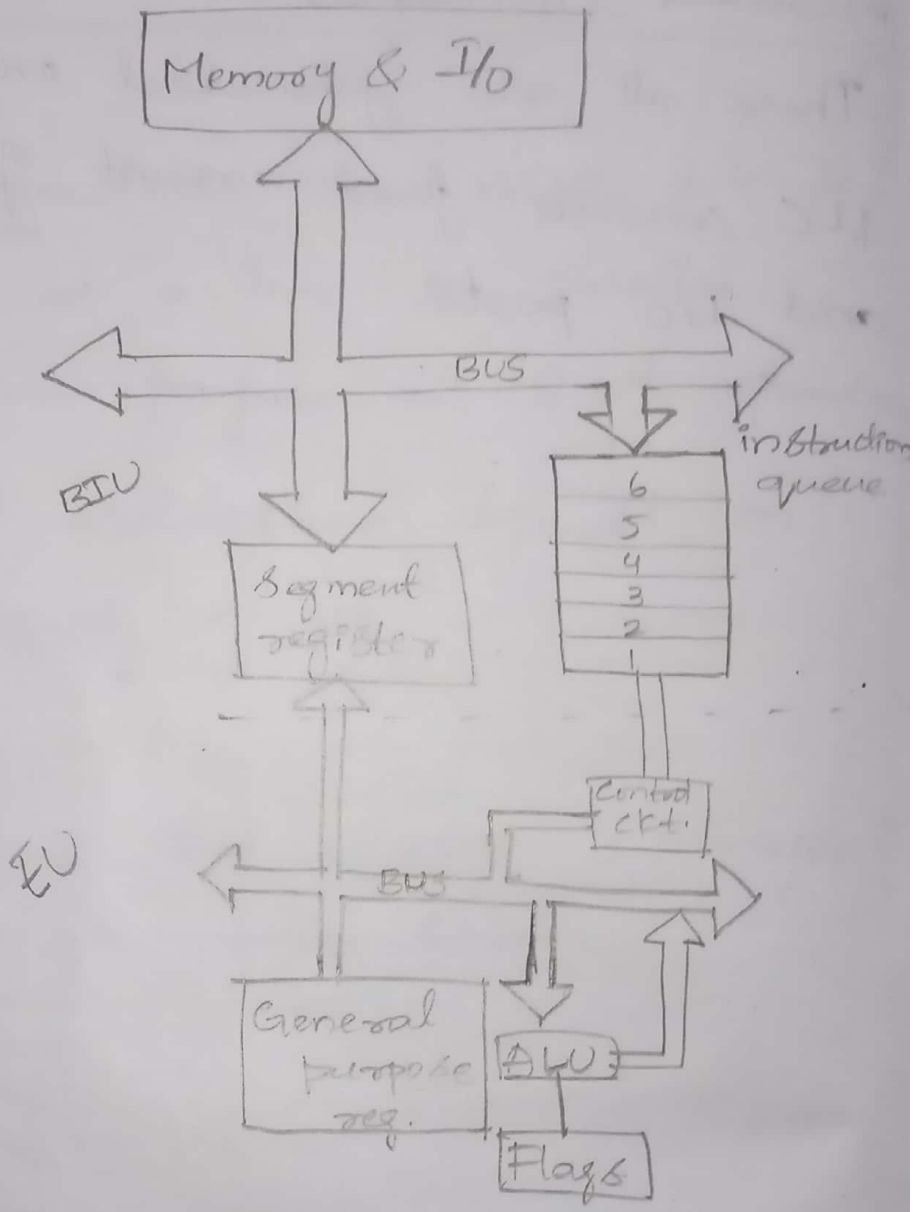
7/12/19.

Unit-1.

8086 Architecture.

Functional Block diagram of 8086:



8086 supports 16 bit ALU, a set of 16 bit registers & provides segmented memory addressing capability, a rich instr. set, powerful interrupt structure, fetched instr. queue for overlapped fetching & execution.

8086 is divided into two separate functional parts. (i) Bus Interface Unit (BIU)
(ii) Execution Unit (EU)

Functions of BIU:

i) It fetches the instr's from the memory
ii) It reads the data from memory (or) I/O devices.
iii) It writes the data into the memory (or) I/O devices
iv) It supports instr. queue.
v) It provides address location facility.

* Functions of EU:

i) EU provides contrd circuit ALU, flag register, instruction decoder, general purpose reg's, pointer & index reg's
ii) EU selects the data depends up on the user app's with the help of general purpose reg's.
iii) It also performs all the arithmetic operations with the help of ALU.

Instruction Queue:

To inc.(↑) the speed of the program execution, the bus int. unit fetches 6 instr. bytes. These prefetched instr. bytes are held for the execution unit in a group of reg.'s is called as queue. With the help of queue it is possible to fetch next instr. when the current instr. is in execution.

Internal function of 8086:

BIU fetches the data from memory & I/O reg.'s. This data will be stored in the segment reg.'s. These reg.'s provide particular address location of data.

BIU fetches the data to the instr. queue, the instr. queue data is decoded by using control ckt.

The decoded data is sent to the execution unit through the system bus & these data is stored in general purpose reg.'s.

the general purp. reg's are used to select the data which operation should be performed depending upon the user app. This data is fetched to ALU. ALU performs all the operations & these results are sent to the BIU, the BIU write the results into the memory. If there is any flag in the o/p of ALU, that will be stored in the flag reg.

The instr. operations is divided by 3 parts.

(i) Fetching
(ii) Decoding
(iii) Executing

18/12 Register Organization:

The 8086 registers are divided into 4 types.

(i) General purpose reg's (AX, BX, CX, DX).
(ii) Segment reg's (CS, DS, ES, SS)
(iii) Flag reg's
(iv) Index (or) Pointer reg's.

(i) General Purpose Registers:

General purpose reg's are 4 types:

AX, BX, CX, DX.

	15 8	7 0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

They are 16 bit reg's. The letter 'x' represents total reg.

AX is used as an accumulator for arithmetic & logical operations.

BX: The reg. is used to store the offset value

CX: CX is general control reg. It is used for looping & counting of no. of instructions

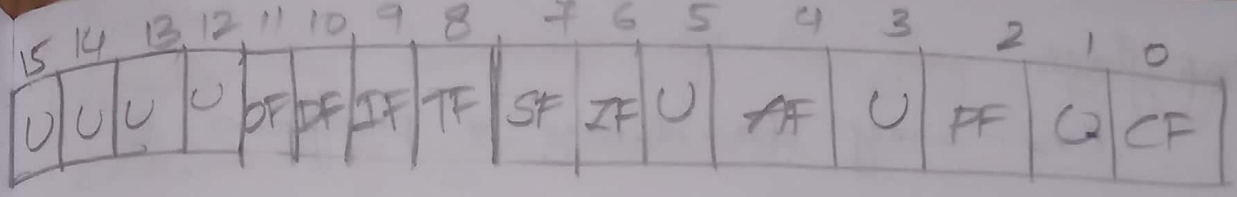
DX: Multiplication & I/O addressing

Flag Registers:

Flag registers are divided into 2 types.

(i) Status flag reg. (CF, PF, AF, ZF, SF, OF)

(ii) Control " " (TF, IF, DF)



CF=1 (Set) when the result contains carry as a MSB bit. Otherwise, CF=0.

Parity flag: It sets when result has even no. of one's. Otherwise, it is zero.

Auxillary flag: This flag is set if there is any carry from lower nibble (D3) to higher nibble (D4). This flag is used for BCD operations & is not availabla for the programmer.

Zero flag: The zero flag sets if the result operation in ALU is zero & flag resets if the result is not zero.

Signed flag:

In result if MSB bit is '1' then the sign is -ve & in result if MSB bit='0' then the sign is +ve.

Overflow flag:

The overflow flag sets the result will be equal to more than 16 bits. Otherwise, it is zero.

Trap flag:

One-way to debug a program & run the program for one instruction at a time is called as single stepping execution. If the machine follows single step execution, then trap flag is set. Otherwise, it is zero

Interrupt flag:

It is set a certain type of interrupt can be recognised by 8086, otherwise these interrupts are ignored.

Direction flag:

It is used for string instr. If DF=0 the str is processed from the lowest address to higher address. It is also called as auto increment process.

If DF=1, the str. is processed from highest address to lowest address. It is also called as auto decrement method.

Problem:

1) Write the contents of the flag reg. after execution of foll. addition.

(1)(1) 0110	(1)(1)(1)(1) 0101	(1)(1) 1101	(1) 0001
0010	0011	0101	1001
(1)000	1001	0010	1010

CF=1 ZF=0

PF=1 SF=1

AF=0 OF=0

Segment reg.'s are divided into 4 types.

(i) Code Segment

(ii) Data "

(iii) Extra "

(iv) Stack "

(i) Code Segment: Code Segment is used in addressing a memory location. It holds the upper 16 bits of the starting address of the segments. It receives the code from memory.

Stack Segment: It is used for the upper 16 bits of the starting address for the program stack.

Extra Segment & Data Segment:

It stores the upper 16-bits of the starting add. of 2 memory seg.'s which are used for data. It receives the data from the memory.

Pointer & index registers:

To get 20-bit physical address, each segment reg.'s are associated with the ptr. & index reg.'s.

The ptr. reg. is IP (instruction ptr.)

SP (Stack ptr.)

BP (Base ptr.)

Index reg.'s are Source index, destination index.

19/12. (Memory Org.)

Memory Segmentation: (Memory Address)

Memory is divided into two types.

(i) Linear addressing method

(ii) Segment " "

(i) Linear Addressing Method:

The entire memory space is available to the processor in one linear array.

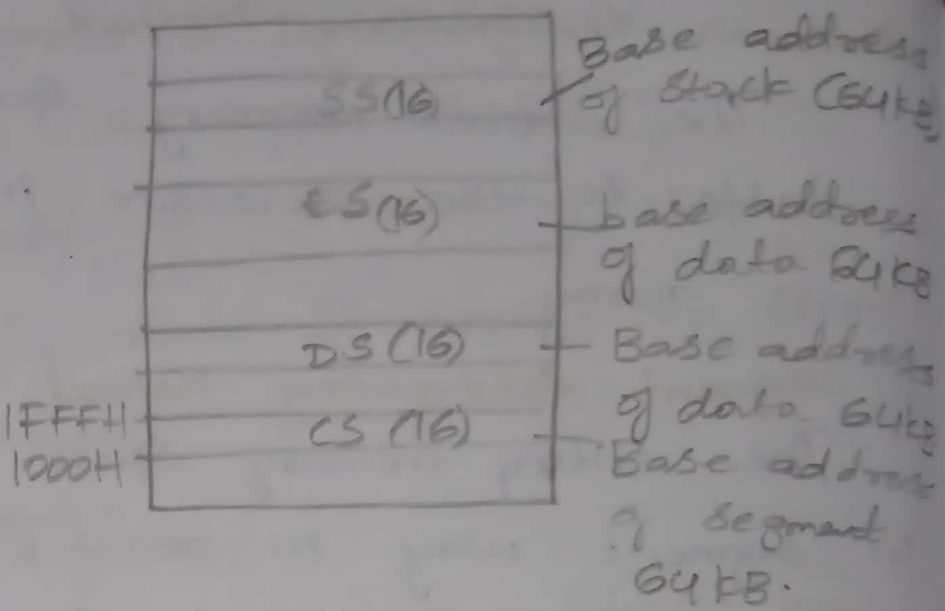
(ii) Segment Addressing Method:

The entire memory space is available in segment way in rows (or) in columns. The MAB memory is divided into 16 logical segments. Each segment is 64 KB. The 16bit contents of the segment register gives the starting (or) base address of a particular segment. To address a specific memory location (or) physical memory location within a segment we need an offset address. The offset address is also 16bits & is provided by one of the pointer (or) index register.

Rules for memory segmentation:

(i) The four segments should not be overlapped

(ii) The segment can start at any memory address which is divisible by 16.



Advantage of Memory Segmentation:

It allows the memory addressing capacity upto 1MB & address is associated with individual segment registers.

Generation of 20-bit address (or) Physical Address Calculation:

To access a specific data from a specific memory location from any segment, we need 20-bit physical address.

Ex: 341AH. (Code Segment)

0011 0100 0001 1010

Left shift & insert Zero.

0011 0100 0001 1010 0000

offset address: 3211H.

0011 0010 0001 0001.

Physical address: CS+IP

0011	0100	0001	1010	0000
+ 0011	0010	0001	0001	
P.A: 0011	0111	0011	1011	0001

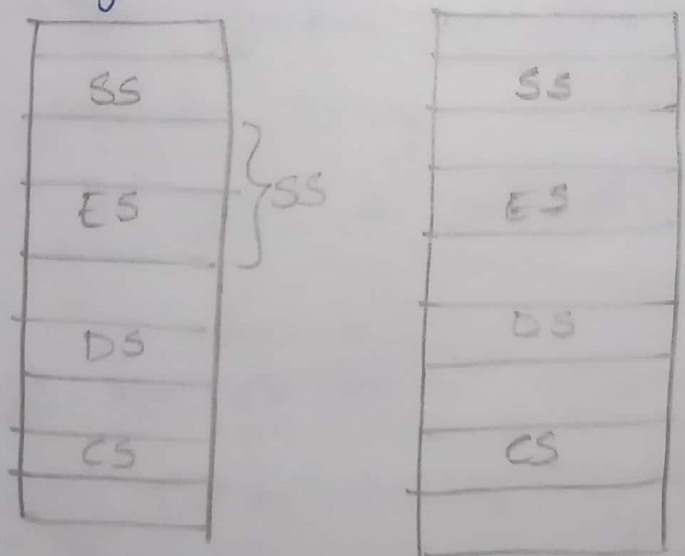
P.A: 373B1H 373B1H

Overlapping Segmentation:

If any two segment regis consist same address memory location, it is called as overlapping Segmentation.

Non-overlapping Segmentation:

If Segments have different memory location, then it is called as non-overlapping Segmentation.



Over-lapping. Non-overlapping.

Default register assignment:

Type of memory	Segment Register	Effective address (or) offset address
instruction fetch.	Code Segment	Ins. Ptr.
Stack operation	Stack Segment	Base Ptr., Stack Ptr.
Data	Data Segment	Source Index (effective address)
String Source	Data Segment	Source Index
String Destination	ES (Extra Seg.)	Dest. Index
BX is a ptr.	DS	BX.

Examples for register assignments:

Q. Segment: CS=1000H, DS=2000H, SS=3000H,
ES=4000H

offset: BP=0010H, BX=0020H, SP=0030H,
SI=0040H, DI=0050H.

dest. Source
(i) MOV AL,[BP]

A. BP=0010H.
SS=3000H. (16-bit)
Left shift: 3 0000 {insert 0}

offset address: 0010

P.A: 30010H. (20-bit) data at this P.A is moved to AL.

→ The instruction copies a bit from the memory location to the AL reg.
→ The effective address for memory location contain BP Reg.
→ BP offset is added to the stack segment.

dest. Source
(ii) MOV CX,[BX]

DS=2000H (16-bit)
left shift: 20000H {insert 0}

offset: 0020

P.A: 20020H (20-bit)

→ Instr. copies a word (16-bit) from memory location to CX reg.
→ BX is associated with DS.

(iii) MOV AL, [BP+SI]
dest source

SS has higher priority than DS. Because it does stack oper.'s also.

BP : 0010
SI : 0040
offset add : 0050H

SS → 3000H

left shift ← 30000H {insert 0}

offset → 0050H

P.A: 30050H.

(iv) MOV CS:[BX], AL : represents addition

BX = 0020H

CS: 1000H

left shift & insert 0 : 10000H.

offset : 0020H

10020H.

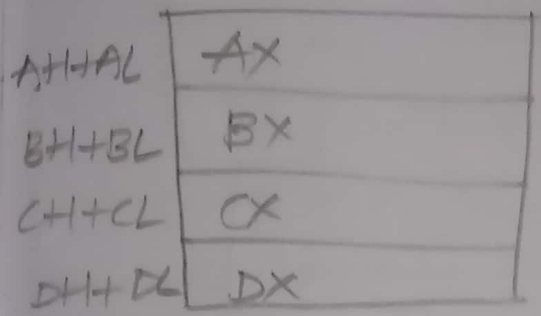
The instruction byte copies from the AL reg. to a memory location. The effective address this mem. location is contained in BX location Default BX will be added to the DS to produce the physical address.

CS:[BX] indicates that BIU wants to add the effective address to the code segment

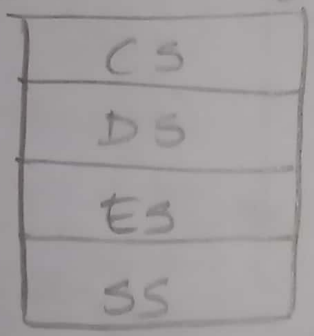
to produce Physical address. This is called as Segment override prefix.

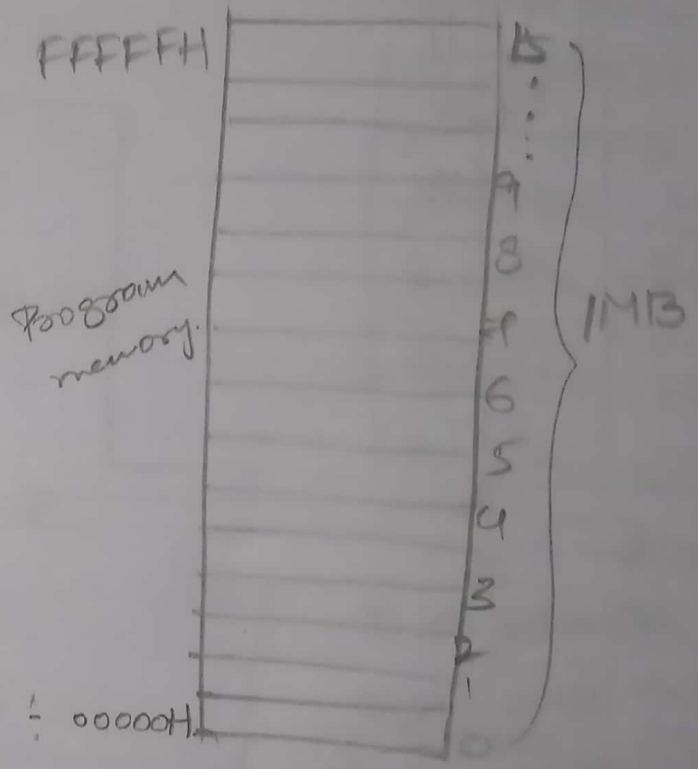
23/12 Programming Model:

General purpose register:

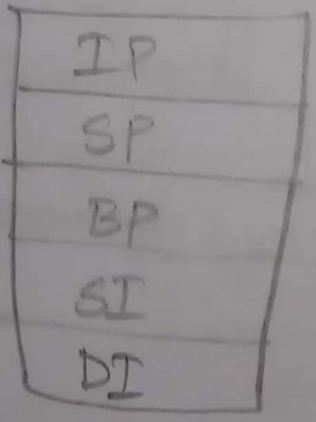


Segment regis:

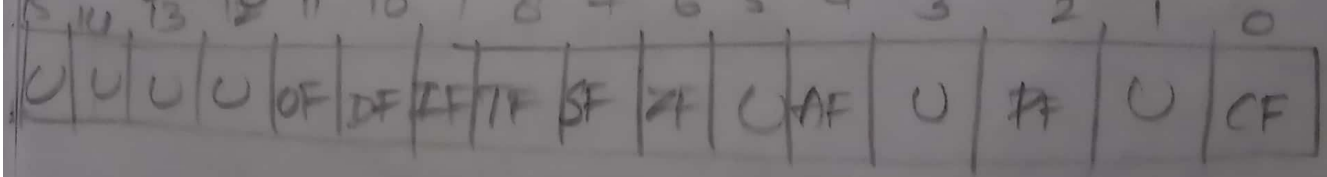




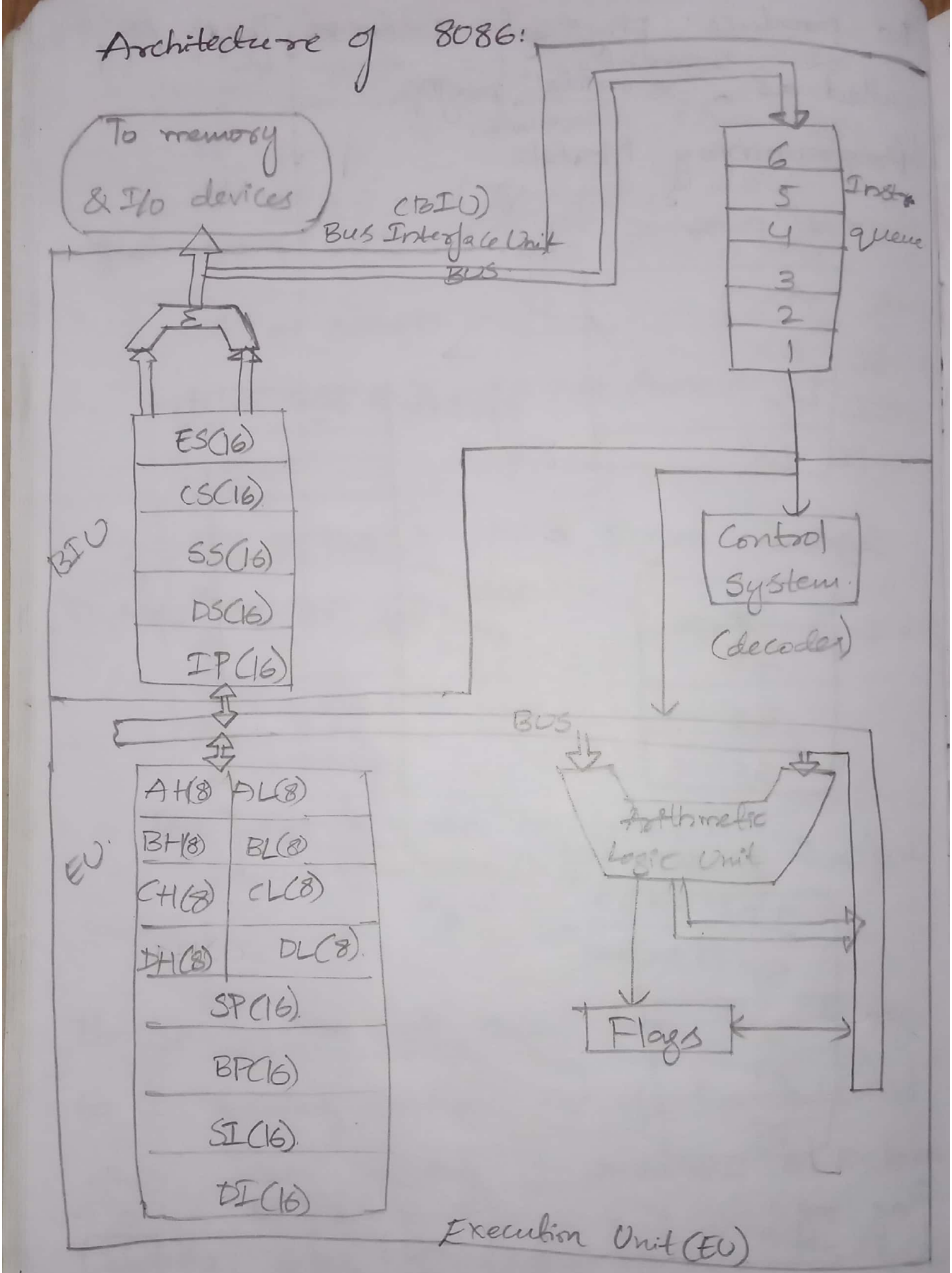
Index & pointer reg. :



Flag reg.:



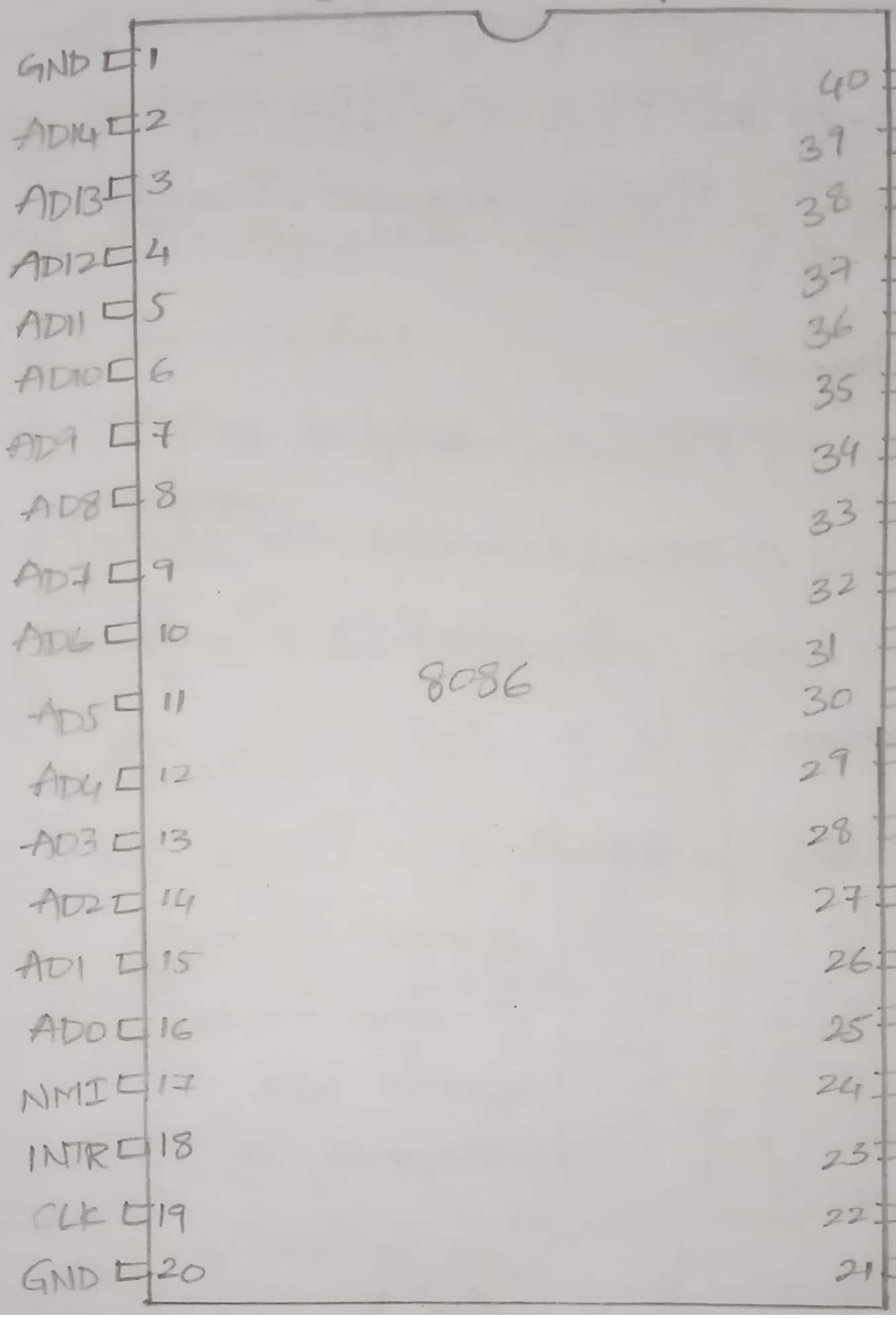
Architecture of 8086:



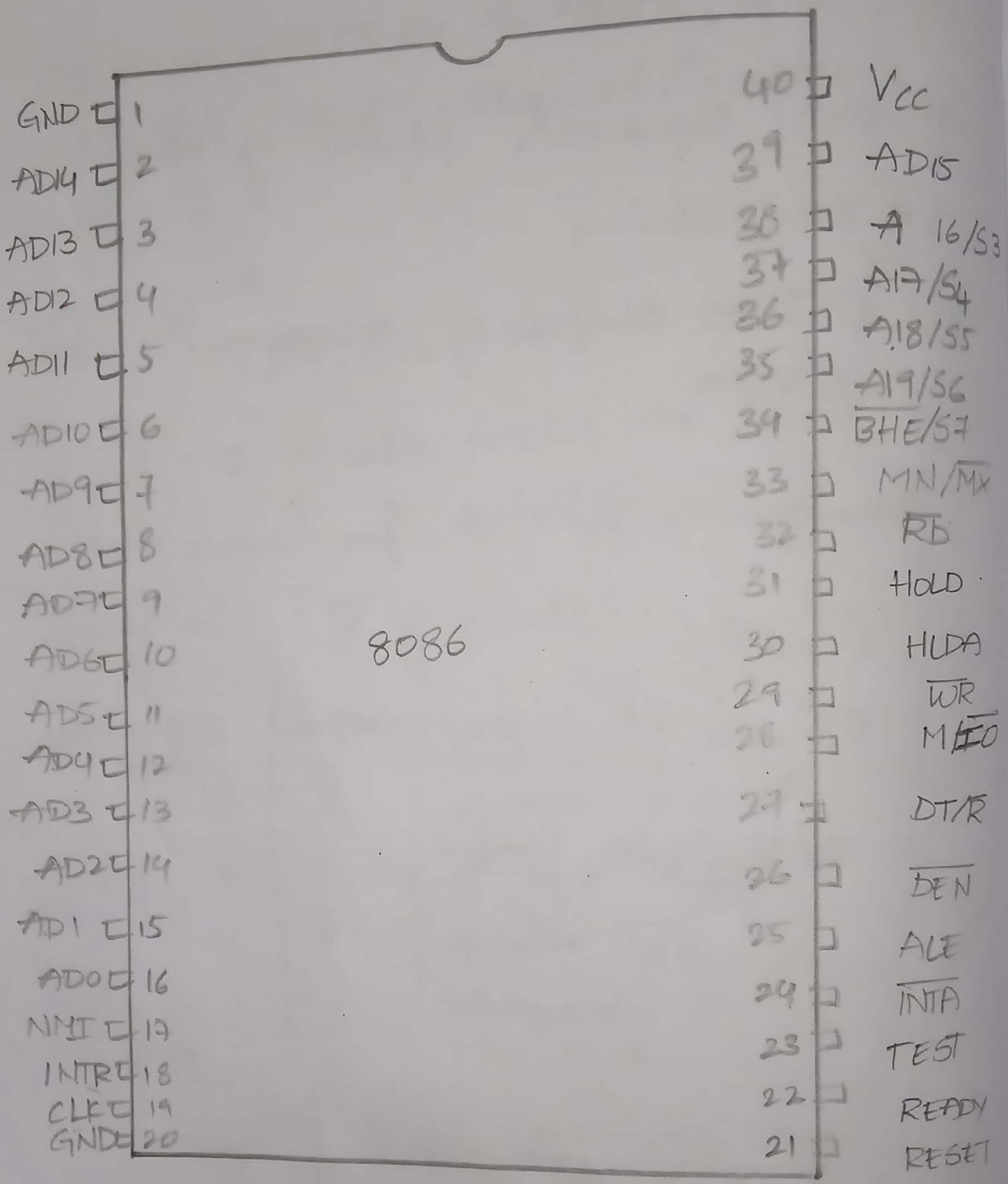
operation is same as 8086 (for block diagram).

2/4/12 Signal Description and Pin diagram of 8086:

(Max. mode of 8086):



Minimum mode of 8086:



8086 is a 40 pin IC. It is available in Dual Inline Package (DIP). It consists 20 address bits & 16 data bits. 8086 µP is operated in 2 modes.

(i) Max. mode: Multiprocessors are connected to I/O and Memory & these are used in large logic devices.

(ii) Min. mode: One processor is connected to I/O devices & memory & this is used in small devices.

In 8086, 32 pins are operated in both min. & max. mode.

8 pins are operated either max. mode (or) min. mode.

Min. mode pins:

31 → HOLD

30 → HLDA

29 → $\overline{WR}$

28 → M/$\overline{IO}$

27 → DT/$\overline{R}$

26 → $\overline{DEN}$

25 → ALE

24 → INTA

Max. mode pins: 31-24

$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$, $\overline{LOCK}$, $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$, $\overline{QS_0}$, $\overline{QS_1}$.

~~I/P~~ AD0 - AD15:

These signals (pins) are bidirectional & multiplexed. During the 1st clock cycle these pins are consist address information & in the remaining cycles, they contain data info. The address is separated by address latch enable pin.

A16/S3: - A19/S6: (Address/Status):

During the 1st clk cycle, they contain address info. In the remaining cycles, they contain status info.

S_3 & S_4 indicates the ~~signal~~ Segment register which is used for generating 20-bit physical address.

S_3	S_4	Segment Register.
0	0	ES
0	1	SS
1	0	CS
1	1	DS.

S5 represents the status of interrupt flag. & S6 is always zero (or) unused.

NMI: (Non-Maskable Interrupt flag):

This is an interrupt request to processor from external devices. It cannot be disabled by using interrupt flag.

INTR (Interrupt Request):

It is a maskable interrupt request to processor from external devices. It can be enabled (or) disabled using interrupt flag.

GND (Ground):

This i/p should be connected to the -ve terminal of the DC supply.

Clk (Clock):

This i/p clock signal provides the synchronisation for the μP. This signal should have a duty cycle of 33% for proper functioning of the μP. The clk freq. depends upon the version of 8086 μP.

Version	Frequency
8086	5MHz
8086-I	8MHz
8086-II	10MHz.

V_{CC}:

This i/p is connected to the +ve terminal of +5V DC supply.

$\overline{BHE}$/S7: (Bus High Enable):

During the 1st clk cycle, it contains no of bits data info.. Remaining cycles, it contains S7.

$\overline{BHE}$	$\overline{BLE}$ (A0)	No. of bits
0	0	Total word
0	1	Upper 8-bits
1	0	Lower 8-bits
1	1	No operation.

S7 is always high.

MN/$\overline{MX}$:

If MN=1, the 8086 is operated in min mode.

If $\overline{MX}$=0, 8086 is operated in max. mode.

$\overline{TEST}$:

The signal is only used by the wait instruction. The 8086 enters into a wait state after execution of the low signal on the $\overline{TEST}$ pin. The test signal is synchronised internally during each clk cycle.

READY:

This i/p signal indicates to the μP whether memory (or) i/p devices is ready for data transfer. If it is high, they are ready. If it is low, μP waits for to become high.

RESET:

This i/p signal resets the μP. At that time, all segment reg's & offset value equal to zero. The code segment reg. is initialised to FFFFH, after reset the processor executes instr's from memory location FFFF0H.

$\overline{RD}$:

$\overline{RD}$=0, then processor reads the data from the external devices.

Maximum mode Pins:

$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$: These signals are used as i/p's & o/p's. Other processors send the request for the system bus & 8086 µp sends the grant for the request using these signals. $\overline{RQ}/\overline{GT_0}$ is having the highest priority than $\overline{RQ}/\overline{GT1}$

$\overline{LOCK}$: This signal indicates the instr. with Lock prefix is being executed & it cannot be used by other processors (or) controllers.

$\overline{S_2}, \overline{S_1}, \overline{S_0}$ (Status bits):

These bits represents status of machine cycle.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Machine cycle.
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Machine cycle.
0	1	1	Halt
1	0	0	instruction fetch
1	0	1	memory read.
1	1	0	memory write
1	1	1	in active.

QS_1, QS_0:-

It represents status of queue.

QS_1	QS_0	Status of queue.
0	0	Queue is idle
0	1	1st bit is enter into 'Q'
1	0	'Q' is empty.
1	1	Subsequently perform 'Q' instr.

Min. mode pins:

HOLD:

This pin indicates a request by I/O devices. If it is high, the HP stops executing instr's.

HLDA: This pin will be high by processor to indicate that hold req. has been accepted.

$\overline{WR}$: When $\overline{WR}$ =0 it writes the data into the memory (or) I/O devices.

M/$\overline{IO}$: It shows the direction of data bus. If M=1, the data will be stored in the memory (or) fetched. If $\overline{IO}$=0, the data will be fetched from the I/O devices.

DT/$\overline{R}$: It represents the data transmission & receiving. If DT=1, the data will be transmitted. If $\overline{R}$=0, the data will be receiving.

