COEP TECHNOLOGICAL UNIVERSITY, PUNE
Wellesley Road, Shivajinagar, Pune-411 005

8th March '23

Name: Abhishek Shinde.
MIS: 112103134
Division: 2 Batch: S3

Experiment - 1

- **Aim:** Study of 8086 based microcomputer trainer kit.

- **Objectives:**

A] To lacate CPU8086, 74373 devices, 4.00 MHz piezoelectric crystal, 8284 clock generator, RAM and ROM devices on the microcomputer board.

B] Find total onboard memory capacity.

C] Use and describe following commands:
D, E, R, T, G and U and A command.
Refer to user manual of 8086 based microprocessor trainer kit.

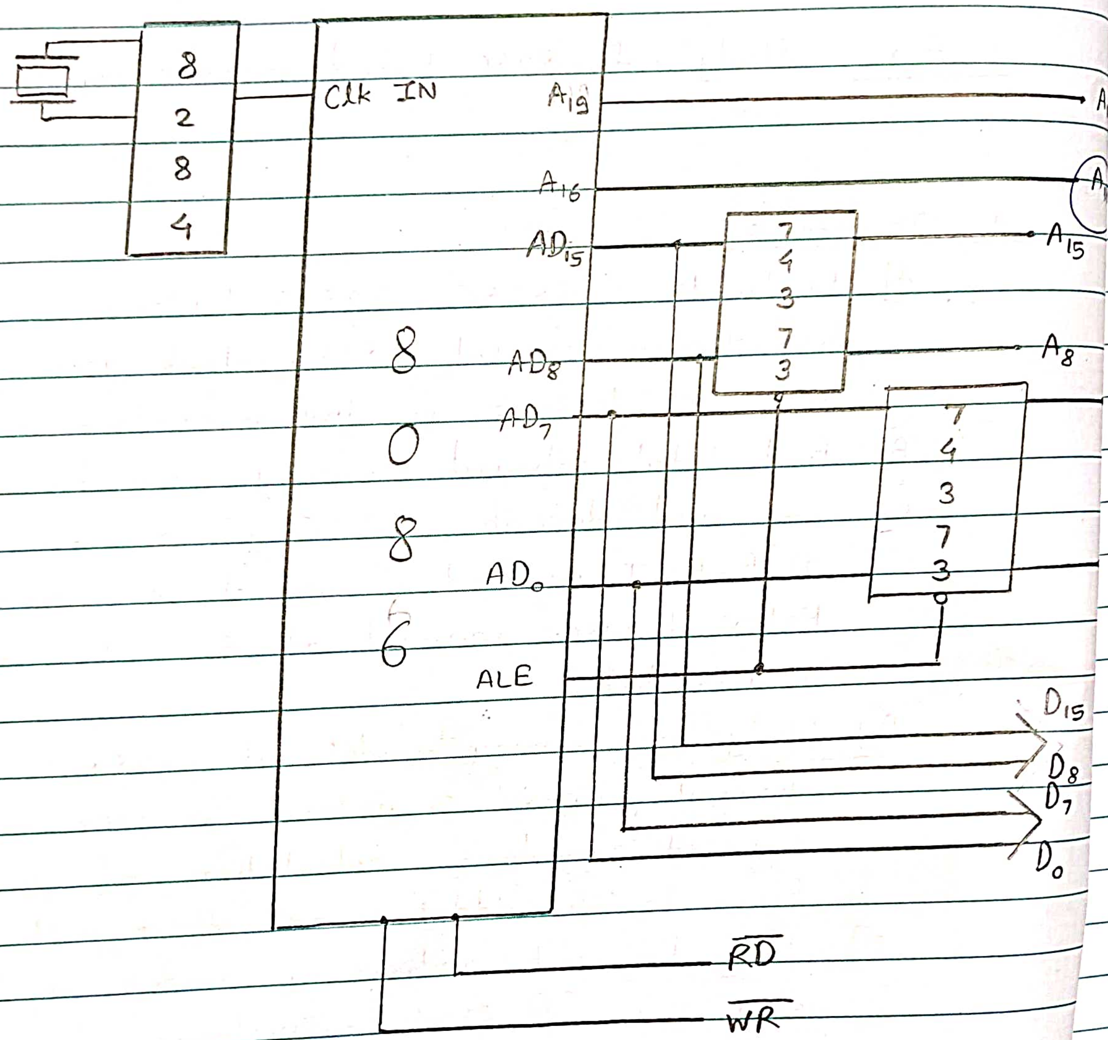
D] Study programming model of 8086:
general purpose registers, pointer and base registers, segment registers, calculation of physical address from logical address or offset and segment address.

E] Study different addressing modes of 8086 by using relevant MOV instructions:
Register addressing, immediate addressing, register indirect addressing, indexed addressing, based and indexed addressing with and without offset.

F] Find opcodes for the instructions in E above using instructions templates of 8086.

• **Theory:**

A] **Microprocessor having CPU 8086, 74373 devices, 8284 clock generator**



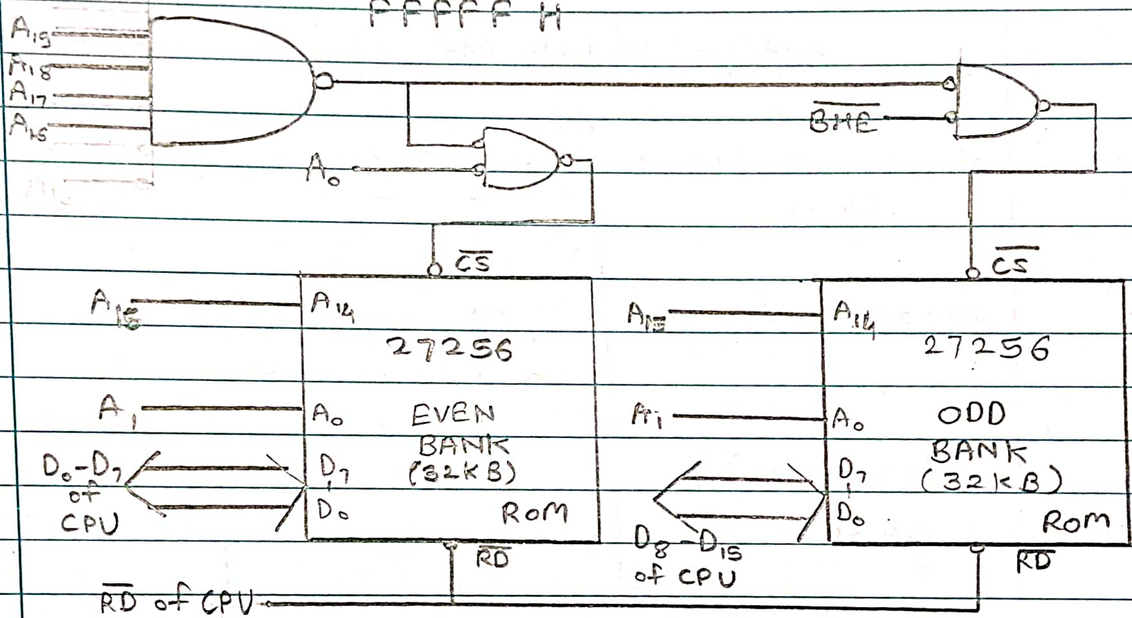
COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Odd and even banks of ROM

F0000 H

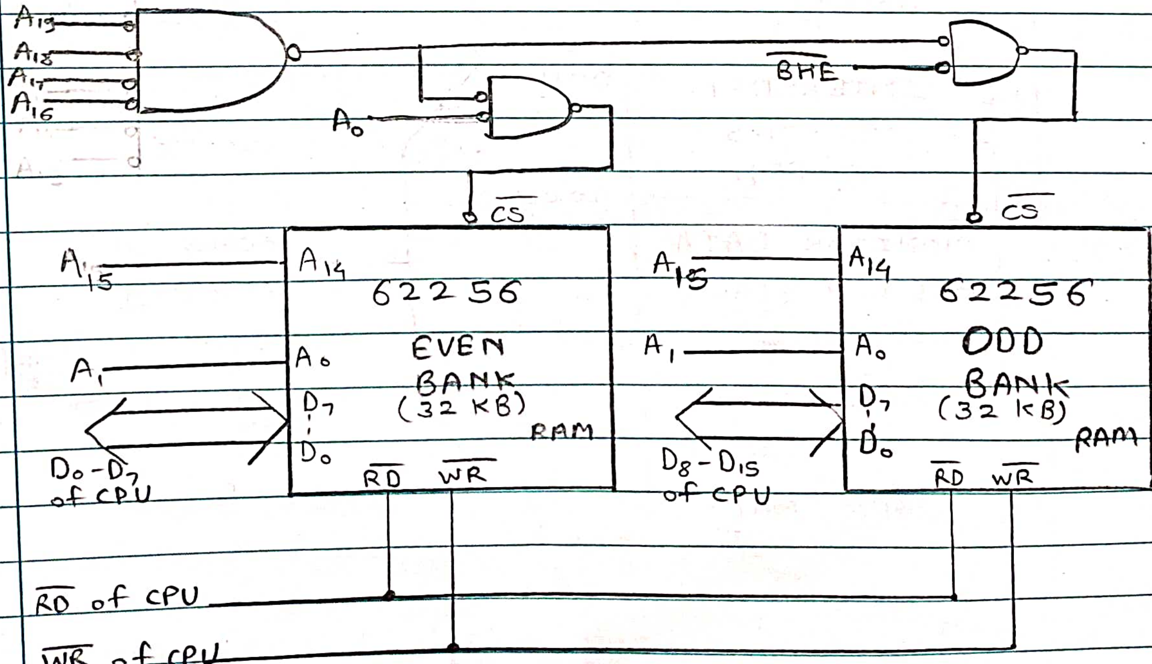
FFFFF H



odd and even banks of RAM:

00000H

0FFFFH



B] • The memory map of DYNA - 86L is shown in the following table :

Address	Socket No.	chips total capacity
00000 - 0FFFF H 64 KB	U31/U33	Battery-backe 62256 SRAM
40000 - BFFFF H	CS2 signal on J7 connector	User Expansion
F0000 - FFFFFH 64 KB	U30/U32	Firmware EPROM 27256

• The memory map is as below :
(the user program should start from 0000:0700 H above).

INTERRUPT VECTORS 0-FFH	00H
MONITOR DATA AREAS / STACK	400H
USER STACK	600H
USER RAM	700 H
	FFFF

→ Reserved Memory 0:0 to 0:6FF

* Total onboard capacity is : 32 + 32 = 64 Kbytes
and 32 + 32 = 64 Kbytes

c] Commands and their functions:

1. D - display command.
Displays the block of memory.

e.g.: > D 0:1000
⇒ 8B C3 00

2. E - Edit memory
Modify the content of memory.

e.g.: > E 0:3000
EE

3. R - Register
Examine/ modify register

e.g.: > R
⇒ shows contents of all registers.

