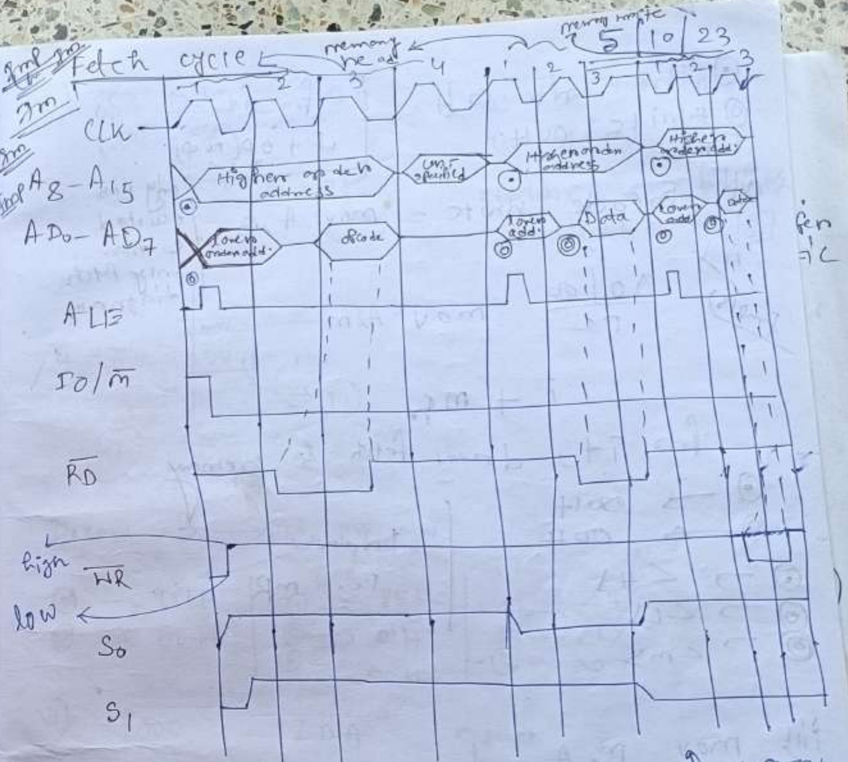
LHLD 8000
F + mR + mR + mR + mR
As 3 byte it is must





F + mR + mR

Fetch cycle



S_1	S_0	Operation
0	0	Halt
0	1	write
1	0	Read
1	1	Fetch

(using as memory cas every thing is taking place on memory) (IO/$\overline{m}$ = 0 for every case)

IO/$\overline{m}$ = 0 → memory
= 1 → I/O

i) 40 (H) 70 (L) mov A, B
PC
for this, from above
Dont draw memory read & write

⊙ Here write 80 H
⊚ write 00 H
then
at opcode write = mov A, B

| for i) -
F + op (no op)
×

only fetch executed so draw only fetch diagram

ii) 90/00 / PC   mov A, m
Imp(s)

F + M$_R$

for it, draw fetch & memory

⊙ → 90 H
⊚ → 00 H
⊙ → <H>
⊚ → <L>
⊚ → <m>

| Draw
FC. MR

iii) mov M, A
Imp(s)

F + MW

| Delete M.R.
write/connect MW with fetch

⊙ → <H>
⊚ → <L>
⊚ → <m>

Draw F.C, M.W

iv) 8500 mvi A, 80
8501
F + MR + op
×

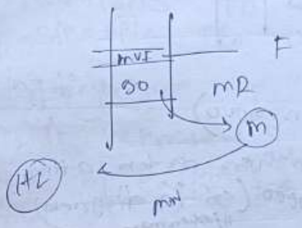
Draw FC, MR

⊙ → 85H
⊙ → 85H
⊙ → 00 H
⊙ → 01H   ⊚ → 80H

v) 8500 mvi m, 90
8501

[90 val store in memory (m) whose address is available in HL] for it

F + MR + OP
(MW)



Draw 3 → FC, MR, MW

⊙ → 85 H
⊚ → 00 H

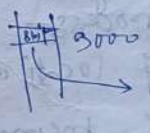
| ⊙ → 85 H
⊚ → 01 H
⊚ → 90 H

| ⊙ → <H>
⊚ → <L>
⊚ → 90

vi) 8000   LDA   3000
Imp(s)   address

8000 (LDA) F
8001 (00) MR
8002 (30) MR

F + MR + MR + OP
(MR)



Draw 3 → FC, MR, MR

⊙ → 80 H
⊚ → 00 H

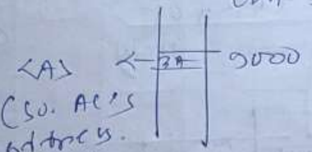
| ⊙ → 80 H
⊚ → 01 H
⊚ → 00 H

| ⊙ → 80
⊚ → 02
⊚ → 30

Draw another ~~Part~~ ~~mst~~ MR

LDA 9000

① → 90
② → 00
③ → Data

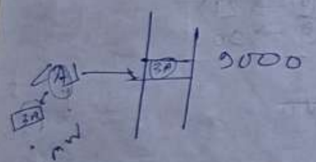


(So. AC's address is read from address 9000)

So. the higher order address is = 9000 (90 in diagram)
Higher order
lower order = 00

Prop (S)

If, STA 9000 F + MR + MR + MW (OP)



memory mapped I/O – It is a way of interfacing external devices with a microprocessor by treating them as memory locations. In this technique unique memory addresses are reserved for I/O operations, and the µP accesses those addresses to talk with the devices.

I/O mapped I/O – In a µP, I/O-mapped I/O is when I/O devices are mapped to a seperate address space from the memory address space. The µP uses special instructions to access the I/O devices using specific I/O address signals.

timing diagram – It is a graphical representation, it represents the execution time taken by each instruction in a graphical format.

Fetch operation – In this step, the µP fetches the ins. from the memory location pointed to by the Program counter (PC). The PC is incremented by one after the fetch operation.

11/10/23

String matching prog

LXI B, 9000	<BC> = 9000
LXI H, 9100	<HL> = 9100
MOV D, M → copying (memory address)	<D> ← <M>
LDAX B	<A> ← (<BC>)
CMP M	<A> compare with <M>
JNZ (no → yes)	Jump if no zero
INX B	<BC> + 01 → <BC>
INX H	<HL> + 01 → <HL>
DCR D	<D> - 01 → <D>
JNZ	Jump is no zero
MVI A, 11	<A> = 11
MVI A, 22	Jump to STA address; <A> = 22
STA 9500	<A> → (9500) H
RST 1	END

(As not effect have to do strings, place it, we have to mention it)

I/P	HL
0000 = 05	0100 = 05
0001 = AA	0101 = AA
0002 = BB	0102 = BB
0003 = CC	0103 = CC
0004 = DD	0104 = DD

O/P
2500 = 1122

It is a Procedure that is used to design a specific delay. There are 4 types ~~of~~ / Processes by which we can design a time delay (Time delay prog in final a time delay prog)

[OUT OF
If μP's freq = 3mHz
10×0.33 μsec
$= 3.3$ μSec

- within 3.3 a sec a 3.3 is submitted to a [ ] & another 3.3 is submitted to another [ ]
- Data 1

msg ⟶ time delay application

(→) Data2]||

- Time delay (Loop)
  - → NOP [1 NOP = 4×3.3]
  - → Register
  - → reg pair
  - → loop inside loop

NOP
- NOP
  NOP
  NOP
  NOP instruction does nothing but take 4T-States of Processer's time to execute
  NOP (ACC. to time delay)

Reg (Imp)
- MVI C, FF
  DCR C
  JNZ
  [this loop is executed FF time]

Here Producer used to designs a specific delay. A reg. is loaded with a no. depending on the time of FF [FF = 255] max value

- MVI C, FF | 7T → [F + mR required 4+3=7]
  DCR C | 4T
  JNZ 16 bit address | 10/7T → [when the condition is false then 7T; F mR mR 4+3+3=10]