$\overline{DEN}$ (Data Enable Pin):

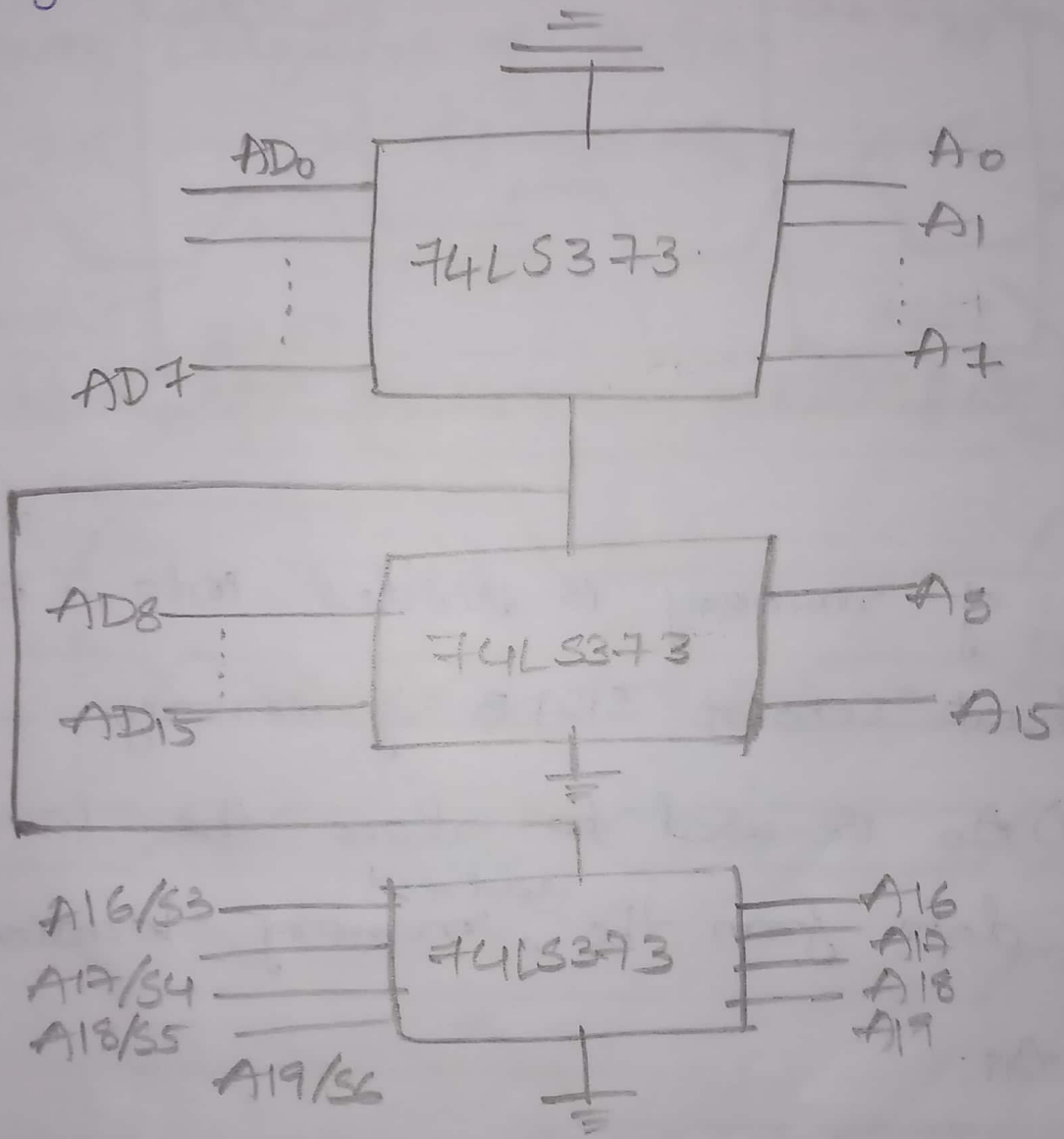
This signal is used for enable the data from AD0 to AD15.

$\overline{INTA}$:

This pin is activated by processor in response to INTR signal. This signal is used to identify the INT. type of the (CPU) processor.

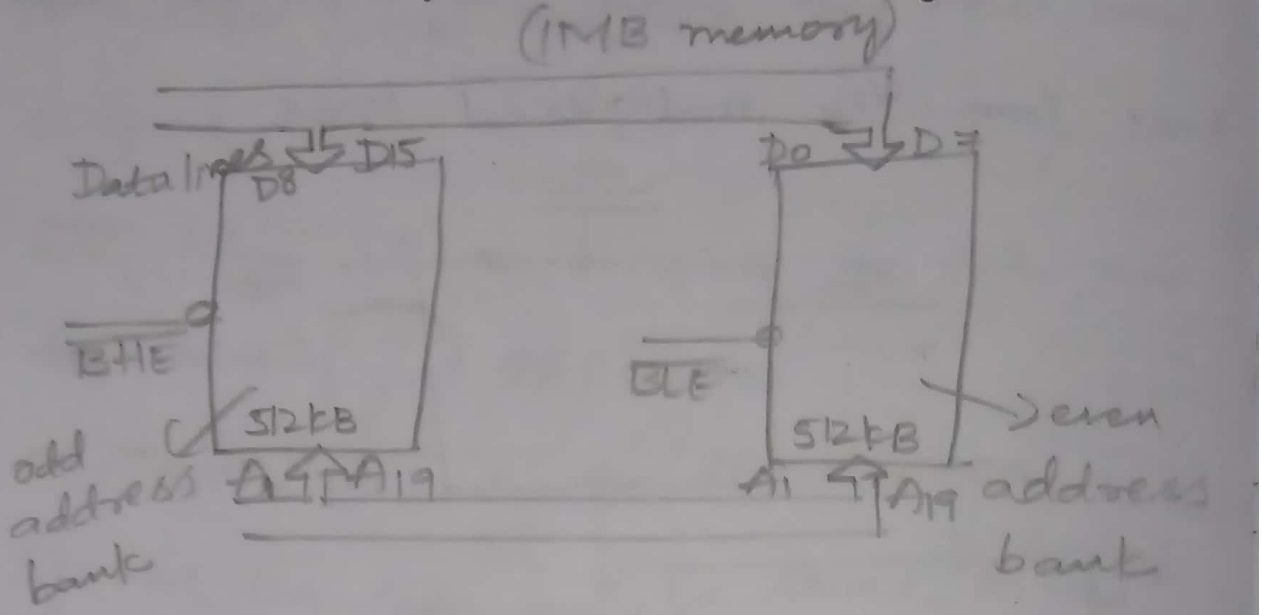
ALE (Address Latch Enable):

It is used to separate address lines from the multiplexed lines.



30/12

Physical Memory Organisation of 8086:



1MB of memory is divided into 2 banks.

Each bank consist 512kB of memory.

$\overline{BLE}$ (or) A_0 is used to allow the lower bits of data from the address memory location A_1 to A_{19}.

$\overline{BHE}$ is used to allow the upper bits of data ($D_8 - D_{15}$) from the same memory address location A_1 to A_{19}.

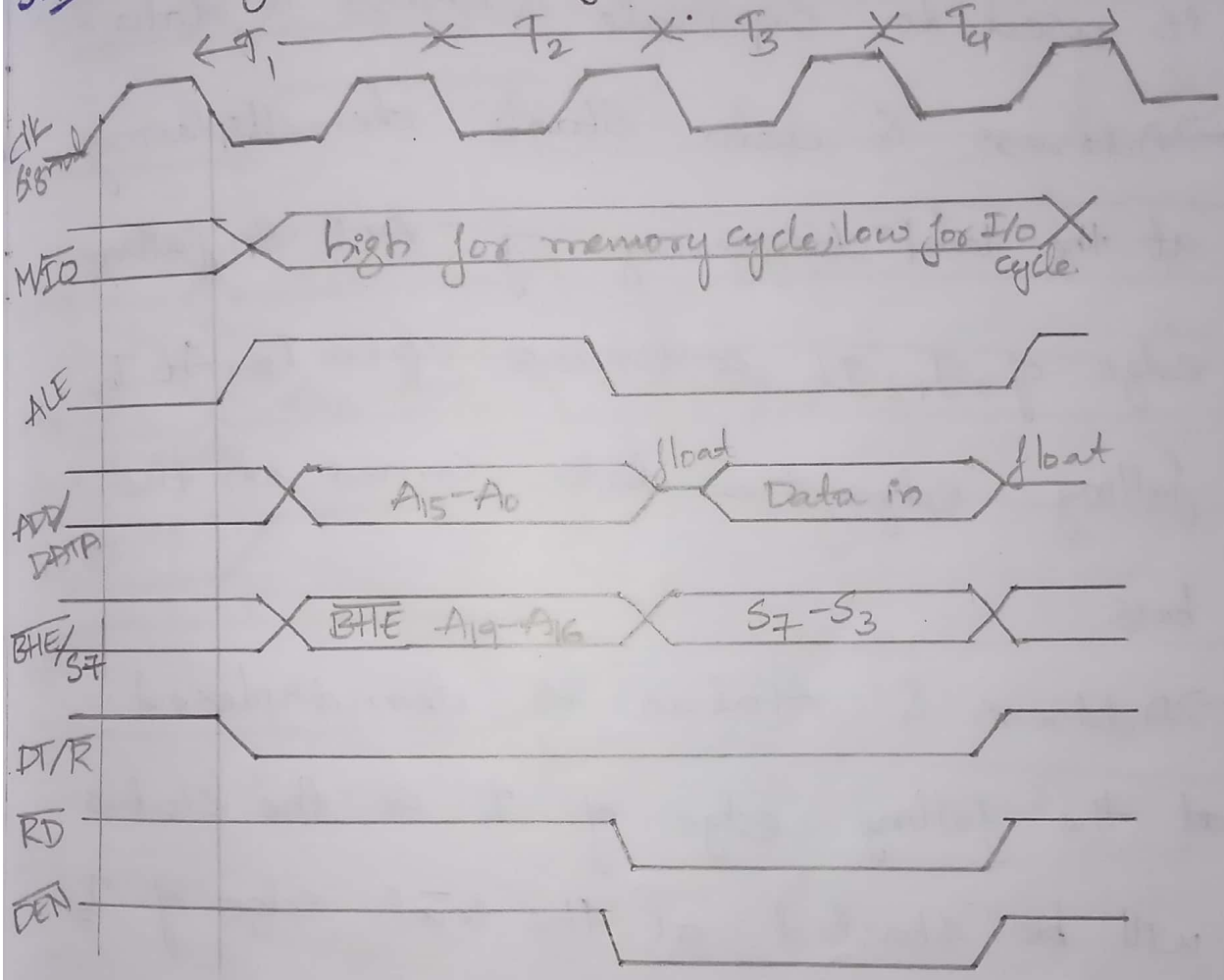
$\overline{BLE}$: even address bank.

$\overline{BHE}$: odd

	$\overline{BHE}$	$\overline{BLE}$	No. of data bits
read/write operation odd address bank	0	1	Upper 8-bits of data
read/write operation even address bank.	1	0	Lower 8-bits of data

	$\overline{BHE}$	$\overline{BLE}$	No. of data bits
read/write operation odd & even bank	0	0	total 16-bits of data.

31/12. Timing Diagrams of 8086:



2/1/20.

Timing diagram for Min. mode read operation:

The processor completes its operation in one cycle. One cycle consists four clk pulses (t_1, t_2, t_3, t_4). The processor sets memory/$\overline{IO}$ signals. It takes the data from the memory when m=1 & it takes the data from IO when $\overline{IO}$=0.

$M/\overline{IO}$ demultiplexed at the falling edge of the T_1 Clk pulse.

The processor sets to ALE signal. ALE is used to separate address & data. (Same in Min)

→ Address & data starts demultiplexing at the level triggering of ALE & falling edge of T_1. It continues upto T_2. At T_3 falling edge, the data comes on the bus.

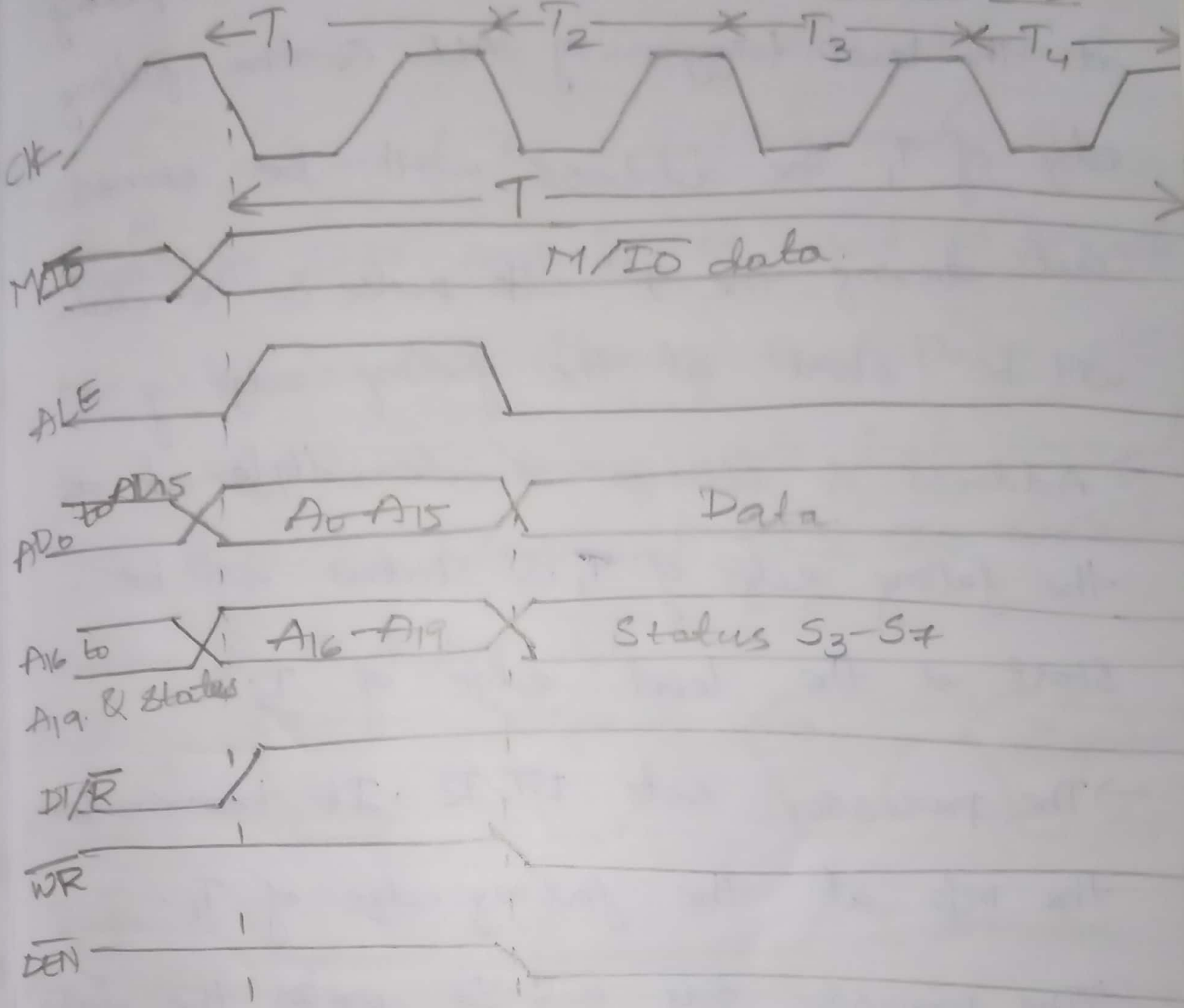
→ Address & Status is demultiplexed at the falling edge of T_1 & the status will be started at the level edge of T_2.

→ The processor sets $DT/\overline{R}$. It receives the info. at the falling edge of T_1.

→ The processor sets $\overline{RD}$. It reads the data when the data comes out on the multiplexed lines.

→ Then the processor sets to $\overline{DEN}$. It enables the data when the data comes from the multiplexed lines.

Min. mode Write Operation: Timing Diag:



3 points from Min. mode read operation.

→ The address & data starts demultiplexing at the level trigger of ALE & the falling edge of T_1. The address will be comes out during the 1st clk pulse & the data will be start at the falling edge of T_2.

→ Address & Status is demultiplexed at the falling edge of T_1 & Status will be start at the level edge of T_2.

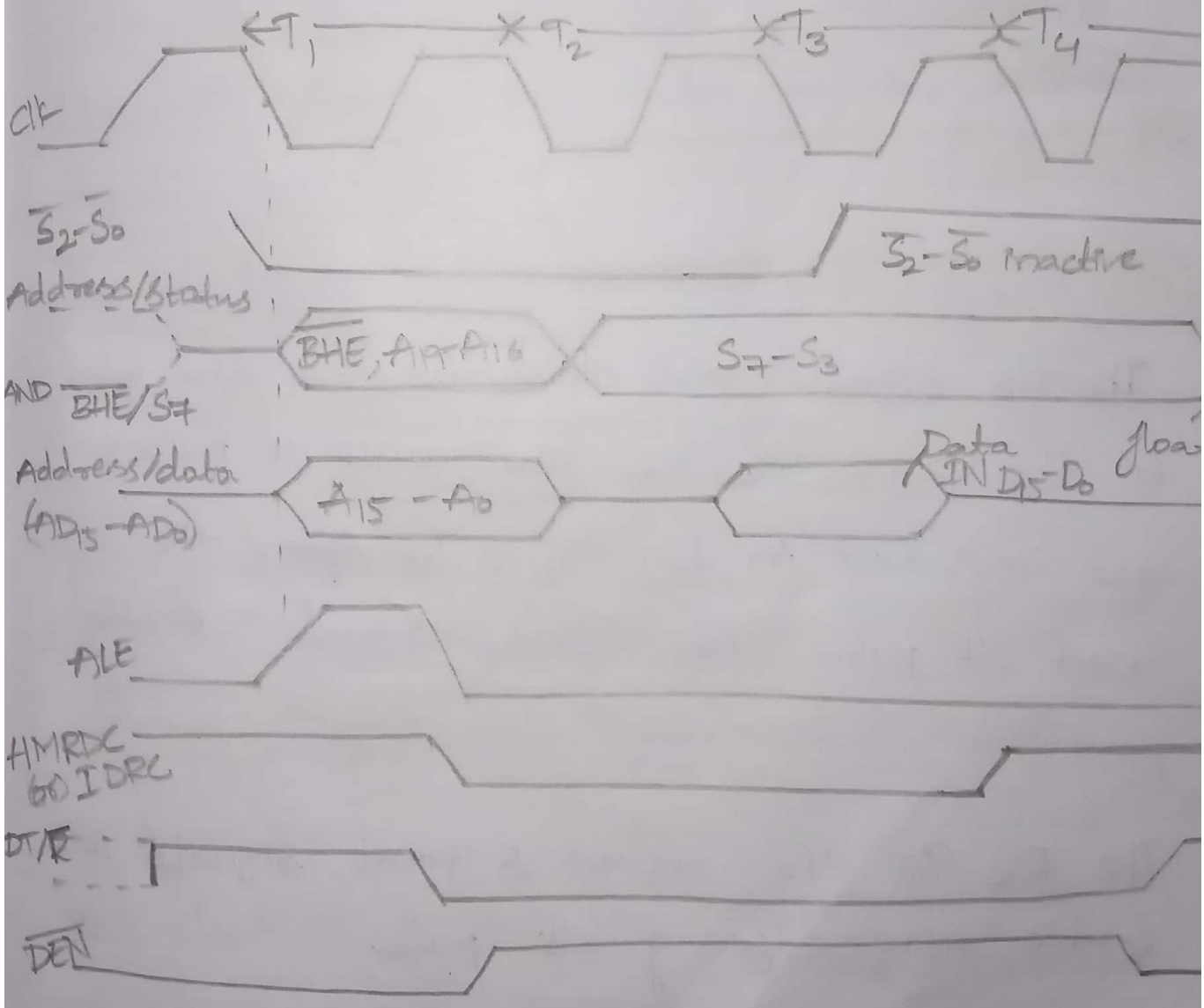
→ The processor sets DT/$\overline{R}$. It transmits the info. at the falling edge of T_1.

→ The processor sets $\overline{WR}$. It writes the data when the data comes out from the multiplexed lines.

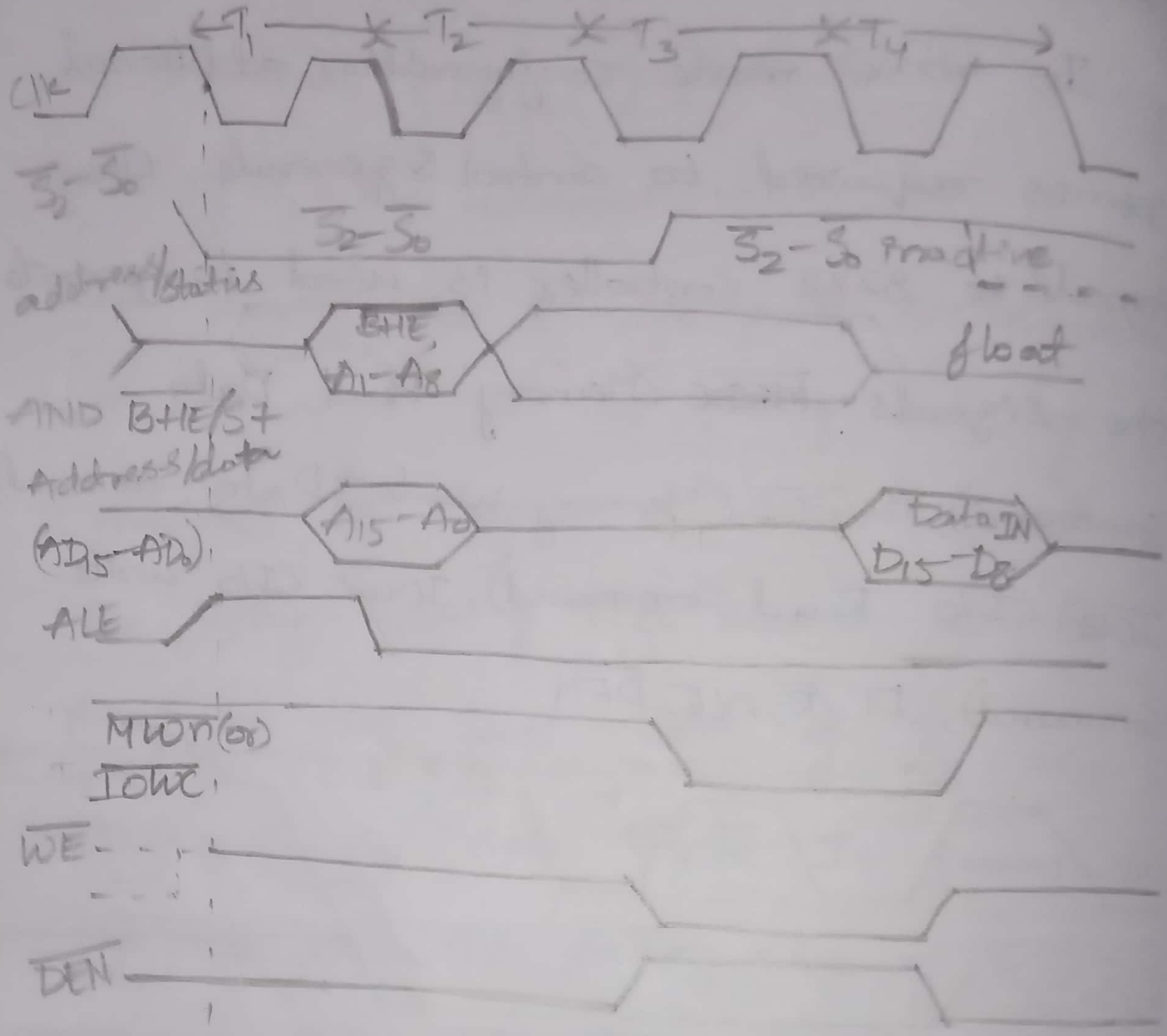
→ The processor sets to $\overline{DEN}$. It enables the data when the data comes from the multiplexed lines.

Maximum mode Read Operation:

In max. mode configuration, additional ckt is required to control & generate the signals ~~is~~ 8288 controller is used to generate the signals $\overline{MRDC}$ (Memory Read Data Command), $\overline{MWDC}$ (Memory Write Data Command), $\overline{IORC}$ (I/O Read Command), $\overline{IOWC}$ (I/O write Command), DT/$\overline{R}$, ALE, $\overline{DEN}$.



Write Operation Max. mode :



The i/p signals are slow to sent the m/c. to the processor. Then the WAIT clk pulses are included in b/w T_3 & T_4. After the wait clk pulse, the ready signal is Enable

The $\overline{RQ}/\overline{GT}$ the request & grant signals are enable rising edge of clk pulse.

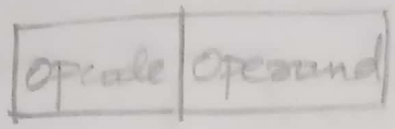
3/1/20.

Unit-II.

Instruction Set and Assembly Language Program of 8086.

Instruction Format:

The instruction format consists OPcode & Operand.



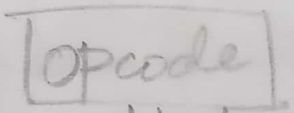
OPcode:

OPcode shows the type of instruction.

Operand shows the data, register, memory location and I/O.

The instruction format length 1-6 bytes.

One byte instruction:



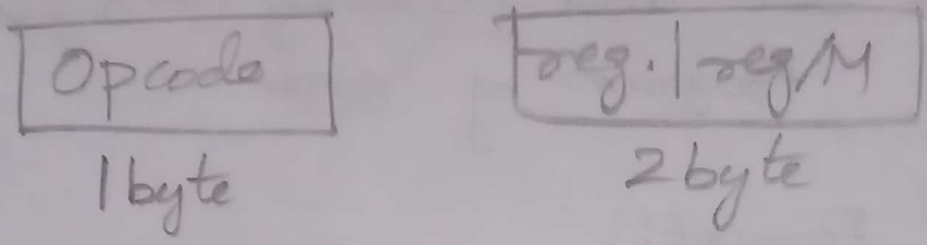
Ex: JMP, JNZ, CBW (Convert Byte to Word)

One byte instruction register to register:

Ex: PUSH BX.

The contents in the BX register will be moved to stack register.

Two byte instruction register to register/M



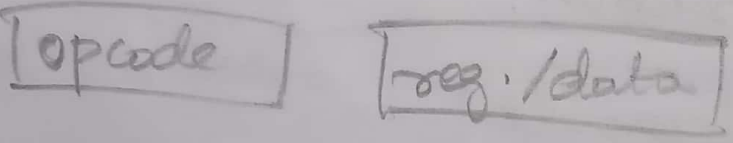
EX: MOV AX, BX.

Two byte instruction register to register/M with no displacement:

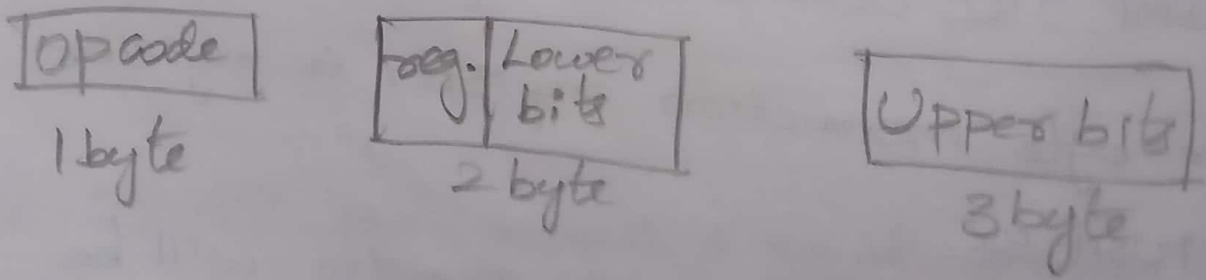
EX: MOV AX, [SI].

Two byte instruction immediate data:

EX: MOV AX, 05
MOV BX, 03
Add AL, BL.



Three byte instructions immediate data:



EX: MOV AX, 1234H.

11/20.

Addressing Modes.

The way of representation of data operand in the instruction format is called as Data Addressing Mode.

Addressing modes are classified into 3 types.

1) Data Addressing Mode
2) Program Memory Addressing Mode
3) Stack " " "

Data Addressing Mode is divided into 8 types.

(i) Immediate Addressing Mode.

MOV AX, 05H

{Copies 05H into AX reg.}

~~AX= AL + AH + AL~~

= 00 05H.

data direction
D. ← S

(ii) Register Addressing Mode:

MOV AX, BX.

{Copies the contents in BX (16-bit reg.) to AX 16-bit reg.}.

(iii) Direct Addressing Mode:

MOV AX, [3000H]

EA : Address / offset address

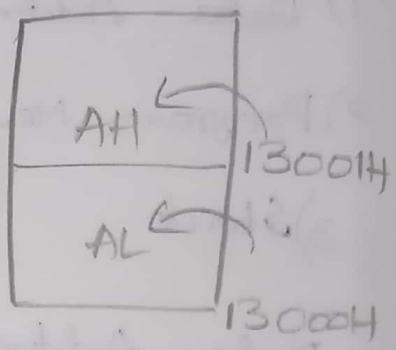
EA is associated with data Segment reg. (default)

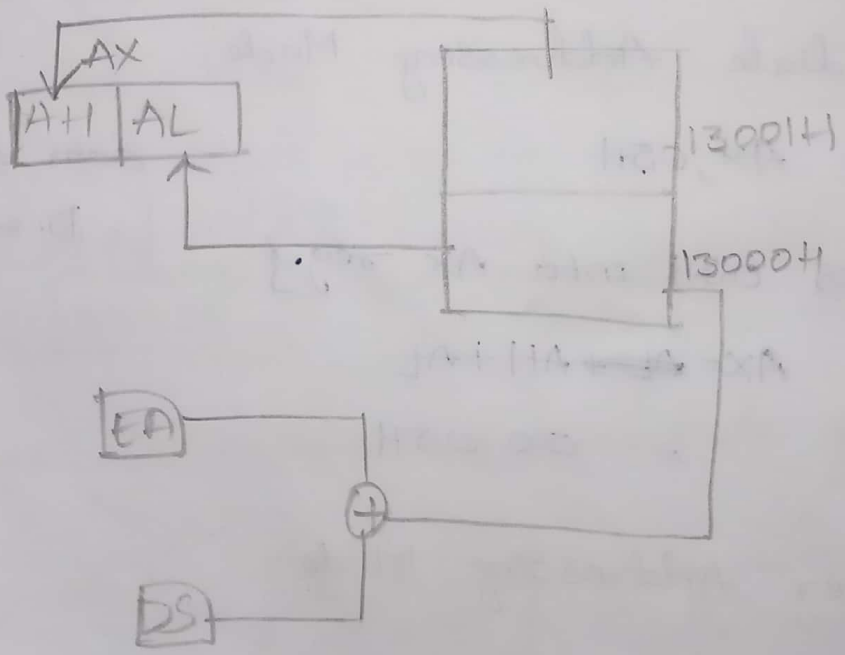
DS = 1000H

10000H

3000

Phy. Address: 13000H





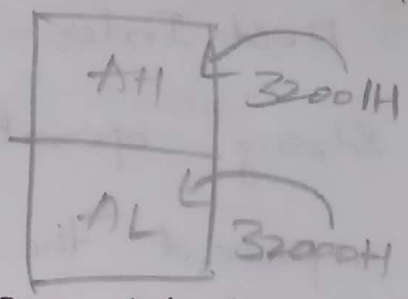
(iv) Register Indirect Addressing Mode:

MOV AX, [BP]

BP = 2000H

Stack Segment = 3000H.

P.A = 30000H
2000H.
32000H.



(V) Base Plus Index Addressing Mode:

MOV AX, [BP+SI]

(or)

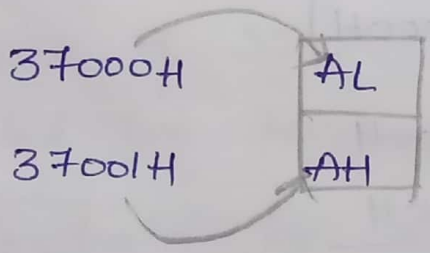
MOV AX, [BX+DI]

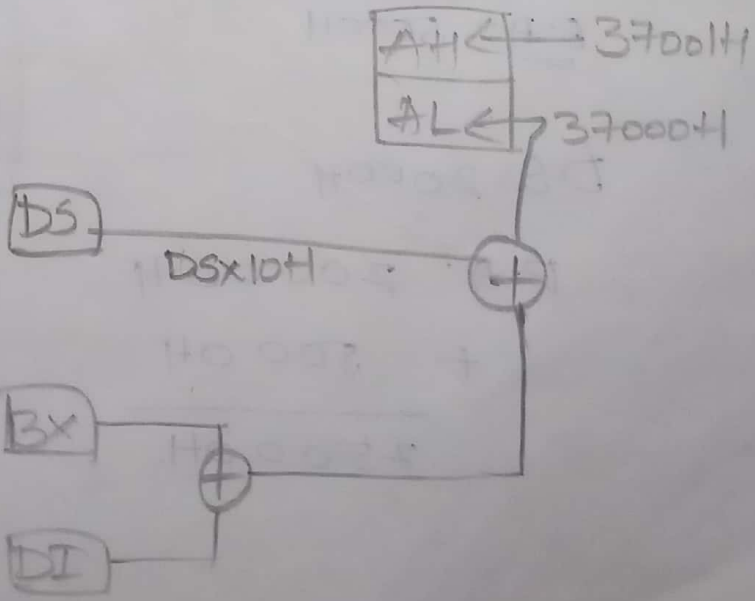
BX = 2000H
DI = 5000H
} E.A

E.A = 7000H.

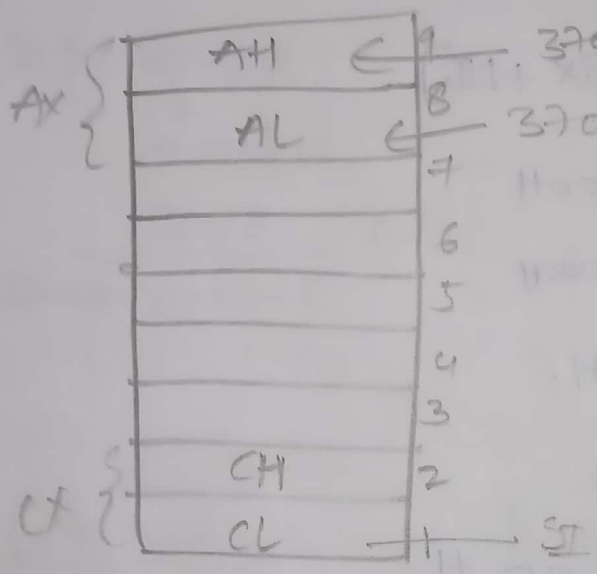
DS = 3000H

P.A = 30000 H
7000 H
37000H





Dest Index, Source Index are used in String operation. S.I P.Address is used to represent the Starting Pt. of the array & D.I represents the ending pt. of the array



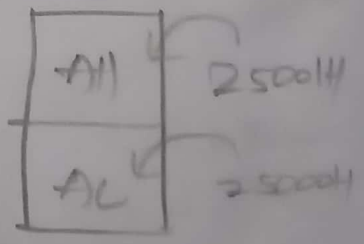
(VI) Register Relative Addressing Mode:

MOV AX,[BX+3000H]

BX = 2000H

3000H

EA: 5000H



DS: 2000H

P.A: 20000H

+ 5000H

25000H.

Register relative

(vii) Base Plus Index Addressing Mode:

MOV AX, [BX+DI+3000H]

BX = 1000H

DI = 5000H

E.A = 90000H.

D.S = 3000H

P.A = 30000H
+ 9000H

39000H

39000H	AL
39001H	AH

(viii) String Addressing Mode:

MOVS BYTE

The contents of SI & DI are automatically incremented (or) decremented by one byte.