4. T - Trace
Single Step execution

e.g.: > T : 1000

5. G - Go
Transfer 8086 control from monitor to the used program.

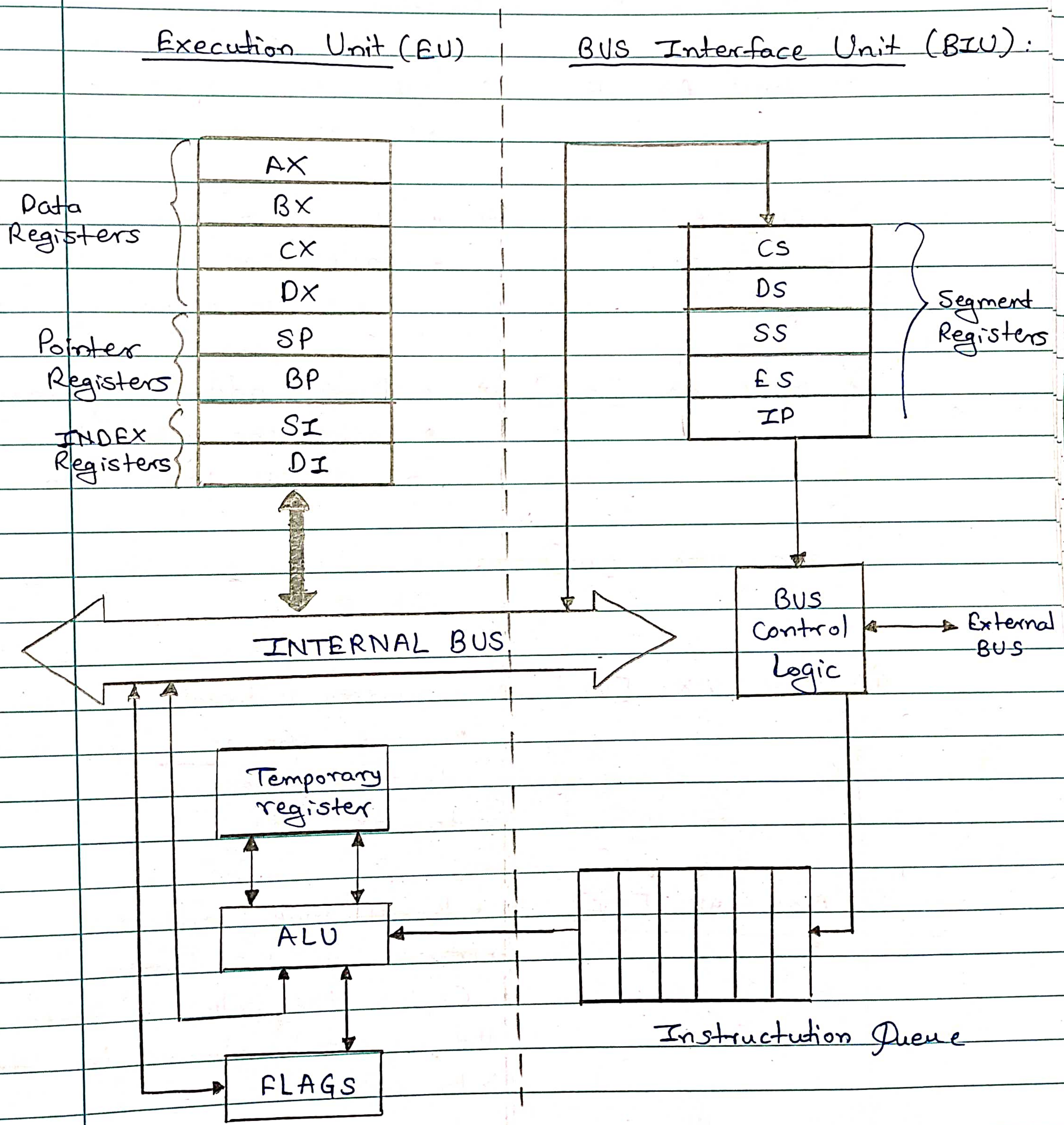
⑥ U – Unassembled
Used to unassemble a memory block.

e.g.: > U : 1000
⇒ Show instruction / data stored at that l

⑦ A – Assembled
Used to assemble to memory A [[seg:] off

e.g.: > A 0:1000
MOV AX, BX
⇒ stored above instruction to that location

D] Internal architecture of 8086 microprocessor :



E] Different addressing modes of 8086 by using relevant MOV instructions :

1) MOV AX, BX → • Register addressing mode
• Copies the data of BX to

2) MOV AX, 1000 → • Immediate addressing mode
• Copies the ~~hexa~~decimal value 1000 to register AX

3) MOV AX, [1000] → • Direct addressing mode.
• Copies the value in memory at address 1000 into AX.

~~4) MOV AX, [BX] → • Based index addressing mod with BX~~

4) MOV AX, [BX] → • Register indirect addressing mo
• Copies the value in memory at the address contained in B to AX.

5) MOV AX, [SI] → • ~~Based~~ index addressing mode ~~with BX as the base register~~
• Copies the value in memory a the address contained in ~~BX~~ into AX.

6) MOV AX, [BP] → • Based index addressing mode with BP as the base register.
• Copies the value in memory at address contained in BP into AX.

7) MOV AX, [BP+SI+4] → • Based index addressing mode with BP and SI as a base register.
• Copies the value in memory at the address contained in BP+SI+4 into AX.

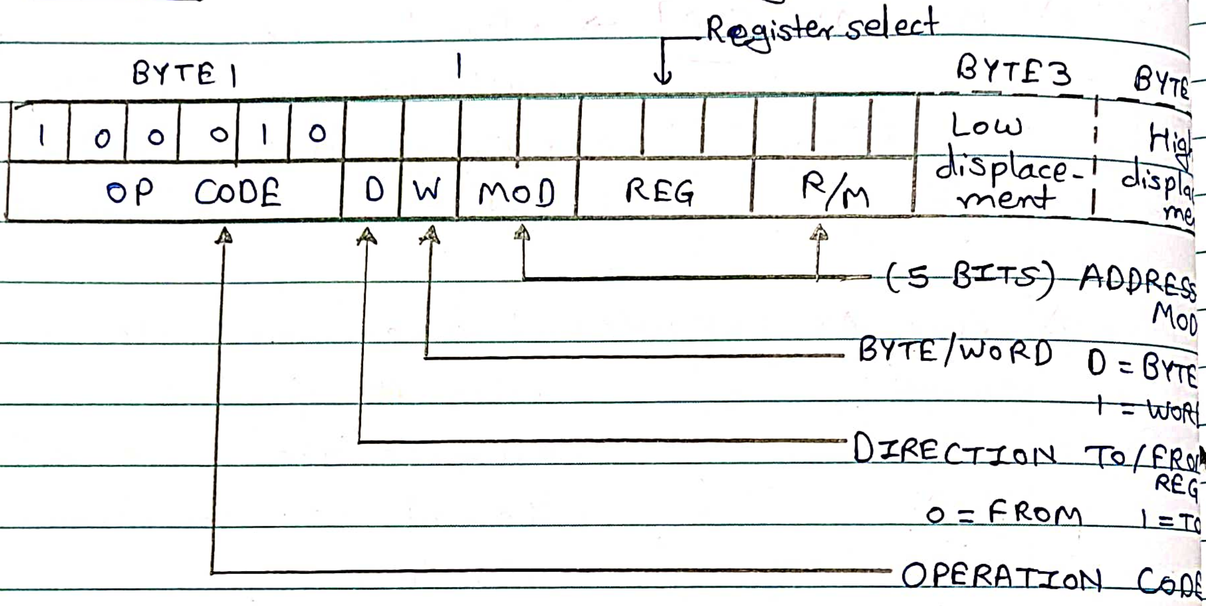
8) MOV [BX], AX → • Register indirect addressing mode.
• Copies the value in AX into the memory location contained in BX.

• Program example:

① 0100 : 0000 B8 ~~22~~ 22 MOV AX, 2222h

This program uses MOV opcodes with an immediate addressing mode to load the value 2222h into the AX register.

F] Example: MOV instruction coding format:



1) MOV AX, BX ⇒ Register to register

~~100010 011 11 011 000~~

$\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{\text{OPCODE}}\ \underbrace{1}_{D}\ \underbrace{1}_{W}\ \underbrace{1\ 1}_{\text{MOD}}\ \underbrace{0\ 0\ 0}_{\text{REG}}\ \underbrace{0\ 1\ 1}_{R/M}$

⇒ 8BC3

2) MOV AX, 1000H ⇒ Immediate ~~Memory~~ to register

$1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ \underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{LOW}}\ \underbrace{0\ 0\ 0\ 1\ 0 \ldots}_{\text{HIGH}}$

⇒ B8 00 10

3) MOV AX, [1000H] ⇒ Memory to register

$\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{\text{OPCODE}}\ \underbrace{1\ 1}_{DW}\ \underbrace{0\ 0}_{\text{MOD}}\ \underbrace{0\ 0\ 0}_{\text{REG}}\ \underbrace{1\ 1\ 0}_{R/M}\ 0000\ 0000\ 0001 \ldots$

⇒ ~~A1 00 10~~

⇒ 8B 06 00 10

4) MOV AX, [BX] ⇒ Register based indirect

$$\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{OPCODE}\ \underbrace{1}_{D}\ \underbrace{1}_{W}\ \underbrace{0\ 0}_{MOD}\ \underbrace{0\ 0\ 0}_{REG}\ \underbrace{1\ 1\ 1}_{R/M}$$

⇒ 8B07

5) MOV AX, [SI] ⇒ Indexed addressing

$$\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{OPCODE}\ \underbrace{1}_{D}\ \underbrace{1}_{W}\ \underbrace{0\ 0}_{MOD}\ \underbrace{0\ 0\ 0}_{REG}\ \underbrace{1\ 0\ 0}_{R/M}$$

⇒ 8B04

6) MOV AX, [BP + SI + 4] ⇒ Based index addressing

$$\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{OPCODE}\ \underbrace{1}_{D}\ \underbrace{1}_{W}\ \underbrace{0\ 1}_{MOD}\ \underbrace{0\ 0\ 0}_{REG}\ \underbrace{0\ 1\ 0}_{R/M}\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0$$

⇒ 8B4204

*** Conclusion:** Thus we studied the architecture of 8086 microprocessor and its components. Also, we used assembly language to carry out various instructions and find their opcode. We also studied various addressing modes.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Name: Abhishek Shinde.

MIS : 112103134

Division: 2 Batch : S3

Experiment - 2

• Aim: A] Complete first five units of tutorials of EMU86.

B] Study instructions of 8086 :

i) Data transfer group.

ii) Arithmetic group.

iii) Logical group.

iv) Program control transfer group.

v) String instructions.

C] Write a program in assembly language of 8086 to add two 32 bit numbers. First number is stored from 0000:2000, second number is stored from location 0000:3000. Store the result from 0000:4000 onwards.

D] Write a program in assembly language of 8086 to transfer a block of memory. Starting address of source block is 0000:2000. Starting address of destination block is 0000:3000. Assume length of the block is stored in CX register.

A] Assemble directives are the instructions to the assembler, linker or loader. They are not part of processor instruction set.

(i) **ORG:** • The assembler ~~#~~ uses a location count to account for its relative position in d or code segment.

• format: ORG Expression

• e.g.: ORG 1000H

This sets the location counter to

• That means, the ORG directive directs assembler to start the memory allotmen for the particular segment from the d address in the ORG statement.

(ii) **END:** • The END directive is put after the last statement of the program to tell the assembler that this is the end of the program.

• The assembler ignores any statement a END directive.

• e.g.:

```
MOV AL, 02H
MOV BL, 04H
ADD AL, BL
END
```

(iii) **RET** :
- It is an instruction used at the end of procedures or subprograms or subroutines.
- This instruction transfers the execution to the caller program.

B

i) Data Transfer Group :

(1) MOV destination, Source

- The MOV instruction copies a word or a byte of data from some source to a destination.
- The destination can be a register or a memory location.
- The source can be a register, memory location or an immediate number.
- MOV instruction doesn't affect any flags.

e.g.: (i) MOV AX, 2000H ; loads the immediate number
; 2000H in BX register.

- Before execution : AH → 00 AL → 00
- After execution : AH → 20 AL → 00

(ii) MOV [3000H], BX ; copies the contents of BX
; register to the location
; 3000 and 3001.

- Before execution : BH → xx BL → yy
- After execution : Location 3000 → xx
Location 3001 → yy

② XCHG destination, source

- The XCHG instruction exchanges the conte[nts] of a register with the contents of an[other] register or the contents of a register [with] a memory location.
- Segment registers can't be used in the[se] instructions.

- e.g.: XCHG BX, CX ; Exchanges word in
  ; with word in CX

ii) Arithmetic Group :

① ADD destination, source

- This instruction adds a number from sou[rce] to a number from destination and puts the result in destination.
- Flags affected → AF, CF, OF, PF, SF, ZF

- e.g.: ADD AL, 20H ; Adds immediate n[umber]
  ; 20H to contents [of AL]
  ; and stores the ad[dition]
  ; in AL register.

② INC destination

- This instruction adds 1 to specified destina[tion]
- The destination may be a register / memory location

- Flags affected → AF, OF, PF, SF, ZF.
- e.g.: INC AL ; Adds 1 to contents
  ; of AL register

- Doesn't affect carry flag. If contents of 8-bit register are FFH and that of 16-bit register are FFFFH, INC instruction makes contents equal to zero.

iii) Logical Group:

① AND destination, source

- This instruction logically AND's each bit of source byte or word with corresponding bit in destination and stores result in destination.
- The source may be register or memory location.
- The destination may be ~~restio~~ register or memory location.
- Flags affected → CF and OF are 0.