It takes 255-1 = 254 time to execute

Time delay in the loop (TL)

Assume
f = 2mHz (If f isn't given then f = 3mHz)
$\therefore T = 0.5\,\mu s$
(time delay inside the loop)
$TL = (14 \times 0.5 \times 10^{-6} \times 255)$ sec $= 1.8$ ms

$\left[T = \frac{1}{f} = \frac{1}{2 \times 10^6} = 0.5\,\mu s\right]$

Loop adjusted → (When the cond will be fal. then only 7T is going to be used, so we have to decrease 3T from TL)
Calculate delay inside the loop $TLA_2 = (TL - 3 \times 0.5 \times 10^{-6})$ sec $\simeq 1.79$ ms $\simeq 1783.5\,\mu s$

total time = delay time inside the loop + delay time outside the loop

$T_{delay} = (T_O + T_{LA})$

$= (7 \times 0.5 \times 10^{-6} + T_{LA})$

$\simeq 1.8$ msec

↳ that type of ins. (Jmp) Jmp(S)

write the prog to generate the $T_L = 1$ ms

↳ In this type, $T_L = (14 \times 0.5 \times 10^{-6} \times n)$ sec

∴ 1 ms = something × x

⇒ x = 142.8 s

- If the value is more than 1.8ms then, reg pair

Reg pair - As mentioned, we can use reg pair to create large delay.

Special case (not in if statement)

```
      LXI H, 2384
loop: DCX B      10T
      MOV A, C   6T
      ORA B      4T
      JNZ        4T
                 10/7T
```

[If f > 2 then T = 20.5 ... If f = 4 T = 20.25 ...]

$T_L = (24T \times 9092)$ → this is loop time

(2384_{16} s = 109.1 ms)

$T_{LA} = (T_L - 3 \times 0.5 \times 10^{-6})$ sec (decimal)

≃ 109 ms

$T_{delay} = (T_{LA} + 16 \times 0.5 \times 10^{-6})$ sec

= 109 ms

2 3 8 4
3 2 1 0

$= 2 \times 16^3 + 3 \times 16^2 + 8 \times 16^1 + 4 \times 16^0 = 9092$

If it is the matter of second then loop inside the loop is used (i.e when more biggers it is given or when we want to increase the T_L).

loop inside the loop

```
Jmp(S) MVI B, 38    7T
loop → MVI C, FF    7T
loop → DCR C        4T
       JNZ          10/7T   Tdelay1
       DCR B        4T
       JNZ          10/7T
                    Tdelay2
```

[1.8 ms is using from 38's decimal value]

14 × T × 255 − 3T

$38 = 3 \times 16^1 + 8 \times 16^0 = 56$

- To create t.d. using a loop inside a loop, 2 count values are set up in 2 regs. the inner loop will execute reg-C times with every one count of reg-B value. T = 0.5

T_{delay} in loop 1

$T_{L_1} = (0.5 \times 14 \times 255 - 3 \times 0.5)$ μsec

= 1783.5 μsec

T_{delay} in loop 2

$T_{L2} = 56(T_{L_1} + 21 \times 0.5)$ μsec

$= 56(1783.5 \times 21 \times 0.5)$ μsec

$= 100.46 \times 10^3$ sec

$T_{L2A} = 10^3 \times 100.46 - 3 \times 10^{-5}$ μsec

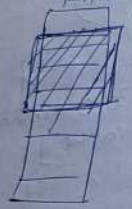
$\simeq 100.4 \times 10^3$ sec

total time delay = $(T_{L2A} + 7 \times 0.5 \times 10^{-3})$ msec

$= 100.4 \times 10^3$ s (1ms)

Stack and Subroutine

Stack = portion of the RAM. It can be used as temp data storage when µp execute the ins. To overcome the limitation of registers it is used.



this are done with the help of 2 ins.

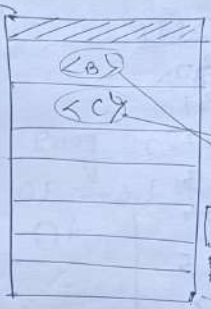
① PUSH regpair
② POP regpair (16 bit data)

A + F ⟶ PSW (Prog. status write)
(Accum ↓ data) (flag 8 bit reg)

Which portion of ~~stack~~ RAM is initialized as the stack is done by SP

LXI SP, 16 bit address.

Between this some regs are used & others are free regs where we can store data.



8FFF + 1 = 0000

[If PC is available for then only prog is possible otherwise not]

(when POP executes then SP is incremented by 2)

~~Prog~~ write a prog to set all flag, for this ~~type~~ ins — S Z AC P CY

1 1 1 1 1

i) MVI C, FF
PUSH B
ii) POP PSW

∴ C = 11111111 = FF

[B contains garbage value, PSW = A + F, B content transfers to the Accumulator & C content to the F, As C = FF, so all flags of the flag store + will be 1]

without affecting ac—
mov B, A
MVI C, FF
PUSH B
POP PSW

Subroutine — It is basically a fixed prog that is ~~used~~ (A fix Prog) which is repeatedly executed

HEX	ASCII
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39
A	41
B	42
C	43
D	44
E	45
F	46

(39) → 30 41

3A 3B 3C 3D 3E 3F 40

the diff. between 39 & 41 = 7

multiple time we are executing a single prog (subroutine). whenever it is needed execute it and back to the memory.

(Imp) CALL 16 bit add → subroutine execution

(Imp) RET → Back to the main prog. the end of subroutine

when the no. is < A add 30 to get the ASCII val (then cy is set)

- - - - - ≥ A , add 37

ASCII Prog 2nd

```
9000 ANI 0F  → [this ins. is used to clear the upper 4 bits of the accumulator] for JC
     CPI 0A
     JC
     ADI 30
     RET
```

```
CPI 0A
JC
ADI 07
ADI 30
RET
```

this is 2 times needed when convert the H into ASCII

1st Prog

```
8000 LDA  8500
8003 MOV  B,A
8004 CALL 9000
8007 STA  8700
800A MOV  A,B
800B RLC
800C RLC
800D RLC
800E RLC
800F CALL 9000
8012 STA  8701
8015 RST  1
```

Chang add (Stored - 8200, will be stored - 8300)

```
LDA  8200
MOV  B,A
CALL 9000
STA  8300
MOV  A,B
RLC
RLC
RLC
RLC
CALL 9000
STA  8301
RST  1
```

3	A
03	0A

0011 1010
0000 1111
0000 1010
0 A

[8004 CALL, 8005 00, 8006 90] ↓ PC value [8007 - PC]

[RLC = Rotate AC left without carry]

[RRC = Rotate AC right without carry]

[8007 is stored to stack & when RET is called it will load the value of stack so we will return there from where we left]

Imp from here – CALL, RET, PUSH, POP

- **Subroutine** – A subroutine is a sequence of program instructions that performs a specific task, packaged as a unit.
- **CALL** – It interrupts the flow of a program by passing control to an internal or external subroutine.

Importance – 1) Allow progs to be modular & organized
2) make it easier to write & maintain complex code
3) Save memory space

- **RET** – The ret instruction transfers control to the return address, located on the stack.

Importance – 1) Loads the PC from top
2) Increments the SP
3) Jumps to the location specified by the PC
4) Pops the return address off the stack
5) Continues execution at that address.