Addressing modes for accessing I/P - O/P ~~modes~~ ports:

Out 05H, AL
S D

In AX, 80H
D S

In AX, DX
D S

Program Memory Addressing Modes:

They are classified into 3 types.

1) Direct Program Memory Addressing Mode
2) Relative " " "
3) Indirect " " "

1) Direct P.M.A.M's:

In this addressing mode, address where to transfer program control is specified within the instruction along with the Opcode

Initially, these addressing modes are associated with Code Segment & Instr. Ptr.

CS = 2000H

IP = 3000H

Phy. Add.: 23000H

The JMP instr. is jump to 23000H memory location for the next instruction.

Ex: JMP

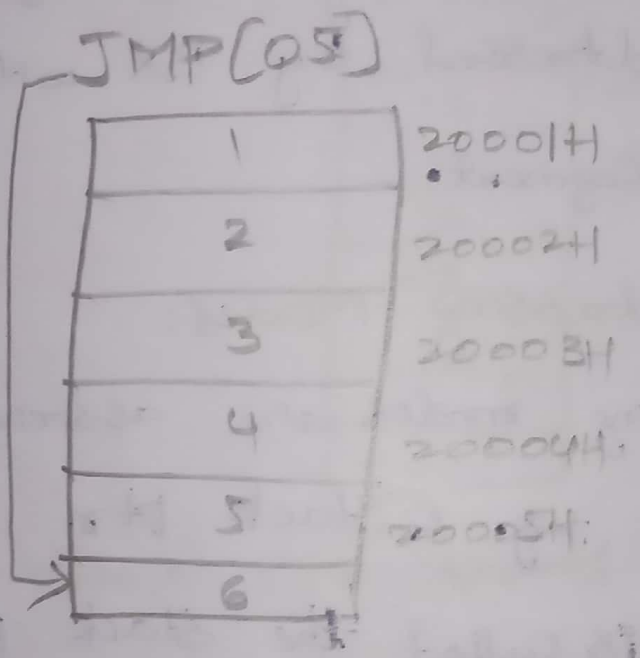
CALL

JNZ (Jump Non-zero to Zero)

Inter Segment Jump:

If the source & dest's are different segment reg's, it is called as inter seg. Jump (or) far jump.

2) Relative, P.M.A.M's:



The JMP instruction skips the 5 bytes of memory, the address in relation to the instr. ptr. ptr. is a file that adds to the instr. ptr.

Intra Segment Jump:

The source & dest's are same Segment reg's, it is called as intra seg. Jump (or) near Jump.

3) Indirect Segment P.M.A.M's:

JMP BX : Jumps to memory location addressed by BX within current Code Seg.

NEARPTR

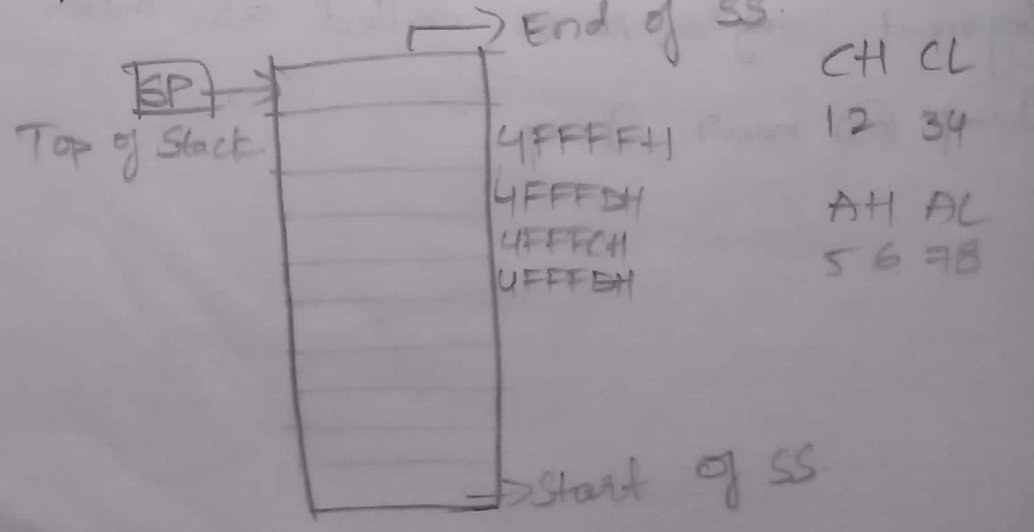
JMP ~~NEARPTR~~ [BX] : Jumps to memory location addressed by the contents of data segment memory location addressed by BX within the current code segment.

Stack Memory Addressing Modes:

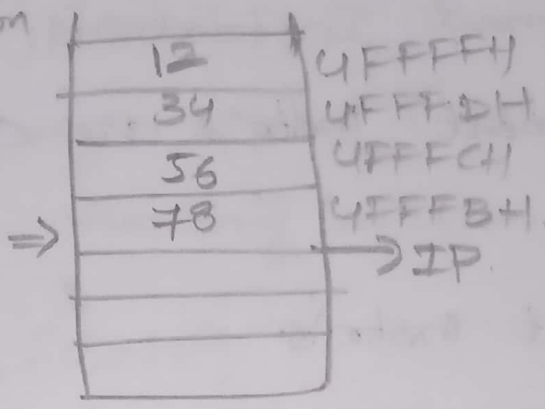
These addressing modes are associated with Stack Segment reg. & Stack ptr.

The Spl. ptr. reg. is called the Stack ptr., during push & pop operations Stack ptr. reg. gives the address of memory where the info. is to be stored & where to be read.

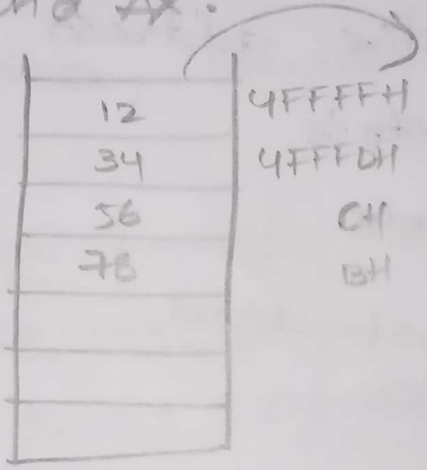
The memory location currently pointed by Stack ptr. is called 'Top of Stack'.



after execution



POP CX and AX:



12 34 56 78

Stack Ptr. moves to top of Stack

9/1

Instruction Set of 8086:

The instr's are classified into 8 types.

They are:

1) Data movement instructions.
2) Arithmetic and logic instr's
3) String instr's
4) Program control transfer instruction
5) Iteration instr's
6) Processor control instr's.
7) External H/w Synchronization instr's
8) Interrupt instr's.

1) Data movement Instructions:

Data movement instr's are classified into 5 types.

(1) Movement instr's.

(2) Stack instr's

(3) Load effective address instr's

(4) String instr's

(5) Other instr's.

(1) Movement Instructions:

Dest, Source

MOV AX, BX

MOV AL, 05H

MOV AX, [BX].

(2) Stack Instructions:

PUSH CX

POP CX.

(3) Load effective address Instructions:

dest source

1) LEA SI, BP

LEA BX, SS: STACK_PTR

Load Source Index register with offset of Base Pointer in Data Segment Reg.

BX is loaded with stack ptr. offset address in Stack Segment reg.

2) LDS BX, [3910H]
(D, S)

Load BX reg. with data which is stored in memory location in DS reg.

3) LES AX, [3901H]
(D, S)

Load AX reg. with data which is stored in memory location in Extra Segment reg.

(4) String Data Transfer Instructions:

1) MOVS / MOVSB / MOVSW.

SI → Source Index, DI → Destination Index.

MOVSB → Moves a byte from SI to DI.

MOVSW → " " word " " " "

2) REP (repeat):

Instruction	Condition exist
REP	CX=0.

The instrs will be repeat until the specific condition exists.

3) LODS / LODSB / LODSW:
(Load String / Load String Byte / LODSword)

MOV SI, OFF_PTR

LODS OFF_PTR (loaded from AX reg.)

The off-pto is loaded with the data which is stored in the AX reg. (default)

4) STOS/STOSB/STOSW: (Store)

```
MOV DI, Sum_th
STOSW Sum_th
```

Sum_th is stores with a word from AX reg.

5) Other Data Transfer Instr's:

1) XCHG destination, Source.
(EXchange)
2) XLAT : Translate a byte in AL

IN & OUT Instr's.

- IN AL, 0983H

(Source represents port no.)

The data is moved to AL reg. from the port no. 0983H.

- OUT 0832H, AL

The data which is stored in AL reg. will be moved to o/p port 0832H.

2. Arithmetic & Logical Instr's:

They are classified into 8 types.

(i) Addition instr's.

(ii) Subtraction

(iii) Multiplication

(iv) Division

(v) BCD & ASCII

(vi) Comparision

(vii) Basic logic instr's (AND, OR, NOT, EX-OR)

~~(viii) AND, OR, NOT, EX-OR~~

(viii) Shift & rotate instr's

(i) Addition Instr's: ADD 05,03

ADD Dest, Source. — ADD AX, BH

ADC Dest, Source (with carry)

20/1/20.

(1) ADD destination, Source

Ex: ADD AX, BX {BX+AX}.

(2) Addition with Carry:

ADC destination, Source

Ex: ADC AL, BL {BL+AL+CF}.

(3) Increment

INC destination

INC AL {the bytes in AL is incremented by 1}

(ii) Subtraction instructions:

(1) Subtraction:

Ex: Sub destination, Source.

Sub AL, BL {BL-AL}.

(2) Subtraction with Borrow:

Ex: SBB destination, Source.

SBB AL, BL {BL-AL-B}.

(3) Decrement:

DEC destination

DEC AL {the bytes in AL decremented by 1}

(4) Negative instruction:

NEG destination

NEG AL {It gives 2's complement of destination}

(iii) Multiplication instructions:

(1) MUL Source:

MUL BX.

(iv) Division instructions:

(1) DIV Source:

DIV BL

(v) Compare instructions:

CMP destination, source

CMP AL, BL.

{it compares the bytes in BL with AL}.

(vi) BCD and ASCII instructions:

(BCD: Binary Coded Decimal)

Two types.

1) DAA {decimal adjust after addition}.

2) DAS {decimal adjust after subtraction}.

1) DAA

DAA destination.

Ex: ADD AL, BL.

AL = 1010 1110

BL = 1110 0001

1 1011 1111

=> 7F

AL: 1001 0110

BL: 0000 0111

1001 1101

=> 9D

+ 0000 0110 Add 6

10100011

=> A3

AL: 0011 1001 ⇒ 39

0001 0010 ⇒ 12

AL: 0100 1011 ⇒ 4B

$D_7 D_6 D_5 D_4$ $D_3 D_2 D_1 D_0$

higher bit lower bit

DAA: 0000 0110

0101 0001 ⇒ 51.

1010 0011

+0110 0000

1 0000 0011

1 03

If $D_0 - D_3 > 9$, add '6' to lower bit

If $D_4 - D_7 > 9$, add '6' to higher bit

⇓

0110.

2) DAS:

DAS destination

Ex: AL = 0011 0010

BL = 0001 0111

If AL > 9, subtract 6

2's comp. of BL: 1110 1000

\+ 1

1110 1001

AL: 0011 0010

BL: 1110 1001

AL: 0011 0010

AL: 1 0001 1011 ⇒ 1B.

- 0000 0110

1 0001 0101

⇒ 15 (BCD)

ASCII Arithmetic

ASCII numbers range in value from 30H to 39H for the numbers 0-9

AAA: ASCII adjust after addition

AAS: ASCII adjust after subtraction

AAM: ASCII adjust after multiplication

AAD: ASCII adjust before division.

AAA instruction: ASCII adjust for addition.

* The numbers from 0-9 are represented as 30H-39H in ASCII code. When you want to add two decimal digits which are represented in ASCII code, it is necessary to mask upper nibble (3) from the code ~~(it is necessary to mask upper nibble (3) fi)~~ before addition.

The 8086 allows you to add the ASCII codes for two decimal digits without masking off the "3" in the upper nibble of each digit. The AAA instruction can be used after addition to get the current result in unpacked BCD form.

Examples:-

AL = 0011 0100 ASCII 4

CL = 0011 1000 ASCII 8

ADD AL, CL AL = 0110 1100

6CH = incorrect temporary result

AL = 0000 0010 unpacked BCD form

AAA

carry = 1 to indicate correct answer is 12 decimal

The AAA instruction updates the AF and the CF, but the OF, PF, SF and ZF are left undefined.

NOTE:- The AAA instruction only works on the AL register

AAS instruction:- ASCII adjust after subtract

The numbers from 0-9 are represented as 30-39 in ASCII code when you count to subtract two decimal digits which are represented in ASCII code, it is necessary to mask, upper nibble (3) from the code before substraction. The 8086 allows you to subtract the ASCII code for two decimal digits without masking

off the '3' in the upper nibble of each digit. The AAS instruction can be used after subtraction to get the current result in unpacked BCD form.

Examples:-

1. AL = 0011 1000 ASCII 8
   CL = 0011 0010 ASCII 2
   SUB AL, CL AL = 0000 0110 BCD 06
   CF = 0
   AAS AL = 0000 0010 = BCD 06
   CF = 0 no borrow required.

AAM instruction:-

ASCII adjust after Multiplication

After the two unpacked BCD digits are Multiplied, the AAM instruction is used to adjust the product to two unpacked BCD digits in AX.

Examples:

AL = 0000 0100 = unpacked BCD 4
CL = 0000 0110 = unpacked BCD 6

MUL CL AL x CL Result in AX.

∴ AX = 0000 0000 0001 1000 = 0018H

AAm AX = 0000 0010 0000 0100 = 0204H

Which is unpacked BCD for 24

Now by adding 3030H in Ax registar we get the result in ASCII form.

AAD instruction:- ASCII adjust before division.

AAD converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. After the division AL will contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder. The PF, SF and ZF are updated. The AF, CF and OF are undefined after AAD.

Example

AAD AX = 0403 unpacked BCD for 43 decimal, CL = 07H

Adjust to binary before division

AX = 002BH = 2BH = 43 decimal

Div CL Divide AX by unpacked BCD in CL

AL = quotient = 06 unpacked BCD

AH = remainder = 01 unpacked BCD

Now by adding 3030H in AX register we get the quotient and remainder in ASCII form.

iii) Logic Instructions:

AND, OR, EXOR, NOT

AND:

AND destination, source.

AND AL, BL {Y = AL · BL}.

AL = 0010 1010

BL = 0010 1010

AL: 0010 1010

OR:

OR destination, source.

OR AL, BL {Y = AL + BL}.

AL = 0010 1010

BL = 1110 1010

AL: 1110 1010

XOR:

XOR destination, source.

XOR AL, BL.

AL = 0110 1010

BL = 0010 1001

AL: 0100 0011

NOT instruction:

NOT destination

NOT AL

AL= 0010 1010

1101 0101

TEST instruction: (AND)

TEST opcode performs AND fn.

TEST destination, source

Ex: TEST AL, BL {Y = AL.BL}

AL= 0010 1010

BL= 0010 1010

AL: 0010 1010

(viii) Shift & rotate Instructions:

① Arithmetic Shift instr's.

② Logical Shift instr's

① Arithmetic Shift Instr:

SAL {Shift left}

SAR {Shift right}

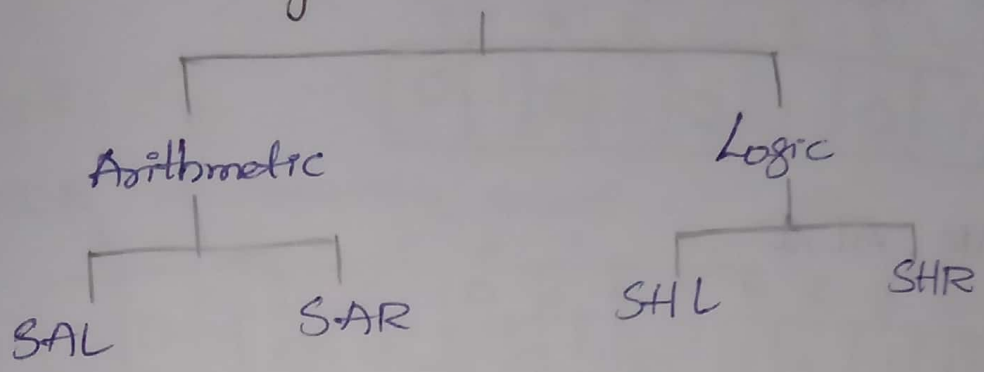
② Logical Shift Instr's:

SHL {Shift left}

SHR {Shift right}.

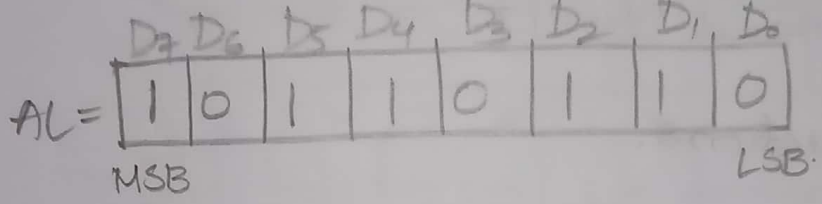
21/1

Shift instructions

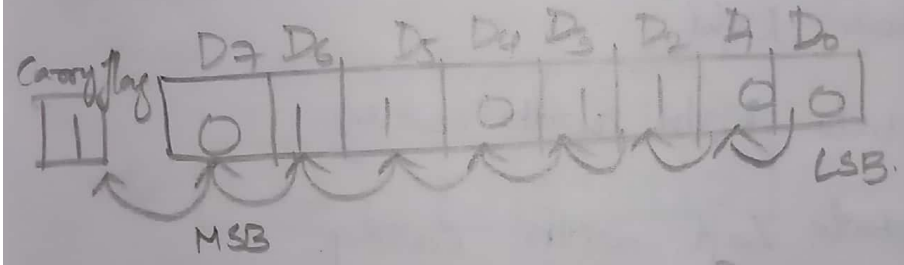


1) SAL / SHL:

SAL/SHL destination, count



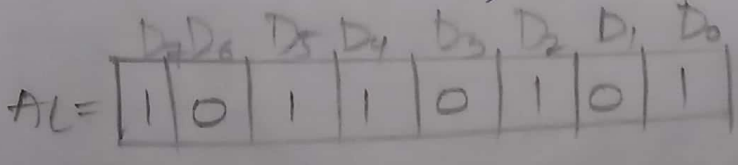

```
SAL/SHL  AL, 1
```



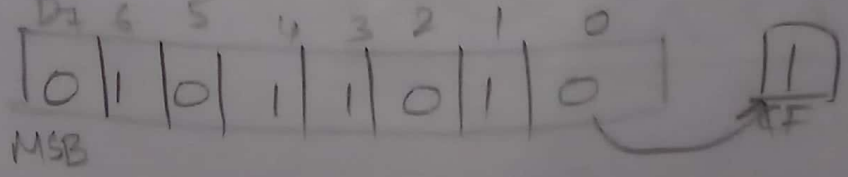
{ MSB is moved to CF
LSB=0 }

2) ~~SAR~~: SHR:

SHR destination, count



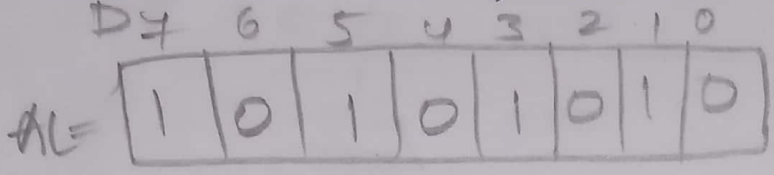
SHR AL, 1



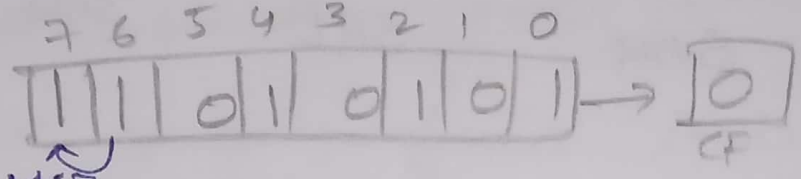
MSB=0, LSB moved to CF.

SAR:

SAR destination, count



SAR AL,1



MSB Stores the previous MSB value.

LSB moved to CF.

Rotate Instructions:

1) ROL → Rotate Left.

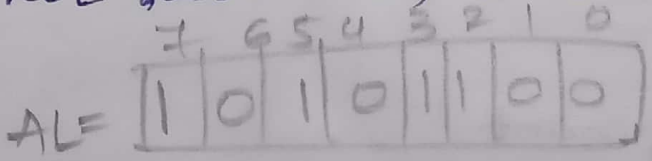
2) ROR → Rotate Right

3) RCR → Rotate Right with carry

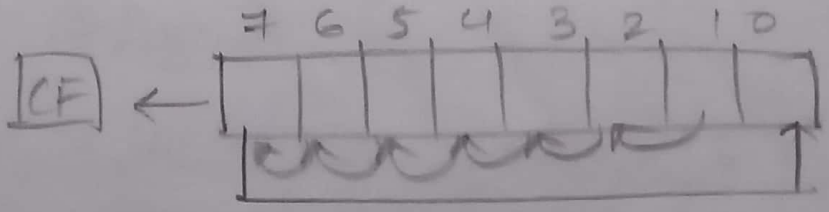
4) RCL → Rotate Left with carry.

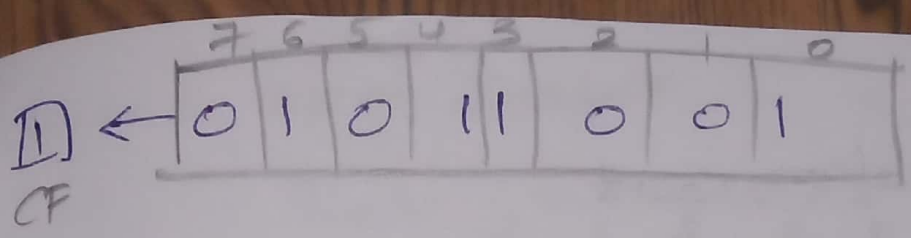
1) ROL:

ROL destination, count



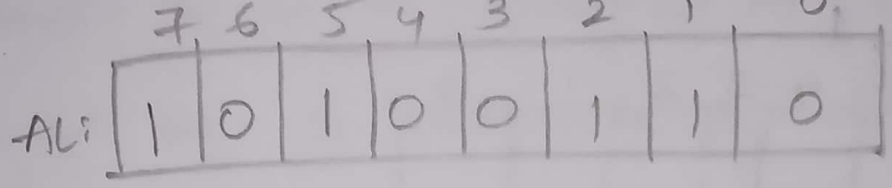
ROL AL,1



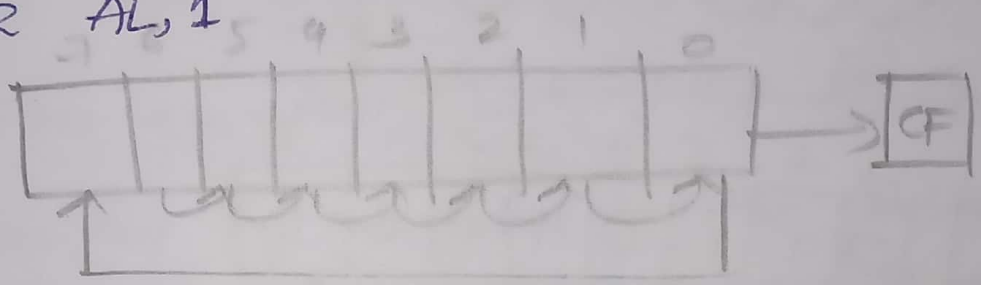


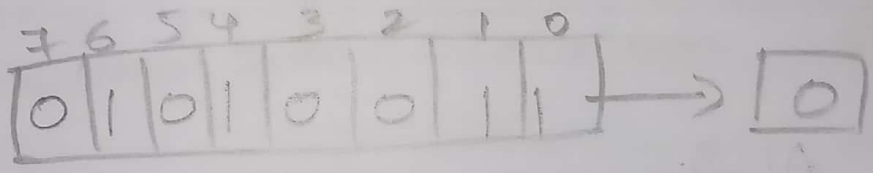
2) ROR:

ROR destination, count



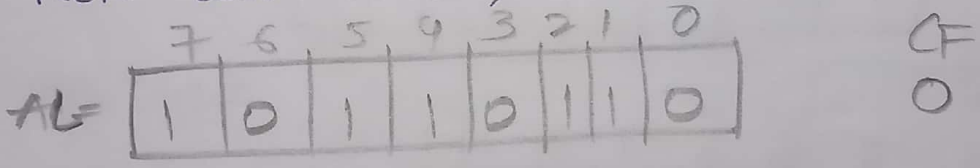
ROR AL, 1



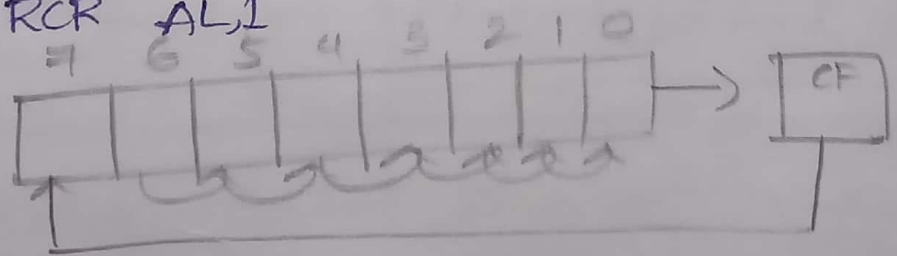


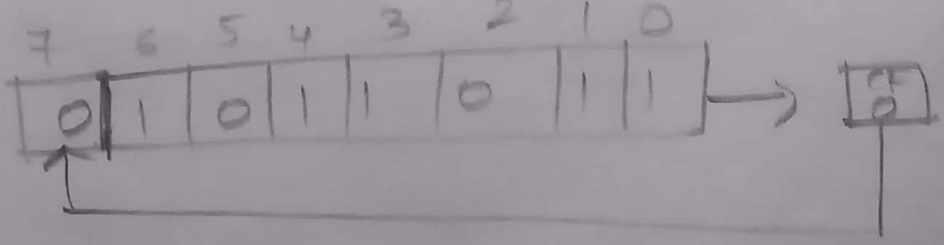
3) RCR:

RCR destination, count



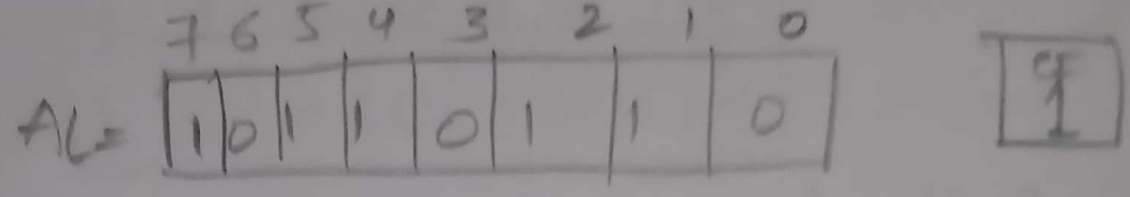
RCR AL, 1



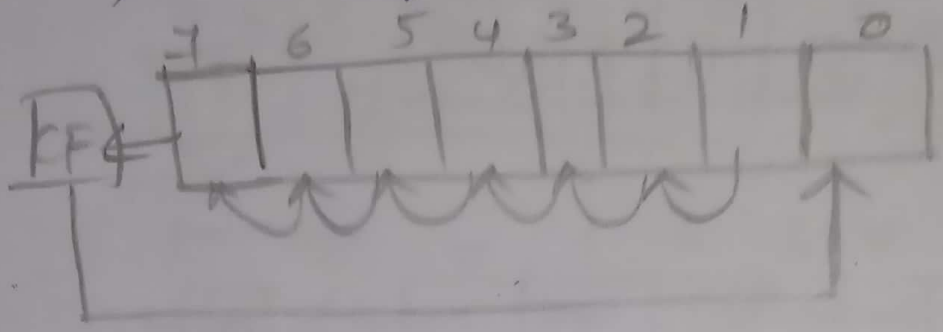


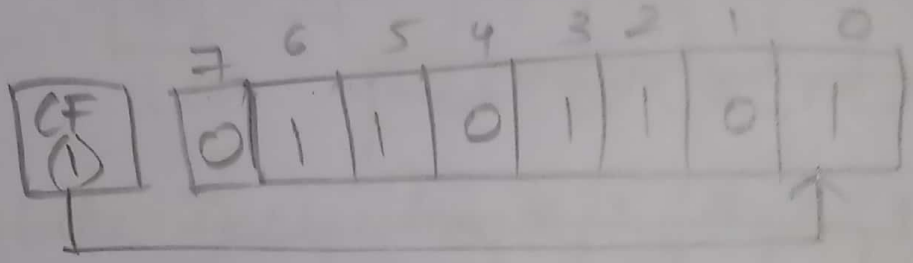
4) RCL:

RCL destination, Count

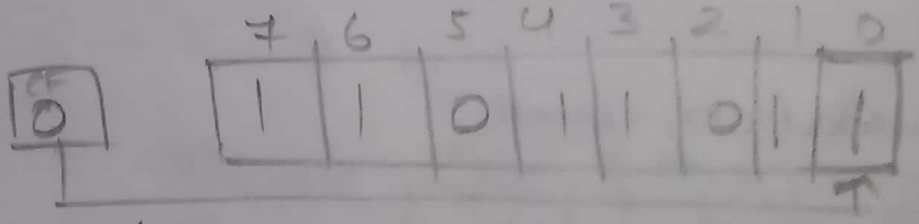


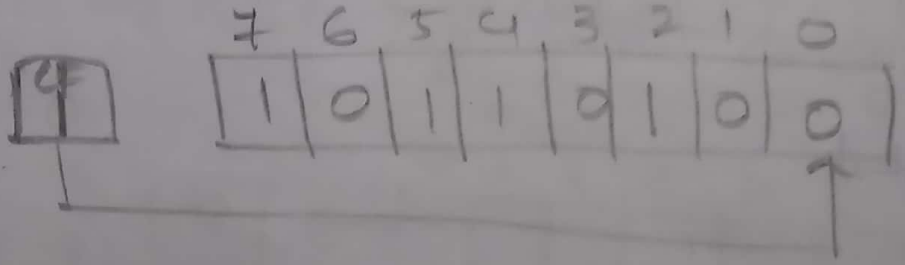
1) RCL AL, 1





2) RCL AL, 3.





23) Iteration Control Loop Instr's:

These instructions are used to execute a series of instr's some no. of times. The no. is specified in the CX register. The CX reg. is automatically decremented by '1', each time after execution of loop instr. Until CX=0, execution will jump to a destination specified by a label in the instr's.

Instruction code	Description	Condition for exist.
Loop.	loop through a sequence of instr.	CX=0
loopE/loopZ	"	CX=0 (or) ZF=0
loopNE/loopNZ	—	CX=0 (or) ZF=1

Processor Control Instr's:

1) STC:
This instr. sets the carry flag.

2) CLC:
This instr. resets the carry flag to zero.

3) CMC:
This instr. gives the compliments of carry flag.

4) Std:

This instr. is used to set the direction flag.

5) Cld:

This instr. is used to reset the direction flag to zero.

6) Sti: This instr. is used to set the interrupt flag.

7) Cli: This instr. is used to reset the interrupt flag to zero.

External H/w Synchronization Instr's:

1) HLT: The HLT instr. is used to 8086 to stop fetching & executing instr's.

2) wait: This instr. executes, the 8086 enters into an ideal condition where it is doing no processing.

3) lock: The lock prefix allows the μprocessor to that another processor doesnot take control of the system bus while it is in the middle of a executing instr's which uses the system bus.

4) nop: At the time of execution of nop instr, no operation is performed except fetch & decode.

5) (escape) esc: When 8086 fetches an esc instruction, the co-processor decodes the instr & carries out the action specified in the instr.

Program control Transfer Instr's:

They are 2 types.

1) Unconditional program C.T.I (CALL, RET, Jmp (without condition)

2) Conditional Program C.T.I (JMP (with condition))

1) Unconditional Program C.T.I:

The procedure is a group of instr's stored as a separate program in the memory & it is called from the main program whenever it requires.

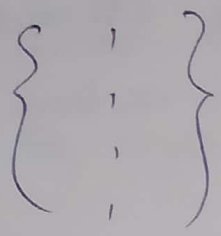
Near Procedure:

Procedure & main program having same code segment

Far procedure:

Procedure & main program having different code segment

EX: Name PROC



Name endp

Call:

The call instr. is used to transfer the execution to a procedure

EX: CALL name of Procedure

Near call:

Procedure & call instr's in same code segment

Far call:

Procedure & call instr's in different code segment

RET:

The RET instr. will return execution from a procedure to the next instr. after the CALL instr. in the calling program.

Jmp:

This instr. will always used to fetch its next instruction from the location specified.

Jmp sqt: It fetches next instr. from address at label sqt.

Conditional Jump: (Short Jmp)
(-128 to 127 bytes).

Instruction code	Description	Condition for jump.
JA/JNBE	Jump if above jump if not below (or) equal.	CF=0 & ZF=0
JAE/JNB	Jump if above (or) equal/ Jump if not below.	CF=0 & ZF=1.
JB/JNAE/JC	Jump if below/jump if not above (or) equal.	CF=1 & ZF=0.
JBE/JNA	Jump if below (or) equal/ Jump if not above.	CF=1 & ZF=1.
JE/JZ	Jump if equal/jump if Zero flag.	ZF=1.
IG/JNLE	Jump if greater/jump if not less than (or) equal.	ZF=0 & CF=0.

JGE/JNL	jump if greater than (or) equal / jump if not less than	SF=0.
JL/JNGE	jump if less than / jump if not greater than (or) equal.	SF≠0
JLE/JNG	jump if less than (or) equal / jump if not greater	ZF=1 & SF≠0 CF=0
JNE/JNZ	jump if not equal / jump if not zero	ZF=0
JNO	jump no overflow	OF=0
JNP/JPO	jump if not parity / jump if parity odd	PF=0
JNS	jump if not sign (or) jump if +ve	SF=0
JO	jump if overflow flag=1	OF=1
JP/JPE	jump if parity / jump if parity even	PF=1
JS	jump if sign flag=1 (or) jump if -ve.	SF=1

JCXZ	Jump if CX is Zero.	CX=0

Interrupt Instr's:

1) Int type: (0-255)

This instr. is used to far procedure.

The term type in the instr. refers to a no. b/w 0 to 255

2) INTO:

If the overflow flag is set INTO is used to handle overflow condition.

3) IRET:

IRET instr. is used at the end of the interrupt service routine to return execution to the interrupted program.

CBW (Convert signed byte to signed word):

AL {AL to AX}.

AX = AL + AH

AL = 1001 1100.

The sign bit of AL (MSB) moved to AH.

AX = 1111 1111 1001 1100.

AH AL

CWD (Convert Signed word to Signed Double word)

AX = 1100 1110 1010 1111 [(MSB) sign bit of AX reg. will be moved to DX].

DX = 1111 1111 1111 1111.

Assembler Directives:

These instrs are given to the assembler, linker, debugger.

1) ASSUME:

Operation: The 8086 at any time can directly address 4 physical segments (this is used to initialize the segment reg's), which includes code segment, data seg, stack seg. & extra segment.

Format: assume cs:code, ds:data, es:extra, ss:stack

2) .code:

Operation: This directive provides shortcut in definition of code segment.

Format: .code [name]

.code [CS1]

It is used more multiple code seg's in one prog.

3) .data:

Operation: Shortcut in definition of the data segment.

Format: DB → Define Byte

DW → Define Word

DQ → Define Quadword.

DT → Define Tenbytes

DD → Define Double word.

Format: Amount DB 10H, 20H, 30H, 40H

1 2 3 4

[Declare array of 4 bytes named amount]

4) DUP:

Operation: This is used to initialize several locations & to assign values to these locations.

Format: name data_type num DUP(value)

Table DB 8 DUP(5555H)

5) END:

This is used to end the program.)

Op: It is return at the last statement of the program.