- e.g.:

```
MOV AL, 90H
MOV BL, 25H
AND BL, AL ; AND byte in BL with
           ; byte in AL, store
           ; the result in BL.
```

② OR destination, Source

- This instruction logically ORs each bit of source byte /word with the corresponding in the destination and stores result in destination.
- The source and destination both can't memory locations in same instruction
- Flags affected → CF, OF, PF, SF, ZF.
  CF = 0, OF = 0.
- e.g.:
```
MOV AL, 90 H
MOV BL, 25 H
OR  AL, BL   ; OR byte in BL w
             ; byte in AL and st
             ; result in AL.
```

iv) Program Control Transfer Group :

① CALL instruction :

- This instruction is used to transfer the exe to a subroutine.
- There are two basic types of CALL : ne and far.
- This instruction involves stack operations store the return address.
- e.g.:
```
CALL PRO   ; PRO is name of the sub
           ; routine. The assemble
           ; determines displacement
           ; transfers execution to P
           ; and pushes RA in stack
           ; which was stored in IP.
```

② Conditional JUMP →

- These instructions cause jump to a label in given instruction.
- e.g.: JNZ BACK ; This instruction makes the execution go to the instruction having lable 'BACK'.

- Types → JNZ, JZ, JNC, JC, JNO, JNP

v) String Instructions:

① MOVS instruction → (MOVB / MOVW) →

- This instruction copies a byte or word from a location in data segmen to a location in extra segment.
- From DS: SI to ES: DI.
- For multiple byte or word moves, the number of elements to be moved is put in CX register.
- e.g.:

```
        CLD          ; Clear direction flag to auto-
                     ; increment SI & DI
MOV AX, 0000H
MOV DS, AX           ; Initialise data segment to 0.
MOV ES, AX           ; Initialise extra sement to 0.
MOV SI, 2000H
MOV DI, 4000H
MOV CX, 05H
REP MOVSB            ; Decrement CX & execute
                     ; MOVSB until CX will be 0.
```

② CMPS instruction :→

- This instruction can be used to compare a in one string with a byte in another st
- Works with DS : SI and ES : DI .
- The comparison is done by subtracting b or word pointed by DI from byte or w pointed by SI.

C]

Address	Hexcodes	Label	Opcode	Operands	Comments
0000	B80000		MOV	AX,0000H	Copy the number 0000H in AX.
0003	8ED8		MOV	DS, AX	Initialize the DS segment from 0000H.
0005	A10020		MOV	AX,[2000H]	Copy the contents at location 2000H into AX register (16-bits of 1st 32-bit number)
0008	8B1E0220		MOV	BX,[2002H]	Copy next 16-bits of 1st number in BX register
000C	8B0E0030		MOV	CX,[3000H]	Copy 16-bits of 2nd number in CX register
~~0010~~ 0010	~~8B160230~~ 8B160230		~~MOV~~ MOV	DX,[3002H]	Copy net 16-bits of 2nd number in DX register.
0014	03C1		ADD	AX, CX	Add contents in CX with AX and store in AX register
0016	13DA		ADC	BX, DX	Add contents in DX with BX and carry and store in BX register

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

0018	A30040		MOV	[4000H], AX	Copy the resul of lower 16-bi to the location 4000H and 400
001B	891E0240		MOV	[4002H], BX	Copy the resul of higher 16-b to the locatio 4002H and 400

• Program:

```
org 100h

mov AX, 0000H
mov DS, AX

Mov AX, [2000H]
Mov BX, [2002H]
Mov CX, [3000H]
mov DX, [3002H]

ADD AX, CX
ADC BX, DX

Mov [4000H], AX
Mov [4002H], BX
ret
```

D]

```
; Program to transfer block of memory from
; 02000H to 03000H and count is stored in CX

org 100h

Mov AX, 0000H
MOV DS, AX
Mov ES, AX
Mov SI, 2000H
mov DI, 3000H

CLD
REP MOVSB

RET
```

Address	Hexcode	Label	Opcode	Operand	Comments
07100	B80000		MOV	AX, 0000H	Store the value 0000H in AX
07103	8ED8		MOV	DS, AX	Initialize DS with value in AX
07105	8EC0		MOV	ES, AX	Initialize ES with value in AX
07107	BE0020		MOV	SI, 2000H	Store value 2000H in SI i.e. make SI point to start of block to transfer
0710A	BF0030		MOV	DI, 3000H	Make DI point to destination address

Address	Hexcode	Label	Opcode	Operand	Comments
0710D	FC		CLD		Clears DF
0710E	F3 A4	REP	MOVSB		Reapetedly copy data from DS:SI to ES:SI till CX becomes zero. Also decrement CX and increment both SI & DI
07110	C3		RET		Return control to O.S.

- Conclusion:

We learnt various instruction of 8086 and also learnt to write basic programs like adding two 32-bit numbers and transferring a block of memory using string instructions.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Name: Abhishek Shinde.
MIS: 112103134
Division: 2 Batch: S3

EXPERIMENT - 3

- Aim: Write an assembly language program in 8086 using EMU86.

A] As a subroutine to add two 32-bit numbers when operand pointers are initialised in main program.
First number is stored from 1000:1000H
Second number is stored from 1000:2000H
Result can be stored from 1000:3000H.

B] Use this subroutine in A above in the main program to add two series of 32-bit numbers. Store the result from 1000:3000H onwards. If carry is generated in double-word addition, store it at third location in sequence of the result.

C] Write a program to add FFH with 01H, analyse the result for reflected changes ~~on~~ in all flag bits.

D] Write a program to read all flags i.e. to read flag register.

A] • Program :

```
; Program to add two 32-bit numbers using
; sub-routine named 'SUBADD' :

org 100h

MOV  AX, 1000H
MOV  DS, AX
MOV  SI, 1000H
MOV  DI, 2000H
MOV  BX, 3000H

CLC
CALL SUBADD

ret

SUBADD:  MOV AX, [SI]
         ADC AX, [DI]
         MOV [BX], AX
         INC SI
         INC SI
         INC DI
         INC DI
         INC BX
         INC BX
         MOV [AX], SI
         ADC [AX], DI
         MOV [BX], AX
ret
```

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Address	Hexcodes	Label	Opcode	Oprand	Comments
07100	B80010		MOV	AX, 1000H	Store the value 1000H in AX
07103	8ED8		MOV	DS, AX	Initialize DS with value in AX
07105	BE0010		MOV	SI, 1000H	Store starting address of first number in SI
07108	BF0020		MOV	DI, 2000H	Store starting address of second number in DI
0710B	BB0030		MOV	BX, 3000H	Store destination address in BX
0710E	F8		CLC		Clear the CF.
0710F	E80100		CALL	SUBADD	Call subroutine SUBADD
07113	8B04	SUBADD	MOV	AX, [SI]	Store the first 16-bits of first number in AX
07115	1305		ADC	AX, [DI]	Add 16-bits of second number and AX with carry
07117	8907		MOV	[BX], AX	Store the result in AX at the memory location pointed by BX.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Address	Hexcodes	Label	Opcode	Operand	Comments
07119	46		INC	SI	Increment SI
0711A	46		INC	SI	Increment SI
0711B	47		INC	DI	Increment DI
0711C	47		INC	DI	Increment DI
0711D	43		INC	BX	Increment BX
0711E	43		INC	BX	Increment BX
0711F	8B 04		MOV	AX, [SI]	Store next 16-b of first number in AX
07121	13 05		ADC	AX, [DI]	Add next 16-b of second num and AX with c
07121	8907		MOV	[BX], AX	Store the result in AX at memory location pointed by BX
07125	C3		RET		Return control to O.S.

B]

```
; Program to add two series of 32-bit numbers
  using Subroutine named 'SUBADD'.
; Assuming there are 2 numbers in each series.

org 100h
MOV AX, 1000H
MOV DS, AX
MOV SI, 1000H
MOV DI, 2000H
MOV BX, 3000H
XOR AX, AX
MOV CL, 02 H

BACK1:  CLC
        CALL SUBADD
        DEC CL
        JNZ BACK1
RET

SUBADD:  MOV AX, [SI]
         ADD AX, [DI]
         MOV [BX], AX
         INC SI
         INC SI
         INC DI
         INC DI
         INC BX
         INC BX
         MOV AX, [SI]
         ADC AX, [DI]
         MOV [BX], AX
```

```
INC SI
INC SI
INC DI
INC DI
INC BX
INC BX

JC LABEL1
MOV [BX], 00H
JMP LABEL2
LABEL1: MOV [BX], 01H
LABEL2: INC BX
RET
```

Address	Hexcodes	Label	Opcode	Operand	Comments
07100	B80010		MOV	AX, 1000H	Store the value 1000H in AX
07103	8ED8		MOV	DS, AX	Initialize DS with value in A
07105	BE0010		MOV	SI, 1000H	Store starting address of first series in SI
07108	BF0020		MOV	DI, 2000H	Store starting address of secon series in DI
0710B	BB0030		MOV	BX, 3000H	Store destinatio address in BX
0710E	33C0		XOR	AX, AX	Make contents of AX 0000H.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Address	Hexcodes	Label	Opcode	Operand	Comments
07110	B1 02		MOV	CL, 02H	Store number/ count of numbers in each series
07112	F8	BACK1	CLC		Clear the CF
07113	E80500		CALL	SUBADD	Call the subroutine 'SUBADD'.
⋮					
0711B	8B04	SUBADD	MOV	AX, [SI]	Store 16-bits of ~~first~~ number in series 1 into AX
0711D	0305		ADD	AX, [DI]	Add 16-bits of number in series 2 into AX
0711F	8907		MOV	[BX], AX	Store the result at location pointed by BX
07121	46		INC	SI	Increment SI
07122	46		INC	SI	Increment SI
07123	47		INC	DI	Increment DI
07124	47		INC	DI	Increment DI
07125	43		INC	BX	Increment BX
07126	43		INC	BX	Increment BX
07127	8B 04		MOV	AX, [SI]	Store next 16-bits of number in series 1 into AX
07129	1305		ADC	AX, [DI]	Add with carry next 16-bits of no. in S2. with that in AX.

COEP TECHNOLOGICAL UNIVERSITY, PUNE
Wellesley Road, Shivajinagar, Pune-411 005

Address	Hexcodes	Label	Opcode	Operand	Comments
0712B	8907		MOV	[BX], AX	Store the res. in AX at the location pointed by BX.
0712D	46		INC	SI	Increment SI
0712E	46		INC	SI	Increment SI
0712F	47		INC	DI	Increment DI
07130	47		INC	DI	Increment DI
07131	43		INC	BX	Increment BX
07132	43		INC	BX	Increment BX
07133	7205		JC	LABEL1	If CF=1 jump
07135	C60700		MOV	[BX],00H	store 00H in next location since carry is 0
07138	EB03		JMP	LABEL2	Jump to LABEL2.
0713A	C60701	LABEL1	MOV	[BX], 01H	Store 01H in next location since carry is 1
0713D	43	LABEL2	INC	BX	Increment BX
0713E	C3		RET		Return control to the caller
⋮					
07116	FEC9		DEC	CL	Decrement CL
07118	75F8		JNZ	BACK1	Jump to Back1 if CL ≠ 0
0711A	C3		RET		Return Control to the OS.

c]

```
; Program to add FFH and 01H :

org 100h

MOV  AX, 1000H
MOV  DS, AX

MOV  AL, 0FFH
MOV  BL, 01H
ADD  AL, BL
RET
```

Address	Hexcode	Label	Opcode	Operand	Comments
07100	B80010		MOV	AX, 1000H	Move value 1000H in AX register
07103	8E D8		MOV	DS, AX	Initialize DS with value in AX.
07105	B0 FF		MOV	AL, 0FFH	Store value FFH in AL
07107	B3 01		MOV	BL, 01H	Store value 01H in BL
07109	02C3		ADD	AL, BL	Add value stored in BL with that in AL
07108	C3		RET		Return the Control to O.S.

Initial Flags	Final Flags
CF : 0	CF : 1
ZF : 0	ZF : 1
SF : 0	SF : 0
OF : 0	OF : 0
PF : 0	PF : 1
AF : 0	AF : 1
IF : 1	IF : 1
DF : 0	DF : 0

⇒ Carry flag, zero flag, parity flag and ~~aug~~ auxiliary carry flag get affected.

⇒ Explanation:

$$\begin{array}{rr} & FF\,H \\ + & 01\,H \\ \hline & 100\,H \end{array}$$

⇒ There is carry as well as overflow is generated during this addition. Also, since carry is generated during first 8 bits, hence auxillary flag is s Since result is ~~no~~ zero, so ZF is s and parity of the result i.e. 00H i even, so PF is also set.

D] ; Program to read flag register :

PUSHF
POP AX
RET

Address	Hexcode	Label	Opcode	Operand	Comments
07100	9C		PUSHF		Push the flag register onto the stack
07101	58		POP	AX	Pop the contents of stack to AX pointed by SI.
07102	C3		RET		Return the control to O.S.

