Experiment no: 03

Experiment no: Study on how to load the machine codes of a sample program to MDA-8086 then execute it and verify the results.

Objectives:

- To observe and verify the results of machine codes with sample programs in MDA-8086 microprocessor.

Theory:

In MDA-8086 microprocessor for various programs the respected generated machine codes will be different. Machine code is a computer program written in machine language. It uses the instruction set of a particular computer architecture. It is usually written in binary, in this case it hexadecimal. Machine code is the lowest level of software programming language. Other programming language are translated into machine code so the computer can execute it. In this particular experiment we will generate machine codes with the mnemonics using the DOSBOX software. In this software, we will give inputs as mnemonics and the respected machine codes will be generated.

Machine codes	Mnemonics
	.model small
0100	.stack 100h
0000	.data
0000	.code
0000	main proc
0000 B8 805E	mov ax, 805Eh
0003 BA 0504	mov dx, 0504h
0006 03 C2	add ax, dx
0008 BB 0050	mov bx, 0050h
000B 2B C3	sub ax, bx

```
000D 42          inc dx
000E 4B          dec bx
000F             main endp
000F             end main
```

Instruction that has been used here:

MOV: for allocating the values to the respected registers.

ADD: for adding the values of the registers.

SUB: for subtracting the values from the registers.

INC: increments the value of the registers by 1.

DEC: decrements the value of the registers by 1.

Task:

1. Generate the codes of adding two hexadecimal numbers using the DOSBOX.
   i. 67EF, 52CD
   ii. AD34, 29A7

2. Generate the codes of subtracting two hexadecimal numbers using the DOSBOX.
   i. CA45, AE49
   ii. BF98, 608A

3. Generate the codes of operation of Multiplication and Division using MUL and I instructions.

Experiment No:04

Experiment name: Study on different types of arithmetic and logic operation in assembly language

Objectives:

a. The *objective* for this lab is to understand the fundamentals of ***logic gates*** and its use in implementing basic Boolean functions.
b. To study and understand the 3 basic gates
c. Implement basic gates
d. The study the specifications of every gate when connected it with one input constant and the other is variable.

Theory:

AND operations: The AND gate is acts in the same way as the logical "and" operator.The output is 1 only when both inputs one AND two are 1.

For example- In assembly language (8086 microprocessor kit) we operate AND operation in this following formula:

Mov Ax, 1010h	Ax=0001 0000 0001 0000
Mov Cx, 2020h	Cx=0010 0000 0010 0000
AND Ax, Cx	AND=0011 0000 0000 0000

This value of this operation will be stored in Ax register

OR operation: In OR operation, the output to be 1, at least input one OR two must be 1.

For example- we operate OR operation in assembly language by using following formula-

Mov Ax, 1010h	AX=0001 0000 0001 0000
Mov Bx, 2020h	BX=0010 0000 0010 00000
Mov Cx, 3001h	CX=0011 0000 0000 0011
OR AX,BX	OR=0011 0000 0011 0000

This value of this operation will be stored in Ax register

NOT operation:

NOT operation takes only one input. Also known as inversions or complementation.

Mov Ax, 1010h NOT Ax	Ax=0001 0000 0001 0000 NOT=1100 1111 1111 1111

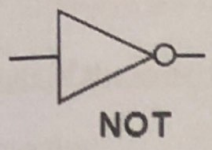
The value of this operation will be stored in Ax register.

Task:

1: Perform the XOR, XNOR TEST

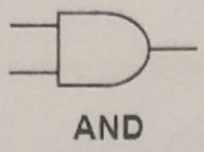
2: Perfrom the NAND operation and verify it.

3: Perfrom and verify the NOR operation.



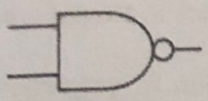
NOT

Input	Output
I	F
0	1
1	0



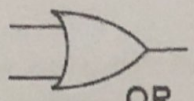
AND

Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1



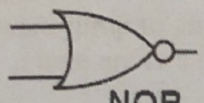
NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0



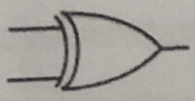
OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1



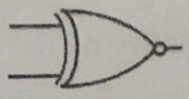
NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0



EXCLUSIVE OR

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Eqipement:

a) 8086 microprocessor
b) A Pc with DosBox software

Code:

Mov Ax, 1010h

Mov Dx , 2020h

Mov Cx, 3001h

Mov Dx,1110h

OR Ax, Bx

AND Ax, Cx

NOT Ax

Output:

Ax=0001 0000 0001 0000

Bx=0010 0000 0010 0000

OR=0011 0000 0011 0000 =3030h=Ax

Ax=0011 0000 0011 0000

Bx=0011 0000 0000 0001

AND=0011 0000 0000 0000=3000h=Ax

NOT (Ax) =1100 1111 1111 1111=0FFFh

References:

1. Internet

2. Lab manual

3. Class lecture

Experiment no: 05

Name of the experiment: Study on the different jump condition of MDA 8086 microprocessor

Objective:

- Knowledge about Jump Conditions.
- Impact of Jump Conditions in a code.

Introduction: A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

Theory: Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions. There are two types of Jump instructions:

- Unconditional Jump Instructions
- Conditional Jump Instructions

Unconditional Jump Instructions

These instructions are used to jump on a particular location unconditionally, i.e. there is no need to satisfy any condition for the jump to take place. There are three types of procedures used for unconditional jump. They are:

- NEAR – This procedure targets within the same code segment. (Intra-segment)
- SHORT - This procedure also targets within the same code segment, but the offset is 1 byte long. (Intra-segment)
- FAR - In this procedure, the target is outside the segment and the size of the pointer is double word. (Inter-segment)

Syntax: JMP procedure name memory location

Example: JMP short target

Conditional Jumps

In these types of instructions, the processor must check for the particular condition. If it is true, then only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements test the flag and jump is the flag is set.

There are following types of conditional jump instructions:

- **JC**: Stands for 'Jump if Carry'.

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

- **JNC:** Stands for 'Jump if Not Carry'.

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.

- **JE / JZ:** Stands for 'Jump if Equal' or 'Jump if Zero'.

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.

- **JNE / JNZ:** Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.

- **JP / JPE:** Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.

- **JNP / JPO:** Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.