Fo: END

6) ENDM:

Op: The directive informs the assembler the end of the macro.

Fo: ENDM.

7) ENDP:

Op: The directive informs the assembler the end of the procedure.

Fo: Name of procedure ENDP

8) ENDS:

Op: end of the Segment

Fo: Segment name ENDS.

24)

9) EQU:

Operation: The directive informs the assembler to equate the variable name with another variablename, (or) immediate data (or) expressions.

Format: variablename EQU 8000H

PORT EQU 8000H

count EQU $\left(\frac{\text{array2} - \text{array1}}{2}\right)$

String variablename EQU 'String'

Soft EQU 'Hyderabad'

{Soft is equal to ASCII value of chars}

10) Even:

Op: The directive informs the assembler to increase the location counter to next given memory address, if it is not in even address

Format: EVEN

Ex: DATA Segment

Num DB 55H (0 location)

LIST DW 50 DUP(0) (1 location)

The list location will be 1. This is an odd address. So the reading of next 50 words location will be slow. It requires two bus cycles.

~~11) EXTRN:~~

DATA Segment

Num DB 55H {0 location}

LIST DW 50 DUP(0) {1' location}

EVEN

even address → LIST DW 50 DUP(0) {2 location}
location

The LIST location will be 2. It requires only one bus cycle.

11) EXTRN:

Operation: The directive is used to access the variables & data from another program modules.

Format: for data segment

EXTRN variable 1: datatypes 1, variable 2: datatypes 2....

for code segment

EXTRN address 1: Near (or) far, address 2: Near (or) far.

12) LABEL:

Operation: It is used to assign the name for the particular location counter.

Format: Labelname LABEL labeltype

Soft LABEL word.

13) OFFSET:

Operation: This loads the offset address of a variable into a register.

Format: offset variablename.

Ex: mov ax, offset array.

14) PROC:

Operation: It informs the assembler to start procedure

Format: PROCname PROC near/far

15) Pointer:

Operation: This directive is used to specify the type of memory address.

Format: Datatype PTR.

16) Segment:

Operation: It is used to start of logical Segment.

Format: Segmentname Segment.

17) MACRO:

Operation: The directive indicates the start of MACRO.

Format: macroname MACRO (Arg1 .... Arg N).

18) Originate:

Operation: It is used to assign a numerical value to the particular location counter.

Format: org numericvalue

EX: org 80

19) PUBLIC:

Operation: This dir. is used to access variables & address from other programming modules.

Format: PUBLIC data segment Variable1 . . . . Variable N

for CS: PUBLIC address1 . . . . address N.

20) ALIGN:

Operation: This directive forces the assembler to align the next segment at an address divisible by specified divisor.

Format: ALIGN number
↓ {divisor}
{2,4,6,8 }

Procedures:

Procedures are 4 types.

1) Single Procedure (it consists only 1 subprogram).
2) Nested " [called nesting: one procedure calling another proc is]
3) Re-entrant "
4) Recursive "

2) Nested Proc:

```
PROC 1
  {
    Proc 2
      {
        Proc 3
          }
      }
  }
```

3) Reentrant PROC:

During this proc, interrupt occurs & program execution is transferred to interrupt service routine.

4) Recursive PROC:

It is a proc, which calls itself

Macros:

Macro is a group of instr.'s & it is stored in particular memory location. This code will be in proper sequence & it no needs of using call & ret instr.'s.

Format: Macroname MACRO

DISP MACRO.

Procedure	Macro.
i) It takes less memory	i) It takes more memory.
ii) It's execution time is more	ii) It's ex. time is less.
iii) It is defined using PROC & ENDP directives.	iii) Using MACRO & ENDM directives.
iv) It is accessed by CALL & RET instr.'s.	iv) It is accessed by its name given to macro when defined

during Program assembling.

v) It will make use of stack.

v) It will not make use of stack

vi) It is used when repetitive instr's are more.

vi) It is used when repetitive instr's are less.

vii) Program flow branches to procedure & comes back to main program.

vii) Prog. flow doesnot branch as macro statement is replaced by its group of instr's. Prog. flow will be sequential.

27/1/20

Unit-III

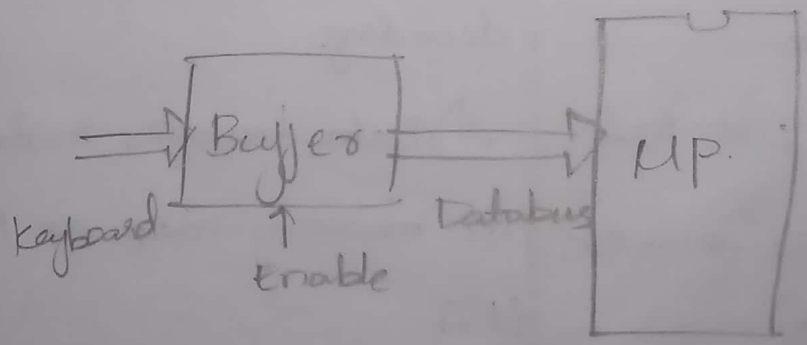
I/O Interface

Introduction:

To execute the instr's, we have to read the data from the i/p devices & the executed data will be applied to the o/p devices. The data transfer b/w the µP is done by using I/O ports.

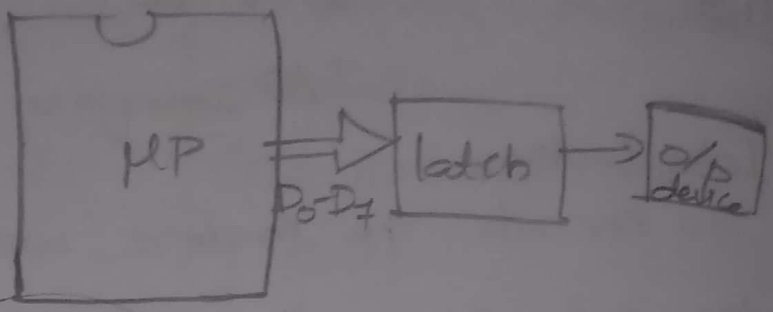
Input Port:

The i/p devices cannot directly connected to the µP, these devices are connected to the µP's by using buffer. The buffer provides some data $D_0 - D_7$. This data is connected to the µP through the data bus.



Output Port:

The o/p devices cannot directly connected to the µP, ~~these~~ µP's ~~devices~~ are connected to ~~the~~ µP ~~these~~ by using latch.



Interfacing is of two types.

(1) I/O mapped i/o interfacing (the i/p is from keyboard, mouse, scanner)

(2) memory mapped i/o interfacing (the i/p is from RAM, ROM)

I/O mapped i/o interfacing	Memory mapped i/o interfacing
i) I/O device is interfacing device & addressed depends upon requirement.	i) In memory mapped scheme device has memory locations & all address lines.
ii) All the available address lines are not used for interfacing device.	ii) All the available address lines used for address decoding.
iii) I/O mapped use in & out instr's for access.	iii) Data transfer instr's in memory map are MOV, LEA
iv) I/O mapped use less H/w for decoding & less no. of address lines.	iv) Memory mapped use complex H/w for decoding.
v) In this we require $\overline{IORD}$ and $\overline{IOWR}$ signals.	v) In this we will use $\overline{MRDC}$ and $\overline{MWDC}$.

vi) $\overline{IO}$ is selected. I_0

this $M=0, \overline{IO}=0$

vi) $M=1, \overline{IO}=1$.

31/1.

8255 (PPI)

Programmable Peripheral Interface:

8255 is 40 pin IC. It has 24 P/p-O/p pins which can be grouped into 3 ports: Port A, Port B, Port C. The 8 bits of port C, can be divided into 4

8255 can be programmed into 2 modes.

1) Bit set/Reset mode
2) I/p-O/p mode.

The i/p-o/p modes divided into 3 modes.

(i) Mode Zero - Simple i/p-o/p.
(ii) Mode one - i/p-o/p with handshaking
(iii) Mode two - bidirectional i/p-o/p data transfer

Features of 8255:

1) The 8255 is a widely used interface device It is compatable with Intel µProcessor.
2) Each port has unique address & data can be read (or) written to a port
3) In mode Zero, port-A & port-B can be used as 8-bit i/p (or) o/p ports without handshaking.
4) The upper port bits of port-C can be added to the port-A & lower bits of port-C can be added to the port-B.

5) In mode-1, port-A & port-B can be act as either i/p (or) o/p & 3 lines of port-C in each group used for handshaking.

6) In mode-2, only port-A can be used as bidirectional port & handshaking signals are provided with 5 lines of port-C. (PC-3 to PC-7).

The 8255 capacity is 2.5mA & 5V.

8255 Pin Description:
(2-6 points)

$\overline{CS}$ - chip selection.

This is an active low which can be enable for data transfer operation blw CPU & 8255.

$\overline{WR}$

When this i/p pin is low, the CPU can write the data on the ports (or) in the control reg. through the data bus buffer.

$\overline{RD}$:

When this pin is low, the CPU can read the data from the ports.

$D_0 - D_7$:

This bidirectional tristate (i/p, o/p, selection line) data bus lines are connected to the system bus, these are used to transfer the data from Microprocessor to 8255 & receive the data from

8255 to 8086.

Reset:

This is an active high i/p used to reset 8255. When reset i/p is high, the control reg. is cleared all the ports & these are set to i/p mode.

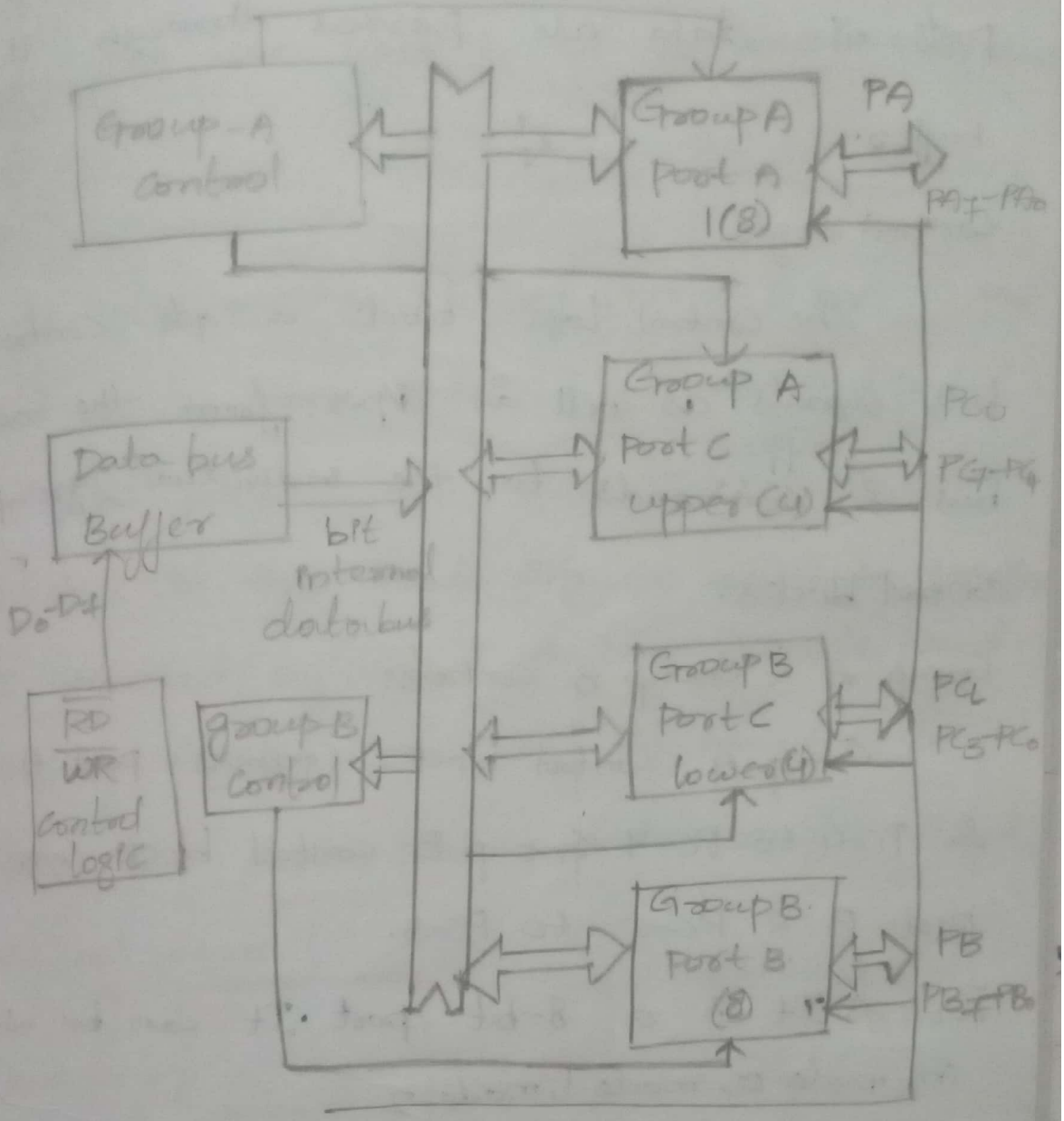
A_0 & A_1:

A_0 & A_1 represents the status of read & write operations.

A_1	A_0	$\overline{WR}$	$\overline{CS}$	$\overline{RD}$	Operation (Read)
0	0	1	0	0	Port A to Data bus buffer
0	1	1	0	0	Port B to Data bus buffer
1	0	1	0	0	Port C to Data bus buffer

A_1	A_0	$\overline{WR}$	$\overline{RD}$	$\overline{CS}$	Operation (write)
0	0	0	1	0	Data bus buffer to port A
0	1	0	1	0	" " to port B
1	0	0	1	0	Data bus buffer to port C
1	1	0	1	0	the data will be moved to control logic

Functional Architecture of 8255:



Data bus Buffer:

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

→ The i/p (or) o/p. instr. is executed by the CPU either read data from buffer (or) write the data into the buffer.

O/p data from the CPU to the ports are control reg. & P/p data to the CPU from the ports all data are passed through the buffer.

Control Logic:

The control logic block accepts control bus signals as well as i/p's from the address bus & commands to the individual group control blocks.

Group A & Group B controls:

Group A control block consists port A & PC-4 to PC-7. Group B control block consists port B & PC-0 to PC-3.

Port A: It is a 8-bit port. It can be used in mode-0, mode-1, mode-2.

Port B: It is a 8bit port. It is used in mode-0 & mode-1.

Port C: It is a 8-bit port. It is divided into 2 ports Upper - PC4 - PC7 & Lower - PC0 - PC3.

17/2/2020

Modes of Operation:

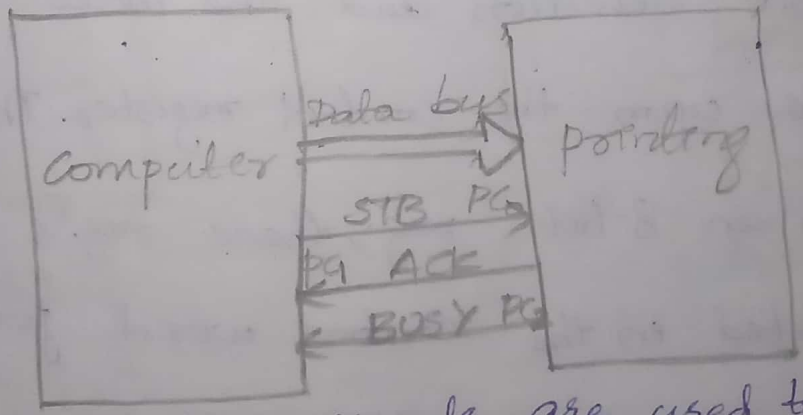
1) Bitset-Reset Mode (BSR)(PC0 to PC7):-

Port-C bits are used to set (or) reset individual bits.

2) I/p-O/p mode:

Mode-0/Simple I/O: In this port-A & port-B are used as 2 simple 8-bit I/O ports and port-C as 4-bit ports.

Mode-1: In this mode I/p (or) o/p data transfer is controlled by hand shaking signals. Hand shaking signals are used to transfer the data b/w devices. Whose data transfer speed are not same.



Ex: These handshaking signals are used to tell computer whether printer is ready to accept the data (or) not. If printer is ready to accept the data then after sending data on databus

Computer uses handshaking signal ($\overline{STB}$) to tell pointer that valid data is available on data bus.

Handshaking Signals Features:

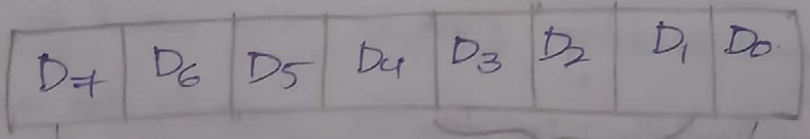
1) 2 ports A & B fn's as 8-bit I/O ports
2) Each port uses 3 lines from port C as handshaking signals.
3) I/p & O/p data are latche

Mode-2 / Bidirectional I:

This mode uses only port-A as the data bus PC3-PC7 are used for handshaking purpose.

Control words for BSR Format:

The bit selection and the mode selection is depends upon the control register. The control reg. is an 8-bit reg; these reg.'s 8-bit is represented in the control word format.



D7=0 then BSR mode

000 - PC0
001 - PC1
010 - PC2
011 - PC3
100 - PC4
101 - PC5
110 - PC6

D0=0 bit is reset

D0=1 bit is set

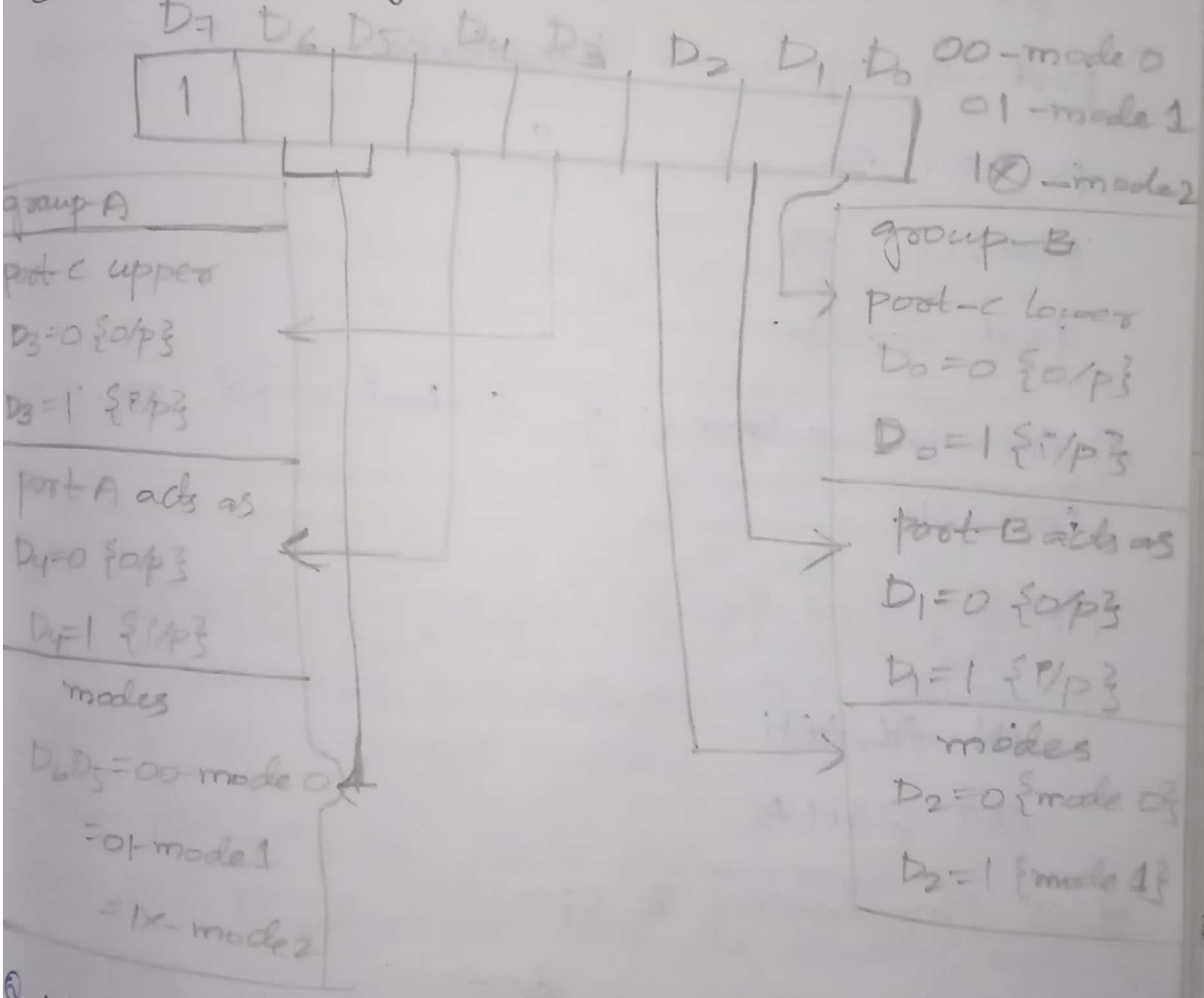
→ D_6, D_5, D_4 is used to represent the mode selection.

→ D_1, D_2, D_3 is used to bit selection line is used in port-C.

Ex: Control word format for set the PC_3 bit

0XX0111.

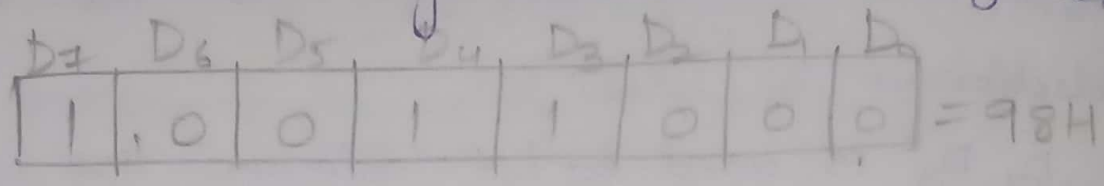
Control word format for I/O mode:



Q. Write a program to initialize 8255 in the configuration given below

a) port-A: Simple i/p
b) port-B: Simple o/p
c) port-C_L: o/p
d) port-C_U: i/p.

A. assume address of control word register is 83H.



IN AL, 98H

OUT 83H, AL

Q. Write a program to initialize 8255 by configuration given below

port-A: O/p with handshake

port-B: i/p " "

port-C_L: O/p

port-C_U: i/p.

A. Assume address of control word reg. of 8255 is 23H.

IN AL, AEH

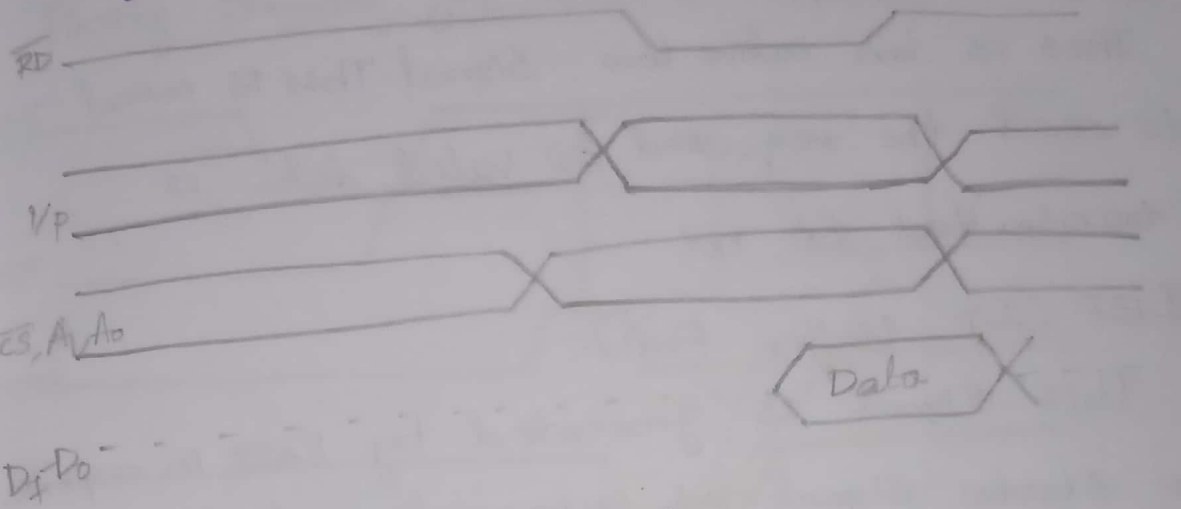
OUT 23H, AL

8255 Programming & operation:

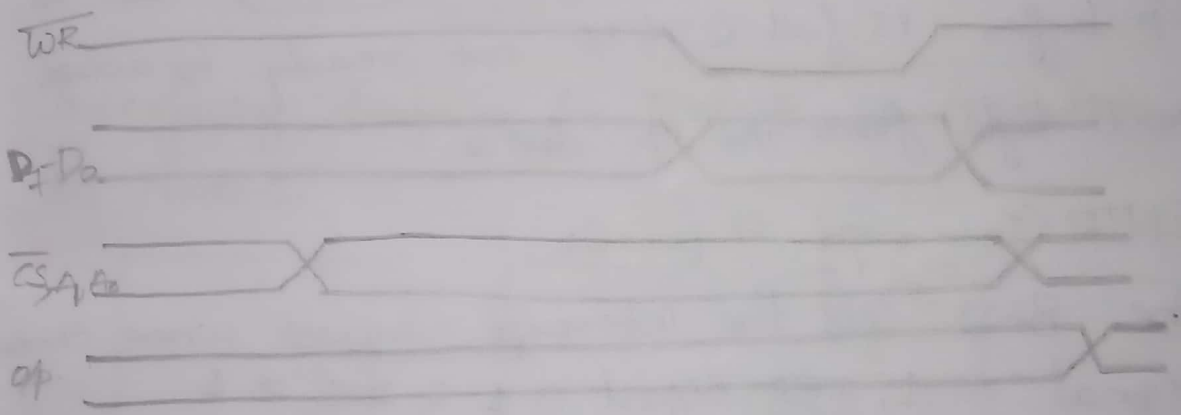
Programming in mode 0:

To operate 8255 in mode-0 set D_5, D_6 & D_2=0

Timing Diagram - i/p mode 0:



Timing diagram - o/p mode 0:



25/2/2020

Programming in mode 1 (I/O with handshaking):-

- port-A and port-B can acts as either I/p or o/p in mode 1 with 3 control signals.
- When port-A & B to be programmed as an i/p port PC_3, PC_4, PC_5 are used for control and PC_6 & PC_7 are used for o/p (or) i/p.
- When port-A is programmed as an o/p port PC_3, PC_6 & PC_7 are used as control pins & PC_4, PC_5 can be i/p (or) o/p.
- When port-B is to be programmed as an i/p (or) o/p port PC_0, PC_1, PC_2 are used for control.

Input Control Signals.

$\overline{STB}$ (Strobe i/p):

This is an active low signal. This is used to check the required (or) valid data is transmitted (or) not.

IBF (I/p Buffer Full):

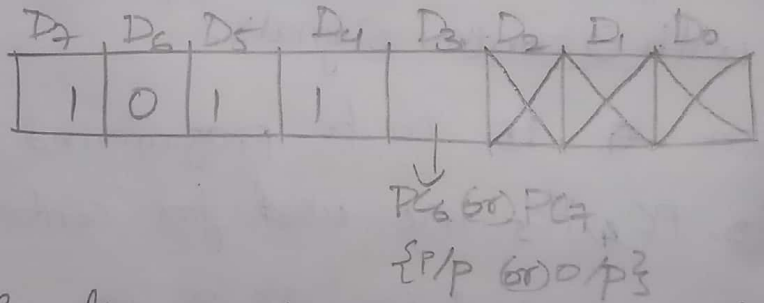
This signal is generated by 8255 in response to strobe signal as acknowledgement to i/p device. It also indicates to the i/p device that the i/p buffer is full & it is not ready to accept next byte from the i/p device.

INTR (Interrupt Request):

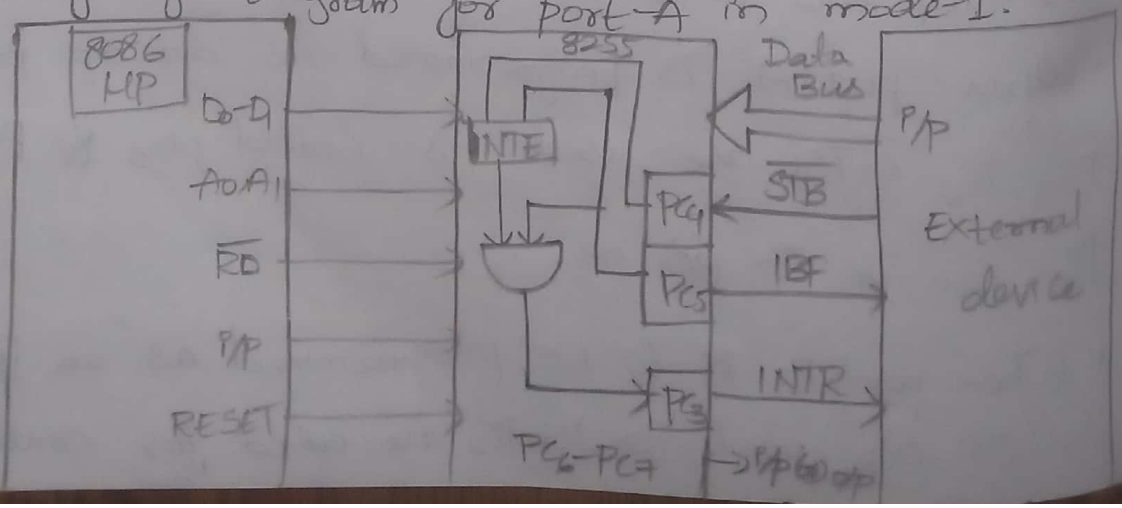
The 8255 sets the interrupt request when Strobe signal is 1, IBF signal is 1 & INTE is 1.

INTE is enable when $\overline{STB}$ = 1 & IBF = 1.

Control Word Format: Port-A in mode-1

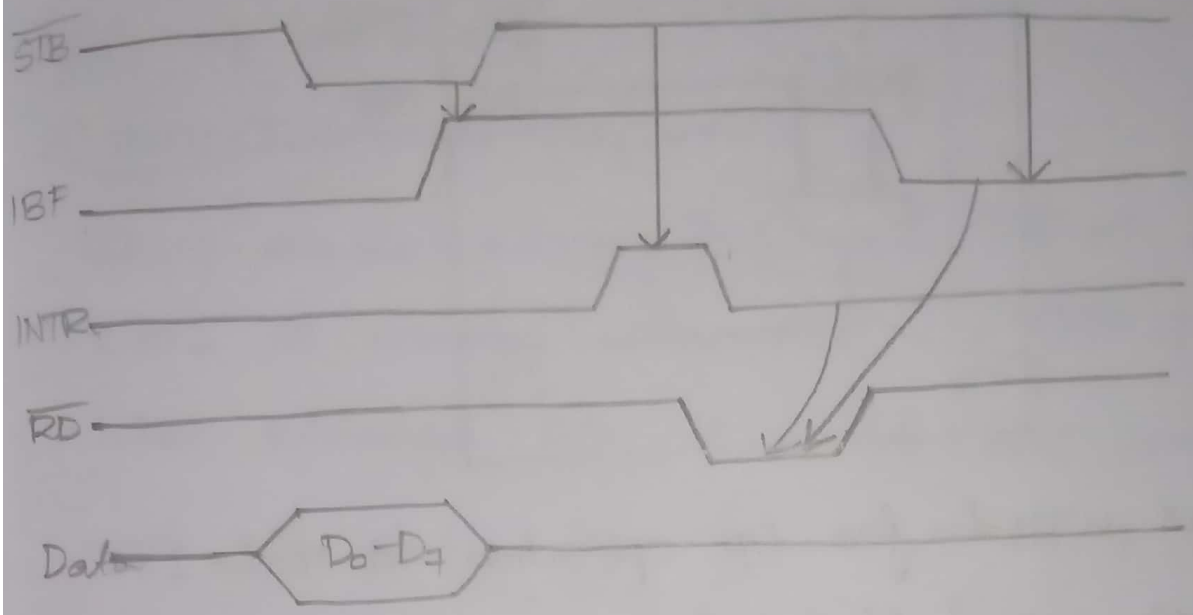


Interfacing diagram for port-A in mode-1:

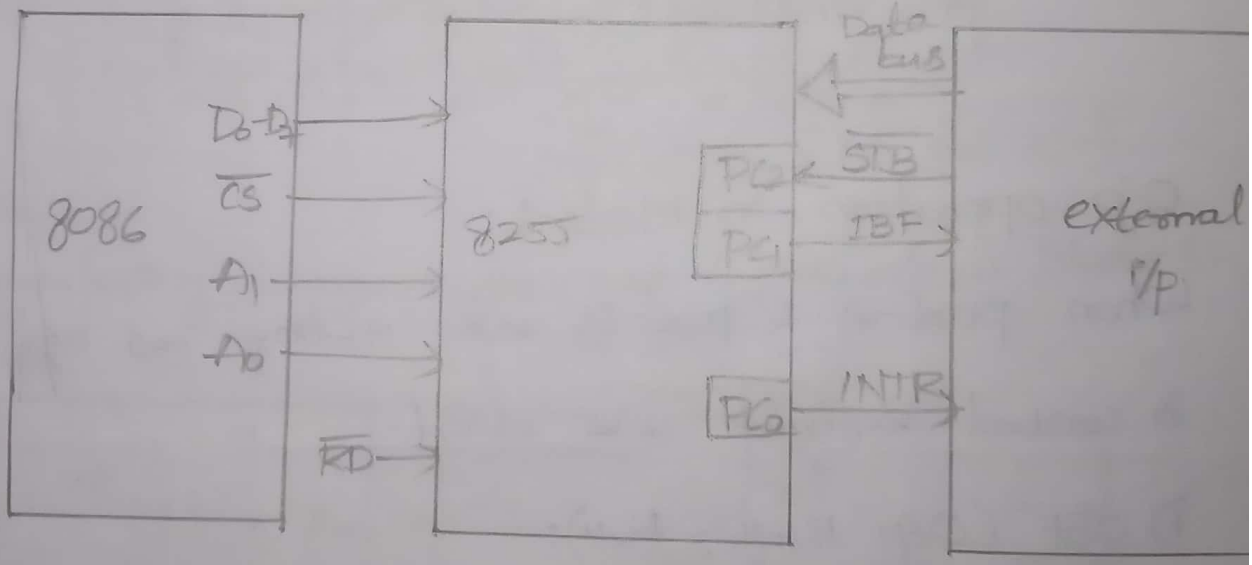


26/2/2020

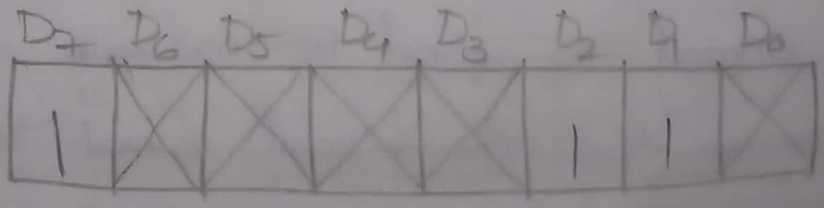
Timing Diagram for Port-A Mode-1:



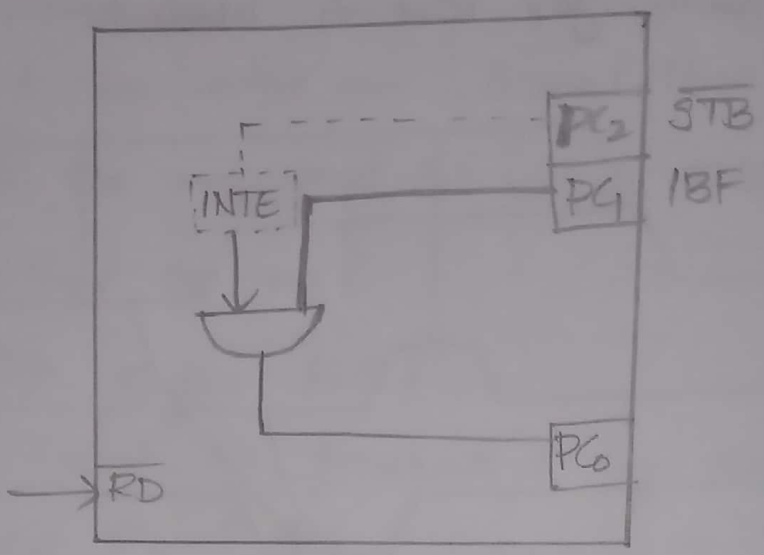
Interfacing diagram for Port-B i/p operation in Mode-1:



Control word format for port-B input in mode-1:



Schematic Diagram:



Status word for i/p operation in Mode-1:

Status word is used to represent the fn. of port-C bits in mode-1

S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
I/O	I/O	IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0

O/p operation in Mode-1:

When port-A & port-B are acting as o/p, 3 control signals are used.