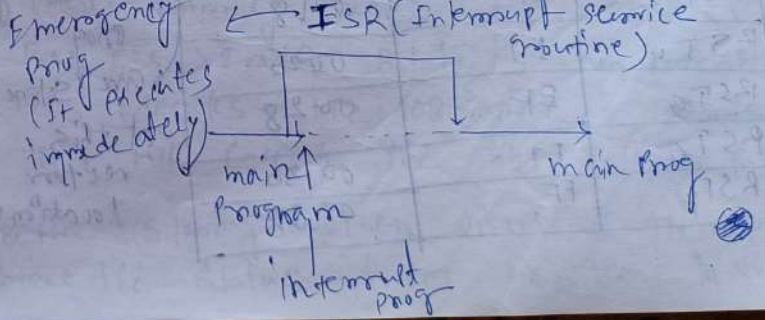
Imp of PUSH & POP

PUSH – 1) Decrements the pointer & copies the data to stack.
2) Creates a new node with the new element and sets the next pointer of the current top node to the new node.
3) can cause a stack overflow error if you try to push an already full stack.

POP – 1) copies data from the stack & then increments the pointer
2) Decrements the index of the top element & returns the value stored at the index.

CW — **Interrupt of 8085 µP** — 1/11/23

Interrupt the operations / suspend all of the operations main



Classification of interrupt

1) Hardware & Software interrupt
2) ~~vector~~ vector & non vector "
3) maskable & non maskable interrupt

Interrupt ~~field~~ of µP - pin

After every machine cycle µP checks for interrupts.

TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
} Emergency signal

Ex of hardware interrupt

Hardware & software interrupt

DI, EI
↓ Disable interrupt
↓ Enable interrupt

These are executed with the help of instructions

Ex - RST0, T1, T2, ... T7

These are Enable & disable by DI, EI.

as well as instruction

S.I (Soft. Int.)

	opcode	vector location
RST_0	C7	0000
RST_1	CF	0008
RST_2	D7	0010
RST_3	DF	0018
RST_4	E7	0020
RST_5	EF	0028
RST_6	F7	0030
RST_7	FF	0038

Intr. Prog into service Prog

Emergency Prog are stored at this vector locations

2) vector & non vector Intr.

	Call loc.
TRAP	0024
RST 7.5	003C
RST 6.5	0034
RST 5.5	002C

7.5 × 8 = 60 = 3C

6.5 × 8 = 52 = 34

5.5 × 8 = 2C

INTR → vector (interrupt loc) (has specific)

It itself isn't used, it is always used with software call locs.

non vector interrupt (those who haven't specific call loc)

3) maskable & non maskable

Those Intr. that are forcefully disable by instructions.

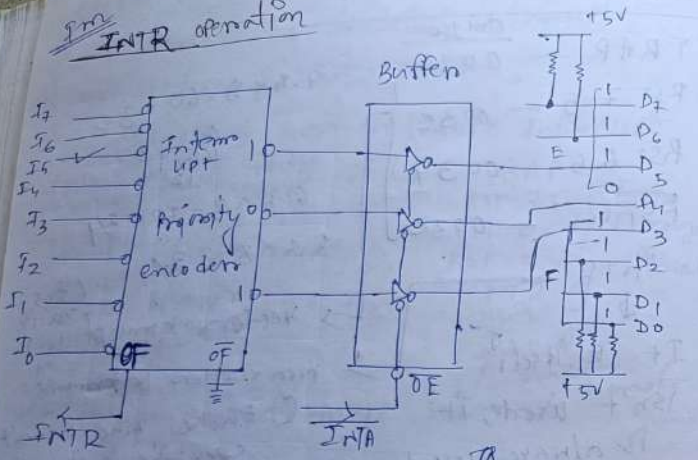
Ex - RST 7.5, 6.5, 5.5, INTR

User can't forcefully disable it

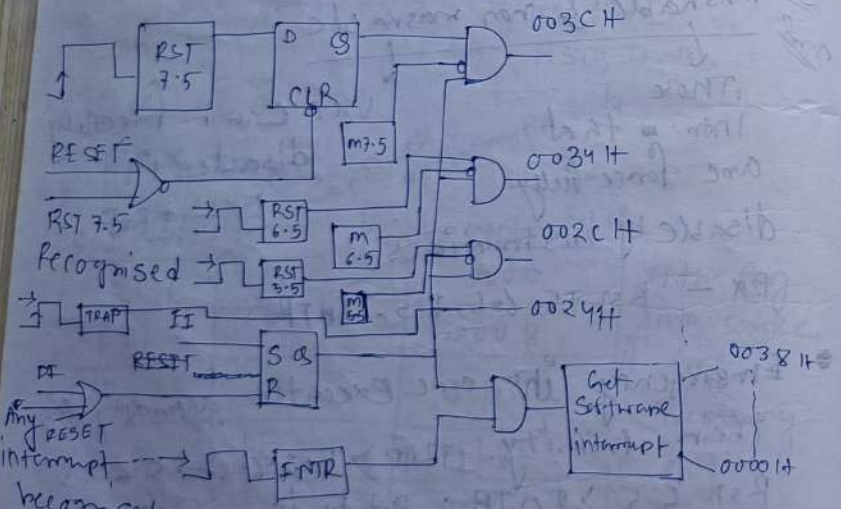
Ex - TRAP

- Emergency Intr are executed according to their priority. TRAP > RST 7.5 > RST 6.5 > RST 5.5 > INTR. But it is non Preemptive. Ist executes the operation it starts then will execute the high priority ins.
- Before starting another operation the µP will store the status of prv operation in the stack.

INTR operation



Interrupt ckt (for this ckt the ISR = 0028)

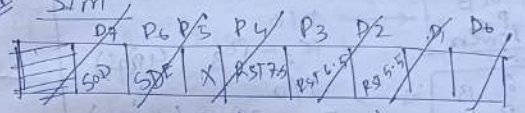


[→ level triggered with edge
→ level triggered
--→ contain type of level triggered]

Advantage of INTR as non maskable interrupt? i) low response time
ii) It is often used when the response time is critical.
iii) It is used in emergency
iv) It can handle higher priority task such as watchdog timers.

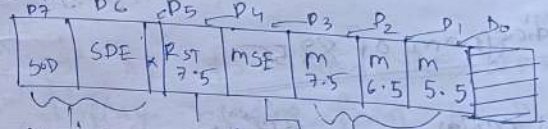
SIM

SIM



(Q diagram isn't needed if it isn't mentioned in the Qn)

SIM → It is an instruction



Serial data out

Reset RST whenever it is 7.5

maskable bit no. RST 5.5 to RST 7.5
if is 0 then enable
if is 1 then mask

maskable set
enable of it is 1

The Prog to enable all interrupts

```
EI
MVI A,08
SIM
```

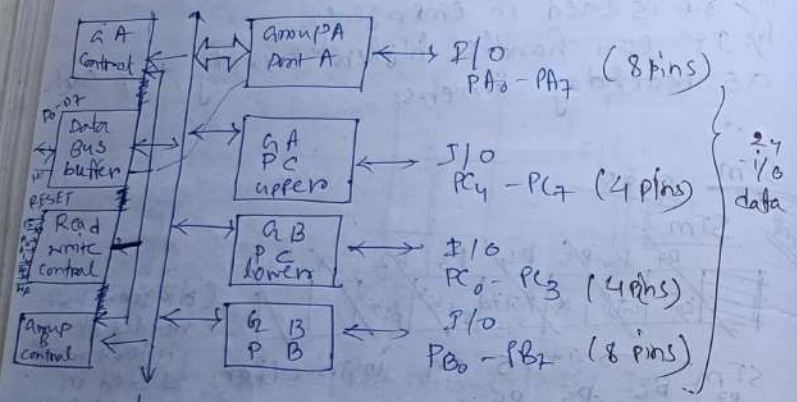
[Here disable means →]