• Conclusion :

We learnt to use subroutines to first add two 32-bit numbers and then extended it to add series of 32-bit numbers.
Then we analyzed flag register for a sum and also learnt to read flag register using PUSHF and POP instructions.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesly Road, Shivajinagar, Pune - 411 005

Name: Abhishek Shinde
MIS: 112103134
Div: 2 Batch: S3

Experiment - 4

• LAB REPORT •

• **Aim:** i) Write a program in assembly language of 8086 to test INT3 interrupt. Initialize the registers AX, BX, DX, DS, ES, SS, SP, BP, SI and DI for any desired values of your choice. Initialize CX = 0000H. Use INT 16 service for BIOS Interrupts of OS to test if B key is pressed.

ii) Write a subroutine, when B key is pressed, insert CC opcode (for INT 3 instruction) after the last instructions of main program where above initialisation is done and executes the main program again till INT3 instruction.

iii) Write a ISS for INT3 to increment CX by two and display contents of AX register on screen using INT10 function call 0A for BIOS interrupt

iv) Modify the ISS to display all registers on screen.

v) Write ISS for single step to perform operations mentioned in iii) & iv).

vi) Write ISS for divide by zero ~~err~~ interrupt to display Divide By Zero Error.

i]

```
org 100H
.MODEL SMALL
.STACK 100H
.DATA
        BLANK_STMT DB 10, 13, "$"
        MSG1 DB 10,13, "B Key was pressed.$"
        MSG2 DB 10, 13, "B key was not pressed! Try

.CODE
        MOV AX, @DATA     } ;Initialize data segment
        MOV DS, AX        }

        MOV AX, 1234H     }
        MOV BX, 3A26H     }
        MOV DX, 984DH     }
        MOV SP, 000EH     } ;Initialize registers with
        MOV BP, 0002H     }  arbitrary values
        MOV SI, 2000H     }
        MOV DI, 4000H     }

        MOV CX, 0000H

        BACK: MOV AH, 00H }
              INT 16H     } ;Read and display press
              MOV AH, 0EH }  key
              INT 10H     }

              CMP AL, 42H } ;Compare pressed key w
              JE NEXT     }  ASCII Value of 'B' i.e.
              MOV DX, OFFSET MSG2
```

```
        MOV AH, 09H      ; Print key not pressed
        INT 21H            message
        MOV DX, OFFSET BLANK_STMT
        MOV AH, 09H
        INT 21H
        JNE BACK         ; Loop until B is pressed

NEXT:   MOV DX, OFFSET MSG1   ; Print key pressed
        MOV AH, 09H             message
        INT 21H

    MOV AH, 4CH
    INT 21H

END
RET
```

ii)

```
org 100h
.MODEL SMALL
.STACK 100H
.DATA
    BLANK_STMT DB 10,13, "$"
    MSG1 DB 10,13, "B key was pressed $"
    MSG2 DB 10,13, "B key was not pressed $"
    FLAG DB 01H

.CODE
    ; Changing CS and IP in IVT at location of INT3
```

```
MOV AX, 0000H
MOV DS, AX
MOV BX, 000CH                      ; changing CS & IP in
MOV [BX], ISR1                       for INT3
MOV [BX + 02H], CS

BACK_TO_START :
MOV AX, @DATA                      ; Initialize data segment
MOV DS, AX

MOV AX, 1234H
MOV BX, 3A26H
MOV DX, 984DH                      ; Initialize registers
MOV SP, 000EH                        with arbitrary values
MOV BP, 0002H
MOV SI, 2000H
MOV DI, 4000H

MOV CX, 0000H

BACK : MOV AH, 00H
       INT 16H                     ; Read & display
       MOV AX, 0EH                   pressed key
       INT 10H
       CMP AL, 42H                 ; Compare pressed k
       JE NEXT                       with ASCII value of
       MOV DX, OFFSET MSG2         ; Print key n
       MOV AH, 09H                   pressed m
       INT 21H
       MOV DX, OFFSET BLANK_STMT
```

```
MOV AH, 09H
INT 21H
JNE BACK               ; Loop until 'B' is pressed

NEXT: MOV DX, OFFSET MSG1   ; Print key
      MOV AH, 09H           pressed message
      INT 21H
      MOV DX, OFFSET BLANK_STMT
      MOV AH, 09H
      INT 21H

      MOV BX, OFFSET LAST   ; Replace NOP
      MOV [BX], 00CCH       instruction with
                            INT3 i.e. CCH

      MOV AL, FLAG          ; Check FLAG, which is
      CMP AL, 00H           used to re-run program
      JE LAST               from beginning once
      MOV FLAG, 00H         after inserting CCH
      JMP BACK_TO_START

LAST: NOP

MOV AH, 4CH
INT 21H

ISR1 PROC                   ; ISS to increment
   MOV CX, 02H              CX register by
   MOV DL, CL               2 and display the
   ADD DL, 30H              result.
   MOV AH, 02H
```

COEP TECHNOLOGICAL UNIVERSITY, P
Wellesly Road, Shivajinagar, Pune - 411 005

```
        INT 21H
        IRET
    ISR1 ENDP

  END
  RET

iii) org 100h
.MODEL SMALL
.STACK 100H
.DATA
   BLANK_STMT DB 10,13, "$"
   MSG1 DB 10,13, "B key was pressed.$"
   MSG2 DB 10,13, "B key was not pressed.$ "
   MSG3 DB 10,13, "Value in CX is : $"
   MSG4 DB 10,13, "Value of AX register: $"
   COUNT DB 04 H

.CODE
   ; Changing CS and IP in IVT at location of INT3
   Mov AX,0000H
   Mov DS, AX
   Mov BX, 0000H
   Mov [BX], ISR1
   Mov [Bx +02H], CS

   MOV AX, @DATA
   MOV DS, AX

   MOV CX, 0000H
```

BACK: MOV AH, 00H
INT 16H
MOV AH, 0EH
INT 10H
CMP AL, 42H
JE NEXT
MOV DX, OFFSET MSG2
MOV AH, 09H
INT 21H
MOV DX, OFFSET BLANK_STMT
MOV AH, 09H
INT 21H
JNE BACK
NEXT: MOV DX, OFFSET MSG1
MOV AH, 09H
INT 21H
~~JNE BACK~~
MOV DX, OFFSET BLANK_STMT
MOV AX, 09H
INT 21H

MOV BX, OFFSET LAST
MOV [BX], 00CCH

LAST: NOP

MOV AH, 4CH
INT 21H

```
ISR1 PROC:
    PUSH AX
    INC CX
    INC CX
    LEA DX, BLANK_STMT
    MOV AH, 09H
    INT 21H
    LEA DX, MSG3
    MOV AH, 09H
    INT 21H
    MOV DL, CL
    ADD DL, 30H
    MOV AH, 02H
    INT 21H

    ; Printing AX
    LEA DX, MSG4
    MOV AH, 09H
    INT 21H

    POP DX   ; Store contents of AX in DX to display
    CALL DISPLAY_REGISTER

    IRET
ISR1 ENDP
```

```
DISPLAY_REGISTER PROC
    BACK2: ROL DX, 04H
           MOV AL, DL
           AND AL, 0FH
           CMP AL, 0AH
           JB DOWN2
           ADD AL, 07H
           DOWN2 : ADD AL, 30H

           MOV BH, 00H
           MOV AH, 0AH
           MOV CX, 0001H
           INT 10H

           PUSH DX

           ; Get cursor position
           MOV AH, 03H
           XOR BH, BH
           INT 10H
           ; Move cursor one position right
           INC DL
           ; Set cursor position
           MOV AH, 02H
           INT 10H

           POP DX
           SUB COUNT, 01H
           JNZ BACK 2
    MOV COUNT, 04H
    RET
DISPLAY_REGISTER ENDP
```

COEP TECHNOLOGICAL UNIVERSITY, P

Wellesly Road, Shivajinagar, Pune - 411 005

iv) To display all registers, ~~abov~~ ISR1 in (iii) can modified as:

```
ISR1 PROC
    PUSH DX
    PUSH CX
    PUSH BX
    PUSH AX

    ; Printing AX register
    LEA DX, MSG3
    MOV AH, 09H
    INT 21H

    POP DX
    CALL DISPLAY_REGISTER

    ; Printing BX register
    LEA DX, MSG4
    MOV AH, 09H
    INT 21H

    POP DX
    CALL DISPLAY_REGISTER

    ; Printing CX register
    LEA DX, MSG5
    MOV AH, 09H
    INT 21H
```

```
POP DX
~~OAS~~ CALL DISPLAY_REGISTER

; Printing DX register
LEA DX, MSG6
MOV AH, 09H
INT 21H

POP DX
CALL DISPLAY REGISTER

IRET
ISR1 ENDP
```

- the display-register subroutine and main program code remains same.
- Data segment now contains -

```
.DATA
    BLANK_STMT DB 10, 13, "$" ~~key pressed~~"
    MSG1 DB 10, 13, "B key was pressed.$"
    MSG2 DB 10, 13, "B key was not pressed!$"
    MSG3 DB 10, 13, "Value of AX register: $"
    MSG4 DB 10, 13, "Value of BX register: $"
    MSG5 DB 10, 13, "Value of CX register: $"
    MSG6 DB 10, 13, "Value of DX register: $"
    COUNT DB 04H
```

V) Program to display 'divide by zero error' :

```
org 100h
.MODEL SMALL
.STACK 100H

.DATA
    MSG1 DB "Can not divide by zero!$"

.CODE
    ;Changing CS and IP in IVT at location of INT0
    MOV AX, 0000H
    MOV DS, AX
    MOV BX, 0000H
    MOV [BX], OFFSET ISR1
    MOV [BX+02H], CS

    MOV AX, @DATA
    MOV DS, AX

    MOV AX, 1234H
    MOV BL, 00H
    DIV BL

    MOV AH, 4CH
    INT 21H

~~ISR1 PROC~~
    ~~MOV AX, @DATA~~
    ~~MOV DS, OFFSET MS AX~~
```

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

```
ISR1 PROC
        MOV DX, OFFSET MSG1
        MOV AH, 09H
        INT 21H
        IRET
ISR1 ENDP

END
RET
```

- Conclusion :
  - I learnt interrupt handling by writing ISS for INT3 and INTO. I also learnt to display the contents of register on screen using INT 10.
  - INTO is used for divide by zero error.
  - INT 3 can be used to transfer control to custom ISS by changing CS and IP in IVT.
  - INT 10 can bu used to print contents of register by using OAH in AH register.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesly Road, Shivajinagar, Pune - 411 005