1) $\overline{OBF}$ (O/p Buffer Full):

O/p buffer full is an active low signal. This signal is used to indicate when the o/p device the o/p signal is not ready for transmission. I/p signal to o/p device.

2) $\overline{ACK}$ (Acknowledge i/p):

This is an active low signal & o/p signal for o/p device.

3) INTR (Interrupt Request):

- It is enable when $\overline{ACK}$ =1 & $\overline{OBF}$ =1 & INTE=1
- INTE is enable when $\overline{ACK}$ =1 & $\overline{OBF}$ =1.
- When interrupt pin is enable, we can't write the data into the o/p device.

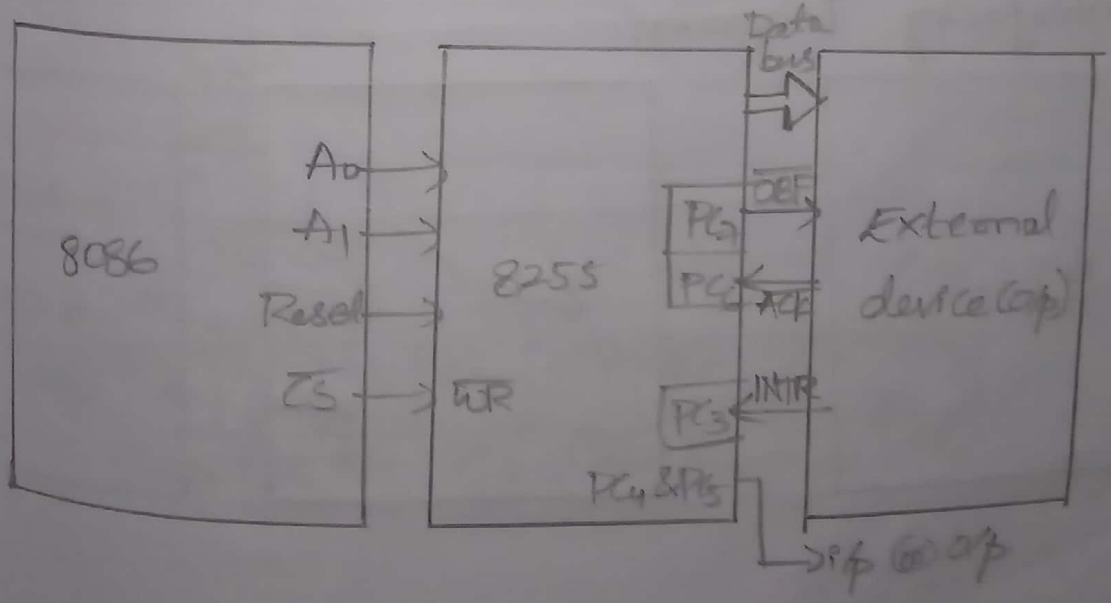
Control Word Format for Port-A as o/p in mode-1:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	I/O	X	X	X

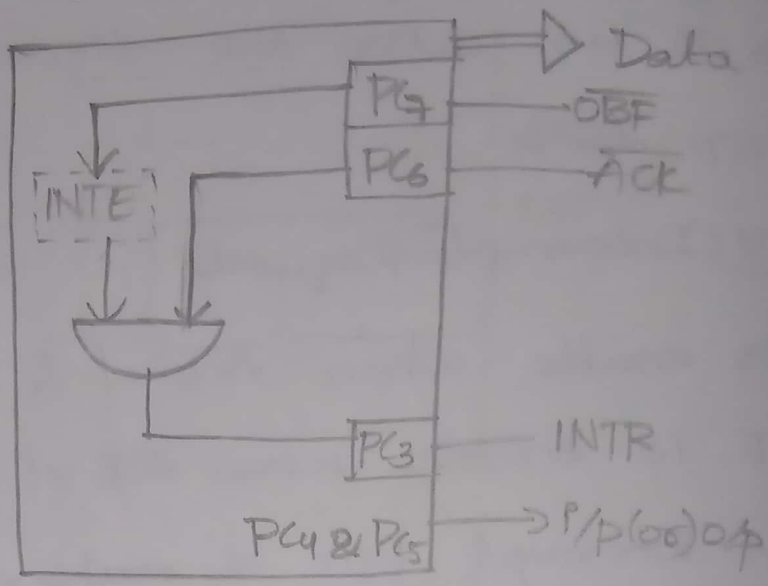
↓

PC_4 (or) PC_5 {i/p (or) o/p}.

Interfacing Diagram for port-A as o/p in Mode-1:



Schematic Diagram:



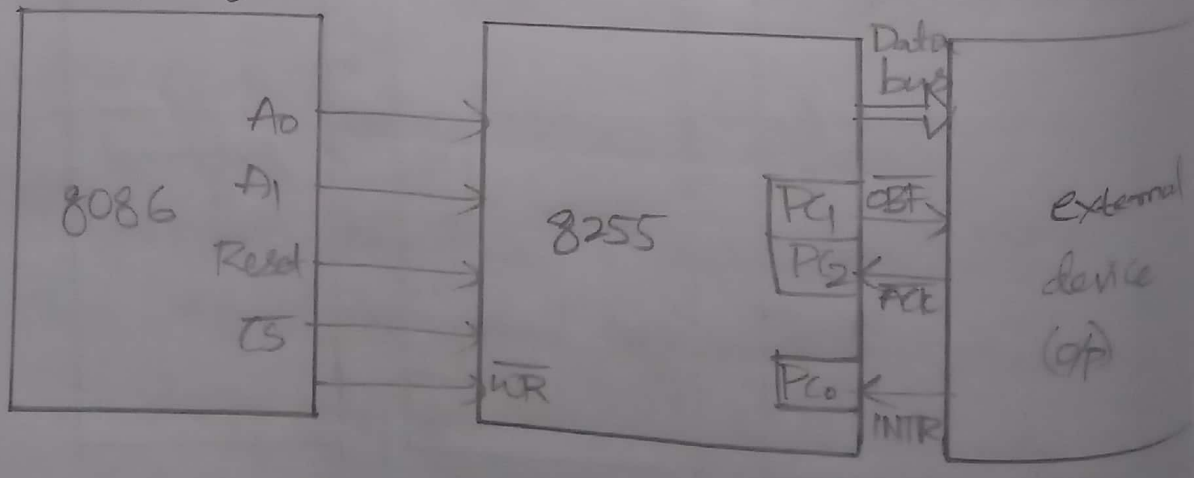
Port-B as o/p in Mode-1:

3 control signals are 1) PC0 / interrupt request
2) PC1 / $\overline{OBF}$
3) PC2 / $\overline{ACK}$

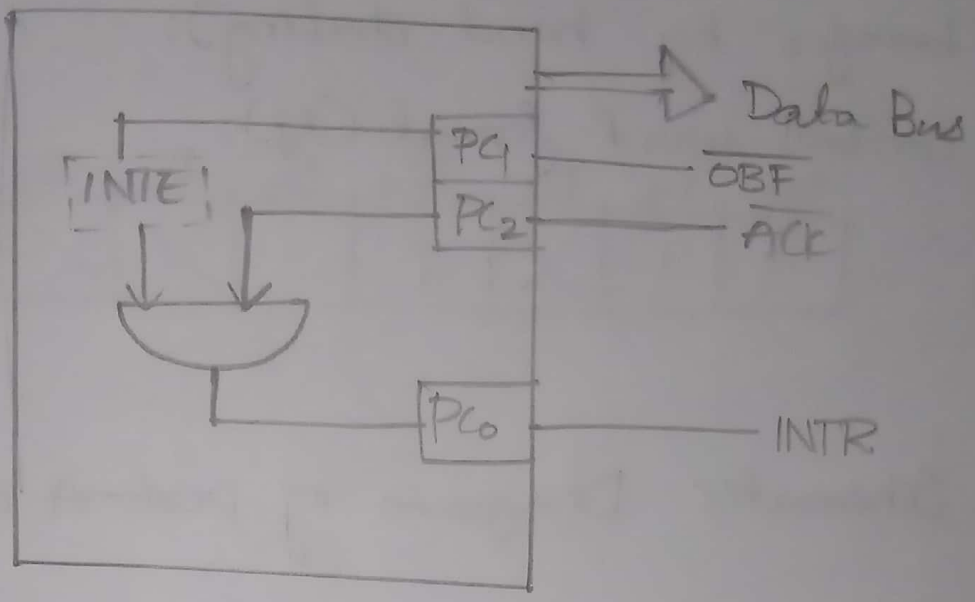
control word:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	X	X	X	X	1	0	X

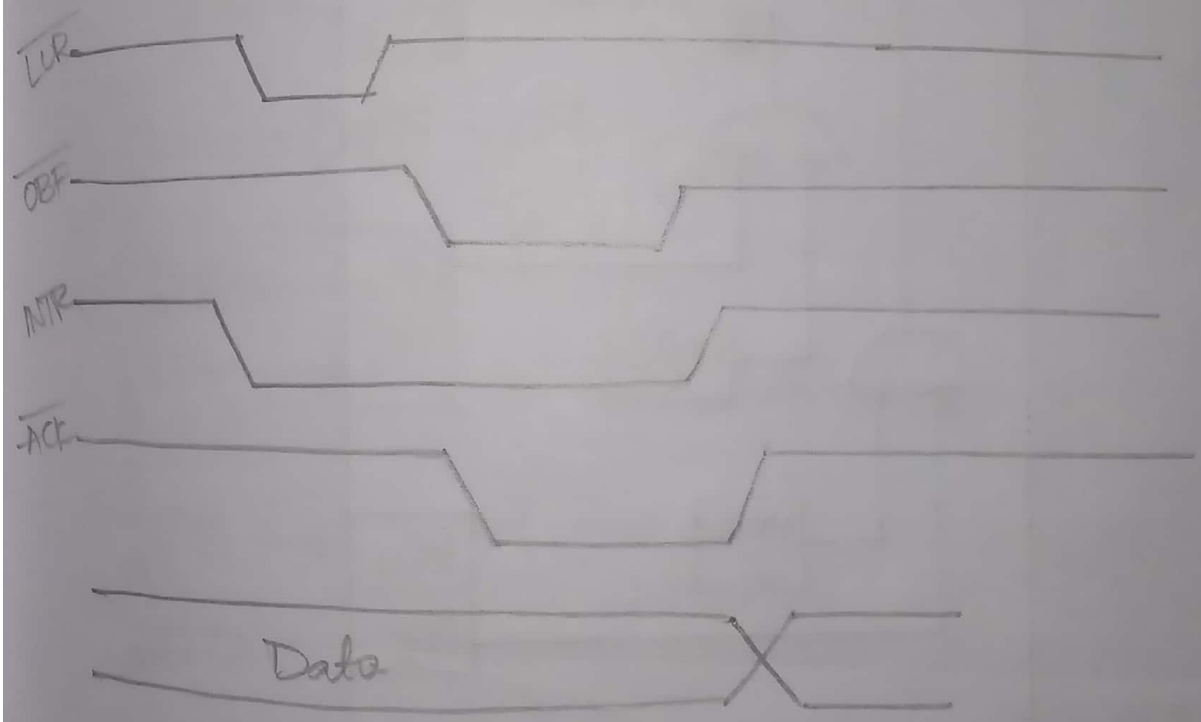
Interfacing Diagram:



Schematic Diagram:



Timing Diagram for o/p:

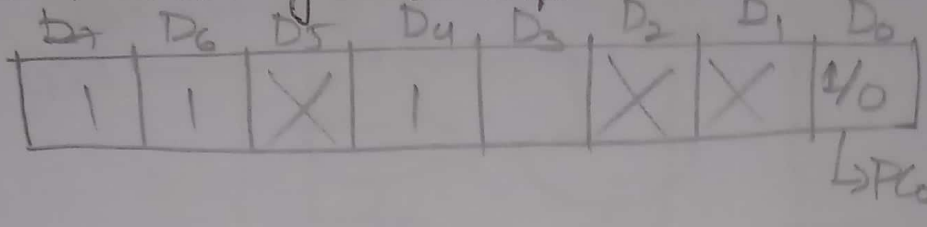


Status word:

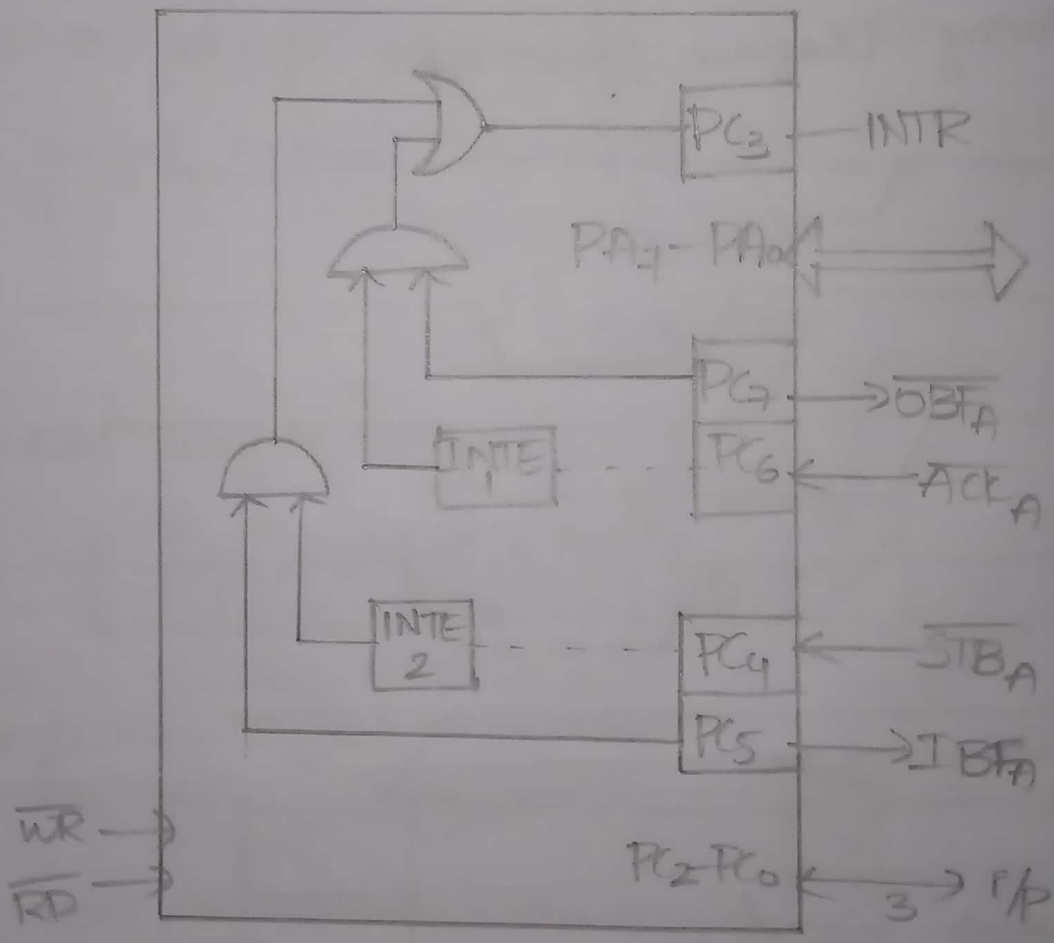
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
$\overline{OBF}_A$	$\overline{ACK}_A$	I/O	I/O	$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0

Programming in Mode-2 (Bidirectional data transfer by hand shaking):

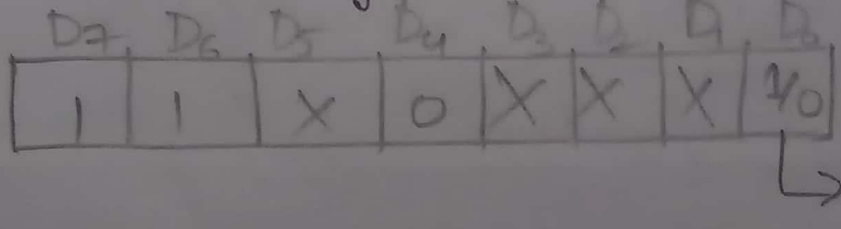
Control word format (i/p):



Schematic Diagram of port-A in Mode-2:



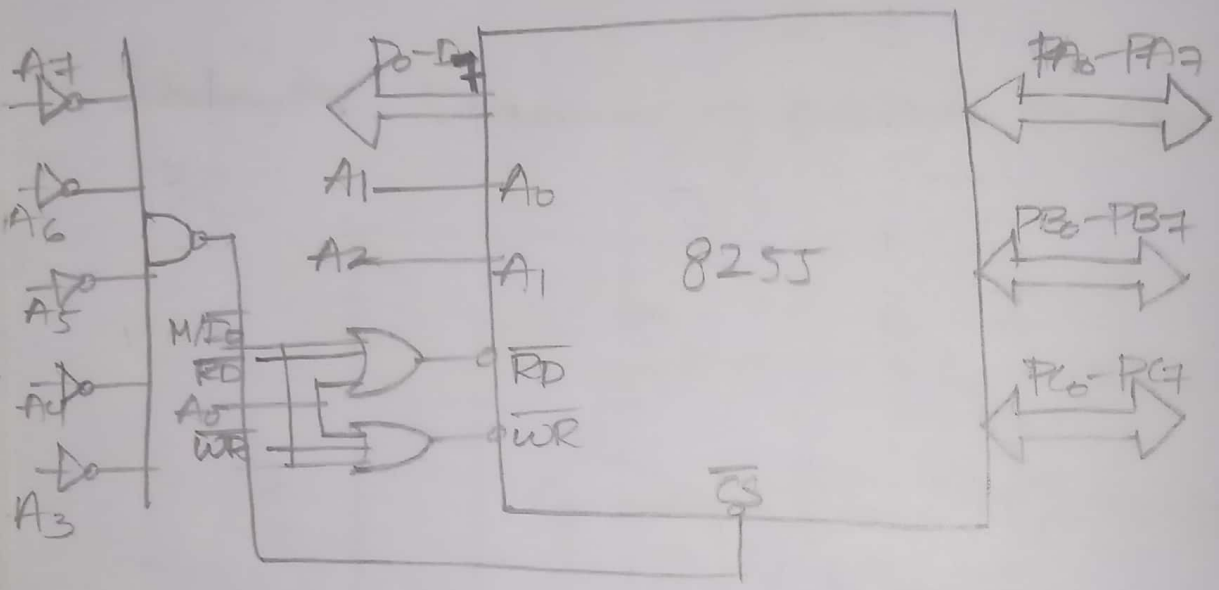
Control word format (o/p):



27/12 Status word for mode 2:

S7	S6	S5	S4	S3	S2	S1	S0
$\overline{OBF}$	$\overline{ACK}$	IBF	$\overline{STB}$	INTR	I/O	I/O	I/O
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC_0

Interfacing 8255 to 8086 in i/o mapped:

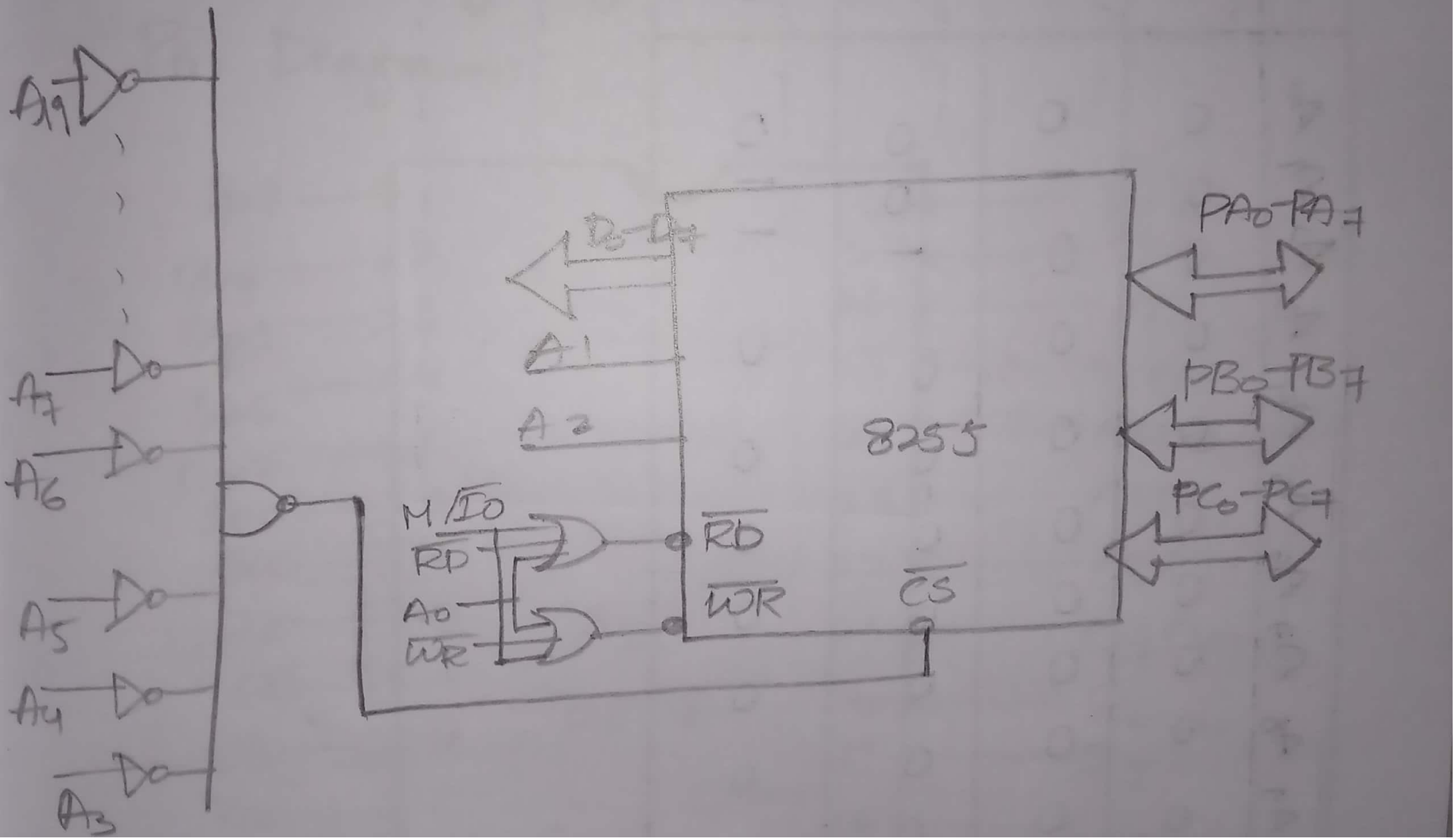


Interfacing Table:

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

Interfacing 8255 to 8086 in Memory mapped:

(20 address locations)



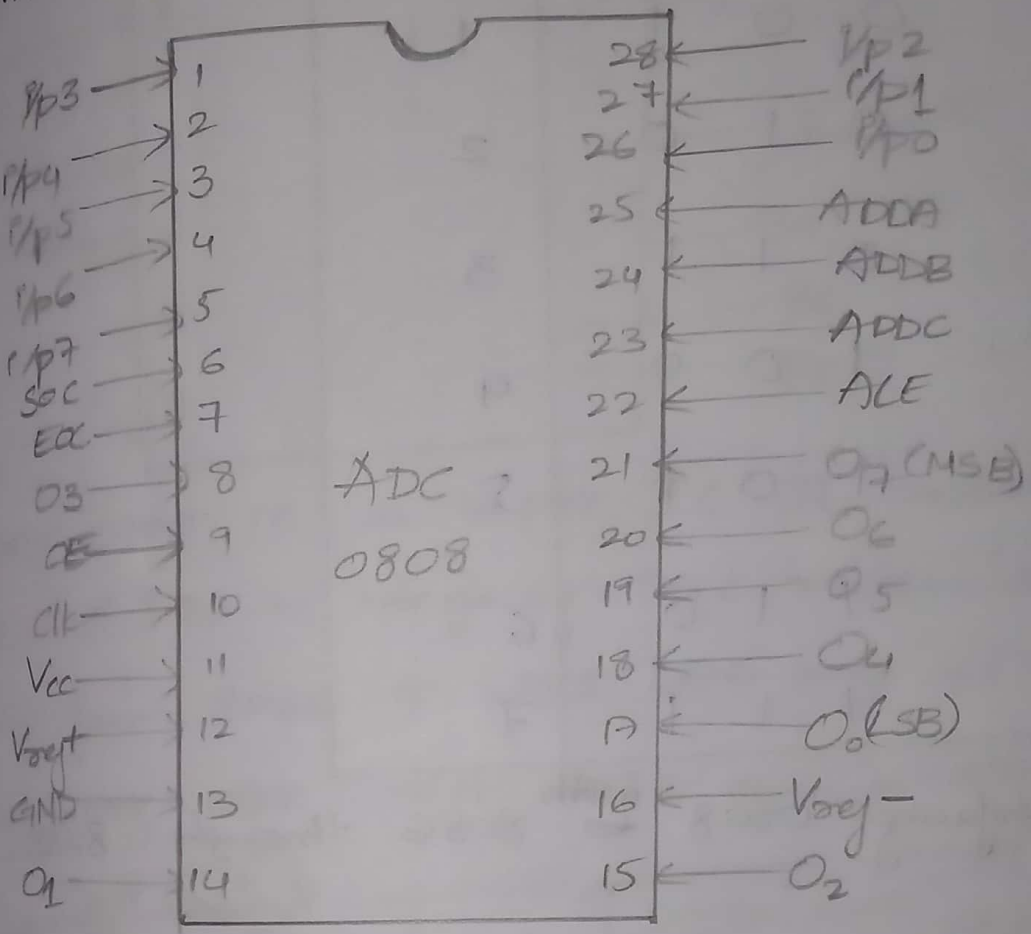
Interfacing Table:

Register	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Port A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
Port B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002H
Port C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0004H
Control Register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0006H

Analog and Digital Converter Interfacing:

Analog I/P ADC Interfacing (0808):

Pin Diagram:



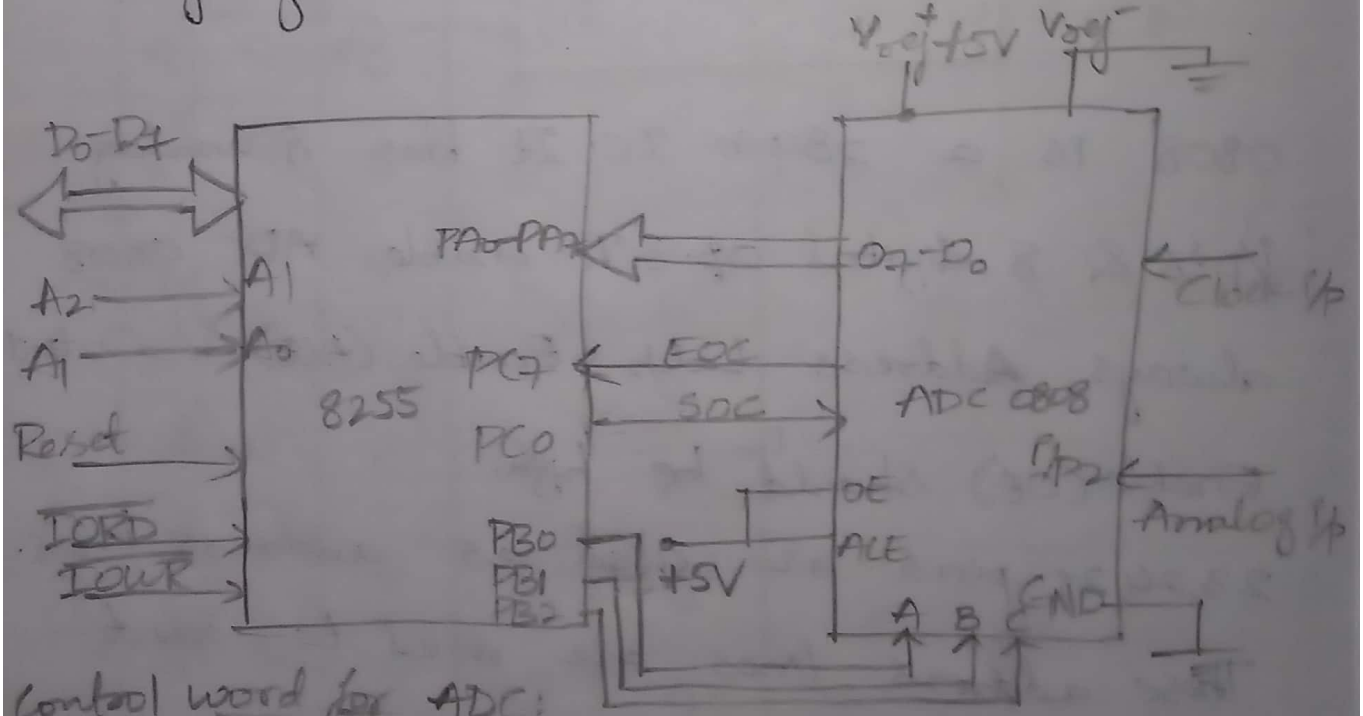
0808 is a 28-pin IC. It has 8 analog i/p's & 8 digital o/p's. To enable ADC 0808 always Address Latch Enable (ALE) & Output Enable (OE) should be high.

23,24,25 pins are used as address lines. These address lines are used to select one of the i/p line.

Port-A is acting as I/p & port-B is acting as O/p.

Address Lines			I/p
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

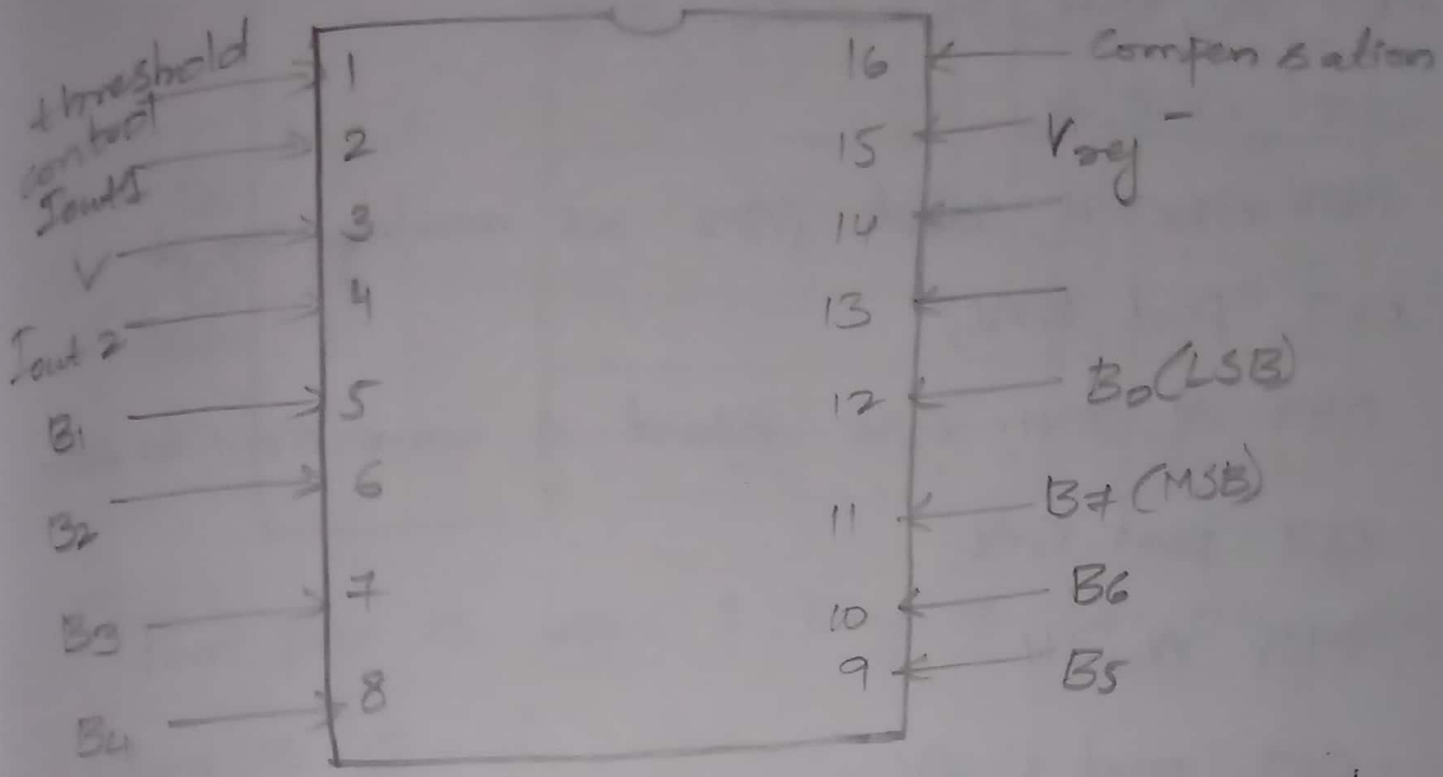
Interfacing 0808 with 8086 through 8255:



Control word for ADC:

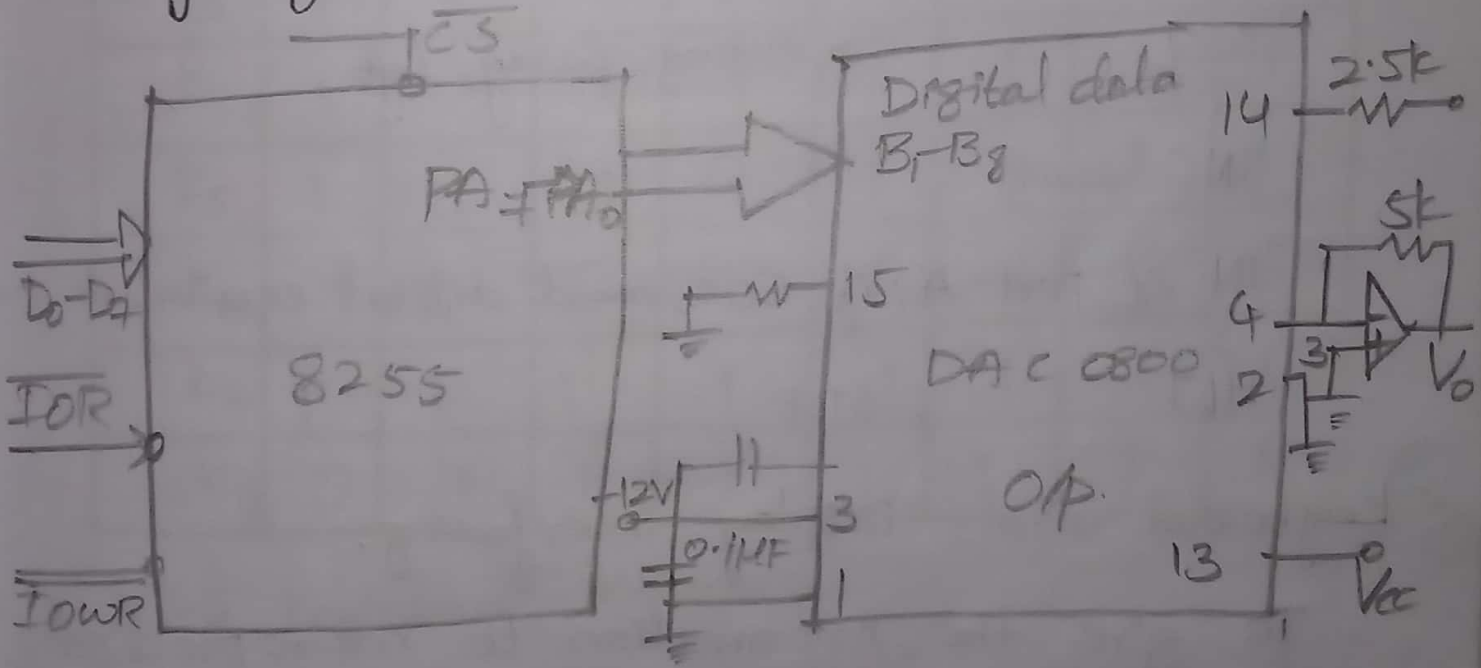
D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	1	0	0	0

DAC Interfacing: (0800) 16-bit IC.