Enable 7.5 & disable 5.5 & 6.5

```
EI
MVI A,0B
SIM
```

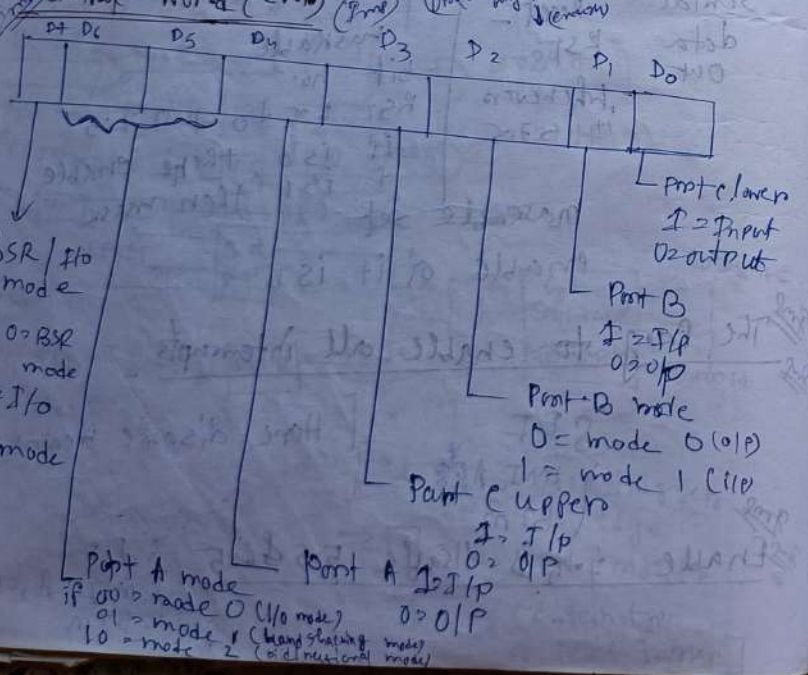
8/11/23

8255 Programmable Peripheral Interface (PPI)



Block diagram of 8255

Control Word (CWR)



mode 0 → I/O mode

mode 1 → Handshaking mode

mode 2 → Bidirectional mode (only Port A is there)

A_1	A_0	
0	0	PA
0	1	PB
1	0	PC
1	1	CWR (Control word register)

2 address bus as 4 i/ps

- BSR_2 Bit set reset
- If,

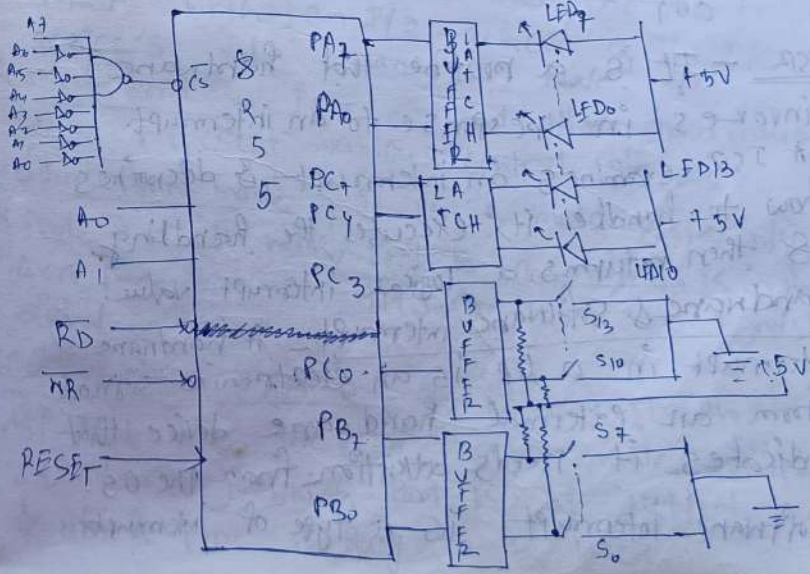
Imp

PA = I/P
PB = O/P
PCL = O/P
PCu = I/P

→ 1 0 0 1 1 0 0 0 → 98

Eg:-

Write A.P to read the DIP switches (Dual in-line package) & display reading from Port B & Port A & from Port C lower at Port C upper.



	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
80	1	0	0	0	0	0	0	0	0 → PA
81	1	0	0	0	0	0	1	0	1 → PB
82	1	0	0	0	0	0	1	1	0 → PC
83	1	0	0	0	0	0			1 → CWR

	7	6	5	4	3	2	1	0	
for this CW =	1	0	0	0	0	0	1	1	→ 83

```
MVI A, 83   → value
OUT 83      → address of control
              bus reg
IN 81
OUT 80
IN 82
~~OUT~~
ANI 0F
RRC
RRC
RRC
RRC
OUT
```

	PC7	-	C	5	4	3	2	1	0
	x		x	x	x	x	x	x	x
	0		0	0	0	1	1	1	1
	0		0	0	0	x	x	x	x

ISR – It is a routine that hardware invokes in response to an interrupt. A ISR examines an interrupt & determines how to handle it, executes the handling & then returns a logical interrupt value.

Hardware & Software interrupt – A hardware interrupt in a µp is an electronic signal from an external hardware device that indicates it needs attention from the OS. A software interrupt is a type of interrupt that occurs when a µp executes certain instructions or encounters certain conditions.

vector & non vector interrupt – A vectored interrupt is a processing technique in computer sci where the interrupting device directs the processor to the appropriate interrupt service routine (ISR).

Non vectored interrupt in a µp is an interrupt where the vector address is not predefined.

maskable & non maskable interrupt – A maskable interrupt is an interrupt that can be ignored or disabled by the instruction of the CPU. A non maskable interrupt is a hardware interrupt that can't be ignored by standard interrupt - masking techniques in the system.

Handshaking mode – It is an I/O control method that synchronizes I/O devices with the µp. It is used to control the µp to operate with an I/O device at the I/O device's data transfer rate.

Control Word – It is a word whose individual bits represent various control signals

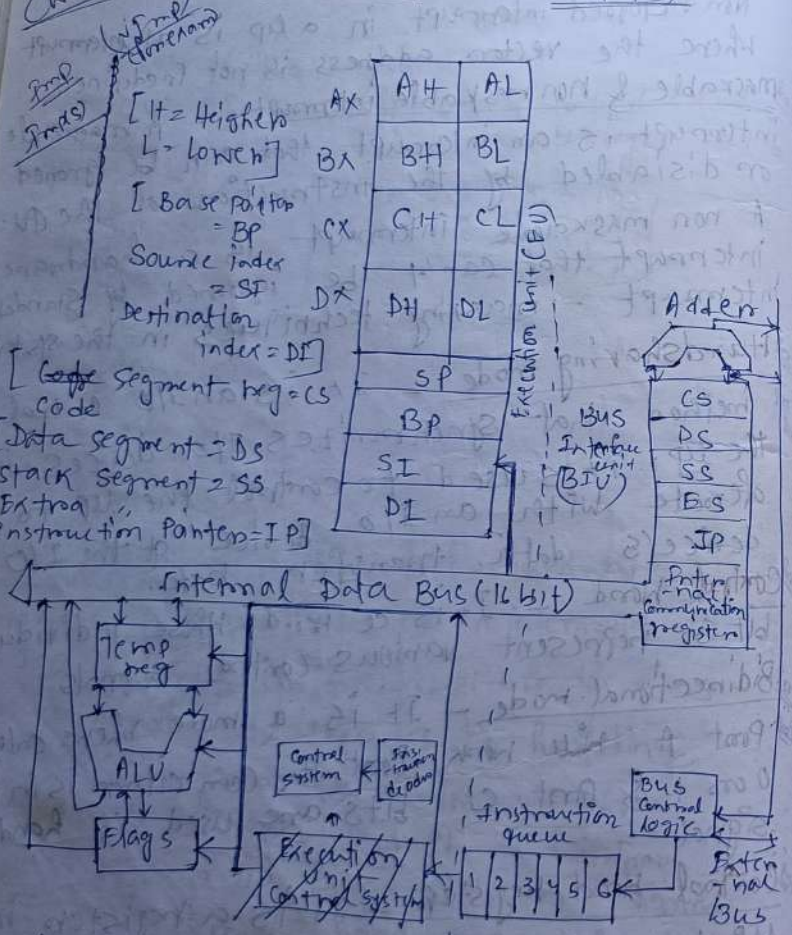
Bidirectional mode – It is a mode where only Port A will work, Port B can either is in mode 0 or 1 & Port C bits are used as handshake signal.