Experiment - 5

• LAB REPORT •

Name : Abhishek Shinde

MIS : 112103134

Div : 2 Batch : S3

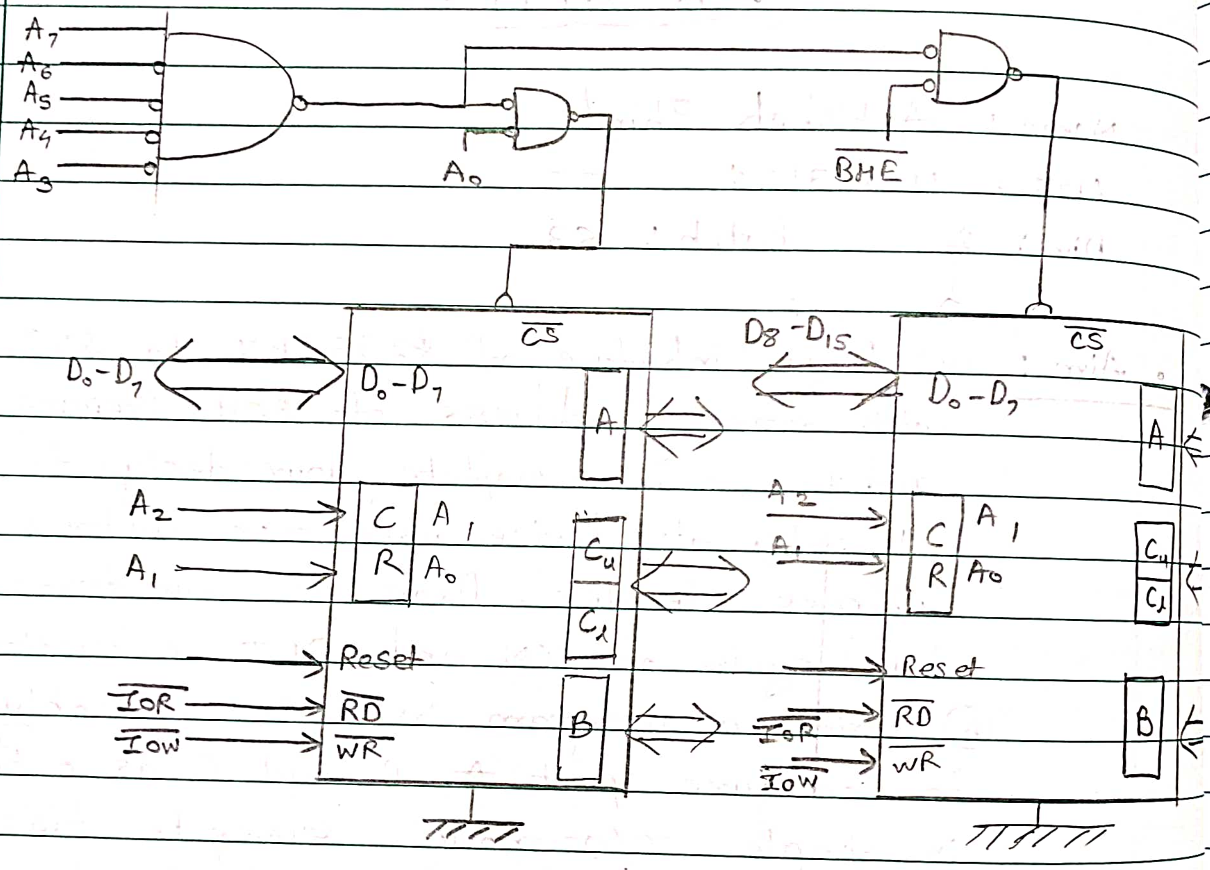
• Aim : (a) Design interface of 8255 PPI to 8086 CPU with starting address of 80H. Prepare the address map. Complete your design to include all relevant devices used for address demux, address decoder. Draw the waveform of last bus cycle of IN and OUT instruction.

(b) Write a program in 8086 assembly language to configure port A, B and C as output ports in simple I/O mode. Generate flashing LEDs at port C lines running lights from right to left at port B and running lights from left to right at port A.

(c) Fine tune the delay between flashing lights to 0.5 seconds.

A]

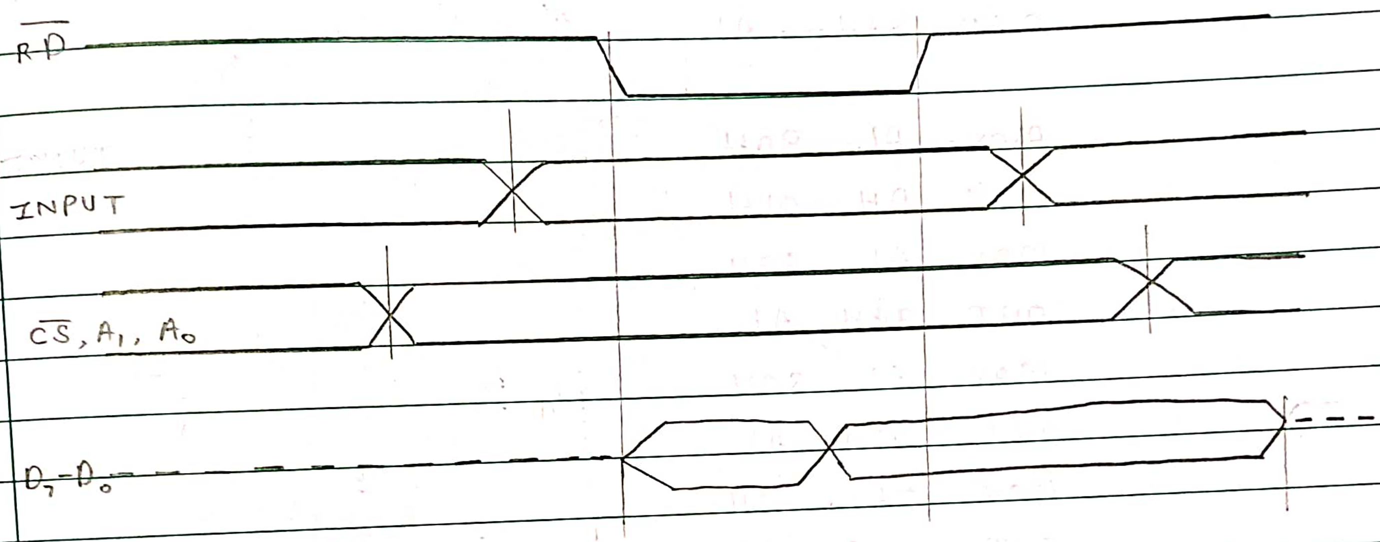
• Interfacing of 8255 to 8086 :



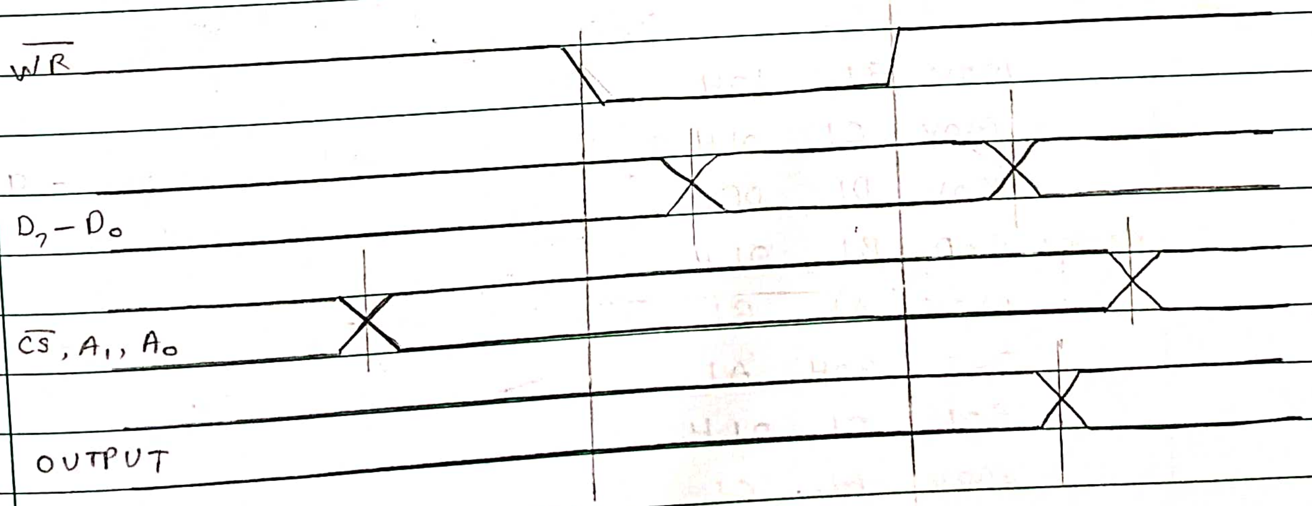
• Address map :

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	Port A
1	0	0	0	0	0	0	1	Port B
1	0	0	0	0	0	1	0	Port C
1	0	0	0	0	0	1	1	C.R.

- Input waveform :



- Output Waveform :



COEP TECHNOLOGICAL UNIVERSITY,
Wellesly Road, Shivajinagar, Pune - 411 005

B)

```
~~MOV AL, 80H~~
~~OUT 33H, AL~~

~~MOV DL, 80H~~
~~MOV DH, 01H~~
MOV AL, 80H
OUT 33H, AL
MOV AL, 80H
OUT 30H, AL
MOV AL, 01H
OUT 31H, AL
MOV AL, 0FH
OUT 32H, AL
CALL DELAY

MOV BL, 10H
MOV CL, 01H
MOV DL, 0FH
BACK: ROR BL, 01H
MOV AL, BL
OUT 30H, AL
ROL CL, 01H
MOV AL, CL
OUT 31H, CL
ROR DL, 04H
MOV AL, DL
OUT 32H, AL
CALL DELAY
JMP BACK
```

~~PROG~~

```
DELAY   PROC
        MOV   AX, FFFFH
        HERE : DEC  AX
               CMP  AX , 00H
               JNZ  HERE
               RET
DELAY   ENDP
```

c]

	T-states
DELAY PROC	
MOV DX, 05H	4 × 1
LABEL1: MOV CX , AD9BH	4×5
LABEL2: DEC CX	2×5×N
JNZ LABEL2	16×5×N
DEC DX	2×5
JNZ LABEL1	16×5
RET	8×1
DELAY ENDP.	

- Conclusion: Thus we learnt interfacing of 8255 with 8086 and configuration of ports as input and output.

* Calculation for part C :

$$4 + 20 + 10N + 80N + 10 + 80 + 8 = \text{Time} \times \text{freq.}$$

$$\Rightarrow \quad 122 + 90N = 0.5 \times 8 \times 10^6$$

$$\Rightarrow \quad N = \frac{4 \times 10^6 - 122}{90}$$

$$\therefore \quad N = 44443.09$$

$\Rightarrow$ In hexadecimal,

$$\boxed{N = AD9B}$$

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesly Road, Shivajinagar, Pune - 411 005

Name: Abhishek Shinde.
MIS: 112103134
Div: 2 Batch: S3

Experiment - 6

• LAB REPORT •

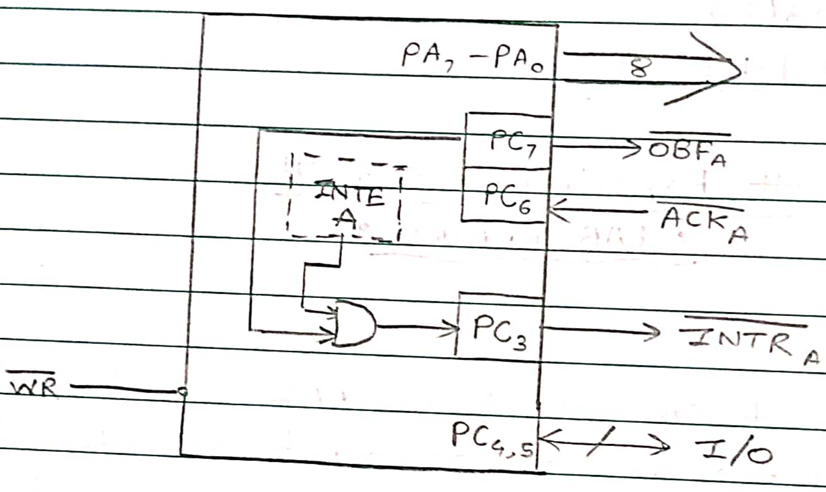
• Aim:

ⓐ For the design of 8255 PPI created in exp5, draw waveforms of relevant signals for verifying mode 1 input (such as STB, IBF and INTRA) and mode 1 output (OBF, ACK and INTRA).

ⓑ Write a program in 8086 assembly language to configure port A as output port in mode 1. Verify operation of this mode by writing a program to send output byte to port A under the control of handshake signals.

ⓒ Write a program in 8086 assembly language to configure port B in mode 1 input. Verify ~~its~~ operation under the control of relevant handshake signals.