DAC 0800 is a 16 pin IC. It consists 7 i/p's & 1 o/p pin interfacing. DAC receives the data from 8086 to 8255 (DAC acts as o/p)

Interfacing 0800 with 8086 through 8255:



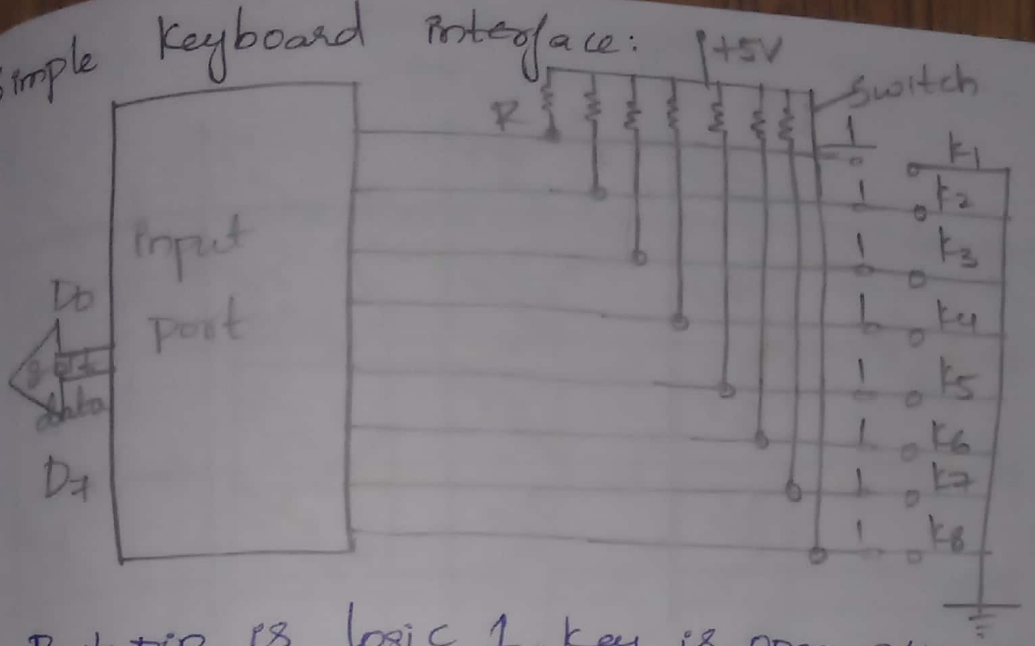
2/3.
Write ALP for interfacing ADC:

```
MOV AL,98H ; Initialize 8255
OUT CWR,AL;
MOV AL,02H ; Select i/p2 as analog input
OUT port B,AL;
MOV AL,00H; Give Start of conversion to ADC
OUT port C,AL
MOV AL,01H
OUT port C,AL
MOV AL,00H
OUT port C,AL
wait: IN AL, 00H : check for EOC
      RCL        : rotating carry left
      JNC wait;
      IN AL, port A; if EOC, read digital equivalent in
      HLT        ; STOP
```

Keyboard and Display interfacing:

To give the information to microprocessor, the key should be pressed (Open (or) close)
OFF (or) ON

Simple Keyboard Interface:



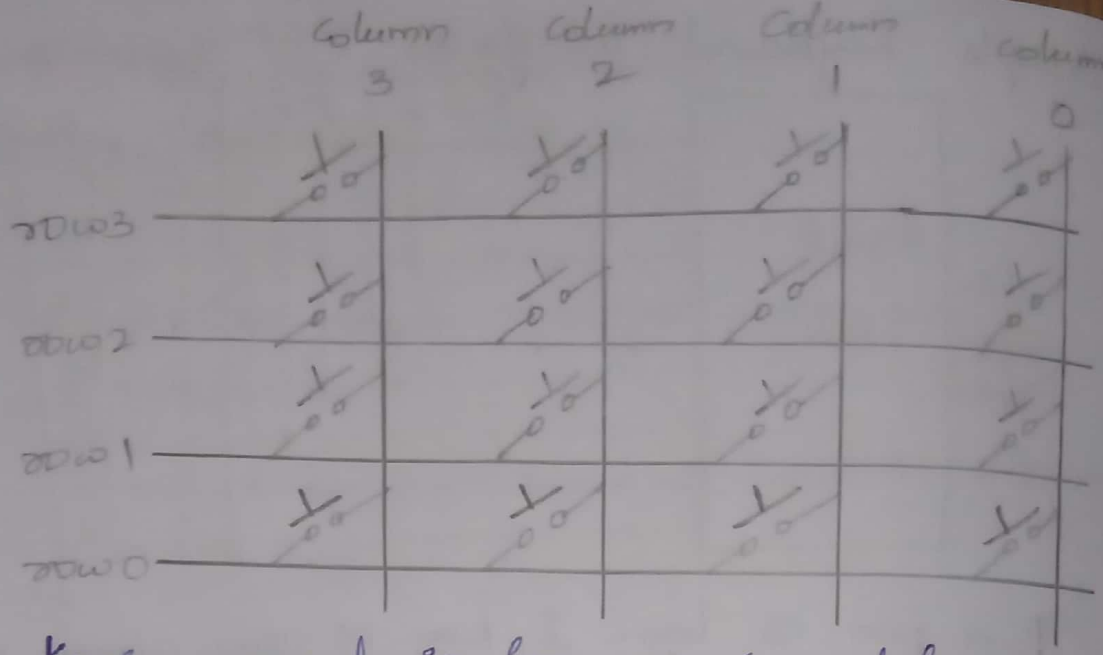
Port pin is logic 1, key is open. Otherwise, key is closed.

Key code.

Key	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
K_1	1	1	1	1	1	1	1	0
K_2	1	1	1	1	1	1	0	1
K_3	1	1	1	1	1	0	1	1
K_4	1	1	1	1	0	1	1	1
K_5	1	1	1	0	1	1	1	1
K_6	1	1	0	1	1	1	1	1
K_7	1	0	1	1	1	1	1	1
K_8	0	1	1	1	1	1	1	1

Matrix keyboard interface:

In simple keyboard interface one I/p line is required to interface one key and this no. will increase with number of keys such technique is not suitable when it is necessary to interface large no. of keys.



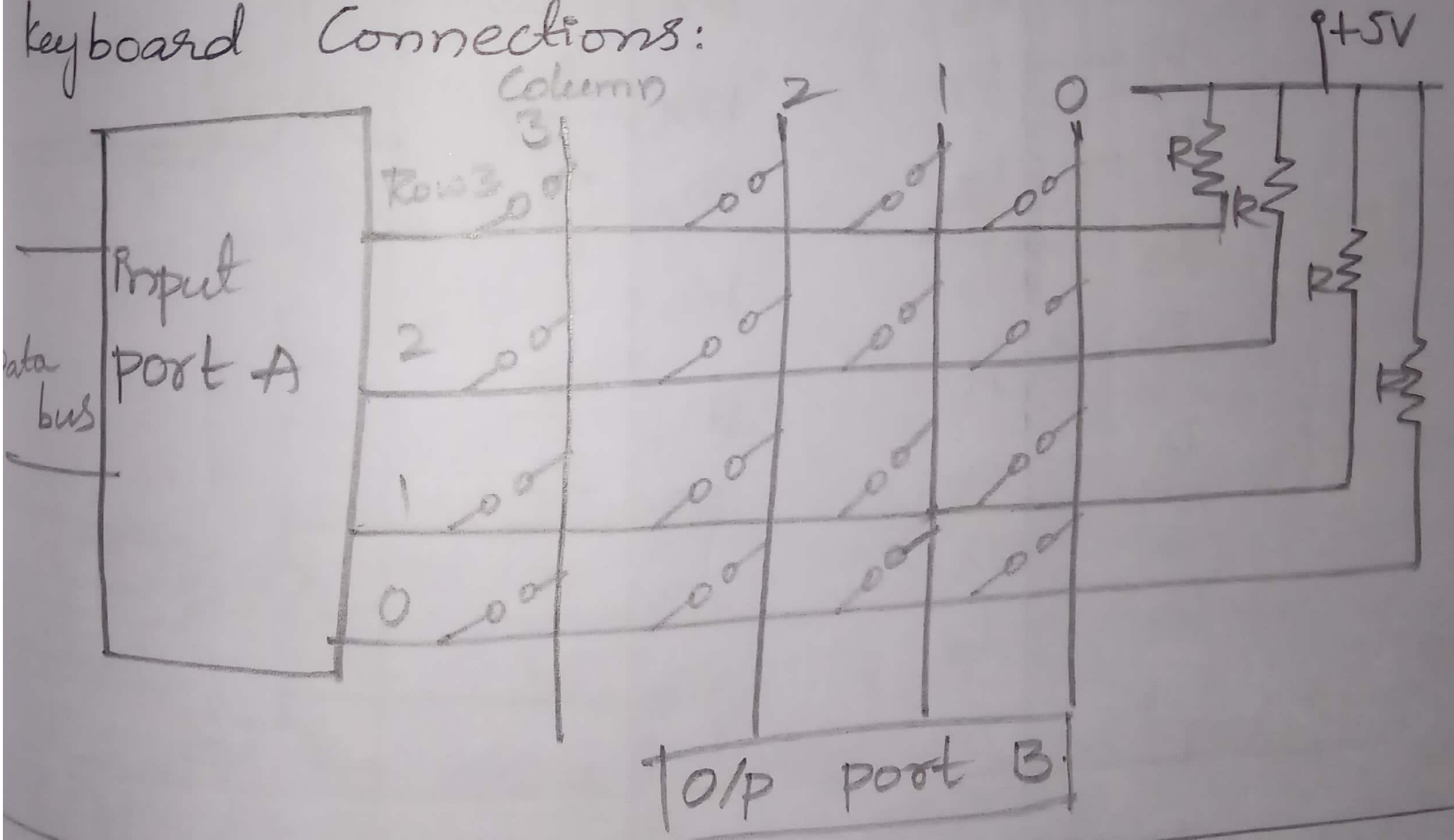
Keys arranged in four rows and four columns. When keys are open, row and column donot have any connection. When key is pressed, it shorts corresponding one row and one column. This matrix keyboard requires eight lines to make all the connections instead of sixteen lines required if the keys are connected individually. Matrix keyboard required two ports.

1) Input port
2) O/p port

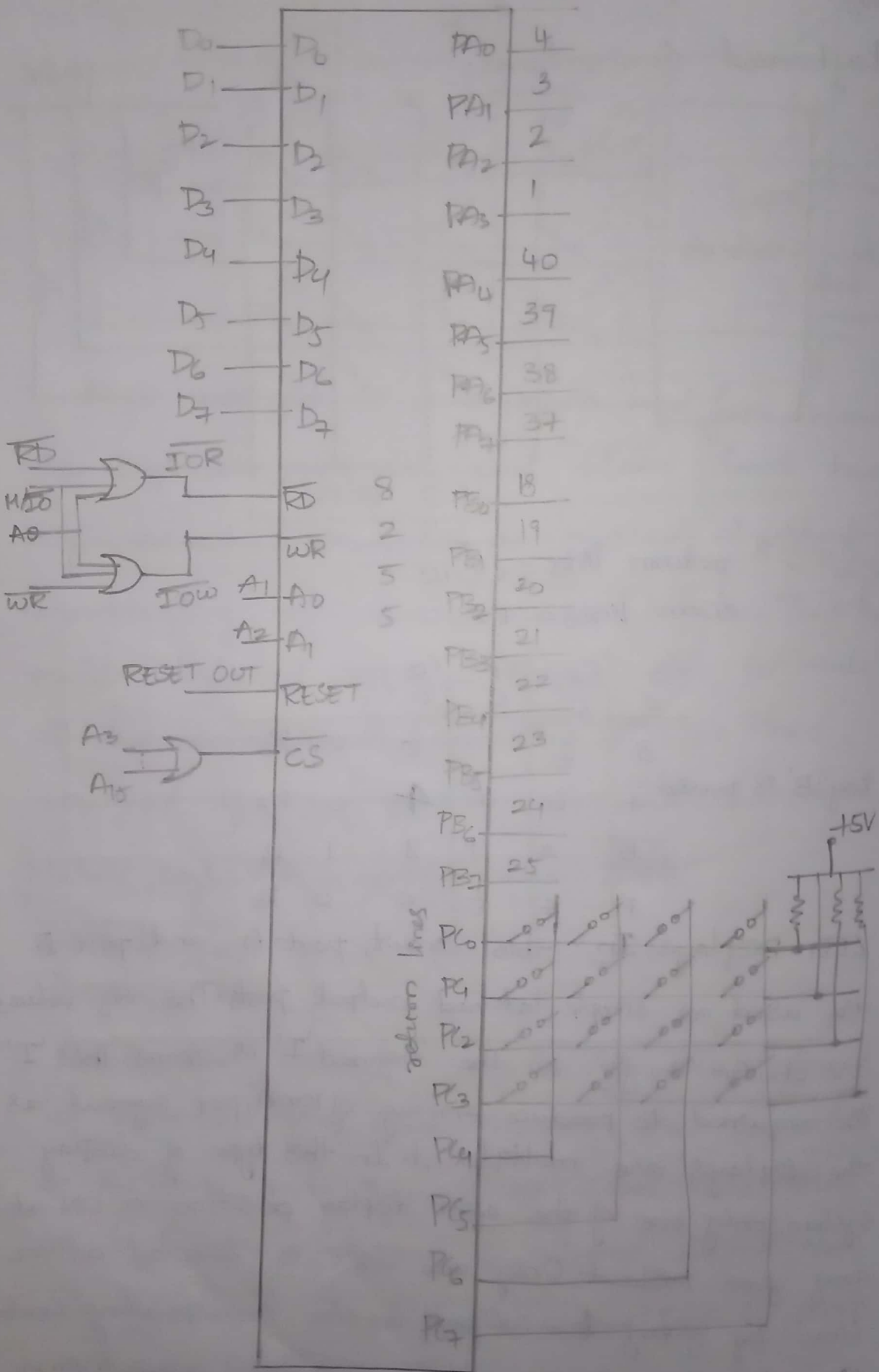
Rows are connected to the I/p port referred as returned lines, and columns are connected to the o/p port referred as scan lines. When all keys are open, row & columns do not have any connection. When key is pressed it shorts corresponding row & column. If the o/p line of this column is low, it makes corresponding row

line low; otherwise the status of row line is high

Keyboard Connections:



Interfacing of 4X9 keyboard with 8086:



Scan lines are connected to port C and return lines are port C0.

Interfacing → contd...

return lines = 0010

Scan lines = 1000

C_3	C_2	C_1	C_0		
0	0	0	0		
3	2	1	0	0	R_3
7	6	5	4	0	R_2
B	A	9	8	1	R_1
F	E	D	C	0	R_0

key 'B' is pressed

LED Displays: In this circuit, port-A and port-B are used as simple latched output ports. The reg. values are chosen in fig. so the segment 'I' is 80mA. This 'I' is required to produce an avg. of 10mA per segment as the displays are multiplexed. In this type of display system, only one of the eight display position is ON at any given instant. Only one digit is selected at a time by giving low signal on the corresponding Control line. Max. anode current is 560mA (7-segments × 80mA = 560mA) but the avg. anode 'I' is 70mA.

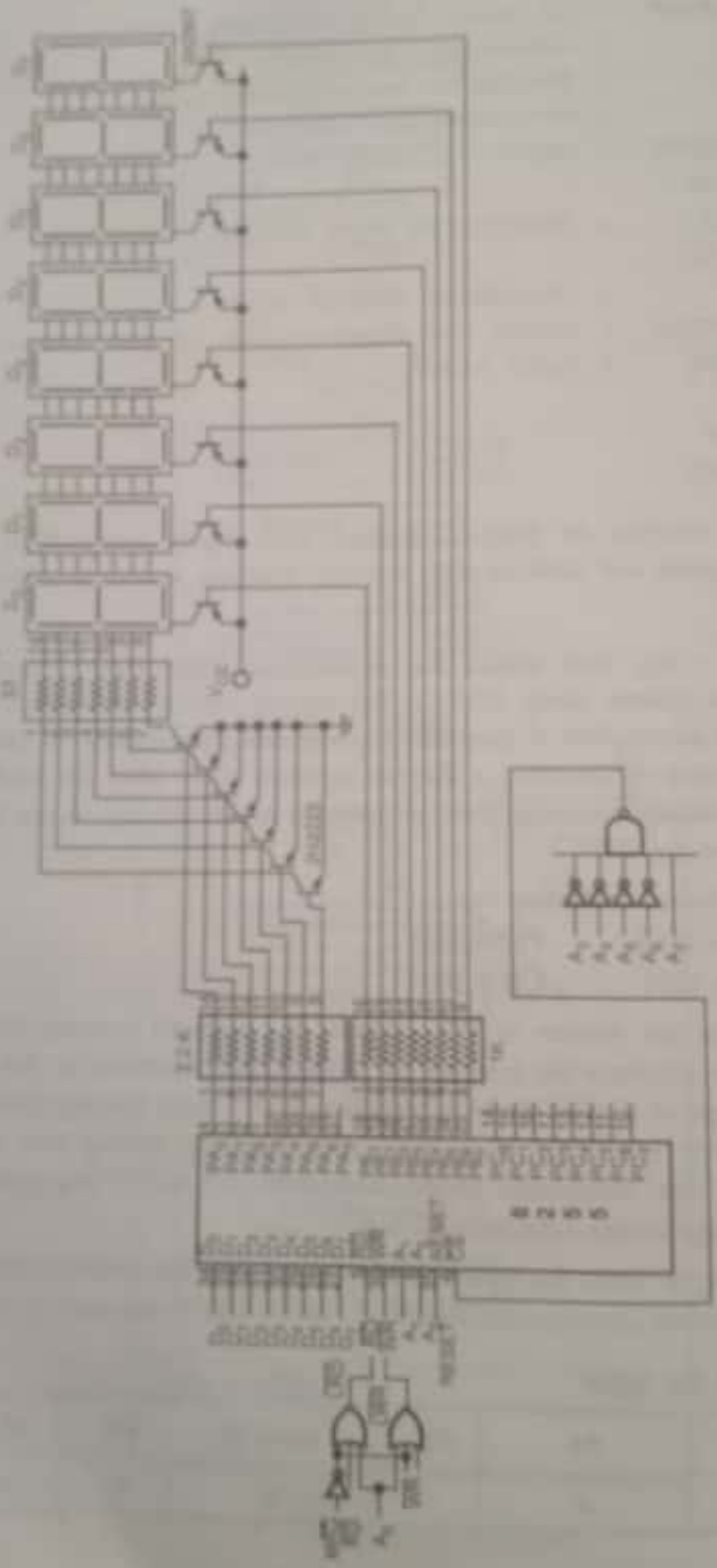


Fig. 5.64 8 Digit 7-segment display interface using 8255

Interrupts.

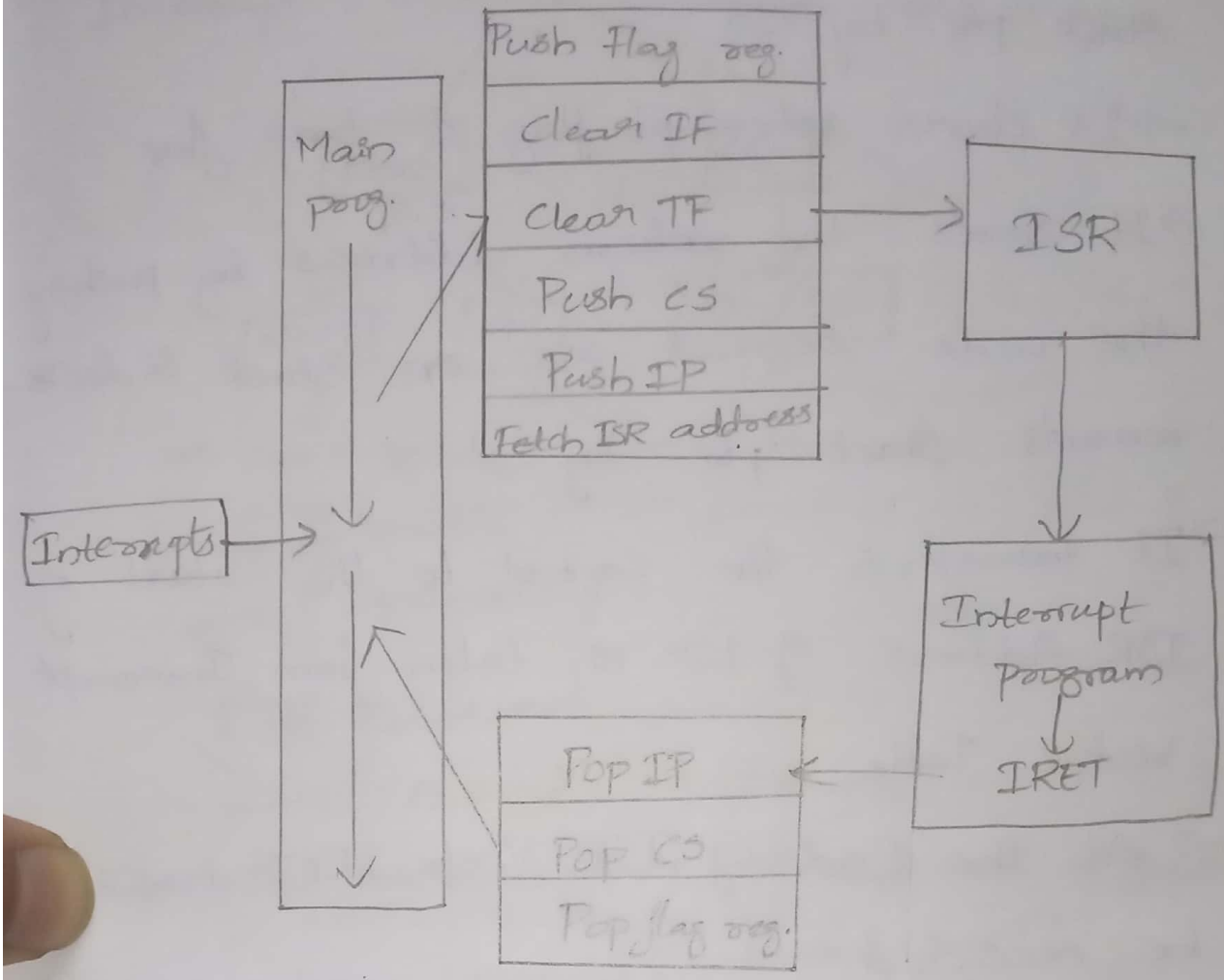
Interrupt is defined as a breakpoint in normal sequence of execution.

Interrupt Service Routine:

When the CPU is executing a program, an interrupt breaks the control sequence of execution of instructions & it diverts its execution to some other program is called as Interrupt Service Routine. After executing ISR, the control is transferred back again to the main program which was been executed at the time of interruption.

Program can be interrupt by 3 ways.

i) By external signal
ii) By Spl. instr. in program
iii) By occurrence of some condition.



ISR Procedure:

→ It completes the execution of current instn.

→ ~~Interrupt~~ ~~se~~ Checks the source of interrupt (Non-Maskable & Maskable)

→ If it is a maskable interrupt, 8086 ignores that interrupt.

→ If it is a NMI, IF flag is set & INTR, $\overline{INTA}$ will be enable.

→ It pushes the flag reg. on stack & decrements

Stack pto. by '2'

→ It clears interrupt flag & trap flag

→ It saves the return address by pushing the code segment reg. on Stack & decrements Stack pto. by '2'

→ It transfers the control to the start of ISR. Address of ISR is taken from Interrupt Vector Table.

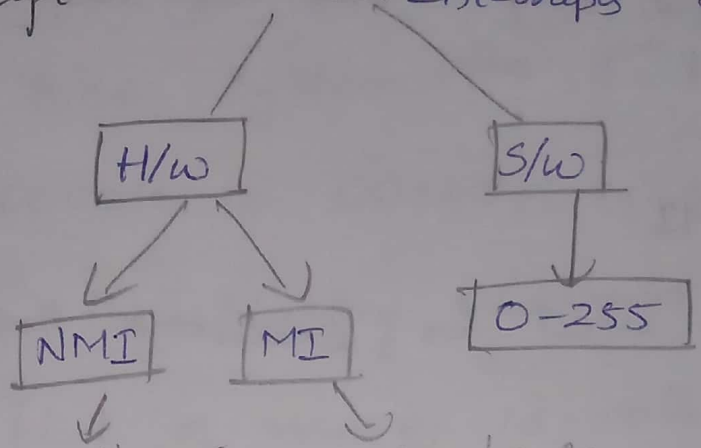
→ At the End of ISR, instr. IRET has to be executed.

→ It pops instr. pto. from Stack & increment Stack pto. by '2'

→ It pops the flag reg. from Stack & increments Stack pto. by '2'

→ It transfers the control back to the program which was interrupted then TF & IF will be enable.

Interrupt Structure: Interrupts are 2 types.



we can't ignore — can be ignored.

INTR } enable
INTA

0-4: Dedicated

5-31: Reserved

224 : Available interrupts.

The 256 interrupts are divided into 3 grps.

1) Type 0 – type 4 interrupts (dedicated int's).

2) Type 5 – type 31 (Reserved int's)

These int's are used in ~~8086~~ 80286.

3) Type 32–255 (Available int's):

Type 0: Divide Error Interrupt.

If the value of quotient is large & cannot fit in respective reg.'s then 8086 does not store it & it executes INT 0 (Zero) instruction

INT type number

Type 1: Single Step Instruction.

This int is used for executing the prog.

on a Single Step mode. For this TF is set

INT 1.

Type 2 NMI:

INT. is used for executing the ISR.

INT 2.

Type 3 Breakpoint Interrupt:

This int. is used for providing breakpoints

in the prog.

INT 3.

This type-4 int. is used to check overflow condition, after any signed arithmetic operation.

Ex: 53 = ~~00~~ 11 0101 0101 (1) 0011 (1)(1)

91 = 0101 1011 1001 0001

144 1001 0000 1110 0100

⇒ E 4.

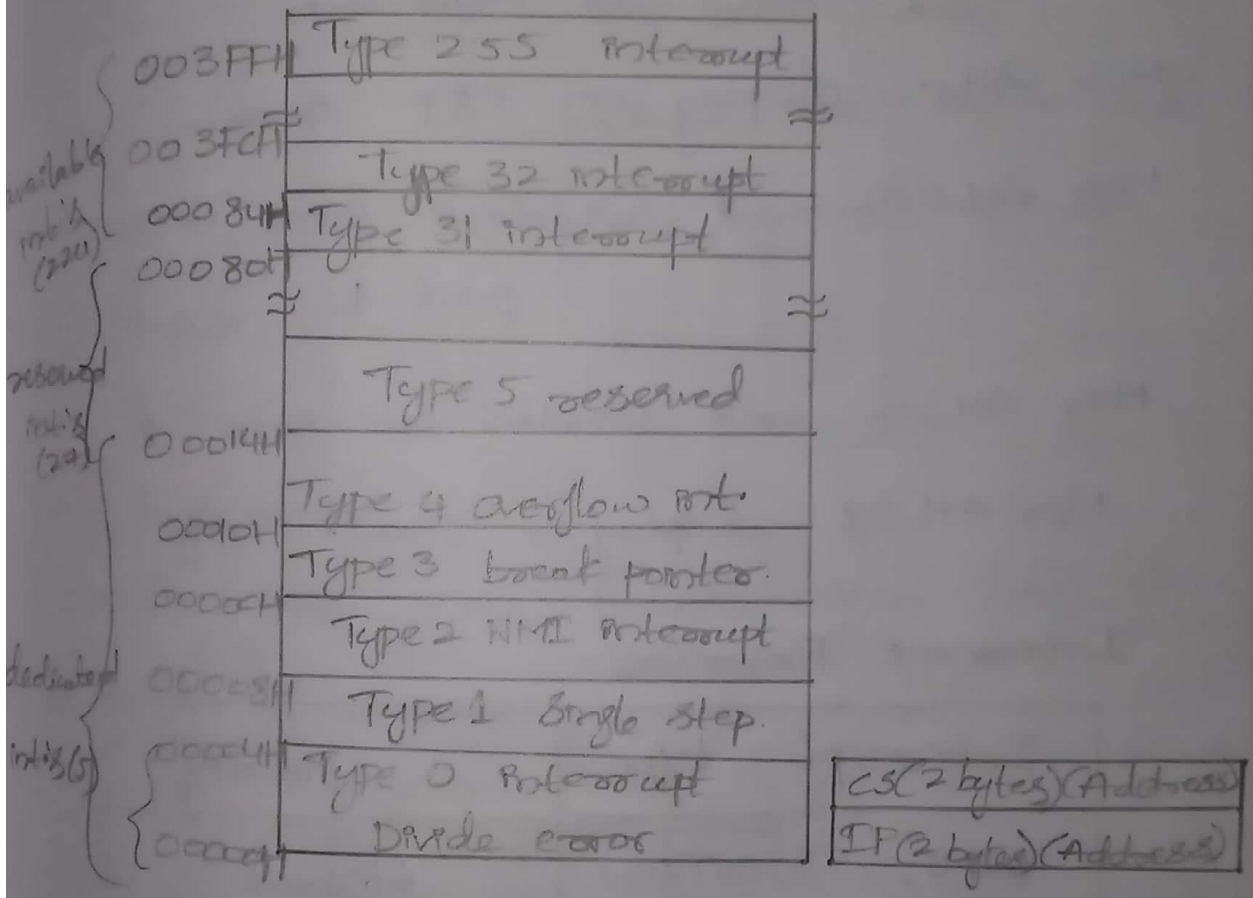
-128 to 128 → if it crosses the range

INT 0.

Interrupt Vector Table:

In 8086 System the 1st KB of memory from 00000H to 003FFH is reserved for storing the starting address of ISR.

This block of memory is called Interrupt Vector Table.



Address of each int. is found by multiplying type of int. by 4.

Ex. for type-11: $4 \times 11 = 44$.

$16 \overline{)44} - 0$

$32 - 12$

$16(44\ 16)\ 4(2$
32
12

00002CH

00C0H.

DOS Interrupt: (Procedure)
This is used to call the Subprog. & macros by using INT 21H

To call any DOS fn; 1st place the fn. no. in AH reg. & load the relevant I/p parameters in specified reg.

MOV AH,01 - read the keyboard

MOV AH,02 - display on CRT screen

MOV AH,03 - read characters from communication port-1.

MOV AH,04 - write chars to comm. -1.

MOV AH,09 - display a char. string.

Interrupt Priorities.

Interrupt type	Priority.
INT 0, INT 3, INT 255, INT O.	Highest
NMI	↓(2nd)
Single Step Type 1	Low.

3/3. Communication Interface.

To avoid the parallel communication, for long distance, we are using Serial communication.

Advantages of Serial Comm.:

→ It is used to reduce the noise

→ It is used for long dist. comm.

→ " " " to reduce the cable cost

Serial comm. is done by 3 methods:

i) Simplex

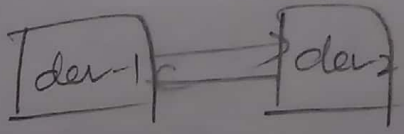
ii) Half-duplex

iii) Full-duplex

i) Simplex: The data transmission occurs only in 1 direction.

ii) Half-dup: The data trans. occurs in 2 way direction but not simultaneous.

Ex: Walky-talky.



iii) Full-duplex: The data trans. occurs in 2 way direction & simultaneously.

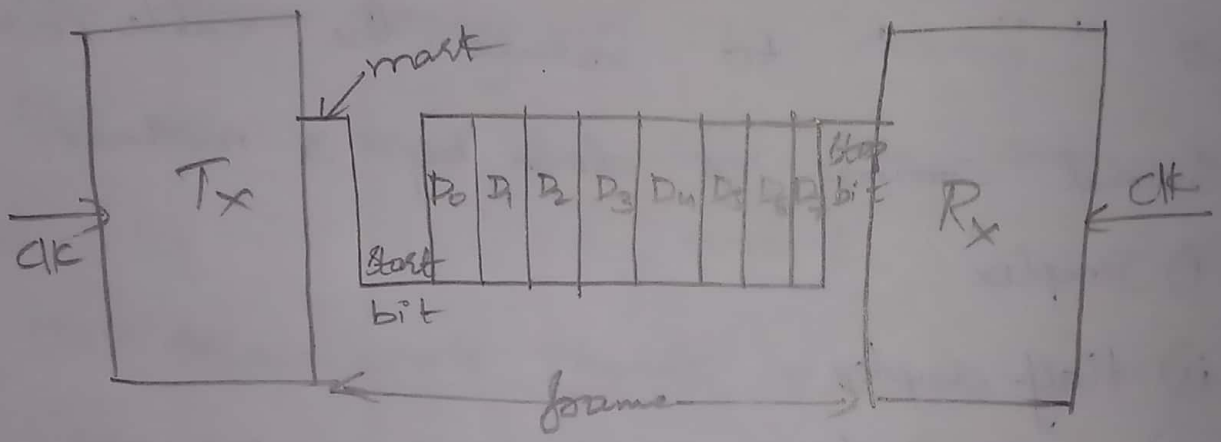
Ex: Telephone

Data trans. formats are 2 types.

i) Asynchronous Data Format

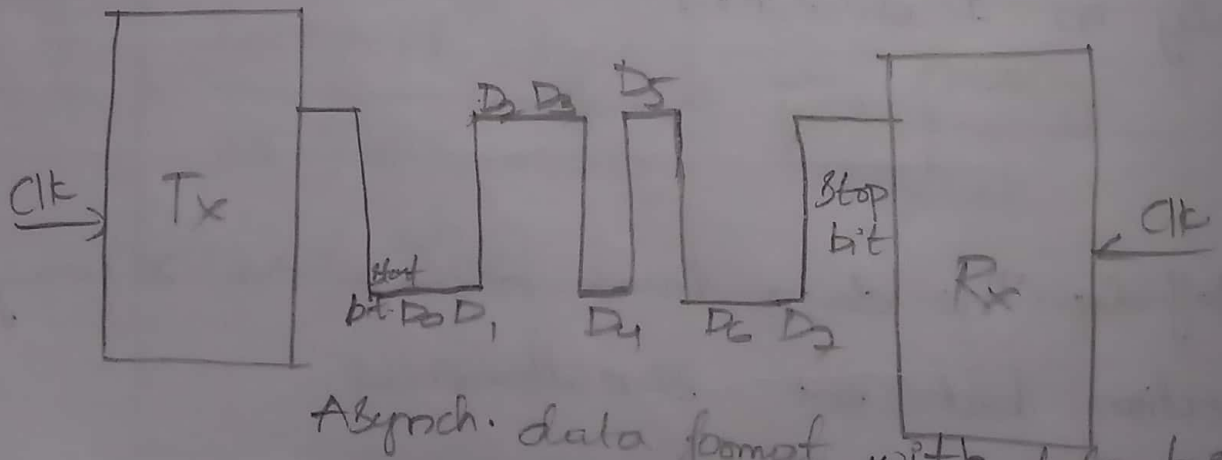
ii) Synchronous " "

Asynchronous Data Format:



Ex: 34:

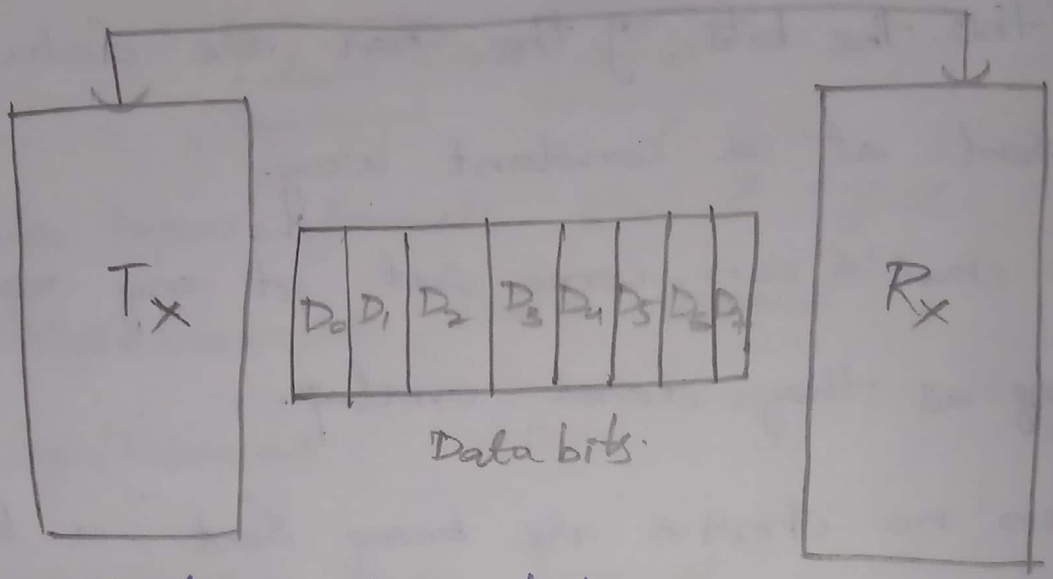
0011 0100.