Control Word register – It is a register in a µp that tells the 8255 Programmable peripheral extender (PE) which port is of output mode & which port is going to be used.

This mode is used to set or reset the bits of Port C only, and selected when the most significant bit (D7) in the control register is 0. This is also called I/O mode.

8086 µP

9/11/21



Architecture of 8086 µP

(Parallel processing or pipelining architecture)

Features of 8086 µP

1) It is a 16 bit µp.
2) 8086 µp has a 20 bit address bus that can access upto 1 MB memory. Address range from (00000 H – FFFFF H)
3) It can support upto 64K I/O modes.
4) It provides 14, 16-bit registers.
5) It has multiplexed address add. & data bus (AD_0 – A_{19}) & (AD_0 – AD_{15})
6) It requires single fetch clock with 33% degree cycle with Provide internal to current.
7) It is designed to operate in 2 mode (min & max mode).
8) It can be feathches upto 6 instruction bytes from the memory & queues, then in order to speedup 8086 instruction execution.
9) A 40 pin dual-in line package (DIP) requires 5V.
10) memory is byte addressable, every byte has a seperate address.
11) It employes parallel Processor (its architecture also called parallel processing or pipelining archi.)
12) It has 2 main functional unit EU & BIU.
13) It has BHE (Bus high enable) Pin.

8086 µp architecture

It has 2 functional units—

EU (Execution unit) – Execution unit gives instructions to BIU, stating from where to fetch the data & then decode & execute those instructions. It's function is to control operations on data using the instruction decoder & ALU.

BIU (Bus Interface unit) – BIU takes care of all datas & addresses transfers on the buses for the EU, like sending addresses, fetching instructions from memory etc. It is connected with internal bus.

General purpose registers – there are 8 general purpose registers, ie., AH, AL, BH, BL, CH, CL, DH & DL. These registers can be used individually to store 8 bit data & can be used in pairs to store 16 bit data.

The valid reg pairs—

AH – AL → referred as AX
BH – BL → BX
CH – CL → CX
DH – DL → DX

AX – It is also known as accumulator reg. It is used to store operands for arithmetic operations.

BX – It is used as a base register. It is used to store the starting address of the memory area within the data segment.

CX – It is referred to as counter. It is used in loop instructions to store the loop counter.

DX – This register is used to hold I/O port address for I/O instruction.

BP – It stands for base pointer. It is a 16 bit register that holds the offset address of any location in the stack segment. It is used to access random locations of the stack.

SI – It stands for source index. It is a 16 bit register that stores the offset address of the source in string manipulation instructions.

DI – It stands for destination index. It is a 16 bit register that stores the offset address of the destination in string manipulation instruction.

Segment registers – BIU has 4 segment buses, i.e, CS, DS, SS & ES. It holds the addresses of instructions & data in memory, that are used by the processor to access memory locations.

CS – It stands for code segment. It is used for addressing a memory location in the code segment of the memory, where the executable prog. is stored.

DS - It stands for data segment. It consists of data used by a program and is accessed in the data segment by an offset address or the content of other register that holds the offset address."

SS - It stands for stack segment. It handles memory to store data & addressings during execution.

ES - It stands for extra segment. Es is additional data segment, which is used by the string to hold the extra destination data.

IP - It stands for instruction pointer. It is a 16-bit register used to hold the address of the next instruction to be executed.

Internal communication register - 8086 µp has 4 groups of the user accessible internal registers. They are the instruction pointer, 4 data reg., 4 pointer & index reg., 4 segment reg.

Adder - It is a digital ckt that performs addition of numbers.

Control system - It controls operations on data using the logic unit (ALU). It uses flags, general purpose reg., operands etc.

Instruction queue - It is 6 bytes buffer that operates on FIFO basis, that stores instructions from memory before they are executed. They also receives the instruction codes from memory.

~~Bus control logic - the control bus~~

External bus - It is a type of data bus that enables external devices & components to connect with a computer.

CW

8086 reg organisation 23/11/23

AX → AH 16 → AH & AL
(8)

BX → BH BL

CX CH CL

DX DH DL

Purpose

Part of AX - i) It is a 16-bit accumulator with lower 8-bit of AX designated as AL & higher 8 bit as AH.

ii) AL can be used as a ac for 8-bit processor.

iii) X is used for of involving i/o & most arithmetic, during execution of 16 bit multiplied & divide ins. it contain the 1 word operand & the result is stored in the ac (accumulators).

Func of BX — i) It is known as base reg.
ii) This is only general purpose reg whose content can be used for addressing 8086 memory.
iii) ~~If~~ It holds the offset address of a location of the memory.
iv) All memory reference utilizing this reg content for addressing usage the DSR (default segment register) resistance.
v) It can also be combined with BP & SI as the base reg. for special addressing.

1 MB → add. of 20 bit

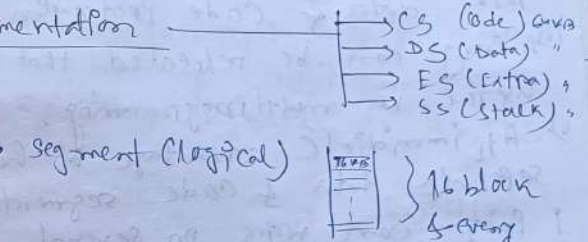
16 segment
64 KB of each

Code, data, Stack, extra segment (4 total segments)
↓
this are used for security purpose.

Func(CX) — i) It is known as default counter reg incase of storing or loop ins. executed.
ii) It may contain a val. to ~~that~~ control the no. of times a loop is repeated when the loop ins. is ~~executed~~ used. or a value to shift bits left to right when shift/rotate ins. is present.

Func(DX) — i) It is called Data reg used to hold 16 bit result.
ii) It used to hold the i/o device address by executing the in & out ins.
iii) It is used to hold a part of the result during the mul op & a part of the result during the division of. dividend

memory segmentation



16 × 216
= 24 × 216
= 2(20) → 1 MB

initial add.
CS → 00000 — OFFFF (only stores ins.)
DS → 10000 — 1FFFF (" — data)
ES → 20000 — 2FFFF (any of but except the code) byte
SS → 30000 — 3FFFF
F0000 — FFFFF ← final add.

Advantage of memory segmentation

i) It allows, the memory addressing capacity to be 1 MB, even though the add. associated with individual ins. is only 16 bit. This is the advantage of ms.

ii) It allows ins. code, data stack & portion of prog to be more than 64 KB long by using more than 1 code, data stack segment & extra segment.

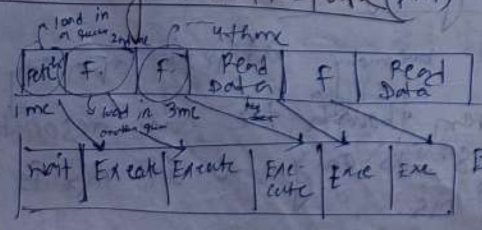
iii) It allows the basic placing of code, data & stack portion of the same prog. in diff. parts of the memory for data & code protection.

iv) A prog can be relocated that is very useful in multiprogramming.

v) An immidiate advantage of having seperate data & code segment is that 1 prog can work on several diff. sets of data.

vi) The greatest adv. of m. segmentation is that progs that have diff. logical add. only, can be loaded and run anywhere in memory. [This is because the logical add. always rent from 00000 - 0FFFF H independ of code segment base.]"

pipelining architecture (Imp) [mc = machine cycle]



Bus interface unit (BIU)

Execution unit (EU)

(6 byte queue is there)

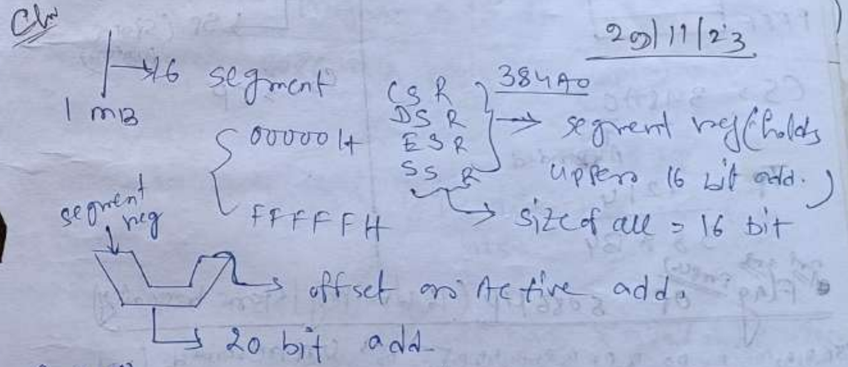
Parallel Enter = Parallel execute

8086 µp - It is a 16-bit CPU introduced by intel in 1978, widely used in early personal computers. It features a segmented memory architecture & it's a fundamental component in x86 architecture development.