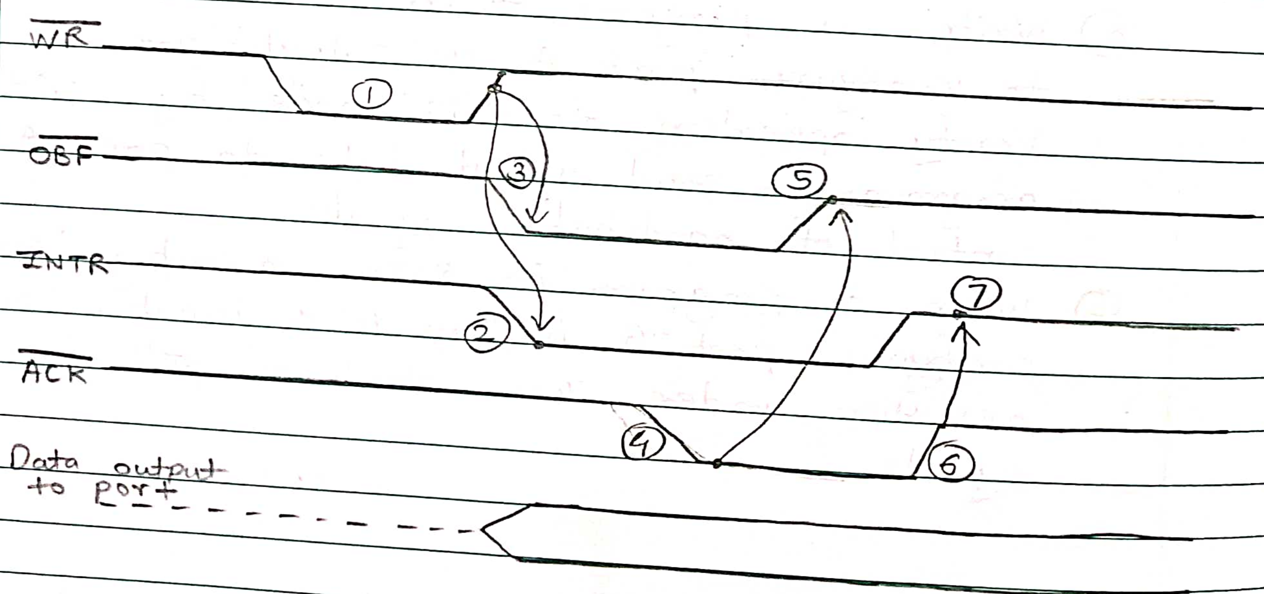
A] • Port A – Mode 1 – output →



In case of Mode 1 output in port A, INTE is controlled by PC_6

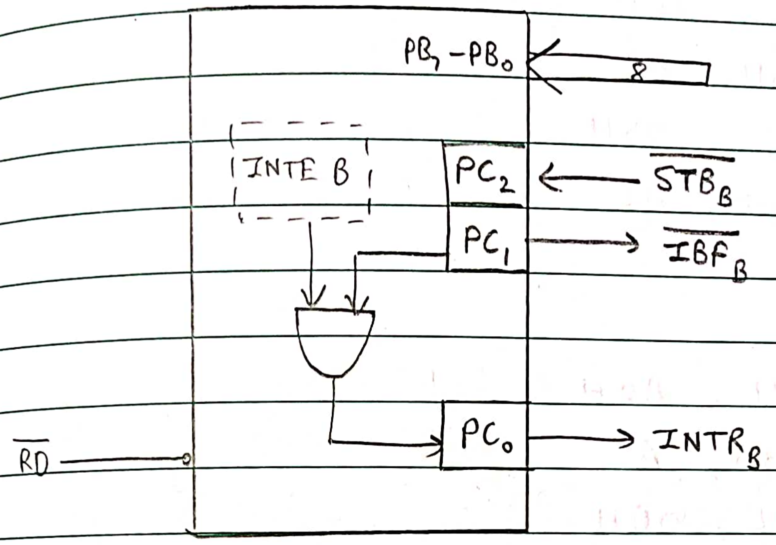
$\underbrace{0000\ 1101}_{0D}$

• Waveforms – mode 1 – output →



When $\overline{OBF} = 1$ and $\overline{ACK} = 1$, then $INTR = 1$.

• Port B – mode 1 – input →

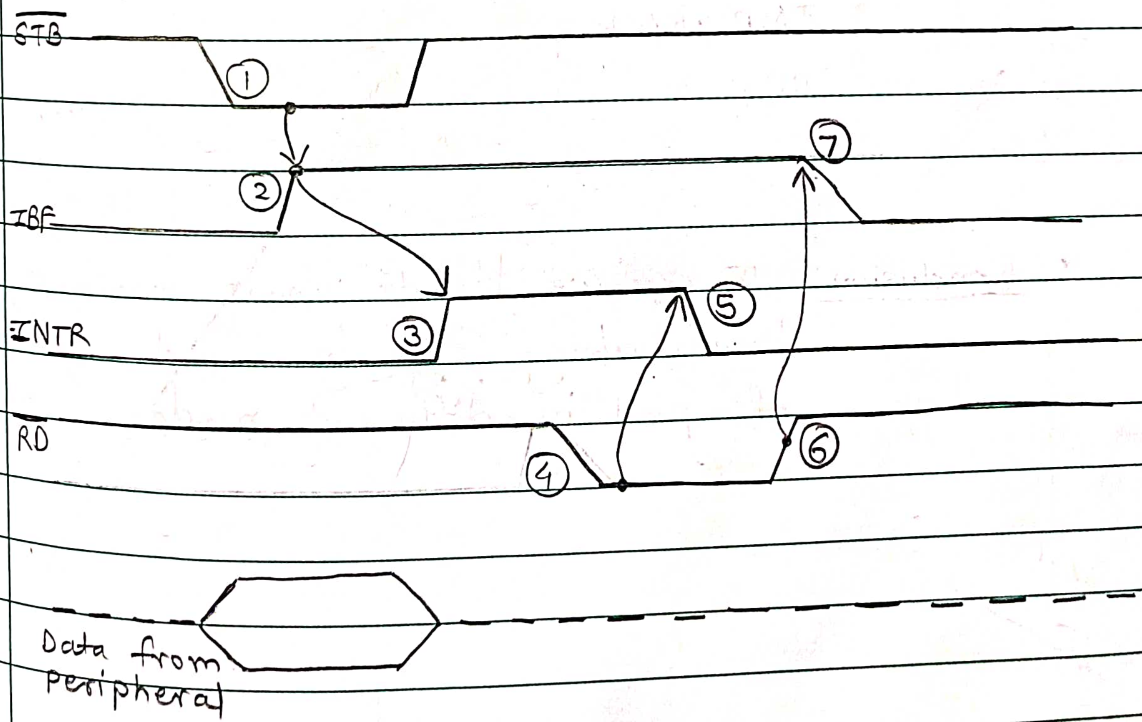


In mode 1 input, in port B, INTE is controlled by PC_2

⇓

$\underbrace{00000101}_{05}$

• waveforms – mode 1 – input →



When $\overline{STB} = 0$ and $IBF = 1$,

$INTR = 1$

C]

MOV AL, 86H } 1000 0110 ⇒ 86H
OUT 33H, AL } out to C.R. → C.W. for M1 input

MOV AL, 05H } 0000 0101 ⇒ 05H
OUT 33H, AL } Bit Set PC_2

B]

MOV AL, A0H } 1010 0000 ⇒ A0H
OUT 33H, AL } out to C.R. → C.W for M1 output PA

MOV AL, 0DH } 0000 1101 ⇒ 0DH
OUT 33H, AL } Bit Set PC_6

```
BACK: MOV AL, F0H
      OUT 30H, AL
      JMP BACK
      RET
```

- Conclusion: Thus, we learnt about various handshaking signals and configuration of port in different modes.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Name: Abhishek Shinde.

MIS: 112103134

Div: 2 Batch: S3

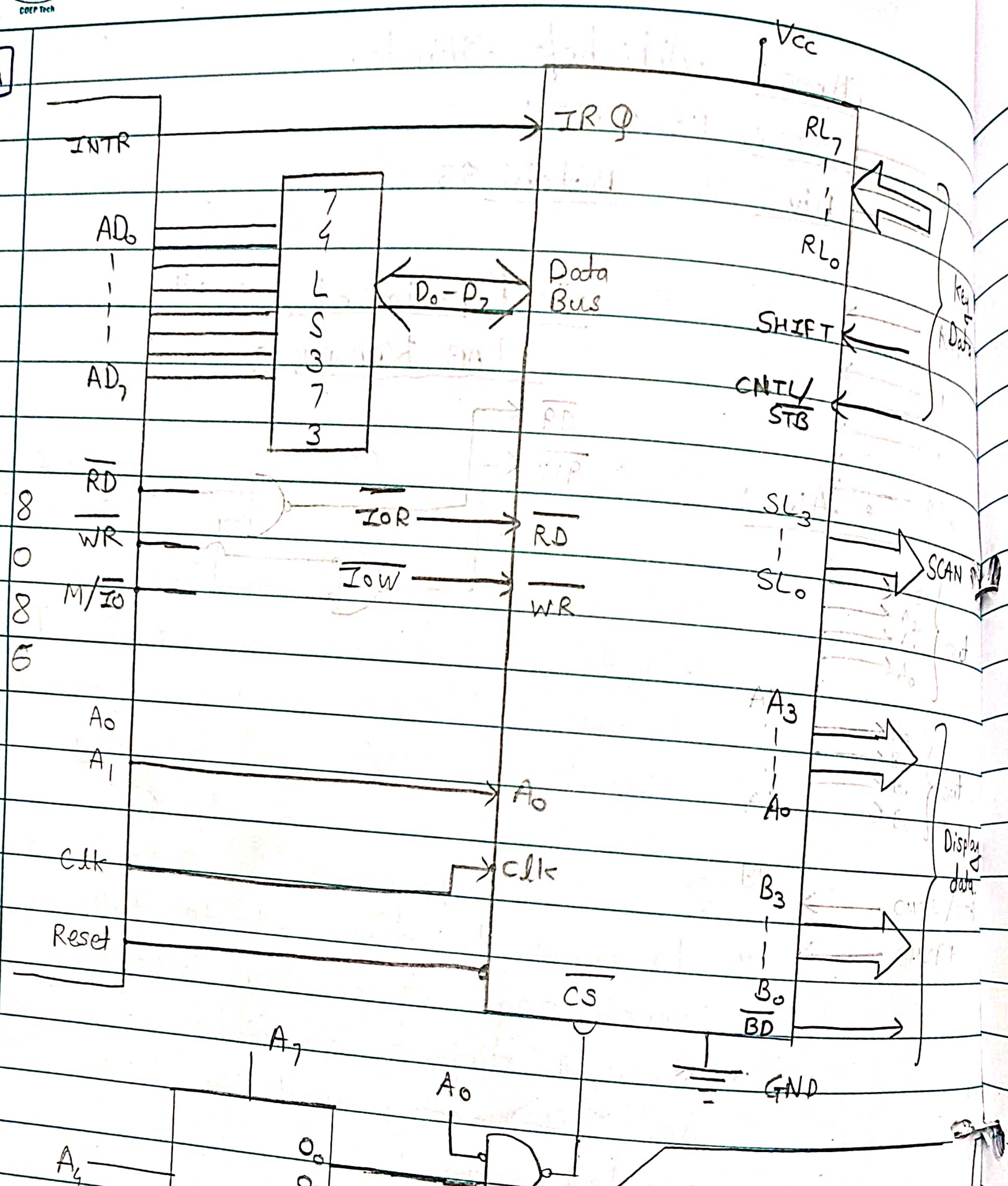
Experiment - 7

• LAB REPORT •

• Aim: (a) Design interface of 8279 PPI to 8086 CPU with starting address of 80H. Prepare the address map. Complete your design to include all relevant devices used for address demux, address decoder, separate control signals for I/O and memory.

(b) Write a program in 8086 assembly language to demonstrate following 8279 seven segment display interface modes. Display digit 0 onwards in a right entry and left entry mode in both decoded and encoded scan.