Operation Instruction:

1. **First Code:**

```
mov ax,3254h;
mov bx,1f48h;
mov cx,412ah;
add ax, cx;
jmp s7;
sub ax, bx;
s7: and ax, bx;
hlt;
```

2. **Second Code:**

```
mov ax,125bh;
mov bx,125bh;
mov cx,412ah;
sub ax, cx;
```

```
jz s5;
div bx;
s5: and ax, cx;
hlt;
```

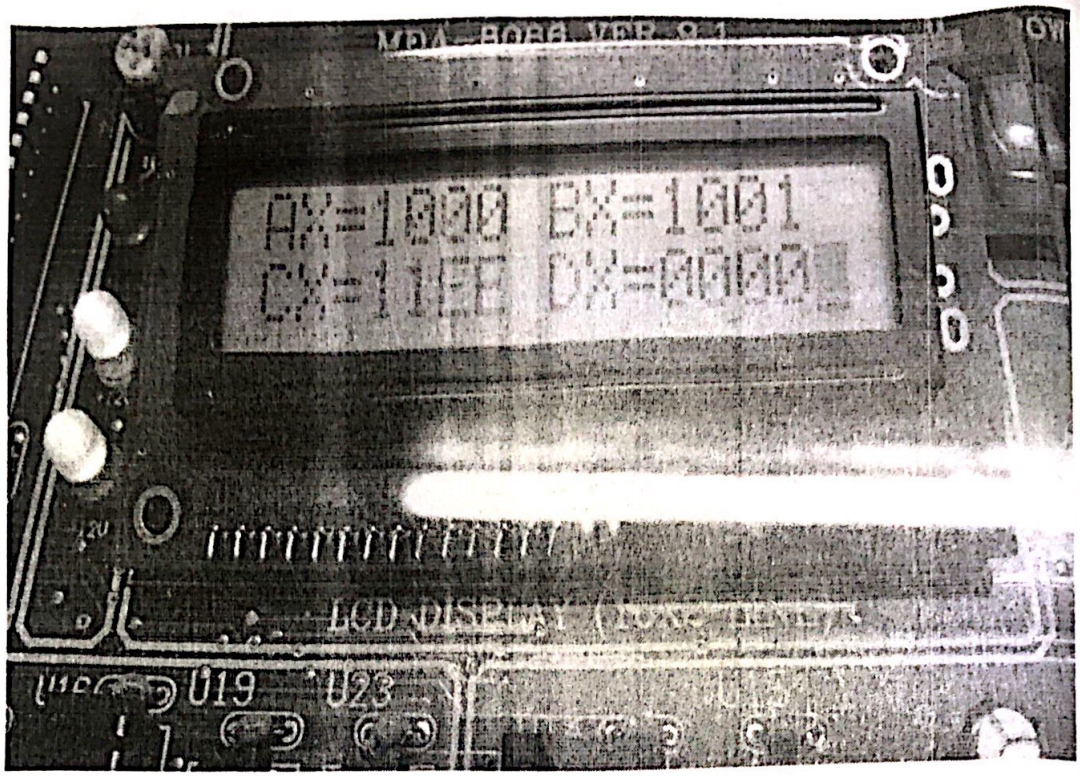
3. **Third Code:**

```
mov ax,7a24h;
mov bx,9543h;
add ax, bx;
jc BAUET;
ICE: or ax,23h;
jnz last;
BAUET: mov cx,0fc7h;
sub ax, cx;
jz ICE;
last: hlt;
```

4. **Fourth Code:**

```
mov ax, 1000h;
mov bx, 1001h;
add ax, bx;
jc polok;
polok: mov cx,11eeh;
jnc polokn;
polokn: mov dx,2233h;
jmp a5;
div cx;
a5: sub ax, cx;
jz hi;
hi: add bx, cx;
jnz hlw;
hlw: add cx, dx;
hlt;
```

Output:



Machine Code:

Microsoft (R) Macro Assembler Version 5.00 1/2/20 00:19:08

Page 1-1

```
                        .model small
0100                    .stack 100h
0000                    .data
0000                    .code
0000                    main proc
0000  B8 1000                   mov ax,1000h
0003  BB 1001                   mov bx,1001h
0006  03 C3               add ax,bx
0008  72 00               jc polok
```

```
000A  B9 11EE          polok: mov cx,11eeh
000D  73 00            jnc polokn
000F  BA 2233          polokn: mov dx,2233h
0012  EB 03 90         jmp a5
0015  F7 F1            div cx
0017  2B C1            a5: sub ax,cx
0019  74 00            jz hi
001B  03 D9            hi: add bx,cx
001D  75 00            jnz hlw
001F  03 CA            hlw: add cx,dx
0021  F4               hlt
0022                   main endp
0022                   end main
```

Experiment No: 06

Experiment Name: Study on Seven segment display operation to display different characters using MDA8086 microprocessor.

Objective:

1.To observe the seven segment display with common anode configuration.

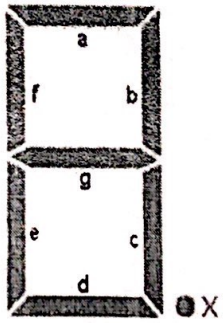
Theory:

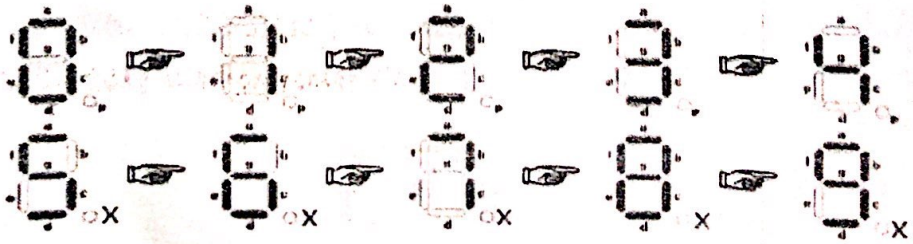
Seven segment display are available in two different configuration such as common cathodes , common anode as shows in figure below is a common anode configuration. The segment are turned on the applying '0' to the required segment LED and common cathode is opposite.