Asynch. data format with data bits

→ Asynch formats are character oriented.
→ In this, the bits of the char. are data word are sent at a constant way.
→ But chars can come out at any rate as long as they donot overlap.
→ When no chars are being sent, a line stays high at logic -1 is called as 'mark' & logic-0 is called as 'space'. The beginning of the char. is indicated by a start bit which is always low.
→ This is used to synchronise the Tx & Rx. After the start bit, the data bits are sent, the stop bit indicates the end of char.
→ Different systems use 1, 1½, 2 stop bits.
→ The combination of start bit, char. & stop-bits is known as a frame.
→ The data rate can be expressed as bits per sec is also called as 'baut rate' & it is used for low speed < 20 bits/sec.

Synchronous Data Format:



The start & stop bits can be eliminated in Synch. data format. The Synch. is provided by the clk signal.

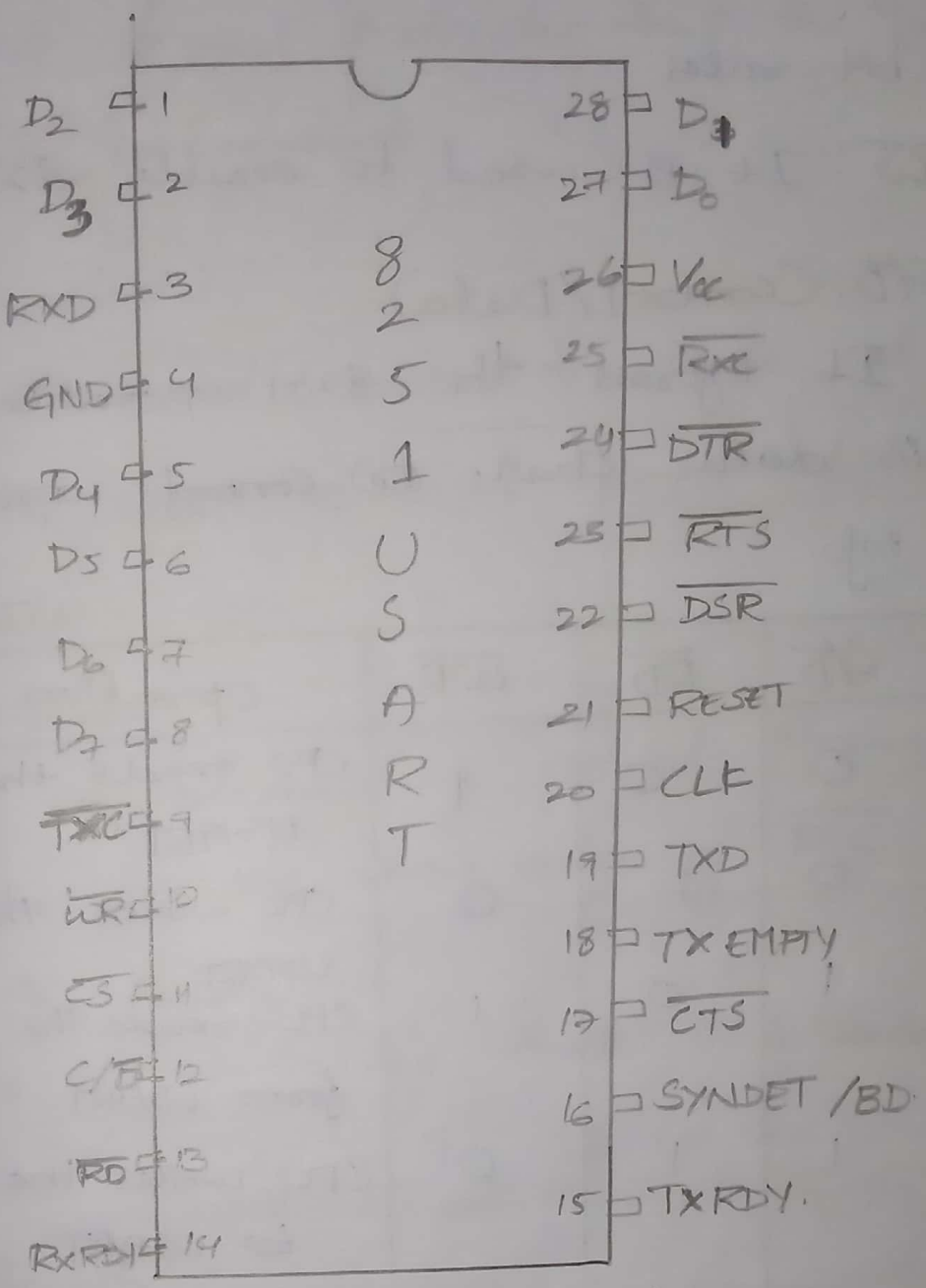
8251 USART (Universal Synch. Asynch. Rx & Tx)

USART is used for ||el to serial converter (During transmission), Serial to ||el converter (during receiving)

4|3

Features of 8251:

Asynchronous: 5 to 8 bit character format
Baud rate 19.2 kbits per sec.
→ Automatic Break Detect & handling.
Synchronous: 5 to 8 bit char. format
Provides error detection logic. Allows full duplex transmission.



Pin Description:

1) Data bus ($D_0 - D_7$): 8-bit data bus is used to transfer of bytes b/w CPU & 8251.

2) $\overline{RD}$: This will allows the CPU to read the data (or) Status byte from 8251.

3) $\overline{WR}$: This will allows the CPU to write the data (or) Command word to the 8251.

4) Clk: This is used to generate the synchronisation in internal device. The freq. of the

Clk must be > 30 times the Rx & Tx data bit rates.

5) $\overline{CS}$: It is used to enable 8251.

6) C/$\overline{D}$: (Control / Data):

It informs the 8251 operation & the word is data char. (or) control word (or) Status info.

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	operation.
0	0	1	CPU reads the data from USART.
0	1	0	CPU writes the data to USART.
1	0	1	CPU reads the Status word from USART
1	1	0	CPU writes the Command word to USART
1	1	1	Idle.

Transmitter Signals:

1) TXD: (Transmit Data):

The o/p signal send a serial stream of data on falling edge of $\overline{TXC}$.

2) TxRDY: (Transmitter Ready):

This o/p signal indicates the CPU that the Tx is ready to accept data char.

3) TXE (Tx Empty):

This o/p signal indicates that the Tx has no ~~o/p's~~ chars to transmit.

4) $\overline{TXC}$: (Tx Clock)

The clk input control the rate at which the char. is to be transmitted.

Rx Signals:

1) RxD: (Rx Data)

This i/p receives a serial stream of data on the rising edge of $\overline{RxC}$.

2) RxRDY: (Rx Ready):

This o/p indicates that 8251 contains a char. is ready to be i/p to the CPU.

3) $\overline{RxC}$ (Rx Clk):

The clk i/p controls the rate at which the char. is to be received.

4) SYNDET/BRKDET: (Synch. Detector / Break Detect)

This pin is used in Synchronous mode for detection of Synch. char. & may be used as either i/p (or) o/p.

MODEM Control Signals:

1) $\overline{DSR}$: (Data Set Ready):

This i/p signal is used to test MODEM conditions such as Data Set Ready.

2) $\overline{DTR}$ (Data Terminal Ready):

This o/p signal is used to tell the MODEM that the data terminal is ready.

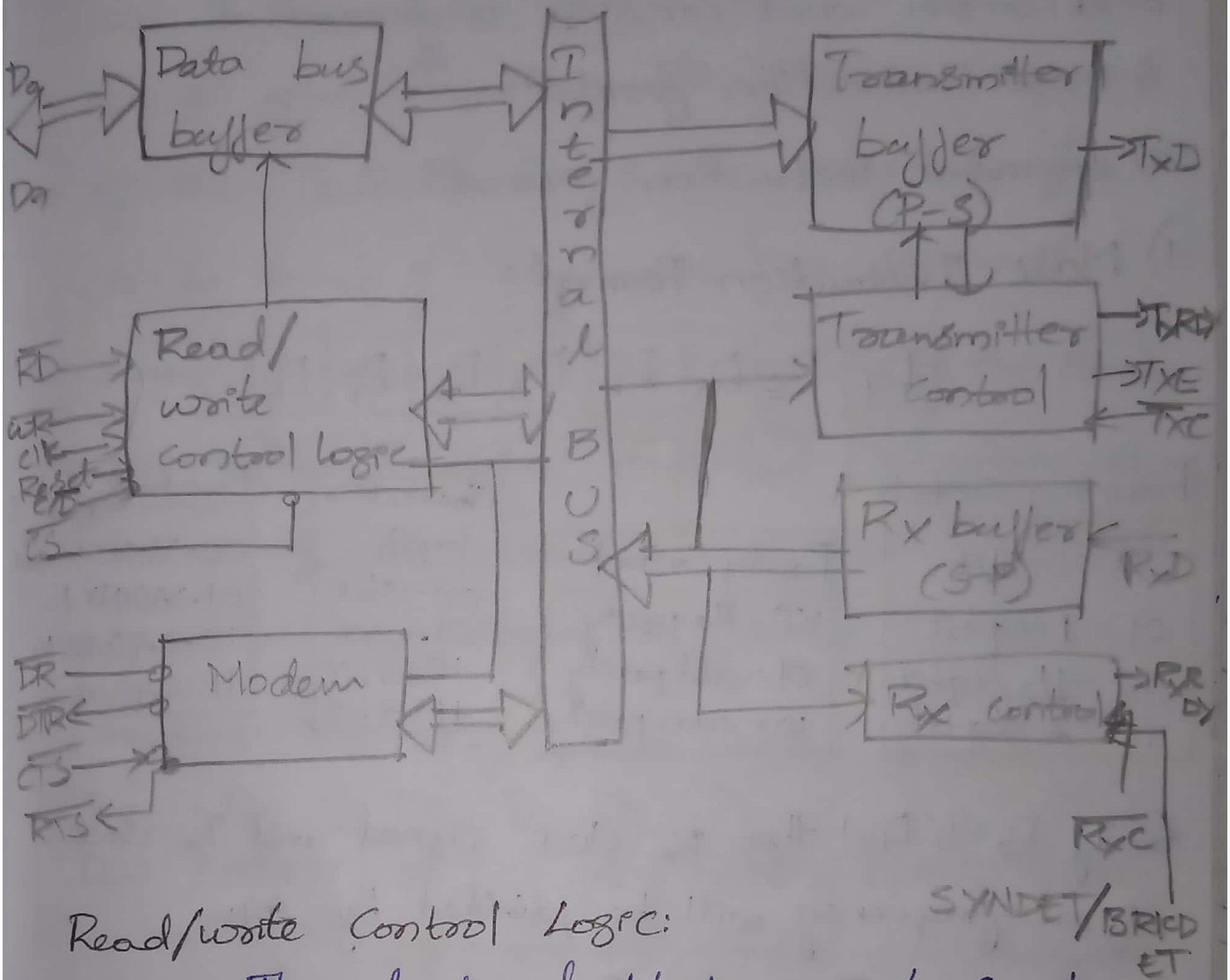
3) $\overline{RTS}$ (Request To Send):

This o/p signal is applied to be in transmission.

4) $\overline{CTS}$ (Clear To Send):

This pin enables the 8251 to transmit serial data if transmission empty bit in the command byte is set to 1.

Functional Block diagram of 8251:



Read/write control Logic:

This functional block accepts inputs from the system control bus & generate control signals for overall device operations.

Transmitter buffer:

It converts parallel data to serial data. It has 2 reg.'s to hold 8-bit & o/p reg. parallel to serial converter.

Rx Buffer:

It is used to convert serial data to parallel data.

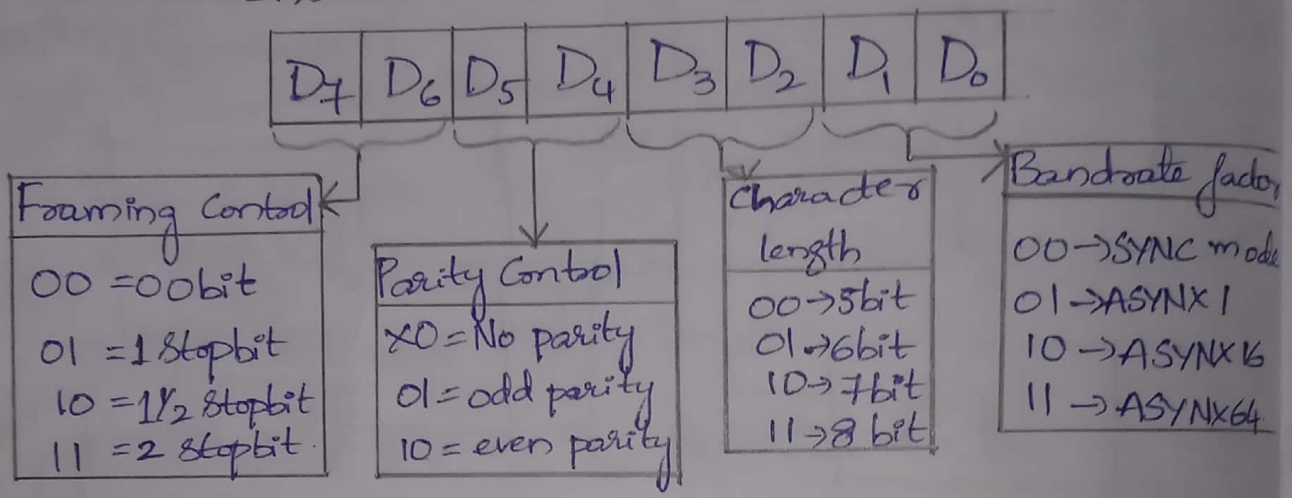
8251 Control word:

8251 control word consists 2 formats:

1) Mode Instruction format

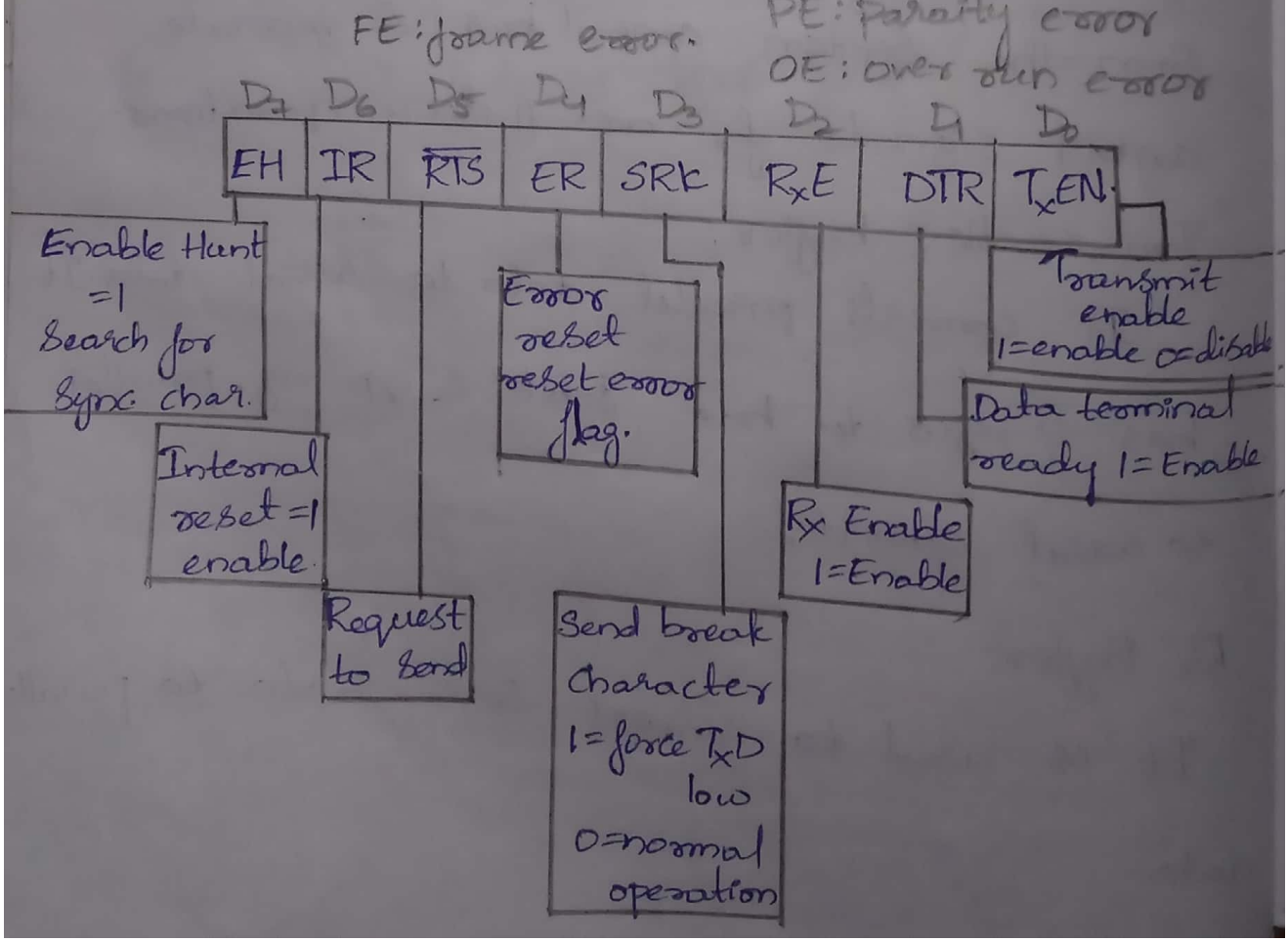
2) Command instruction format.

1) Mode Instruction Format:



Ex: If $D_0 = 1$, $D_1 = 1$ then R_x clock signal and T_x clock signal frequency will be divided by 64.

2) Command Instruction Format:



8251A Status Word:

In the data communication systems it is often necessary to examine the "status" of the Tx & Rx. It is also necessary for CPU to know if any error has occurred during communication. The 8251A allows the programmer to read above mentioned info. from the status reg. any time during the functional operation. Fig(1) shows the format of status register.

Fig(1) Status register format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DSR	SYNDET/ BRKDET	FE	OE	PE	TxEMPTY	RxRDY	TxRDY.

Note 1 → Same definitions as I/O pins

Data Set Ready:
Indicates that the DSR is at zero level.

Framing Error: (ASync only).
The FE flag is set when a valid stop bit is not detected at the end of every char. It is reset by the ER bit of the command instruction. FE does not inhibit the operation of the 8251A.

Overrun Error:
The OE flag is set when the CPU does not read a char. before the next one becomes available. It is reset by the ER bit of the command instr. OE does not inhibit operation of the 8251A. However, the previously overrun char. is lost.

Parity Error:

The PE flag is set when a parity error is detected. It is reset by the ER bit of the command instruction. PE does not inhibit operation of the 8251A.

Error Definitions:

Parity Error: At the time of transmission of data an even (or) odd parity bit is inserted in the data stream. At the Rx end, if parity of the char. does not match with the pre-defined Parity, parity error occurs.

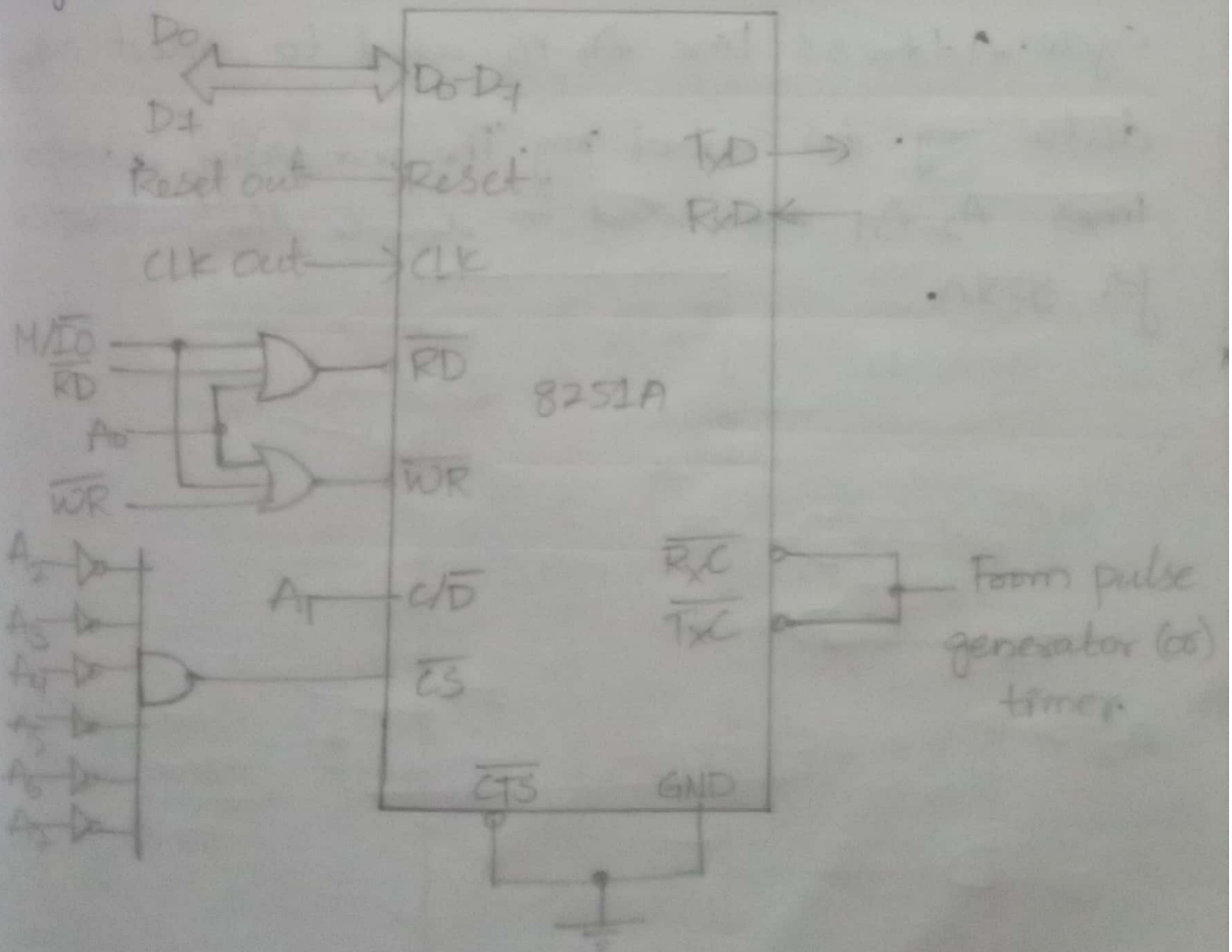
Overrun Error: In the Rx section received char is stored in the Rx buffer. The CPU is supposed to read this char. before reception of the next char. But if CPU fails in reading the char. loaded in the Rx buffer, & the next the received char. replaces the previous one and the OVERRUN Error occurs.

Framing Error: If valid Stop bit is not detected at the end each char. framming error occurs.

All these errors, when occur, set the corresponding bits in the Status reg. These error bits are reset by setting ER bit in the command ins

Interfacing 8251A to 8086 in I/O Mapped I/O Mode:

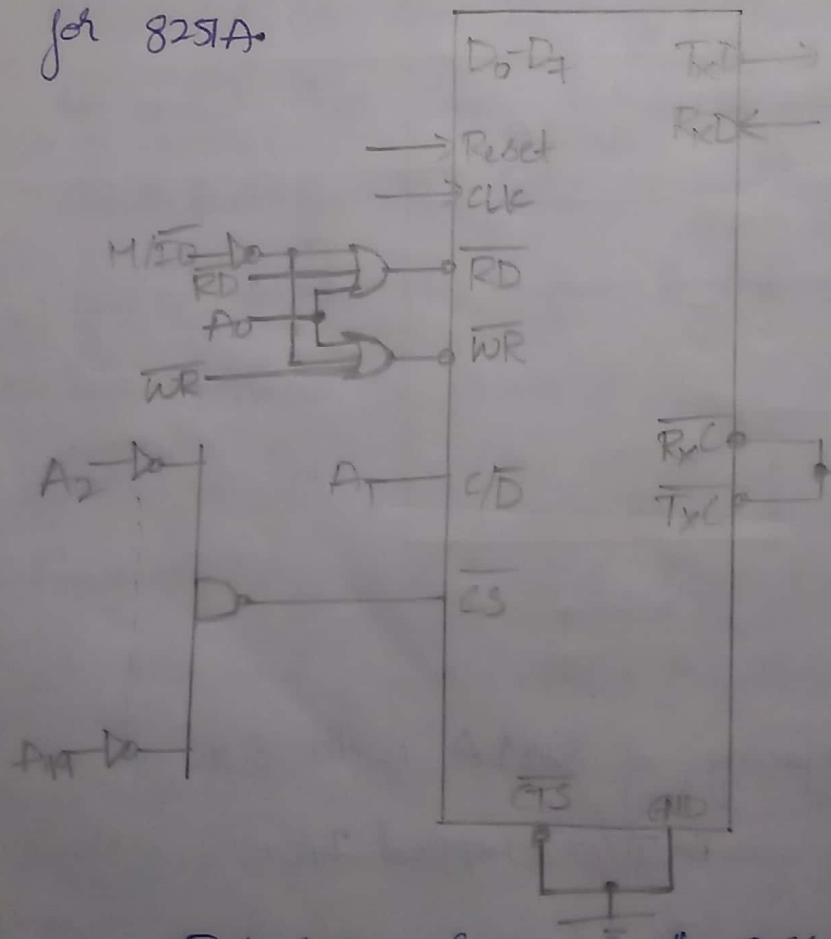
The below fig. shows the interfacing of 8251A with 8086 in I/O mapped I/O technique. Here, $\overline{RD}$ & $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is low, indicating I/O bus cycle. Only lower data bus (D_0-D_7) is used as 8251A is 8bit device. Reset out signal from clk generator is connected to the reset signal of the 8251A



Interfacing of 8251A with 8086 in I/O mapped I/O.

Interfacing 8251A to 8086 in Memory Mapped I/O:

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device; an I/O device is connected as if it is a memory reg. The 8086 uses same control signals & instrs to access I/O as those of memory. The below fig. shows the interfacing of 8251A with 8086 in memory mapped I/O technique. Here, $\overline{RD}$ and $\overline{WR}$ signals are activated when $M/\overline{IO}$ signal is high, indicating memory bus cycle. Address line A_1 is used to select either data reg. (or) control reg. The remaining address lines A_2–A_{19} are used to decode the addresses for 8251A.



Interfacing of 8251A with 8086 in mem. mapped I/O

6/3/2020

Unit-IV

Micro Controller (μC) → (8051)

Micro Controller: μC consists all the features are found in μP and it also has built in ROM, RAM, parallel I/O, serial I/O, counters & clk ckt.

Differences b/w μP & μC:

μP	μC
1) μP contains ALU, CU & reg.'s & interrupt ckt.	1) It consists of μP, memory I/O interfacing & Peripheral devices.
2) It has many instructions to move data between memory & CPU.	2) It has one (or) two instructions to move the data b/w memory & CPU.
3) It has one (or) two bit handling instructions.	3) It has many bit handling instructions.
4) Access time for memory & i/p devices are more	4) Less access time for memory & I/O devices.
5) It requires more hardware & flexible design point of view.	5) Less size hardware & decrease in PCB sides & increase reliability & less flexible in design point of view.

Features

→ 4kB of ROM (prog. mem.)

→ 128 bytes of RAM data Mem.

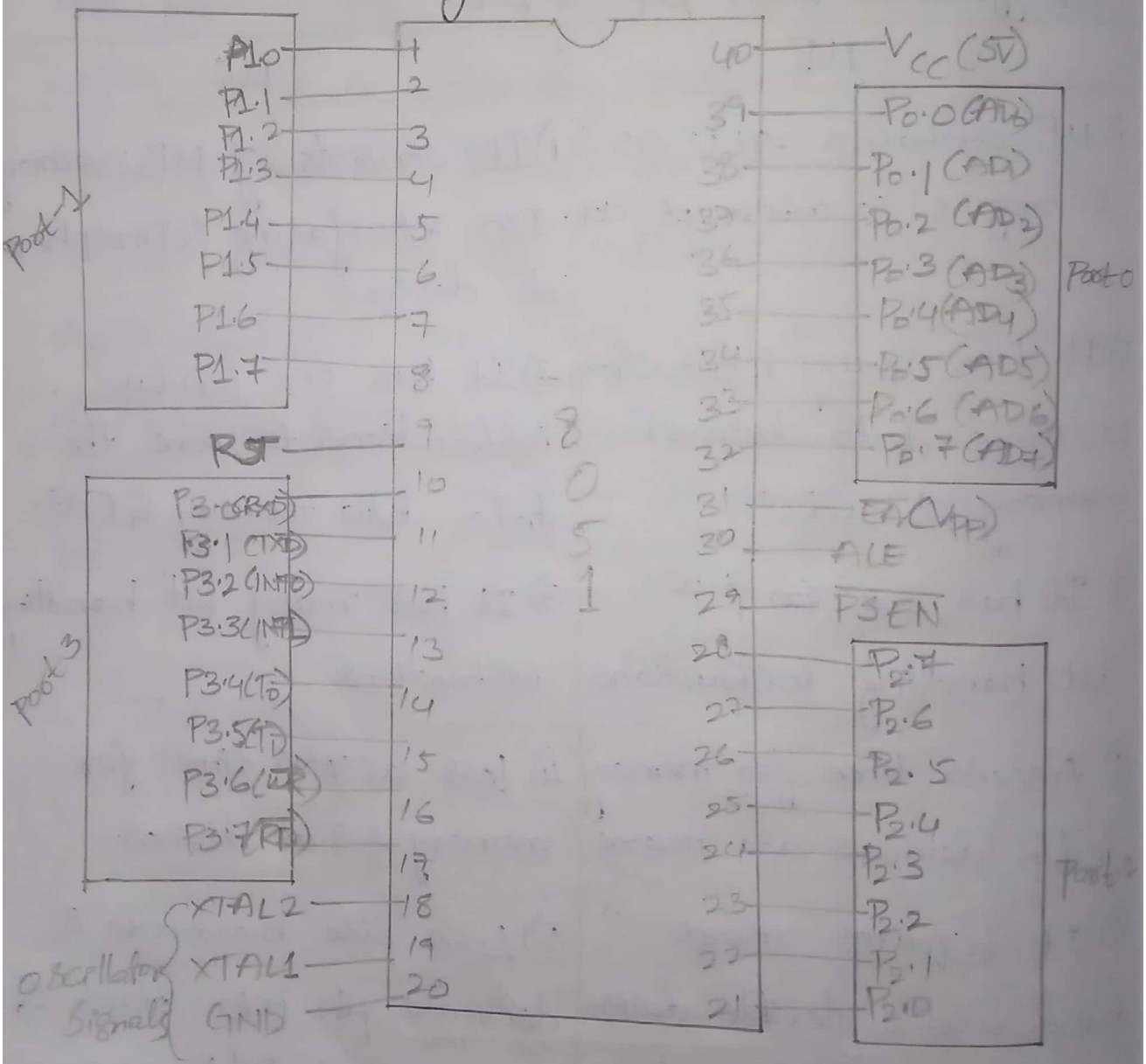
→ 128 user defined flags.

→ 32 bidirectional I/O lines (4 ports - 4x8 = 32 bits)

→ 216 bit timers / counters.

→ 2 level interrupts

Pin Description of 8051:



32 I/O pins. (4 ports)

Each port consists Latch, o/p drivers & i/p buffers.

Port-0:

Port-0 can be used as i/o pins. It acts as external memory from the i/p buffers & o/p drivers.

It consists AD0 – AD7

Port-1:

It can be used as i/o pins.

Port-2:

It consists higher order address (A8-A15)

Port-3:

It has spl. fn.ls (RXD, TXD, INT0, INT1, T_0, T_1, $\overline{WR}$, $\overline{RD}$)

Oscillator pins: XTAL2, XTAL1.

For generating external clk signal, the external oscillator is connected to these 2 pins.

ALE: It is used to separate address & data lines.

RST: Reset the 8051 upto 2 machine cycles.

$\overline{PSEN}$: (Programmable Store ENable) It is used to enable the EPROM & ROM.

$\overline{EA}$: It is used to access the external mem. upto 64KB.

When 8051 in data mode it access memory from the external devices. When it is in programming mode, it is connected to V_{PP}(+5V).

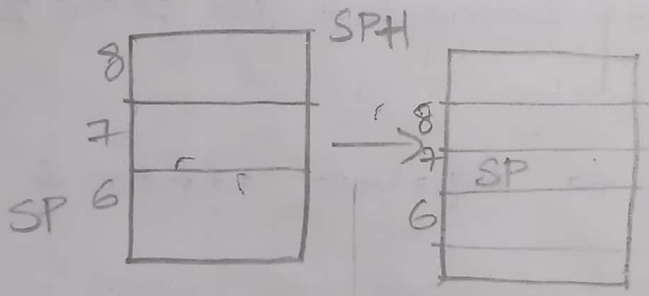
~~8051~~ 8051 Internal Functional Block Diagram:

~~A~~ Register:

→ It is used to store the result of addition & subtraction.

B register: It is used for multiplication & division.