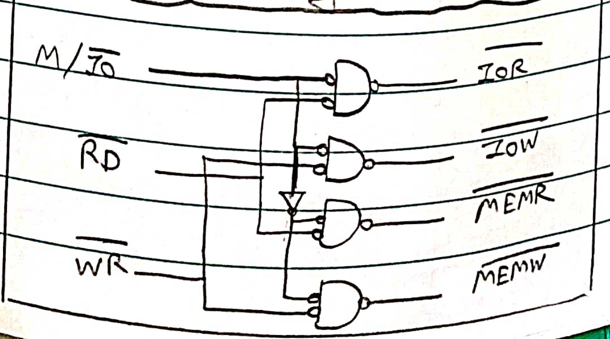
A]



Address map:

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
80 ⇒	1	0	0	0	0	0	0	0
82 ⇒	1	0	0	0	0	0	1	0

Control Signal for I/O:



B] i) Left Entry encoded 16.8 character display -

0	0	0	0	1	0	0	0
			D	D	K	K	K

⇒ 08

• Data stored at CS: 2000 →

FA	0A	B6	F4	6C	DC	DE	70
for '0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'
FE	FC	7E	CE	9A	76	B7	17
'8'	'9'	'A'	'B'	'C'	'D'	'E'	'F'

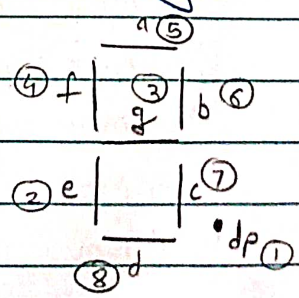
• Program -

```
      MOV  CL, 0FH
      MOV  AL, 08H
      OUT  31, AL
      MOV  AX, CS
      MOV  DS, AX
      MOV  BX, 2000H
BACK: MOV  AL, [BX]
      OUT  30, AL
      CALL DELAY
      INC  BX
      DEC  CL
      JNZ  BACK
HERE: JMP  HERE
```

• Delay -

```
DELAY PROC
        MOV DX, FFFFH
LABEL1: DEC DX
        CMP DX, 00
        JNZ LABEL1
        RET
DELAY ENDP
```

• Seven segment:



d	c	b	a	f	g	e	dp

ii) Left entry decoded display —

			d	d	k	k	k
0	0	0	0	0	0	0	1

⇒ 01 H

• Program:

```
      MOV  CL, 0FH
      MOV  AL, 01H
      OUT  31, AL
      MOV  AX, CS
      MOV  DS, AX
      MOV  BX, 2000H
BACK: MOV  AL, [BX]
      OUT  30, AL
      CALL DELAY
      INC  BX
      DEC  CL
      JNZ  BACK
HERE: JMP  HERE
```

iii) Right entry encoded 16-8 character display —

0	0	0	1	1	0	0	0

⇒ 18 H

```
      MOV  CL, 0FH
      MOV  AL, 18H
      OUT  31, AL
      MOV  AX, CS
      MOV  DS, AX
      MOV  BX, 2000H
BACK: MOV  AL, [BX]
```

```
        OUT 30, AL
        CALL DELAY
        INC BX
        DEC CL
        JNZ BACK
HERE:   JMP HERE
```

iv) Right entry decoded display –

0	0	0	1	0	0	0	1

$\Rightarrow$ 11 H

```
        MOV CL, 0FH
        MOV AL, 11H
        OUT 31, AL
        MOV AX, CS
        MOV DS, AX
        MOV BX, 2000H
BACK:   MOV AL, [BX]
        OUT 30, AL
        CALL DELAY
        INC BX
        DEC CL
        JNZ BACK
HERE:   JMP HERE
```

- **Conclusion:** Thus we learnt the interfacing of 8279 with 8086 microprocessor. We also learnt to display characters in right entry and left entry mode both in encoded and decoded scan.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Name : Abhishek Shinde.

MIS : 112103134

Div : 2 Batch : S3

Experiment - 8

- Aim : ⓐ Write to explain different command words and status word of 8279. Design interface of 8279 PPI to 4x4 key matrix and 4 common cathode 7-segment displays in encoded scan mode.
Repeat the process for 16 sensor's matrix and 4 displays in decoded scan mode. Complete your design to include all relevant devices.

ⓑ Write a program in 8086 assembly language to demonstrate :

i) Encoded key matrix

ii) Decoded scan key matrix

iii) Sensor matrix.

A) • **8279 Command Words:**

a) **Keyboard/ display mode set** –

0	0	0	D	D	K	K	K

DD ⇒ 00 → 8 – 8 bit character display – Left entry
01 → 16 – 8 bit character display – left entry
10 → 8 – 8 bit character display – Right entry
11 → 16 – 8 bit character display – Right entry

KKK ⇒
000 → Encoded scan – 2 key lockout
001 → Decoded scan – 2 key lockout
010 → Encoded scan – N-key Rollover
011 → Decoded scan – N-key Rollover
100 → Encoded scan – Sensor matrix
101 → Decoded scan – Sensor matrix
110 → Strobe input, encoded display scan
111 → Strobe input, decoded display scan.

b) **Program clock** –

0	0	1	P	P	P	P	P

- All timing and multiplexing signal for 8279 generated by an internal pre scalar.
- The prescalar divides the external clock by the programmable integer.
- Bits PPPPP determine the value of this integer which range from 2 to 31.

(c) Read FIFO/ Sensor RAM :

0	1	0	AI	X	A	A	A

- Then in sensor matrix mode, the RAM address bits AAA select one of the 8 rows of the sensor RAM.
- If AI flag is set (i.e. AI = 1), each successive read will be from the subsequent row of the sensor RAM.

(d) Read display RAM :-

0	1	1	AI	A	A	A	A

- The address bits AAAA select one of the 16 rows of the display RAM.
- If AI = 1, this row address will be incremented after each following read / write to the display RAM.

(e) Write display RAM –

1	0	0	AI	A	A	A	A

- After writing Command with $A_0 = 1$, all sub-sequent writes with $A_0 = 0$ will be to the display RAM.

(f) Display write inhibit / blanking –

1	0	1	X	IW (A)	IW (B)	BL (A)	BL (B)

- IW bits can be used to mask nibble A & B in applications requiring separate 4-bit display ports.
- BL flags are used to blank the display.

(g) Clear –

1	1	0	C_D	C_D	C_D	C_F	C_A

- C_D bits are used to clear the display. (Clear all rows of the display RAM).
-

C_D	C_D	C_D	
	0	0	} All zeroes
	0	1	} All zeroes
	1	0	→ AB = Hex 20 (0010 0000)
	1	1	→ All ones

- If $C_F = 1$, FIFO status is cleared and interrupt output line is reset.
- C_A – clear bit has combined effect of C_F & C_D.

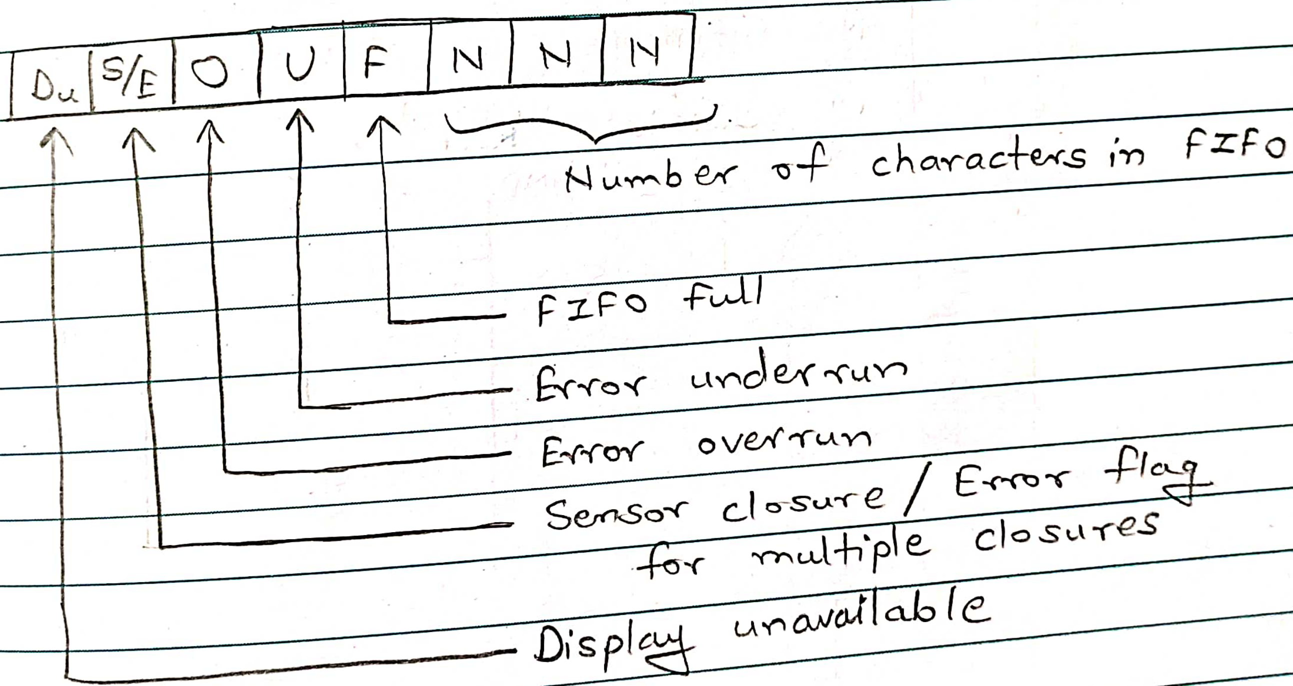
(h) END interrupt / Error MODE SET –

1	1	1	E	X	X	X	X

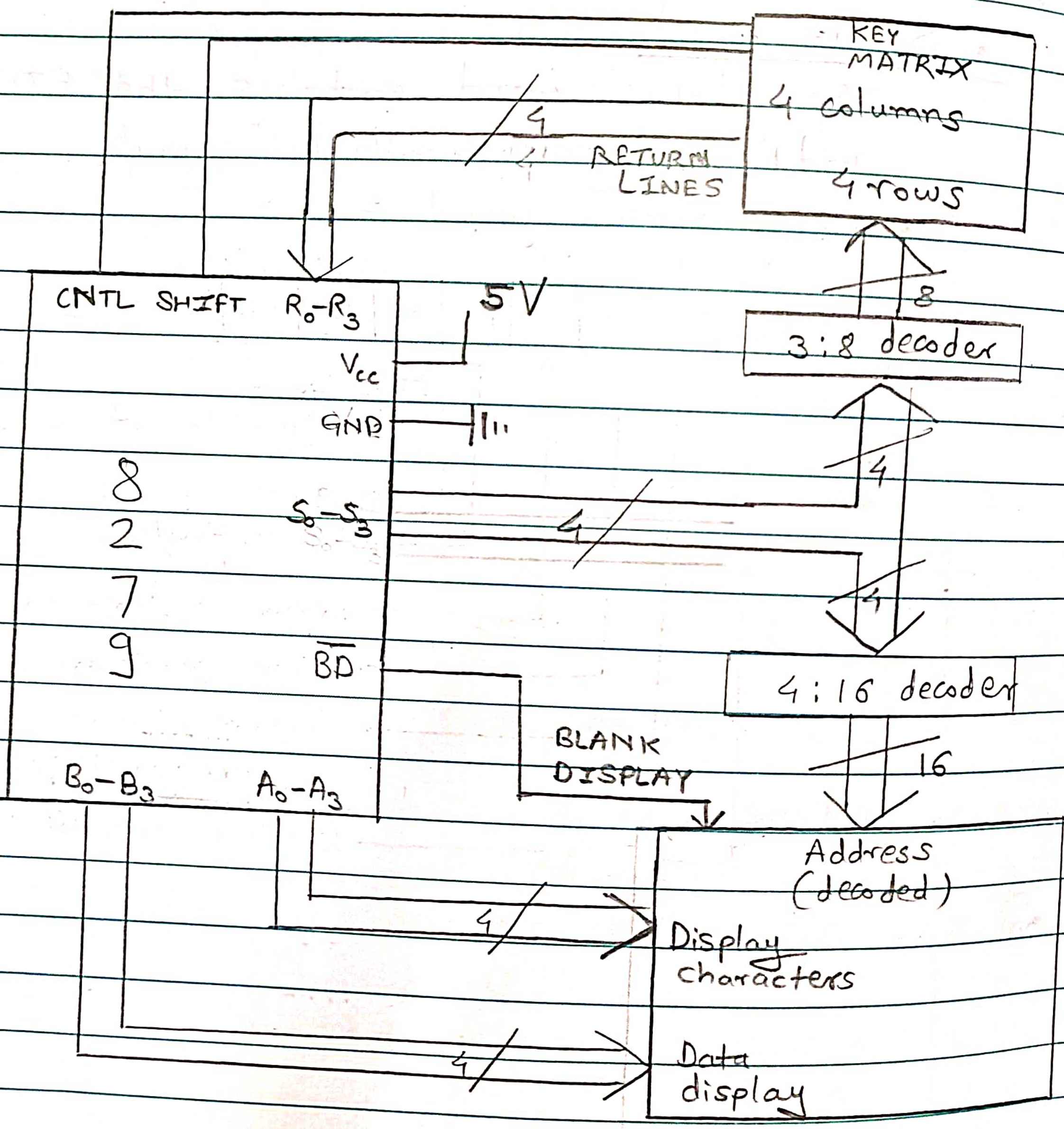
- For sensor matrix, this command lowers the IRQ line and enables further writing into RAM.

- for N-key rollover mode,
  if E-bit is set, the chip will operate in the special error mode.