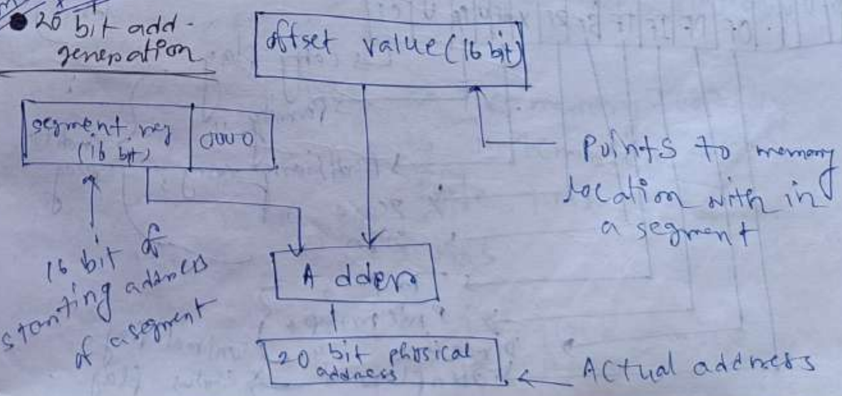
Memory segmentation - It is the process of dividing memory into smaller parts. The 8086 µP has a segmented memory architecture. It divides memory into 4 segments - CS, SS, DS, ES.

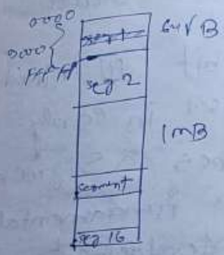
Cl

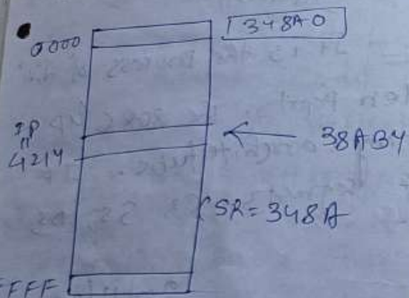
29/11/23



Imp (Imp) 20 bit add. generation







DS → BX
DS → SI
DS → DI

ES → DI

SS → BP (Base Ptr)
SS → SP (Stack pointer)

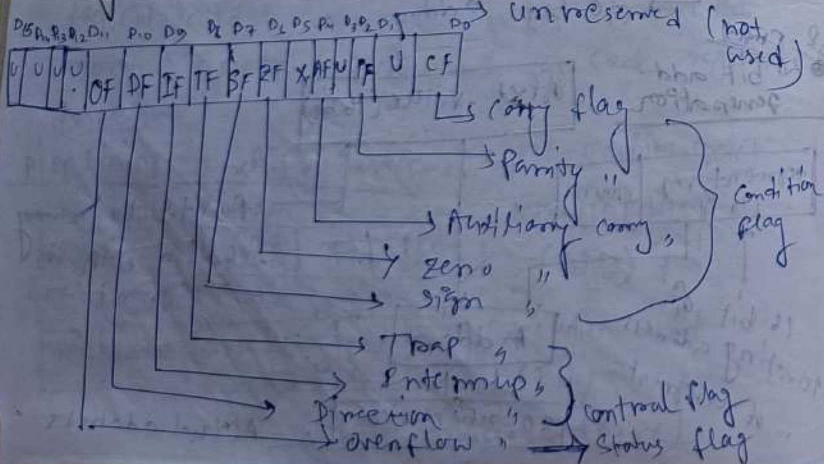
CS → IP

CS = 348A0 (0 Appended)

IP = 4214

38AB4

- Flag (Imp(s)) of 8086 MP (16 bit register basically)



→ active flags here

If 0, Counting Starting from 0 (lower) to 1 (higher)

If 1 . . . . . . 1 (higher) to 0 (lower) — Direction flag for IC

- Diff betw 8086 with 8088
- 8088 features
  → read

Exchange DE with HL by Push & POP —
PUSH D
PUSH H
POP D
POP H

- Von Neumann & Harvard architecture

- 8085 architecture →

8086 ——→ (no need to read that much) (Can come in exam)

- Block transfer, fibonacci, String matching, odd/even seperation — Programmes to do (Imp)

Diff. between 8086 with 8088 µp Imp(S)

8086 µP	8088 µP
1) The data bus is of 16 bit	1) The data bus is of 8 bits.
2) It has 3 available clock speeds (5mHz, 8mHz & 10 mHz).	2) It has 2 available clock speeds (5mHz, 8mHz)
3) The memory capacity is 512KB. (or) memory space is organised as 2,512KB (2×512KB = 1MB) banks.	3) The memory capacity is implemented as a single 1M × 8 memory banks. (or) It is implemented as a single 1mB bank.

4) It can read/write 8/16 bit data at a time.

5) Instruction queue is 16 bit long.

6) It has BHE (Bus high enable) pin.

7) It draws max supply current of 360mA.

4) It can only do so for 8 bit data.

5) Instruction queue is 4 bit long.

6) The pin is replaced by status output (SSO), since it can read or write only 8 bit data.

7) It draws max supply current of 340 mA.

Features of 8088 MP

Imp(s)

It is a 16 bit MP that has used in IBM PC's & PC clones. The features of 8088 MP-

i) clock speed = 5-10 MHz
ii) Registers = 16-bit
iii) Address bus = 20-bit
iv) External data bus = 16-bit
v) memory = 1 MB
vi) transistors = 29,000
vii) Data types = 8, 16, 32-bit
viii) Address space = very large
ix) memory management unit = integrated
x) object code: compatible with all 8086 family MP
xi) virtual 8086 mode: Allows running of 8086 software in a protected & based system.

- SIM - It is an instruction of 8085 MP. It stands for set interrupt mask. It is used for control interrupts. this instruction allows the MP to selectively enable or disable interrupts by setting the appropriate bits in the IMR (Interrupt mask register). It -
  i) set masks for RST 7.5, RST 6.5, & RST 5.5
  ii) Read the contents of accumulator reg.
  iii) Based on the contents of the ac reg. it enable or disable interrupts.

- RIM - It is an instruction of 8085 MP, that stands for read interrupt mask. It is used for control interrupts. It -
  i) Is used to read the status of interrupts 7.5, 6.5 & 5.5.
  ii) Read the condition of the serial input data (SID) bit.
  iii) mask or unmask RST 7.5, RST 6.5 & RST 5.5.
  iv) Read contents of interrupt mask register (IMR)

20 bit address of 8086 MP - It has 20 bit address line. So the max value of address that can be addressed by 8086 is $2^{20} = 1$ MB. So 8086 can address the locations ranging between 00000 H - FFFFF H. This 1MB memory is divided into 16 logical segments, each into a memory of 64KB.

Note that (Ans from M1922)

Flag of 8086 µP - Flag reg is a 16 bit reg, but there are only 9 flags available in 8086 µP. The rest 7 bits are hence left idle. There are 2 types of flag reg or flags in 8086 µP.