Equipment:

1. A PC with DOSBOX software installed.
2. MDA 8086 microprocessor kit.

Figure:





Data table and operation code:

DIGIT	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	1	1	0	0

Code: (For generating digit 7)

```
C       EQU  1FH
B       EQU 1BH
A       EQU 19H
ORG     F000H
MOV     AL,100000000
OUT     C,AL
MOV     AL,11111000
OUT     A,AL
```

Output :

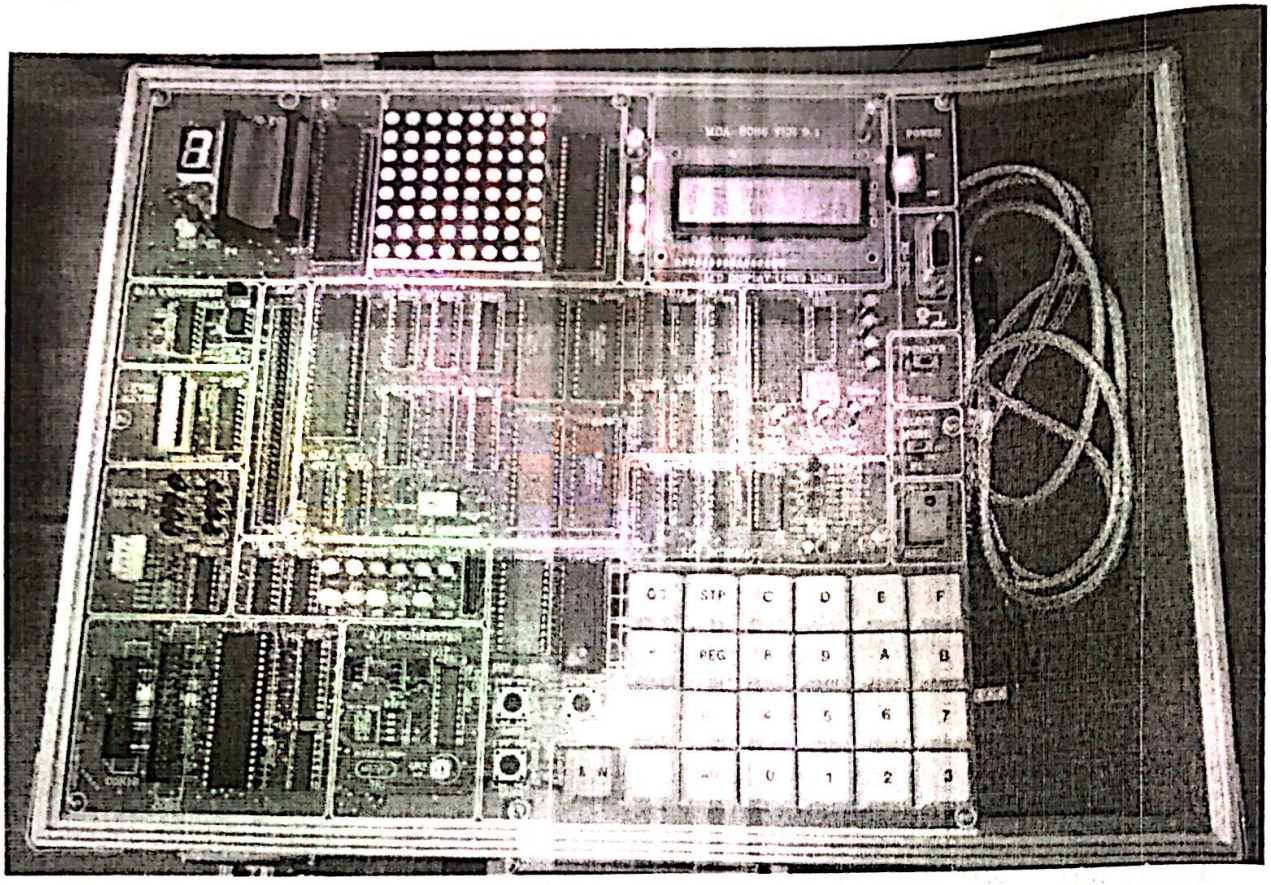


Fig: Displaying 7

Discussion:

1. In this experiment we operate the seven segment display.
2. We generate a machine code with respect to assembly language operation code by the DOSBOX software.
3. To display a different number such as software generate different code.
4. The 8086 microprocessor cannot operate the sever segment display along. So we need code from software.

Conclusion:

Different machine code generate foe each number. This code load into display MDA-8086 microprocessor. We see the 7 segment display in digital devices like watch, calculator etc.

Task:

1.Do print 2,9.

2. Do print 1 to 9 continuously.

Experiment no: 07

Experiment name: Study on the bit shifting of seven segment display in MDA 8086 using different conditional instruction

Objectives:

- Knowledge about seven segment
- Implement seven segment
- Displaying all the numbers by this code

Introduction: Microprocessor is an integrated circuit that contains all the functions of a central processing unit.

Theory: A seven segment display in a form of electronic display device for displaying decimal numbers that is an alternative to the more complex dot matrix displays. It is used in digital color, clocks, electronics meter, basic calculator etc. It is available in two different configuration command cathode, common anode.

number	X	g	f	E	D	C	b	a
0	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	1
2	1	0	1	0	0	1	0	0
3	1	0	1	1	0	0	0	0
4	1	0	0	1	1	0	0	1
5	1	0	0	1	0	0	1	0
6	1	0	0	0	0	0	1	0
7	1	1	1	1	1	0	0	0
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0

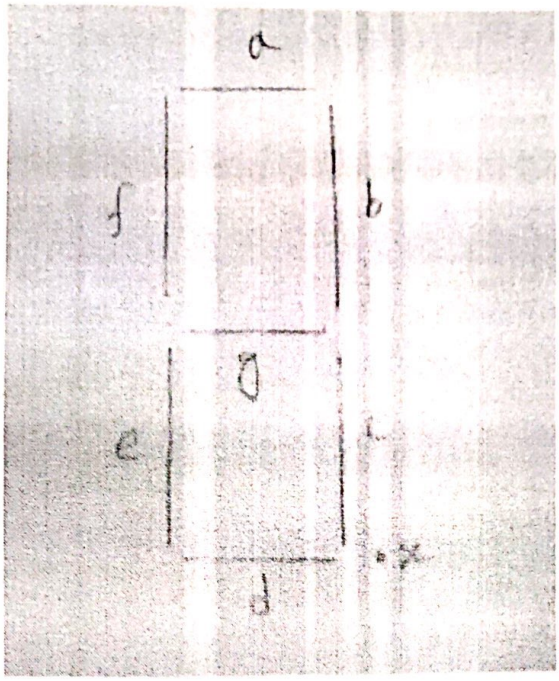


Fig: Seven segment display

As show in fig is a common anode configuration. The segment are turned on by applying 0 to the required segment LED and common cathode is opposite.

Equipment:

A pc with DosBox software and 8086 file.

MDA -8086 microprocessor.

Assembly Language: The assembly programming language is a low level language which is developed by using mnemonics. The microprocessor can understand only the binary language like 0's or 1's there for the assembler convert the assembly language to binary language are store it the memory to perform the task. Before writing the program the embedded designers must have sufficient knowledge on particular hardware of the controller or processor so first we required to know hardware of 8086 processor.

Instruction:

JMP: Use to jump instruction to provided address to proceed to the next instruction. The instruction during execution with some condition.