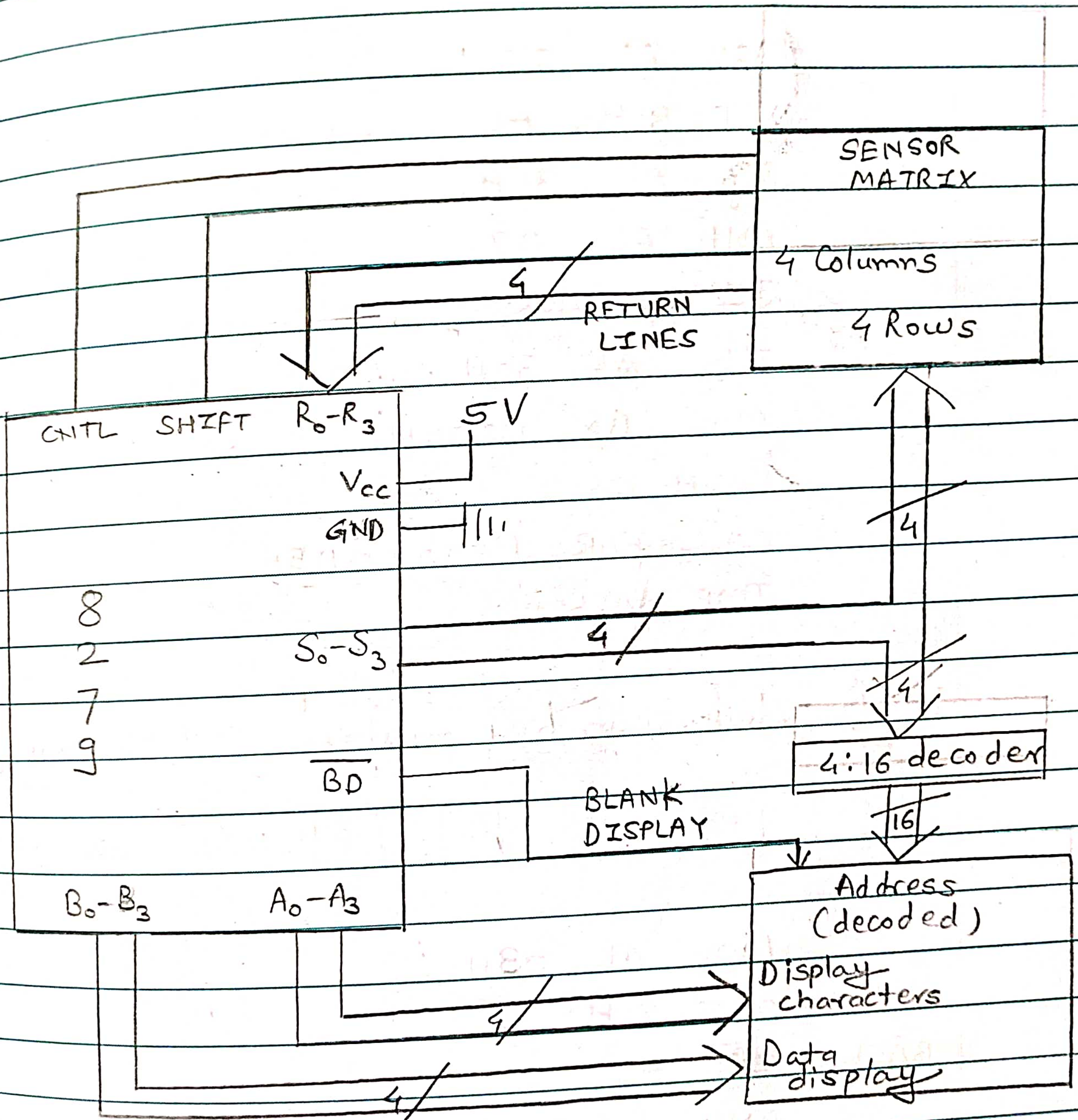
- **Status word :**
  - The status word contains the FIFO status errors and display unavailable signals.
  - FIFO status word –



- Interface of 8279 PPI to 4x4 key matrix and 4 common cathode 7-segment displays (encoded scan mode).



- Interface of 8279 PPI to 16 sensor's matrix and 4 displays in decoded scan mode :



B] i) Encoded key matrix — (2 key lockout)

d	d	k	k	k			
0	0	0	0	0	0	0	0

⇒ 00 H

```
      MOV AL, 00H        ; Encoded key matrix-2 key lockout
      OUT 31H, AL        ; Configure 8279
BACK: IN AL, 31H         ; Input status word in AL
      AND AL, 07H        ; Check for last 3 bits
      JZ BACK            ; If even 1 bit is 1, it escapes loop
      IN AL, 30H         ; Input data port contents into AL
      MOV BX, 0200H      ;
      PUSH CS            ; Function call to display contents of AL
      CALL FAR F000:0D5E ;
      JMP BACK           ; Continue running the program
```

ii) Decoded scan key matrix — (N-key rollover)

0	0	0	0	0	0	1	1

⇒ 03H

```
      MOV AL, 03H
      OUT 31H, AL
BACK: IN AL, 31H
      AND AL, 07H
      JZ BACK
      IN AL, 30H
      MOV BX, 0200H
      PUSH CS
      CALL FAR F000:0D5E
      JMP BACK
```

iii) Sensor matrix
(decoded scan sensor matrix mode)

			d	d	k	k	k
0	0	0	0	0	1	0	1

$\Rightarrow$ 05H

```
        MOV AL, 05H      ; C.W. for decoded scan sensor matrix
        OUT 31H, AL      ; Configure 8279
BACK :  IN AL, 31H       ; Check status word
        AND AL, 07H      } Check last 3 bits
        JZ BACK          }
        MOV AL, 41H      ; To scan 1st row of keyboard
        OUT 31H, AL      ; Configure read FIFO
        IN AL, 30H       ; Input data port content into AL
        MOV BX, 0200H        }
        PUSH CS              } ; CALL function to
        CALL FAR F000:0D5E   }   display contents
        JMP BACK         ; Continue running the program
```

• Conclusion :-

1) We studied the 8 command words, FIFO status word as well as data word of 8279.
2) We studied the interfacing of 8279 to keyboard matrix and common cathod 7-segment display.
3) We also learnt to program 8279 to read user input through encoded key matrix, decoded key matrix and sensor matrix.

COEP TECHNOLOGICAL UNIVERSITY, PUNE

Wellesley Road, Shivajinagar, Pune-411 005

Name : Abhishek Shinde

MIS : 112103134

Div : 2 Batch : S3

Experiment - 9

• LAB REPORT •

Aim - ① Write to explain different Command word and status word of 8259.

② Write a program in 8086 assembly language to demonstrate fully nested mode.

• 8259 Command word :

i) ICW_1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D	A_7	A_6	A_5	1	LTM	ADI	SNGL	IC_4

$\underbrace{A_7 \; A_6 \; A_5}_{\text{Not required for 8086}}$

IC_4 → 1 : ICW_4 needed

0 : ICW_4 not needed

SNGL → 1 : single

0 : cascaded

ADI → Call address interval :

1 : address of 4 (Interval of 4)

0 : Interval of 8

LTM → 1 : Level triggered

0 : Edge triggered

- ICW_1 tells whether 8259 is going to work as edge triggered or level triggered.
- It also tells whether ICW_3 and/or ICW_4 will be given or not.
- It also initializes 8259 in single or cascaded mode.

ii) ICW_2 -

$A_0 = 1$

					×	×	×
N_7	N_6	N_5	N_4	N_3	N_2	N_1	N_0

N_7–N_3: 5 bits of the type no. of IR_0
N_2–N_0: Irrelevant

- ICW_2 provides the type no. of IR_0 and that assigns type no. to $IR_1 - IR_7$ sequentially.

iii) ICW_3 - Master :-

$A_0 = 1$

S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

1 = IR input have slave
0 = IR input have no slave

- It tells the master 8259 on which interrupt request lines (IR) are the slaves connected.

ICW_3 - Slave -

$A_0 = 1$

×	×	×	×	×	ID_2	ID_1	ID_0

ID_2–ID_0: Identification number of slave.

- Each slave is given an identification number and that number is same as its line number of the interrupt request line.

iv) ICW4 –

x	x	x	SFNM	BUF	M/$\overline{S}$	AEO	μPM

μPM: 0 = 8085, 1 = 8086

SFNM: 1 = SFNM, 0 = FNM

AEO: 1 = Auto EOI, 0 = Normal EOI

BUF: 1 = Buffer mode, 0 = Non-buffer mode

M/$\overline{S}$: 1 = Master, 0 = Slave, x = Single mode

v) OCW1 :- OCW1 is issued if programmer wants to mask any interrupt.

$A_0 = 1$

m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0

1 = mask
0 = unmask

vi) OCW2 :

R	SL	EOI	0	0	L_2	L_1	L_0

$L_2\ L_1\ L_0$: IR line no.

R	SL	EOI	
0	0	1	→ Non-specific EOI mode
0	1	1	→ Specific EOI mode
1	0	1	→ Rotate on non-specific EOI
1	0	0	→ Rotate in automatic EOI (set)
0	0	0	→ Rotate in automatic EOI (clear)
1	1	1	→ Rotate on specific EOI
1	1	0	→ Set priority command
0	1	0	→ No operation.

vii) OCW3 -

0	0	ESMM	SMM	0	1	P	RR	RIS

Special mask mode (ESMM, SMM)

- 0 0 } No action
- 0 1 }
- 1 0 = Reset Special mask
- 1 1 = Set special mask.

Read register command (RR, RIS)

- 0 0 } No action
- 0 1 }
- 1 0 → Read IR Reg
- 1 1 → Read IS Reg

P:

- 1 = POLL Command
- 0 = NO POLL Command

• **Status word** -

The status word of 8259 PIC provides info about current status about interrupts and the PIC itself. It includes the following information

IRQ : These status bits include which interrupt are currently requesting service.

ISR : These status bits represent which interrupts are currently being serviced.

IMR : These status bits represent which interrupts are currently masked.

```
MOV AL, 1DH   } ICW1 Master    0 0 0 1 1 1 0 1
OUT 30H, AL   }
MOV AL, 80H   } ICW2 Master    1 0 0 0 0 0 0 0
OUT 31H, AL   }
MOV AL, 20H   } ICW3 Master    0 0 1 0 0 0 0 0
OUT 31H, AL   }
MOV AL, 03H   } ICW4 Master    0 0 0 0 0 1 0 1
OUT 31H, AL   }
MOV AL, 00H   } OCW1 Master    0 0 0 0 0 0 0 0
OUT 31H, AL   }
MOV AL, 1DH   } ICW1 Slave     0 0 0 1 1 1 0 1
OUT 20H, AL   }
MOV AL, 88H   } ICW2 Slave     1 0 0 0 1 0 0 0
OUT 21H, AL   }
MOV AL, 05H   } ICW3 Slave     0 0 0 0 0 1 0 1
OUT 21H, AL   }
MOV AL, 01H   } ICW4 Slave     0 0 0 0 0 0 0 1
OUT 21H, AL   }
MOV AL, 00H   } OCW1 Slave     0 0 0 0 0 0 0 0
OUT 21H, AL   }
STI
RE: JMP HERE
```

ISR →

```
2000 : CLI
2001 : LEA SI, [3000]   ; Display 'INT'
2005 : CALL FAR F000 : 0D3B
200A : LEA SI, [3010]   ; Display 'O'
200E : CALL FAR F000:0D3B
2013 : JMP 2013
```

2015 : CLI
2016 : LEA SI, [3000] ; Display 'INT'
201A : CALL FAR F000:0D3B
202F : LEA SI, [3012] ; Display 'SLAVE0'
2023 : CALL FAR F000:0D3B
2028 : JMP 2028
202A : CLI
202B : LEA SI, [3000] ; Display 'INT'
202F : CALL FAR F000:0D3B
2034 : LEA SI, [3019] ; Display 'SLAVE1'
2038 : CALL FAR F000:0D3B
203D : JMP 203D

- 'INT' is stored at 0000:3000 →

0000:3000 - 49
3001 - 4E
3002 - 54
3003 - 00

- At location 0000:3012 → 'SLAVE0' is stored
- At location 0000:3019 → 'SLAVE1' is stored
- At location 0000:3010 → '0' is stored

0000:3010 - 30
3011 - 00
3012 - 53
3013 - 4C
3014 - 41
3015 - 56

0000:3016 - 45
3017 - 30
3018 - 00
3019 - 53
301A - 4C
301B - 41

0000: 301C - 56
301D - 45
301E - 31
301F - 00

- Interrupt Vector Table -

Type no. → 80H ⇒ 80 × 4 = 0200H

Therefore, 0000:0200 - 00
0201 - 20
0202 - 00
0203 - 00

Type no. → 88H ⇒ 88 × 4 = 0220H

Therefore, 0000:0220 - 15
0221 - 20
0222 - 00
0223 - 00

Type no. → 89H ⇒ 89 × 4 = 0224H

Therefore, 0000:0224 - 2A
0225 - 20
0226 - 00
0227 - 00

• **Conclusion** - Thus we studied different command words & status words of 8259. We also implemented the fully nested mo