1) condition flags - This are stored reset after many arithmetic or logical operation is performed on an 8 bit or 16-bit no. This category consists of following 6 flags. i) CF, ii) PF, iii) AF, iv) ZF, v) SF,

vi) overflow flag - It will be set if the reg get overflowed with data after an arithmetic or logical operation. This happens in cases when the carry is getting in, in MSB, but there is no space in the reg. to store the carried out bit.

2) Control flags - the control flags are used to navigate the µP for certain operations. There are 3 types of control flags

i) Trap flag (TF) - This flag is used if we need single-step debugging in your code. If the TF is set, then the execution will be done step by step. Otherwise, the free-running operation will be done.

ii) Interrupt flag (IF) - the flag is used to enable the interrupt. The µP is capable of handling interrupts only if this flag is in the set mode.

iii) Direction flag (DF) - this flag is used for string operations. If this flag is set, the string will be read from higher-order bits to lower order bits & vice versa.

2023

µP (internal of ECE)

1) Explain the instructions with suitable example. (Data transfer & Arithmetic & logic ins)

a) LDAX: Load accumulator indirect

general format: LDAX regpair

Eg - LDAX B
LDAX D etc

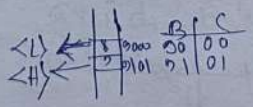
Working: It loads the accumulator content of regpair to the accumulator in indirect way.

b) LHLD: Load HL reg pair direct (" ins)

general format: LHLD 16 bit memory address

Eg - LHLD 9000

Working: It will load the content of 16 bit address in lower order address [i.e <L>] & then automatically it will load the content of next 16 bit address in higher order address (i.e <H>)



c) XCHG: Exchange the HL content with DE register pair. (" ins)

general format: XCHG (H, L) (D, E)

Eg –

D E
(AB CD)
(12 34)
H L

9500 } CD → E
9501 } AB → D
9600 } 34
9601 } 12

Working: It will exchange the content of HL with the DE reg pair content.

For the upper case,

9500 E ← 34
9501 D ← 12

Exchange
L → 32 → E
H → 12 → D

9600 (34) load → E
9601 12 → D

From this Prog is –

```
LHLD 9500
XCHG
LHLD 9600
RST 1
LHLD 9500
MOV E,L
MOV D,H
LHLD 9600
RST 1
```

d) DAD: Double addition (A & L ins.) (Arith)

general format: DAD regpair

Eg – DAD B <HL> + <BC> → <HL>

Working: It will add the regpair content to the content of <HL> & store the value in <HL>. As 2 addition is performed here it is called double addition.

e) STA: Store accumulator direct (Data transfer ins.)

general format: STA 16 bit address

Eg – STA 9000 <A> ← [32] 9000

Working: It stores the address of the memory. The storing address is given directly to the instruction.

Q) What is the need to demultiplexing the address/data bus? How it is done?

Ans) The address/data bus is demultiplexed to avoid mixing up of address / data bits.

~~Delivering the received segments at the receiver side to the correct app layer~~

It is needed because it needed to fit in a 40-pin package. Demultiplexing is used to collect the digital infos from

a 16 bit reg, available in ... hence left flag reg. on ...
store reset logical operation n 16-bit no. 6 flags.
... if the ... on arithmetic s in cases MSB, left store the ... re used ... ations.
... of we . If the done ...

Java notes
... flex no. prog (in phn) ... also write ... nic class

multiple Sources & integrate them into a single Source as the output.

How it is done -
ALE signal is used to demultiplex the lower order address bus (AD0 - AD7).
diagram

3) Draw & explain the flag registers of 8085

4) Explain the function of HOLD, S_0, S_1, RESET IN & READY signal of 8085 μP.

6) Draw & explain the timing diagram for the instruction INR m. (INR m ↓ Increment content of m content by 1)

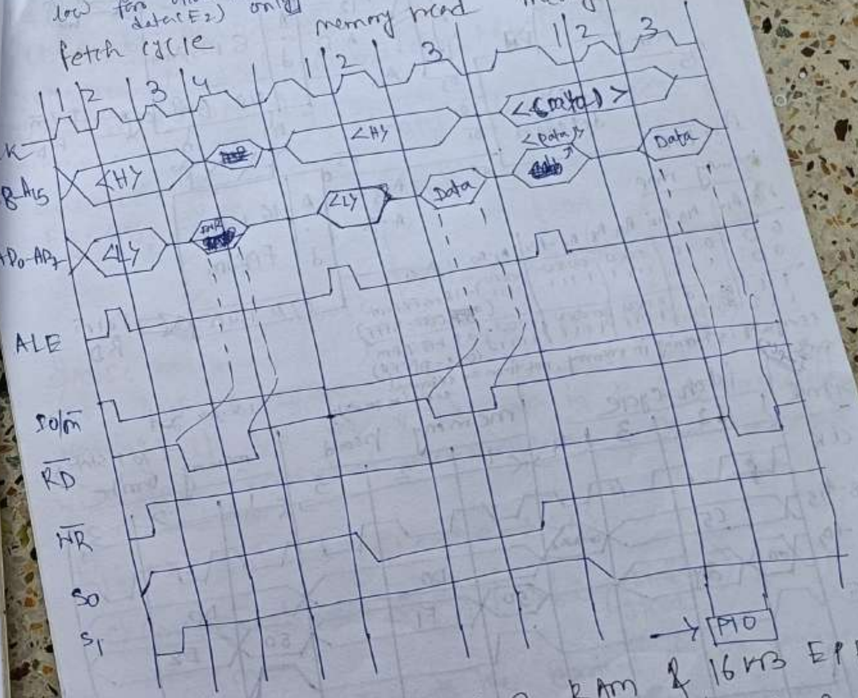
It is a 1 byte (8 bit) instruction, here opcode - INR, operand - m

So, here we need to perform F (fetch) + mR (memory read) + mW (memory write) timing diagram.

Explanation (not compulsory):
[INR m = opcode fetch (INR m) → memory read (go to the add. of m & read the data in it) [E1H] → memory write (while incrementing val to D050H here)
Do it's mem (E1H → E2H)
Let Add C050H contains INR m (ins. INR m)
INR is pointing this mem. add. D050H ← m, it contains val. E1H → INR m → E2H
Let's say opcode of INR m = 34H (opcode is also a data, E1 & one data → so while reading them) the RD will be active low

as E2 is going to be write not read. same condition will goes for WR, it is also active low for opcode write, data write, here we need to write the data (E2) only

fetch cycle | memory read | memory write



5) Interface one 8KB RAM & 16KB EPROM with 8085 in absolute decoding technique.

$8\ KB = 2^3 \times 2^{10} \times 8$ bits $= 2^{13} \times 8$ bits

$16\ KB = 2^4 \times 2^{10} \times 8$ bits $= 2^{14} \times 8$ bits

reg. is a 16 bit reg.
flags available in
bits are hence left
types of flag reg. on

this are store result
ic or logical operation
bit or 16-bit no.
following 6 flags.
SF,
be set if the
ta after an arithmetic
happens in cases
in, in MSB, left
to store the