JZ: Use to jump if equal / zero flag zf=1

AND: The and instruction is used for supporting logic expression by performing bitwise AND operation.

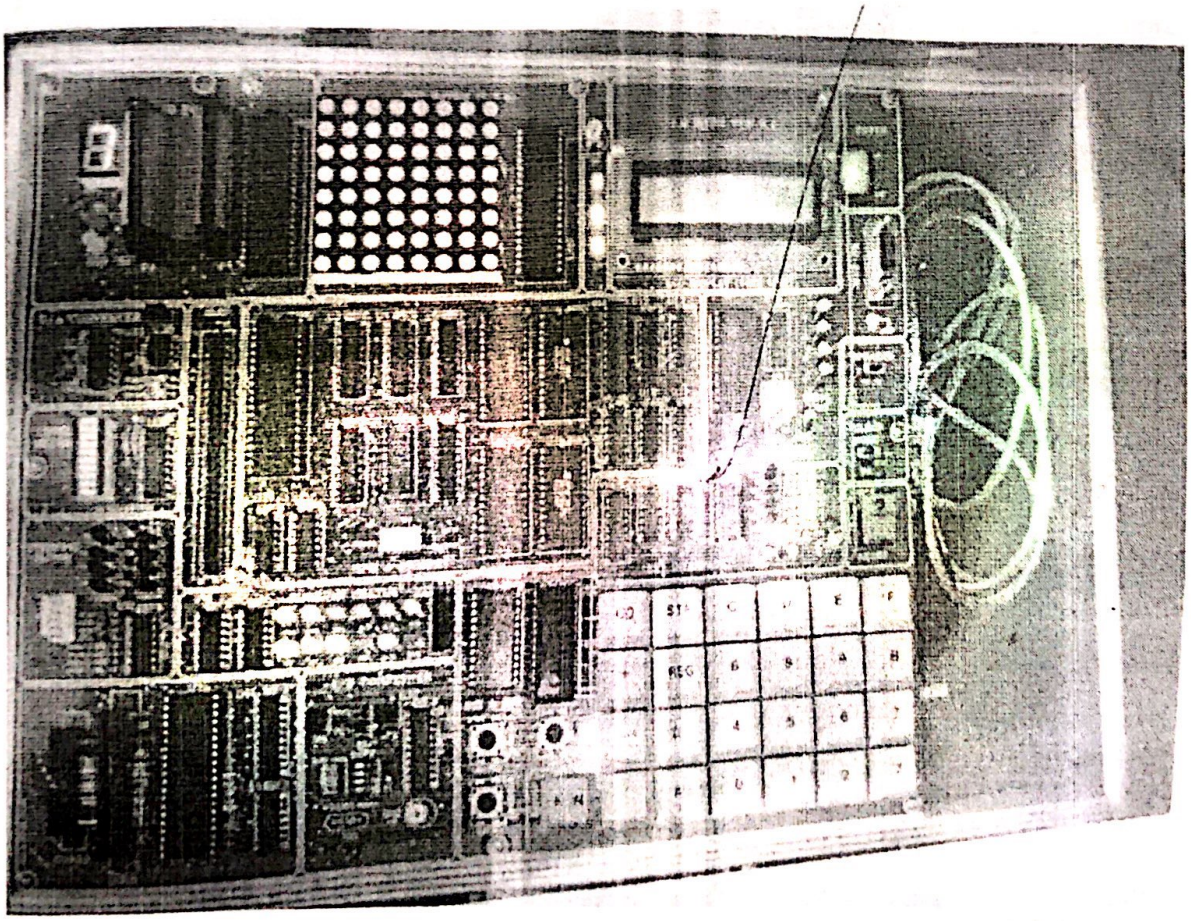
OR: The and instruction is using for supporting logic expression by performing bitwise OR operation.

NOT: The instruction implements the bitwise NOT operation. NOT operation reverse the bit in and operand.

CODE:

```
                        .model small
0100                    .stack 100h
0000                    .data
0000                    .code
0000                    main proc
= 001F                      ppic_c equ 1fh
= 001B                      ppib equ 1bh
= 0019                      ppia equ 19h
1000                    org 1000h
1000  B0 80              mov al,10000000b
1002  E6 1F              out ppic_c,al
1004  B0 FF              mov al,11111111b
1006  E6 19              out ppia,al
1008  B0 F1              l1:mov al,11110001b
100A  E6 19              l2:out ppia,al
100C  EB 09 90                jmp timer
100F  D0 E0              l3:shl al,1
1011  A8 10              test al,00010000b
1013  75 F3              jnz l
1015  EB F3              jmp l2
```

```
1017 90          timer:nop
1018 90          nop
1019 90          nop
101A 90          nop
101B 90          nop
101C 90          nop
101D 90          nop
101E 90          nop
101F EB EE       jmp l3
1021             main endp
1021             end main
```



Experiment No: 08

Experiment Name: Study on row shifting of dot matrix display using MDA 8086 microprocessor.

Objective:

i. To familiar with dot matrix and work with that

ii. Generating different shapes in dot matrix display

Theory:

A dot-matrix display is an electronic digital display device that displays information on machines, clocks and watches, public transport departure indicators and many other devices requiring a simple alphanumeric (and/or graphic) display device of limited resolution.

The display consists of a dot matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot matrix controller converts instructions from a processor into signals which turns on or off indicator elements in the matrix so that the required display is produced.

Command OR Instruction: Here use many type of command:

- **MOV:** The MOV instruction is the most important command in the 8086 because it moves data from one location to another.

format:
MOV destination,source

destination	source	example
register	register	mov ax,bx
register	immediate	mov ax,10h
register	memory	mov ax,es:[bx]
memory	immediate	mov aNumber,10h
memory	register	mov aDigit,ax

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)

- a memory location cannot be copied into another memory location (i.e. mov a number, a Digit)
- CS cannot be copied to (i.e. mov cs, ax)

- **JMP:** Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.
- **There are two types of Jump instructions:**

1. Unconditional Jump Instructions
2. Conditional Jump Instructions

1.Conditional Jump Instruction:

In these types of instructions, the processor must check for the particular condition. If it is true,only the jump takes place else the normal flow in the execution of the statements is maintained.
The ALU operations set flags in the status word (Flag register). The conditional jump statements tests the flag and jump is the flag is set.
There are following types of conditional jump instructions:
i) JC : Stands for 'Jump if Carry'
It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.
ii) JNC : Stands for 'Jump if Not Carry'
It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.
iii) JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'
It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.
iv) JNE / JNZ : Stands for 'Jump if Not Equal' or 'Jump if Not Zero'
It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.
v) JP / JPE : Stands for 'Jump if Parity' or 'Jump if Even Parity'
It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.
vi) JNP / JPO : Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'
It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.
vii) STC : Stands for 'Set Carry Flag'
Used to set carry flag CF to 1. Operation is the same in all modes.