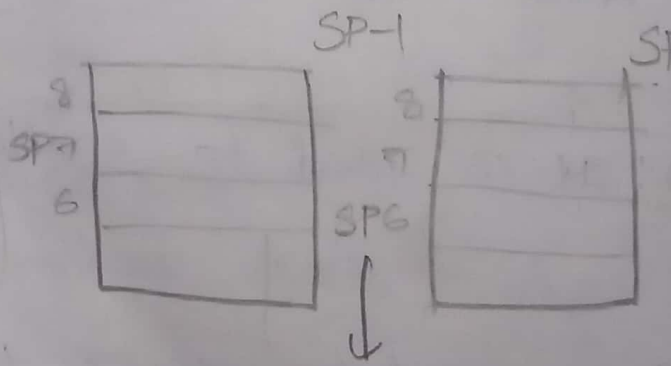
Stack ptr.: It is a temp. storage device while performing push operation, stack ptr. is incremented by ~~1~~ while performing pop operation Stack ptr. is decremented by ~~1~~



after reset

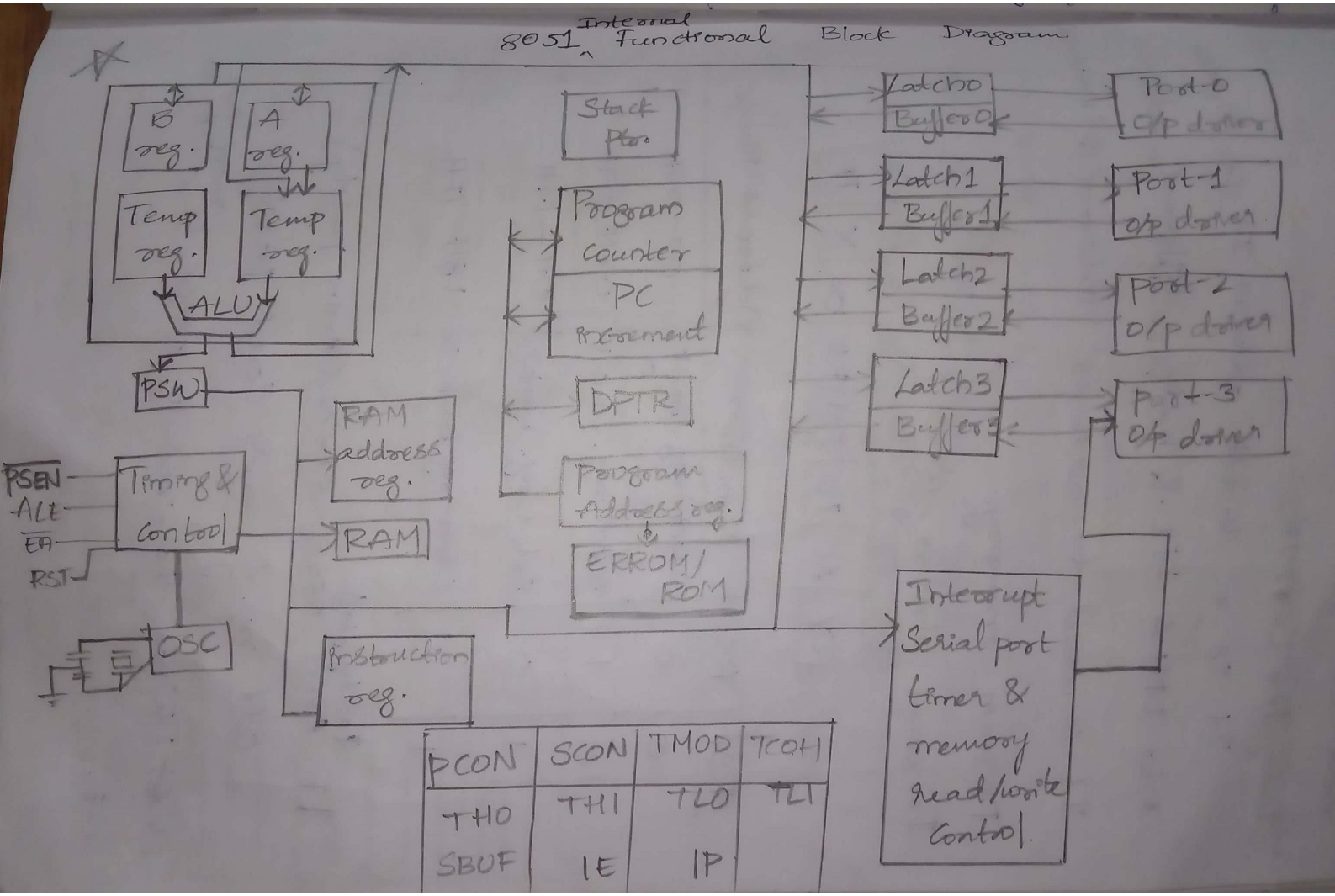
while performing push operation, Stack ptr. is incremented by '1' after it inserts data

~~8051~~



after reset

while performing pop oper, it delete data after it is decr. by '1'



Program Counter:

It is used to hold the address of next instr to be fetched.

Data Ptr (DPTR):

It is a 16-bit address. It is divided into DPH & DPL. & having different address.

It is used to address internal & external program memory.

Program Status word:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CY	AC	F0	RS1	RS0	OV	X	PF

RS_1	RS_0	Function	Address Range
0	0	Select reg. bank 0.	00-07H
0	1	Select reg. bank-1	08H-0FH
1	0	Select reg. bank2	10H-17H
1	1	Select reg. bank-3	18H-1FH.

PF: (Parity flag) Even no. of one's (even parity)

Odd no. of one's (odd parity).

OV (Overflow flag): When the result range is more than -128 to 128.

F0: User defined flag. It depends upon the user instr's.

~~Ans~~ The selection of reg's depends upon the RS0 & RS1.

Memory Organisation

1) RAM → 128 bytes

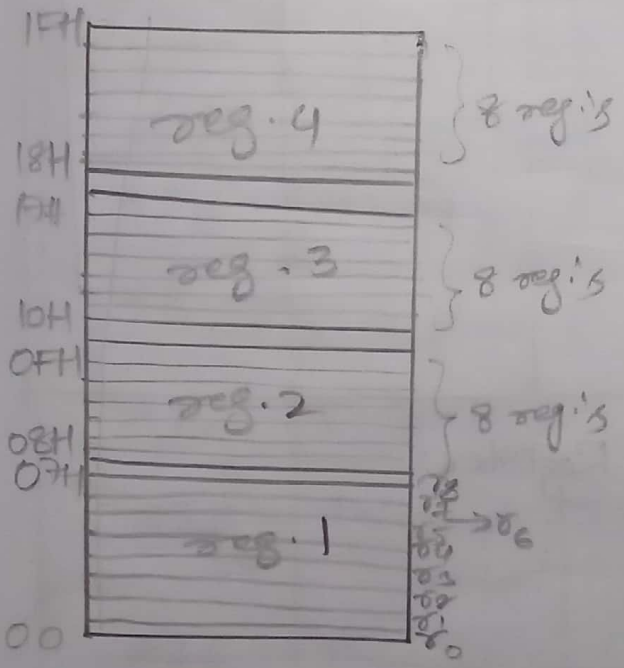
2) ROM → 4k bytes.

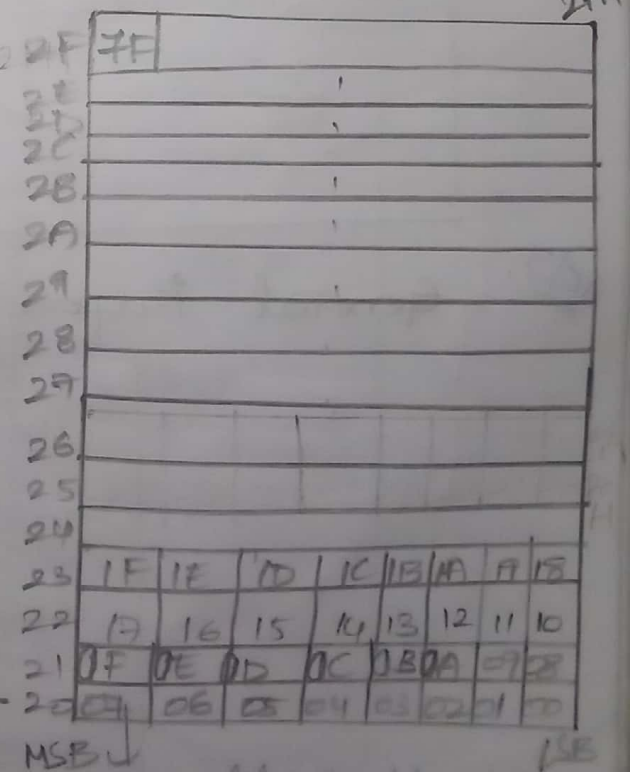
RAM is divided into 3 types:

1) Working reg. (32 bytes) (00–1FH).

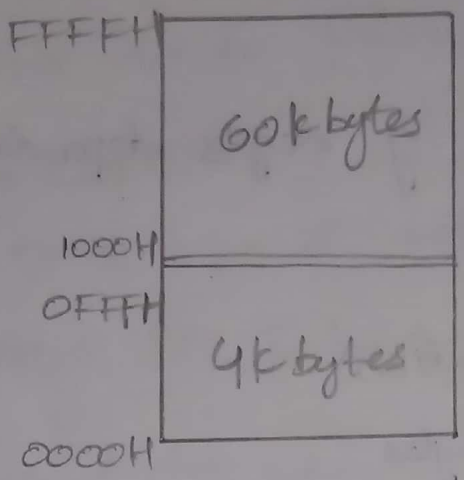
2) bit addressable (16 bytes) (20H to 2FH)

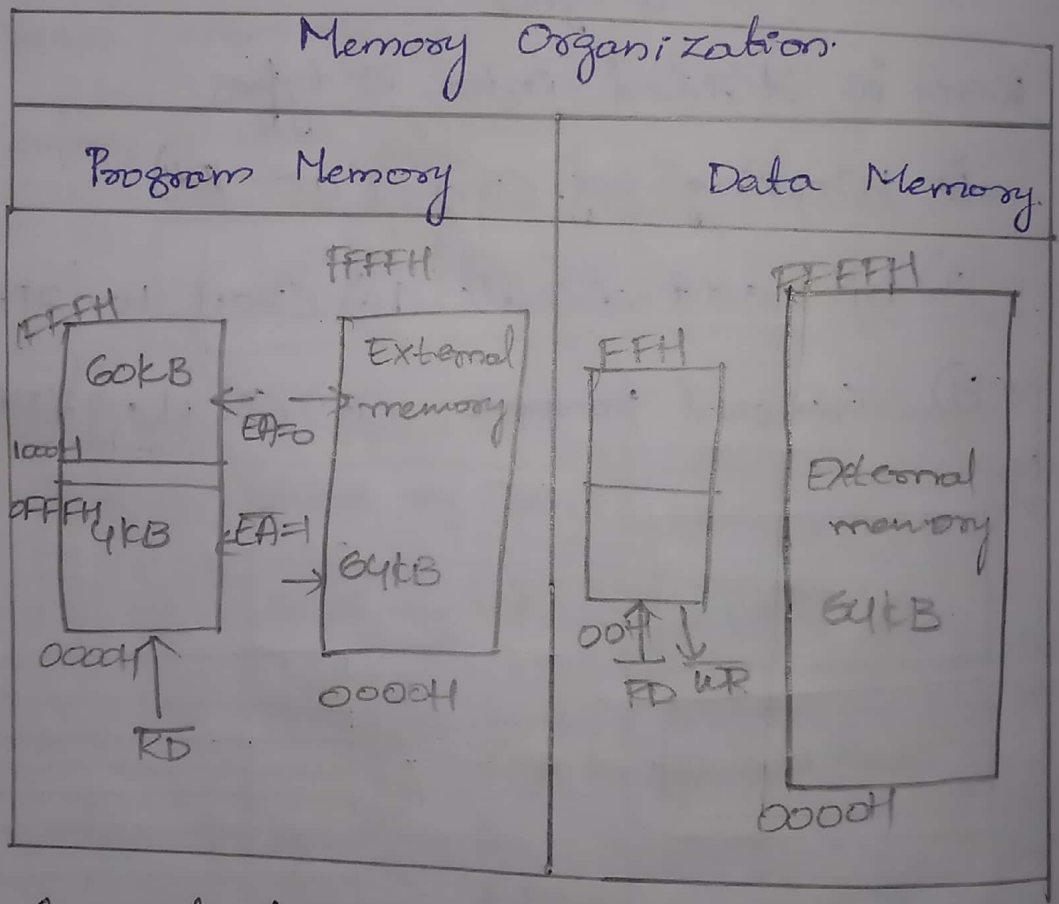
3) General purpose (80) (30H → 7FH)



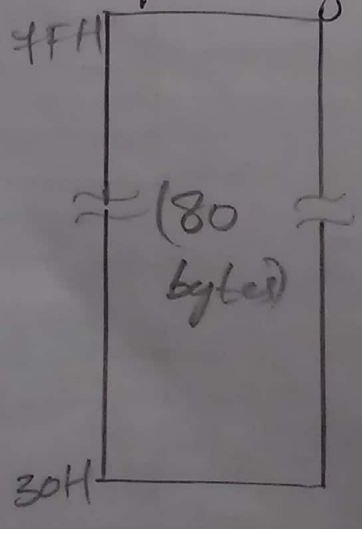


ROM Organisation:





103. General Purpose Registers:



Special Function Registers (128 bytes).
(80 - FFH)

~~Register~~ Bit Addressable Registers:
'*' represents which reg.'s are acting as bit addressable reg.'s.

In bit addressable reg.'s, each & every bit having separate location.

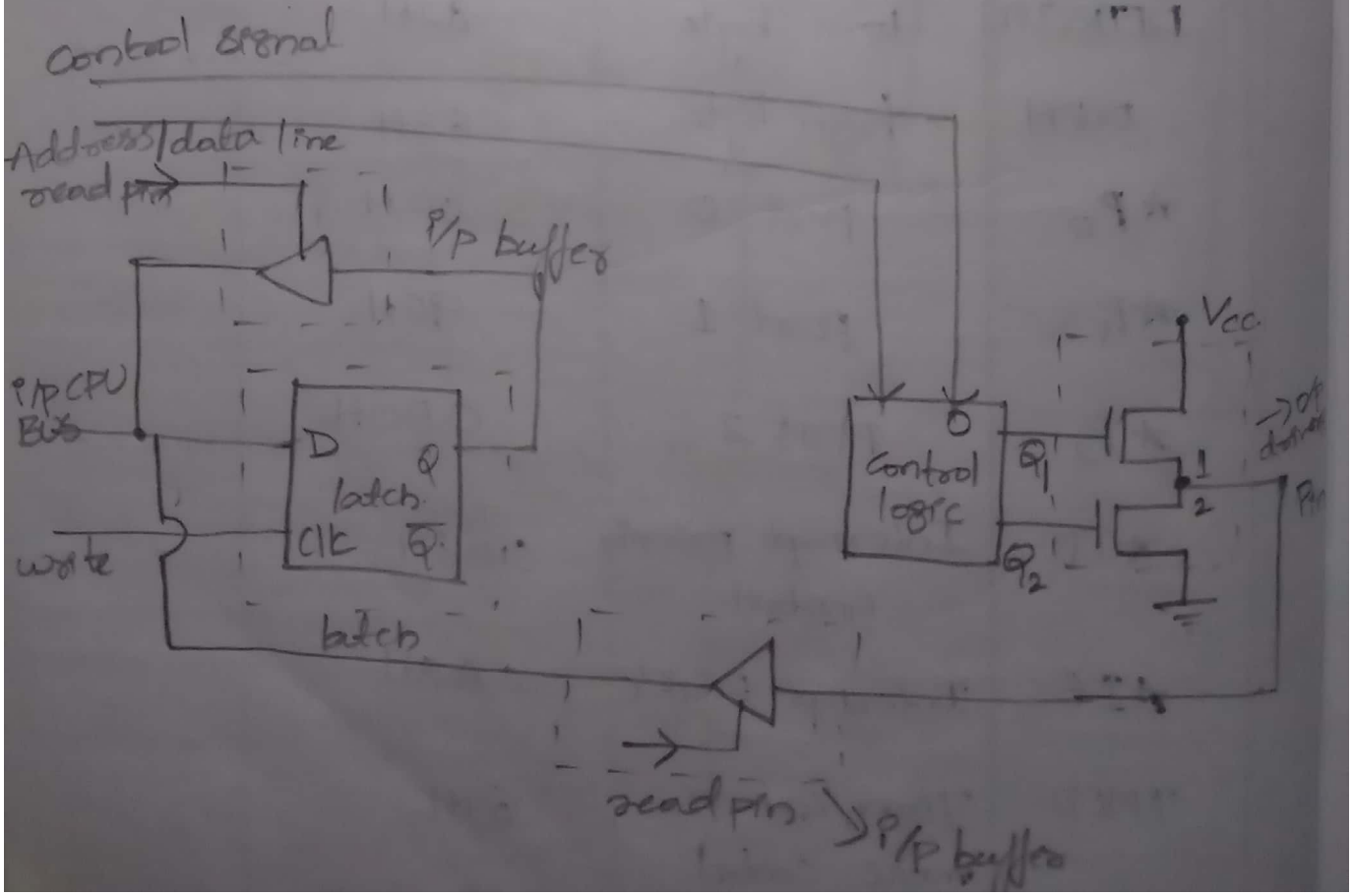
Register	Name	Address.
*ACC	Accumulator	0E0H | E0 | E1 | E2 | E3 | ... | E7
*B	B register	0F0.
*PSW	Program Status word.	0D0H.
SP	Stack pointer	81H | 81.0 | 81.1 | 81.2 | ... | 81.7
DPTR(DPL)	low byte	82H
DPH	high byte	83H
*P_0	port 0	80H
*P_1	port 1	90H
*P_2	port 2	0B0H
*IP	Interrupt priority control.	0B8H
*IE	Interrupt Enable	0A8H.
TMOD.	Timer/counter mode control	89H
*TCON	Timer/Counter control	88H.

TH0	Counter 0 high byte	8CH
TL0	Low byte	8AH.
TH1	Counter 1 high byte	8DH
TL1	Low byte	8BH.

SCON	Serial contol	98H
SBUF	Serial data buffer	99H
PCON	Power contol	87H.

11.3. Input and Output Ports:

Port 0: It acts as i/p & o/p and it consists of address & data lines.

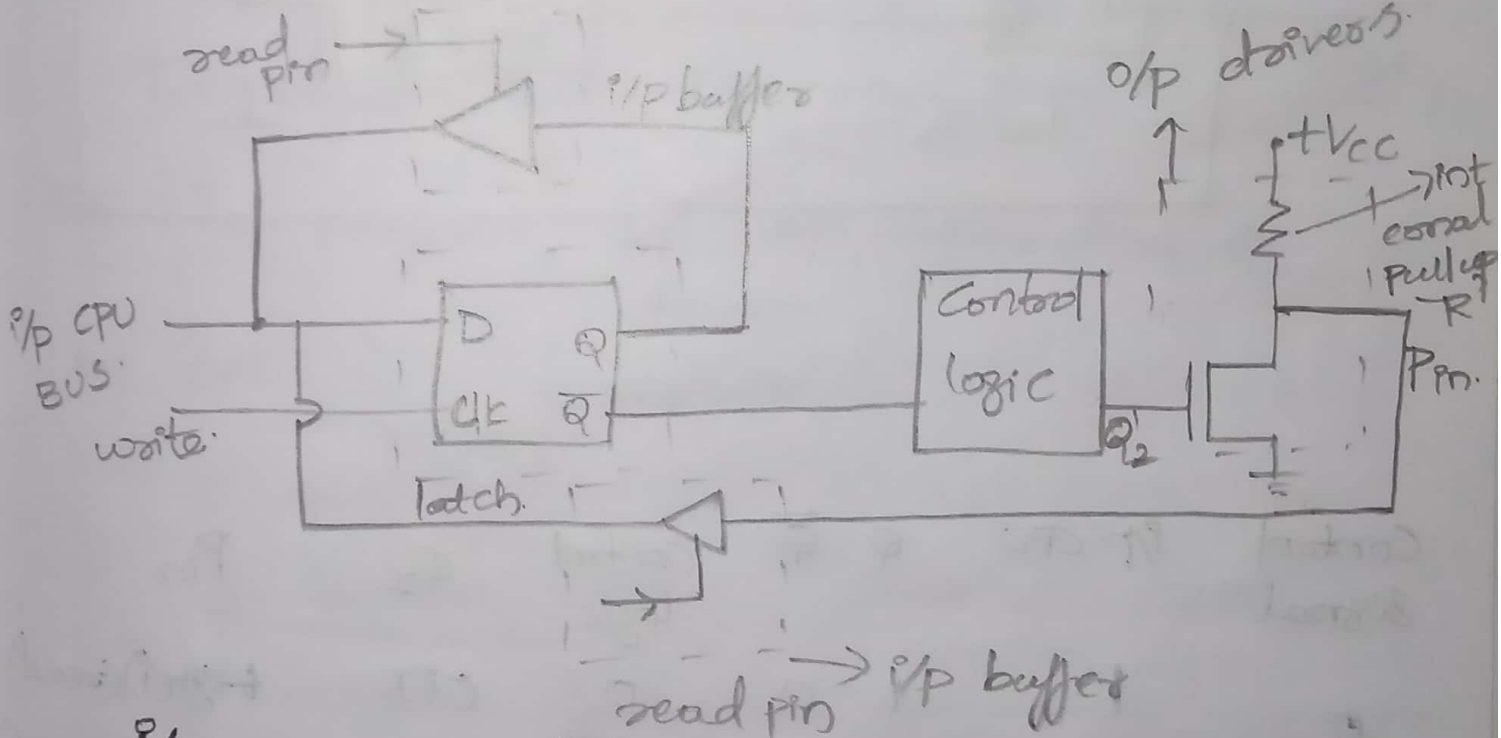


Control signal	I/p CPU BUS	Q	$\overline{Q}$	Control logic	Q_1	Q_2	Pin
0	1	1	0	0	OFF	OFF	High 'Z' State (1)
0	0	0	1	1	OFF	ON	Low

Address line	Q_1	Q_2	Pin
1	ON	OFF	ON (High) (Address)
0	OFF	ON	OFF (Low) (Data)

Port-1:

Port-1 is acting as only i/p & o/p.

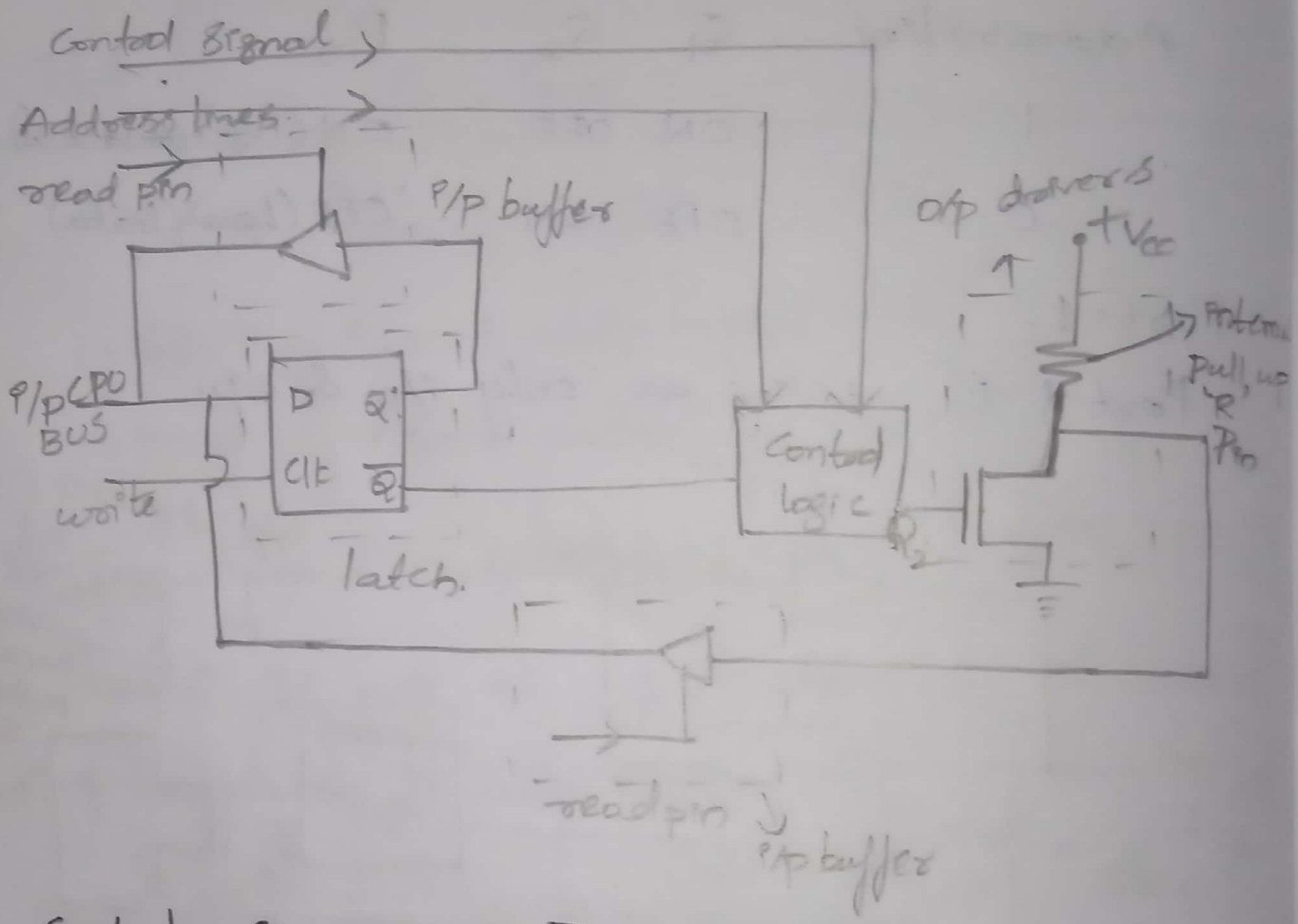


I/p CPU BUS	Q	$\overline{Q}$	Q_2	Pin
1	1	0	OFF	High (read)
0	0	1	ON	Low (write)

Port-2:

Port-2 is acting as I/p & O/p.

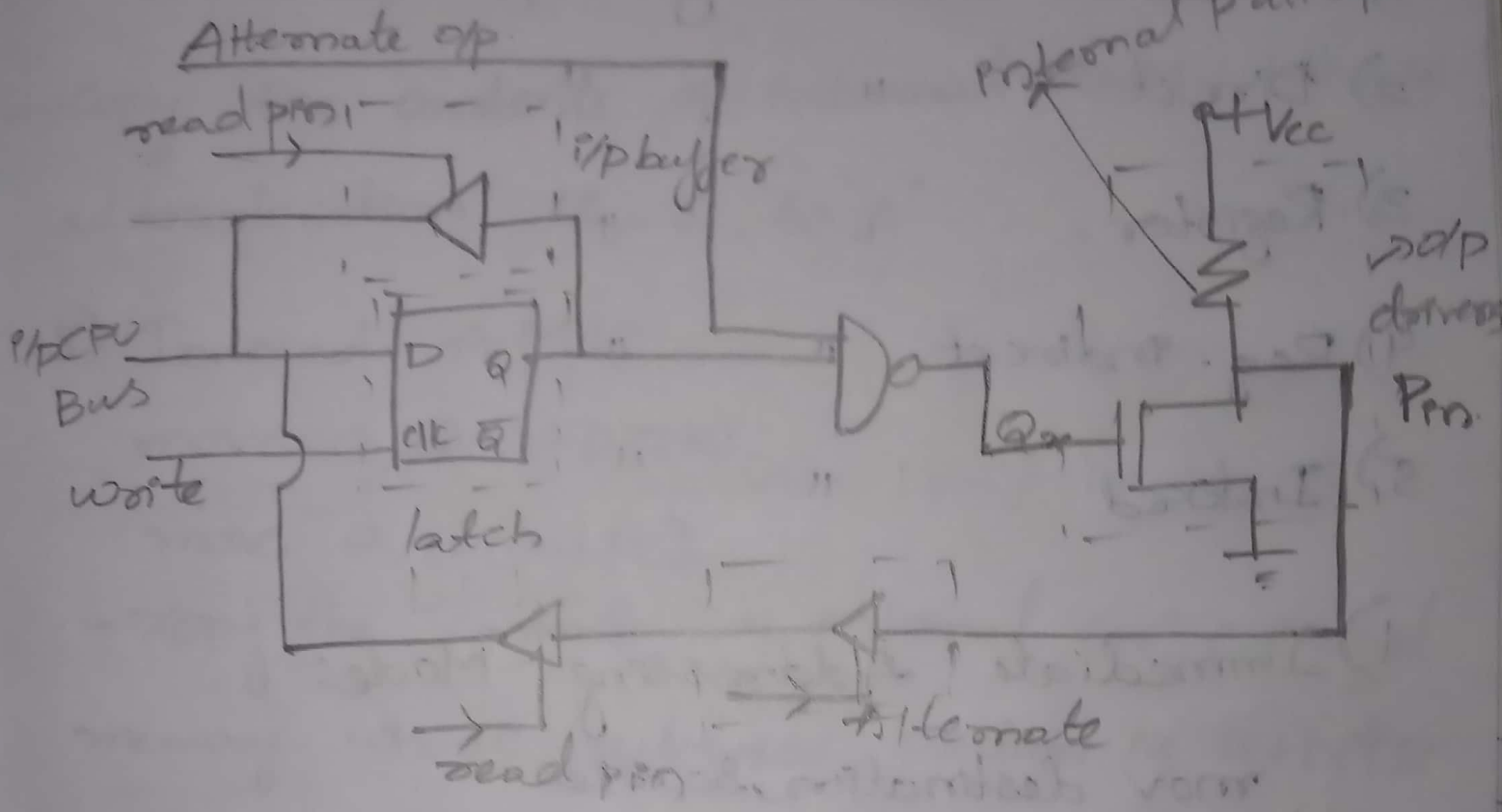
It contains higher order address lines.



Control Signal	I/p CPU	Q	$\bar{Q}$	Control logic	Q_2	Pin.
0	1	1	0	0	OFF	High(1)(read
0	0	0	1	1	ON	Low(0)(write

Address lines	Q_2	Pin
0	ON	Low.
1	OFF	High

Port-3: Port-3 is acting as i/p & o/p.
It consists alternate i/p-o/p pins.



To perform alt. i/p operation, Q_2 will be OFF. To perform alt o/p operation, Q_2 will be ON.

Addressing Modes:

1) Immediate Addressing mode
2) Direct " "
3) Register " "
4) Reg. indirect " "
5) Indexed " "

1) Immediate Addressing Mode:

```
mov destination, source
mov a, #59H
mov r0, #38H.
```

→ Copy 59H to accumulator.

2) Direct A.M:

```
mov a, SBUF        mov a, TMOD.
mov a, mem
```

→ Copy data from SBUF to accumulator.

3) Register Direct A.M:

```
mov a, r0
mov r1, a
```

→ Copy contents of 'r0' to 'a'.

4) Register Indirect A.M:

```
mov a, @r0
```

→ Copy the contents of internal RAM whose

address location is r_0 to 'A'.

movx a, @r_1.

→ Copy the contents of external RAM whose address location is r_1 to 'A'.

5) Indexed A.M:

movc a, @(a+DPTR)

movc a, @(a+PC).

→ Copy the contents of internal (or) external memory whose address location is a+DPTR (or) a+PC to 'A'.

Prog. Counter

Instruction Set of 8051:

1) Data transfer instructions
2) Arithmetic instr's.
3) Byte level instr's.
4) Bit level instr's (Boolean instr's)
5) Program execution transfer control instr's.

1) Data Transfer Instructions:

(internal, external, all addressing mode instr's.)

mov a, #55H

mov 30H, @R0

mov 8AH, 70H { Spl. Fn. Reg TL0

→ The data which is stored in the 70 address location will be copied to Spl. fn. reg. TL0.

Push:

mov SP, #60H. {SP is denoted at 60H}

mov R_2, #55H { copy the data to R_2}

PUSH 02 {SP is incremented by 1, 55H inserted into stack}.

POP 40H {55H deleted and it moves to 40H address location}.

Exchange Instructions:

XCH destination, source

XCH R_3, R_4.

XCH 70H, @r0 {exchange data b/w internal memory whose address location r0 and address location 70H}

Exchange Digit Instr's:

XCHD destination, source

XCHD r0, r1.

→ Lower nibbles will be exchanged, upper nibble will be same.

r0 = 1 [2] r1 = 3 [4] → r0 = 3 [2] r1 = 1 [4]

Arithmetic Operations:

Addition:

Add destination, source.

Add a,b.

Add a,#59H.

Add r_0, r_1.

Increment operations:

INC destination

INC A {incremented by 1}.

Addition with Carry:

ADDC destination, source.

mov r_1, #59H

mov r_2, # 32H.

ADDC r_1, r_2.

Decimal Adjust after addition:

DA destination

mov r_1, #59H

mov r_2, # 32H

ADD r_1, r_2.

DA A

Subtraction:

Borrow

SUBB destination, Source

mov a, #70H

SUBB a, #59H.

Decrement Instruction:

DEC destination.

mov R4, 20H

DEC R4 {20H-1 = 1FH}.

Multiplication Instruction:

mov A, #09H.

mov B, #01H

MUL AB.

Division Instruction:

mov A, #02H

mov B, #01H.

DIV AB.

Byte Level Instructions:

There are 2 types.

1) Logical Instr's
2) Rotate & Swap.

Logical Instructions:

(1) AND Logical:

Anl destination, source

Ex: mov a, #20H

mov r0, #31H

Anl a,r0.

0010 0000
0011 0001
0010 0000

=>20

(2) OR.

ORL destination, source.

```
mov a,#21H
mov b,#02H
ORL a,b.
```

a = 0010 0001
b = 0000 0010
a = 0010 0011
a = 23H.

XOR:

(1) XRL destination, source

```
mov a,#20H
mov R0,#31H
XRL a,R0.
```

(2) Clear Instructions:

CLR destination

CLR A {'A' will be set to zero}

(3) complement Instr's:

CLP destination

CLP A

(4) No operation Instr's:

NOP destination

Nop A

Rotate & Swap Instructions.

1) RL - Rotate Left;

2) RLC - Rotate Left with Carry;

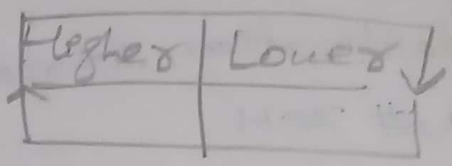
3) RR - Rotate Right.

4) RRC - Rotate Right with Carry.

Swap Instructions:

Swap destination.

Swap A



mov a, #02H

(Lower & higher nibbles)

Ex: Swap a

a=20H.

Bit Level Instructions:

1) AND Instr's:

ANL destination, source

ANL C,b {performs 'and' operation b/w specified bit & Carry flag}

ANL C,/b {performs 'and' operation b/w complement of specified bit & CF}

2) OR instructions:

ORL C,b

{performs the 'OR' in b/w specified bit & CF}.

ORL C,$\overline{b}$ {b/w ~~Spe~~ complement of Specified bit & CF}.

3) Clear Instruction:

CLR b {clears the Specified bit} =0.

CLR C {clears the CF}. CF=0

4) Complement Instruction:

CLP b { complement of Specified bit}

CLP C { " " CF}.

5) No Operation:

NOP b

NOP C

6) SET bit operation:

SETB b {set b to 1}

SETB C {set CF=1}.

7) MOV:

MOV b,C

{CF copied to Specified bit b}.

Prog. Execution Transfer Control Instn's:

DJNZ (decrement & jump if not zero.)

CJNE (compare & jump if not equal.)

Interrupt instn. RETI (Return from interru

(Conditional & uncondi instn's as same as 808

Unconditional Transfer instructions

1) Jmp : Jump:

Syntax: Jmp @A+DPTR

opes:- This instruction will come 8051 to fetch next instruction from an address, which is by addition accumulator and DPTR contents.

2) SJmp: short Jump:

syntax: SJmp destination

it fectch next instruction from an address with in the range of 128 to 127

3) AJmp: Absolute Jump.

Syntax: AJmp destination

it fetch next instruction from an address in the same page.

4) LJmp : Long Jump

Syntax: LJmp destination

it fetch next instruction from anywhere in the 64KB area of program memory.

5) ACLL: Absolute call

Syntax: ACALL Destination

It is used to transfer execution program to subroutine specified by the destination

RET: Return

Syntax: RET

This instruction is used to return to main program from a subroutine.

conditional transfer instructions:-

→ JC → jump if carry

Syntax: JC (related symbol) {JC rel}.

→ JNB:- Jump No bit

Syntax:- JNB b, rel
JNB b, Label1

→ JZ: Jump if accumulator is zero

syntax: Jz rel → E.g:- Jz A
(destination)

→ JB: Jump if direct bit is set

syntax: JB bit, rel
(source)

→ JC: Jump if carry flag is set

sytax: JC rel

→ JNC: Jump if carry flag is not set

Syntax: JNC rel

write rel
∴ is nothing
but for condition
we are checking
condition for code
rel → represents
(code).
rel → destination
(assume)