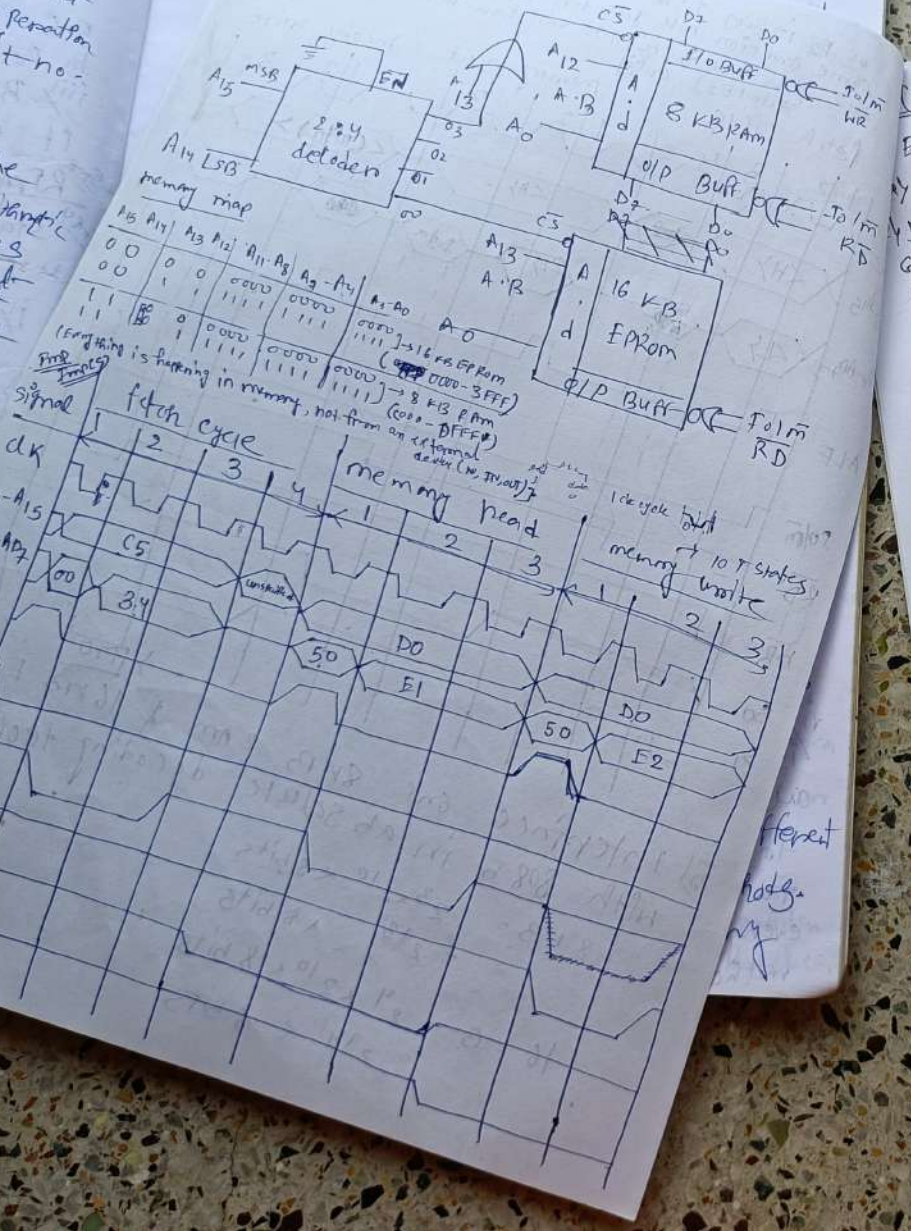
ags are used
operations.
flags
sed of we
code. If the
be done ALE
uming the
enable
handel
the
use
ref
S0
S1

Java notes

For complex no. prog (in phn)

can also write

public class



(Internal (2nd slot))

Explain the instruction with suitable example.

a) LDA X, b) LHLD - Done

DAA: Decimal adjust accumulator (A & L (logical) ins)

General format: DAA reg/m

Working: It converts a binary value into BCD automatically.

Eg: (12)$_2$ = 1100
↓ DAA
(12) BCD = 0001 0010

(18)$_2$ = 10010
↓ DAA
(18)BCD = 11000

ADI: Add immidiate 8-bit data (A & L (Arithmetic))

GF: ADI 8-bit data

Working: It add the 8-bit data in the ins immidiately with the content of accumulator & store the result in accumulator.

Eg: ADI 90 → <A> + 90 → <A>

c) CMC: Complement carry (A & L (Arithmetic) (logical))

GF: CMC

Working: It complements the content of the carry. [1's complement]

Eg: <CY> ⇐ 0101
↓ CMC
<CY> ⇐ 1010

2) What is the need to demulti pleating the address/data bus? How it is done? - Done

3) Explain the flag structure of 8085 µp with suitable example. - Done

flags available in
bits are hence left
s of flag reg. on

are stored reset
on logical operation
t on 16-bit no.
storing 6 flags.

set if the
after an arithmetic
ons in cases
in MSB, left
store the

are used
erations.
)s-
of w
. If
d

Java notes

For complex no. prog (in phn)

we can also write
at public class

4) Explain the func of IO/M, HOLD, READY, SIM, PC register of 8085 µp. — Done

5) Interface one 16KB RAM & 8KB EPROM with 8085 in absolute decoding technique. — Done

Just change in memory map
for 16 KB ~~EPR~~ RAM → (C000 - FFFF)H
for 8 KB EPROM → (0000 - 1FFF)H
[Exchange the Prev positions of chip & also RAM will be of 214 × 8 bits & EPROM will be of 213 × 8 bits & there will be a A13 with it]

6) Draw & explain the timing diagram for the instruction INR m. — Done

8p (internal (1st slot))

1) Explain the instructions with suitable example.

a) LDA: load accumulator direct (data transfer instructions)
OF: LDA 16 bit address
working: It loads the content available in the 16-bit address of the instruction to the accumulator.
Eg: LDA 9000H <A> ← 9000

b) SHLD: Store HL register pair direct (data transfer instructions)
OF: SHLD 16-bit memory address
working: Same as LHLD, but it will store the <L> content in the 16 bit address available in the ~~mem~~ instructions & then automatically store the content of <H> in the next 16 bit add. -ress.
Eg: SHLD 9100

	B	C
<L> → D1 9100	91	00
<H> → D2 9101	91	01 (next 16-bit address)

c) XCHG - Done

d) ANI: Logical AND 8-bit data with accumulator
(A&L (logical instruction))
OF: ANI 8-bit data
working: It will do logical AND of 8-bit data available in the given instruction with accumulator & store the result in the accumulator.
Eg: ANI 00H ⇒ <A> · 00 → to <A> (res=00)

CMA: Complement accumulator (ASL (logical) instruction)

AF: CMA

Warning: It complements the content of the accum[ulator]. (1's complement)

Eg: Let's say, <A> = 0101

↓ CMA

<A> ← 1010

2) Explain in detail the function of ALE signal & where it is used?

Func of ALE – It stands for address latch enable. It goes high during first T state of a machine cycle & enables the lower order 8-bits of address, if it's value is 1, otherwise data bus is activated. It is used to demultiplex the lower order address bus (AD_0 – AD_7). Demultiplexing is used to collect the digital infos from multiple sources & integrate them into a single executable source as the output.

It is used for demultiplexing the lower order address bus.

Demultiplexing ckt diagram – Done

3) Explain the flag structure of 8086 MP with suitable example. – Done

4) Explain the function of ALE, IO/M̄, RESET OUT, READY signal & PC register of 8085 MP. – Done

5) Interface one 8KB RAM & 8KB EPROM with 8085 in absolute decoding technique.

Everything will be same just for EPROM,

8KB EPROM = $2^3 \times 2^{10} \times 8$ bits = $2^{13} \times 8$ bits

∴ the 8KB EPROM's AD will be from = A_0 – A_{12}

& there will be a $\bar{A}_{13}$ & rest of things will be like prev diagram.

And, memory map.

for, 8KB RAM = (C000 – DFFF)H

for 8KB EPROM = (0000 – 1FFF) H

Rest – Done

6) Draw & explain the timing diagram for the instruction DCR M.

DCR M (Decrease the content ~~of the~~ available in M (memory address) by 01)

DCR M → 1 byte instruction

Here we need to perform

DCR M = Opcode fetch (F) → memory Read (MR) → memory write (MW)

Same as INR M [eg of INR M, only change E1H → E0H]

timing diagram –

Complex_NO in exchange of
Complex_NO as if nothing is mentioned