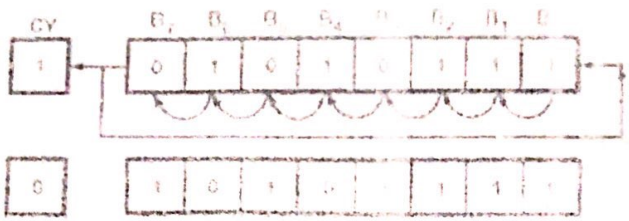
- **Rotate and Shift Instruction:**

The 8086 Rotate Instructions are namely

- **ROL**
- **ROR**
- **RCL**
- **RCR**

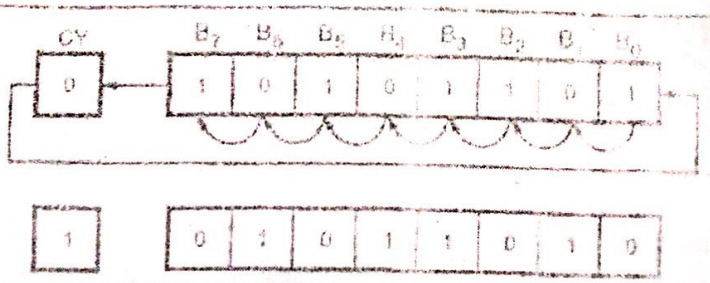
ROL Instruction: ROL destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.
Diagram shows ROL instruction for byte rotation.



The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROR Instruction: ROR destination, count.
This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. LSB is placed as a new MSB and a new CF.
The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.
Diagram shows ROR instruction for byte rotation.



Shift:
SHR: Shift Right
The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).
Syntax: SHR Register, Bits to be shifted
Example: SHR AX, 2
Working:

SHL: Shift Left

The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHL Register, Bits to be shifted

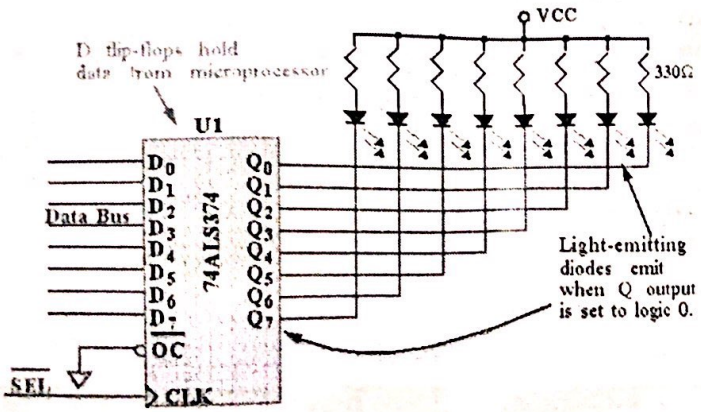
Example: SHL AX, 2

Working:

➢ **OUT:** Used to send out a byte or word from the accumulator to the provided port.

Basic I/O Interface

- Basic Output Interface:



Equipment:

i. 8086 microprocessor
ii. A PC
iii. DOSBox Software

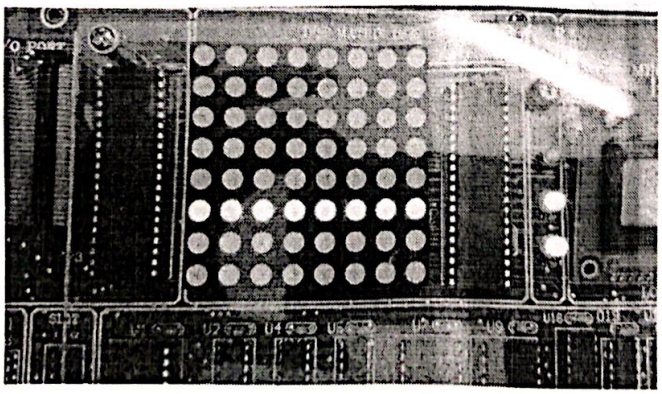
Code:

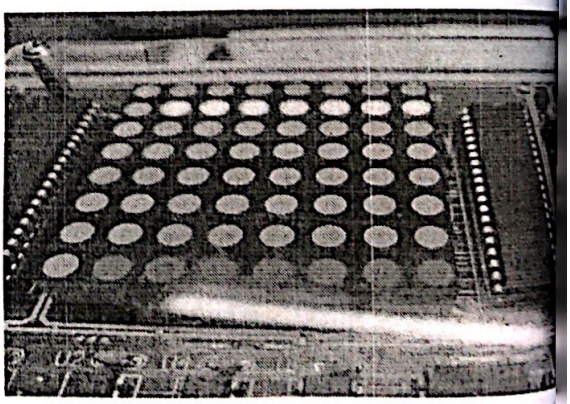
```
= 001E
= 001C
= 001A
= 0018

1000
1000  B0 80
1002  E6 1E
```

```
PPIC_C      EQU   1EH; control register
PPIC  EQU   1CH
PPIB  EQU   1AH
PPIA  EQU   18H
;
ORG   1000H
MOV   AL,10000000B
OUT   PPIC_C, AL
;
```

```
                                MOV  AL,11111111B
1004  B0 FF                     OUT  PPIC, AL
1006  E6 1C                     ;
                                MOV  AL,11111111B
1008  B0 FF                     OUT  PPIB, AL
100A  E6 1A                     ;
100C  B0 FE             L1:     MOV  AL,11111110B
100E  E6 18             L2:     OUT  PPIA, AL
1010  EB 09 90                   JMP  TIMER
1013  F9                L3:     STC
1014  D0 C0                     ROL  AL, 1
1016  72 F6                     JC   L2
1018  EB F2                     JMP  L1
                                ;
101A  CC                        INT  3
                                ;

101B  90                TIMER:  NOP
101C  90                        NOP
101D  90                        NOP
101E  90                        NOP
101F  EB F2                     JMP L3
                                ;
1021                            CODE ENDS
                                END
```

Figure:





Task:

1.Shift the matrix from third row

2.Change the color of row red to green and orange

3.Show two row with different color

Experiment No: 09

Experiment Name: Study on column shifting of dot matrix display using MDA 8086 microprocessor.

Objective:

iii. To familiar with dot matrix and work with that

iv. Generating different shapes in dot matrix display

Theory:

A dot-matrix display is an electronic digital display device that displays information on machines, clocks and watches, public transport departure indicators and many other devices requiring a simple alphanumeric (and/or graphic) display device of limited resolution.

The display consists of a dot matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot matrix controller converts instructions from a processor into signals which turns on or off indicator elements in the matrix so that the required display is produced.

Command OR Instruction: Here use many type of command:

- **MOV:** The MOV instruction is the most important command in the 8086 because it moves data from one location to another.

  format:

  MOV destination,source

destination	source	example
register	• register	mov ax,bx
register	immediate	mov ax,10h
register	memory	mov ax,es:[bx]
memory	immediate	mov aNumber,10h
memory	register	mov aDigit,ax

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)
- a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
- CS cannot be copied to (i.e. mov cs,ax)

- **JMP:** Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.
- **There are two types of Jump instructions:**
  3. Unconditional Jump Instructions
  4. Conditional Jump Instructions

1.Conditional Jump Instruction:

In these types of instructions, the processor must check for the particular condition. If it is true,only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements tests the flag and jump is the flag is set.

There are following types of conditional jump instructions:

i) JC : Stands for 'Jump if Carry'

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

ii) JNC : Stands for 'Jump if Not Carry'

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.

iii) JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.

iv) JNE / JNZ : Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.

v) JP / JPE : Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.

vi) JNP / JPO : Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.

vii) STC : Stands for 'Set Carry Flag'

Used to set carry flag CF to 1. Operation is the same in all modes.

➢ **Rotate and Shift Instruction:**

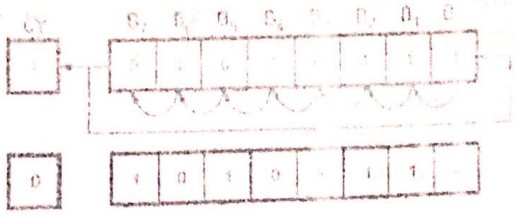
The 8086 Rotate Instructions are namely

- **ROL**
- **ROR**
- **RCL**
- **RCR**

ROL Instruction: ROL destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Diagram shows ROL instruction for byte rotation.



The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count

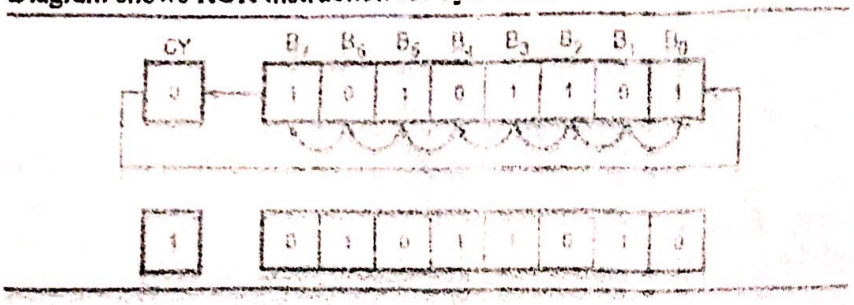
position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROR Instruction: ROR destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. LSB is placed as a new MSB and a new CF.

The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

Diagram shows ROR instruction for byte rotation.



Shift:

SHR: Shift Right

The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHR Register, Bits to be shifted

Example: SHR AX, 2

Working:

SHL: Shift Left

The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHL Register, Bits to be shifted

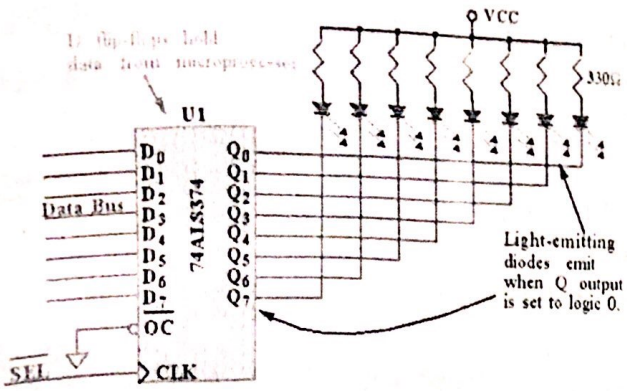
Example: SHL AX, 2

Working:

- **OUT:** Used to send out a byte or word from the accumulator to the provided port.

Basic I/O Interface

- Basic Output Interface:



Equipment:

iv. 8086 microprocessor
v. A PC
vi. DOSBox Software

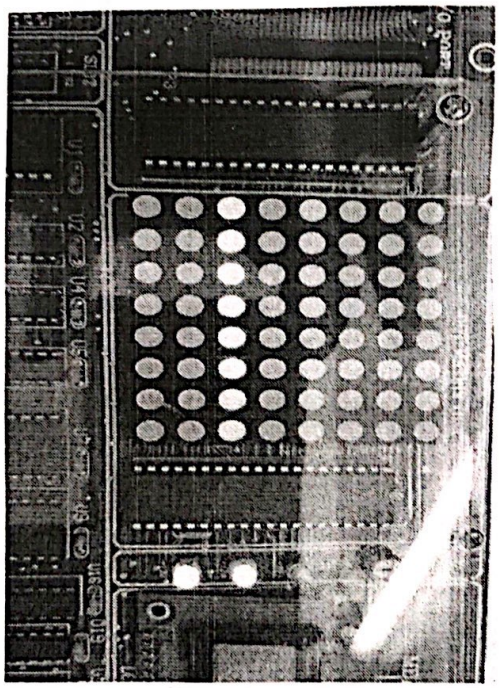
Code:

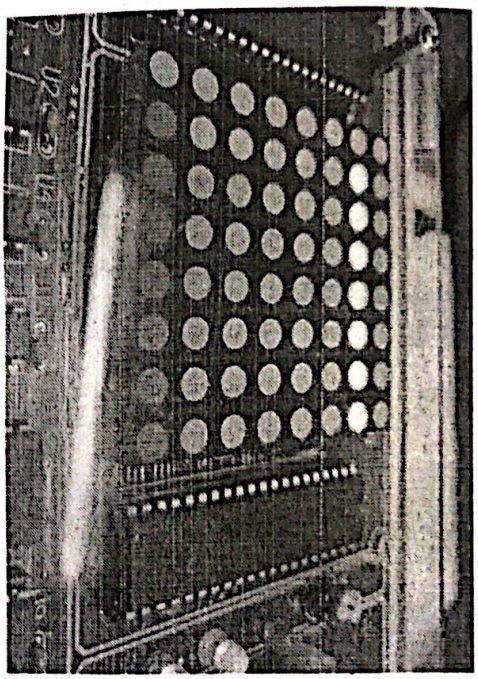
```
= 001E
= 001C
= 001A
= 0018

1000
1000  B0 80
1002  E6 1E

1004  B0 FF
1006  E6 1C

1008  B0 FF
100A  E6 1A

100C  B0 FE
100E  E6 18
1010  EB 09 90
1013  F9
1014  D0 C0
1016  72 F6
1018  EB F2

101A  CC
```

```
PPIC_C    EQU   1EH; control register
PPIC  EQU   1CH
PPIB  EQU   1AH
PPIA  EQU   18H
;
ORG   1000H
MOV   AL,10000000B
OUT   PPIC_C, AL
;
MOV   AL,11111111B
OUT   PPIC, AL
;
MOV   AL,11111111B
OUT   PPIB, AL
;
L1:   MOV   AL,11111110B
L2:   OUT   PPIA, AL
      JMP   TIMER
L3:   STC
      ROL   AL, 1
      JC    L2
      JMP   L1
;
      INT   3
```

```
                        ;
101B 90      TIMER:     NOP
101C 90                 NOP
101D 90                 NOP
101E 90                 NOP
101F EB F2              JMP L3
                        ;
                        CODE ENDS
1021                    END
```

Figure





Task:

1.Shift the matrix from third row

2.Change the color of row red to green and orange

3.Show two row with different color

Experiment No: 10

Experiment Name: Study on printing different letters in dot matrix display using MDA 8086 microprocessor.

Objective:

v. To familiar with dot matrix and work with that

vi. Generating different shapes in dot matrix display

Theory:

A dot-matrix display is an electronic digital display device that displays information on machines, clocks and watches, public transport departure indicators and many other devices requiring a simple alphanumeric (and/or graphic) display device of limited resolution.

The display consists of a dot matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot matrix controller converts instructions from a processor into signals which turns on or off indicator elements in the matrix so that the required display is produced.

Command OR Instruction: Here use many type of command:

- **MOV:** The MOV instruction is the most important command in the 8086 because it moves data from one location to another.

  format:

  MOV destination,source

destination	source	example
register	register	mov ax,bx
register	immediate	mov ax,10h
register	memory	mov ax,es:[bx]
memory	immediate	mov aNumber,10h
memory	register	mov aDigit,ax

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)
- a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
- CS cannot be copied to (i.e. mov cs,ax)

- JMP: Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.
- There are two types of Jump instructions:

5. Unconditional Jump Instructions
6. Conditional Jump Instructions

1.Conditional Jump Instruction:

In these types of instructions, the processor must check for the particular condition. If it is true,only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements tests the flag and jump is the flag is set.

There are following types of conditional jump instructions:

i) JC : Stands for 'Jump if Carry'

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

ii) JNC : Stands for 'Jump if Not Carry'

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.

iii) JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.

iv) JNE / JNZ : Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.

v) JP / JPE : Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.

vi) JNP / JPO : Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.

vii) STC : Stands for 'Set Carry Flag'

Used to set carry flag CF to 1. Operation is the same in all modes.

- **Rotate and Shift Instruction:**

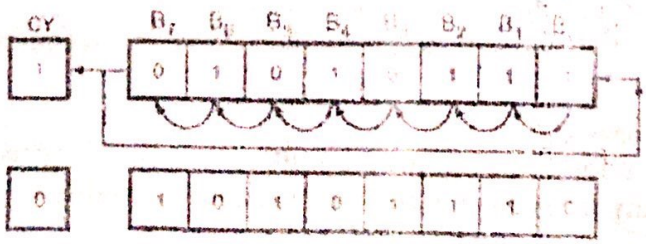
The 8086 Rotate Instructions are namely

- **ROL**
- **ROR**
- **RCL**
- **RCR**

ROL Instruction: ROL destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Diagram shows ROL instruction for byte rotation.



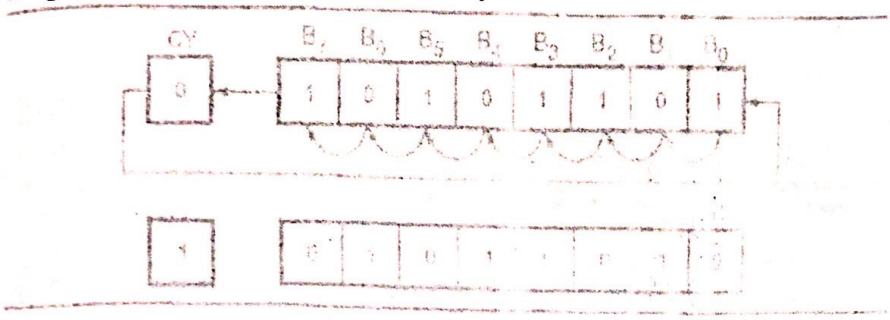
The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROR Instruction: ROR destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. LSB is placed as a new MSB and a new CF.

The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

Diagram shows ROR instruction for byte rotation.



Shift:

SHR: Shift Right

The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHR Register, Bits to be shifted

Example: SHR AX, 2

Working:

SHL: Shift Left

The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHL Register, Bits to be shifted

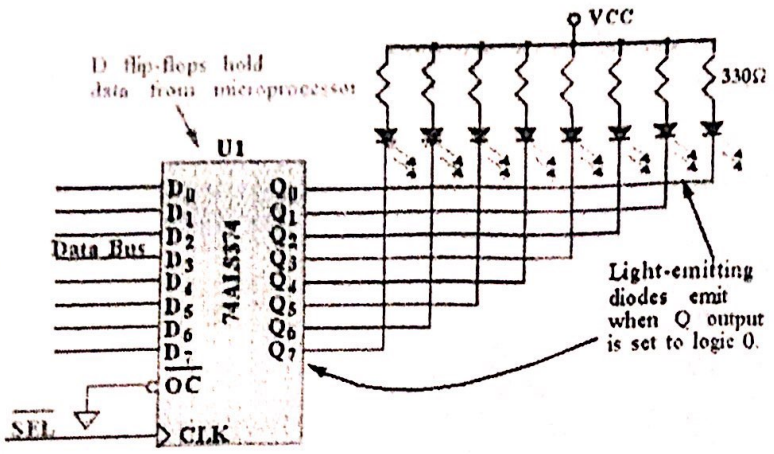
Example: SHL AX, 2

Working:

➢ **OUT:** Used to send out a byte or word from the accumulator to the provided port.

Basic I/O Interface

- Basic Output Interface:



Equipment:

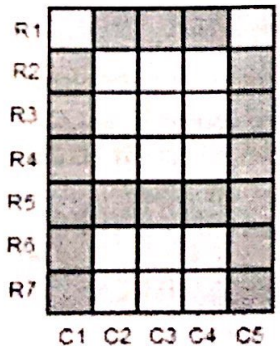
vii. 8086 microprocessor
viii. A PC
ix. DOSBox Software

Code:

```
= 001E                          PPIC_C      EQU   1EH; control register
= 001C                          PPIC  EQU   1CH
= 001A                          PPIB  EQU   1AH
= 0018                          PPIA  EQU   18H
                                ;
1000                            ORG   1000H
1000  B0 80                     MOV   AL,10000000B
1002  E6 1E                     OUT   PPIC_C, AL
                                ;
1004  B0 FF                     MOV   AL,11111111B
1006  E6 1C                     OUT   PPIC, AL
                                ;
1008  B0 FF                     MOV   AL,11111111B
100A E6 1A                      OUT   PPIB, AL
                                ;
100C  B0 FE             L1:     MOV   AL,11111110B
100E  E6 18             L2:     OUT   PPIA, AL
```

```
1010 EB 09 90                JMP   TIMER
1013 F9              L3:     STC
1014 D0 C0                   ROL   AL, 1
1016 72 F6                   JC    L2
1018 EB F2                   JMP   L1
                             ;
101A CC                      INT   3
                             ;

101B 90              TIMER:  NOP
101C 90                      NOP
101D 90                      NOP
101E 90                      NOP
101F EB F2                   JMP L3
                             ;
1021                         CODE ENDS
                             END
```

Figure



Task:

1.Shift the matrix from third row

2.Change the color of row red to green and orange

3.Show two row with different color

Experiment No: 11

Experiment Name: Study on how to scroll a letter from left to right in dot matrix display using MDA 8086 microprocessor.

Objective:

vii. To familiar with dot matrix and work with that

viii. Generating different shapes in dot matrix display

Theory:

A dot-matrix display is an electronic digital display device that displays information on machines, clocks and watches, public transport departure indicators and many other devices requiring a simple alphanumeric (and/or graphic) display device of limited resolution.

The display consists of a dot matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot matrix controller converts instructions from a processor into signals which turns on or off indicator elements in the matrix so that the required display is produced.

Command OR Instruction: Here use many type of command:

- **MOV:** The MOV instruction is the most important command in the 8086 because it moves data from one location to another.

  format:

  MOV destination,source

destination	source	example
register	register	mov ax,bx
register	immediate	mov ax,10h
register	memory	mov ax,es:[bx]
memory	immediate	mov aNumber,10h
memory	register	mov aDigit,ax

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)
- a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
- CS cannot be copied to (i.e. mov cs,ax)

- JMP: Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.
- There are two types of Jump instructions:

7. Unconditional Jump Instructions
8. Conditional Jump Instructions

1.Conditional Jump Instruction:

In these types of instructions, the processor must check for the particular condition. If it is true,only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements tests the flag and jump is the flag is set.

There are following types of conditional jump instructions:

i) JC : Stands for 'Jump if Carry'

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

ii) JNC : Stands for 'Jump if Not Carry'

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.

iii) JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.

iv) JNE / JNZ : Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.

v) JP / JPE : Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.

vi) JNP / JPO : Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.

vii) STC : Stands for 'Set Carry Flag'

Used to set carry flag CF to 1. Operation is the same in all modes.

➢ **Rotate and Shift Instruction:**

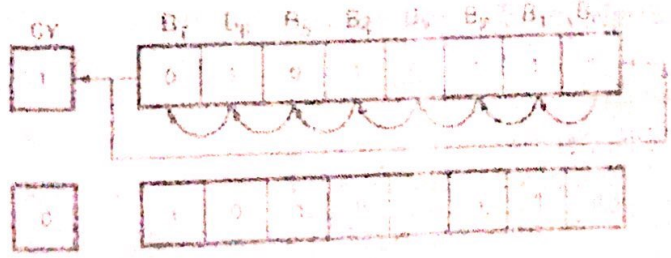
The 8086 Rotate Instructions are namely

- **ROL**
- **ROR**
- **RCL**
- **RCR**

ROL Instruction: ROL destination, count.

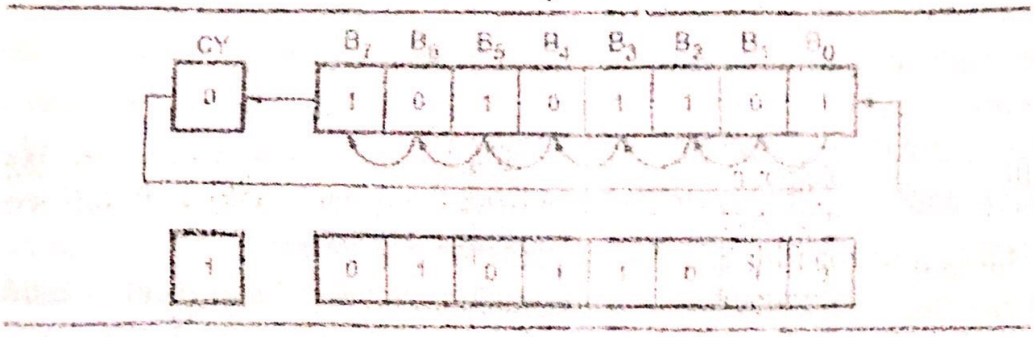
This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Diagram shows ROL instruction for byte rotation.



The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROR Instruction: ROR destination, count.
This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. LSB is placed as a new MSB and a new CF.
The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.
Diagram shows ROR instruction for byte rotation.



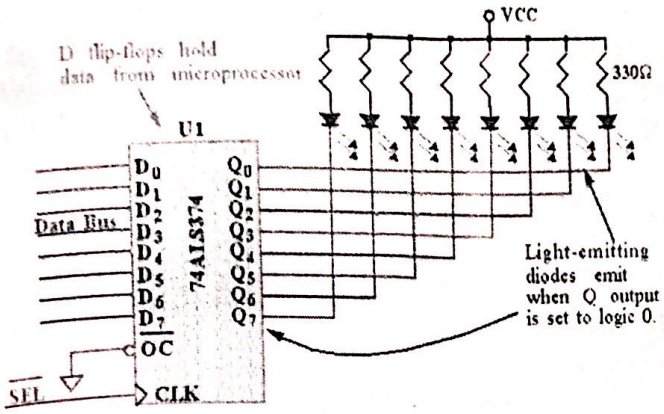
Shift:
SHR: Shift Right
The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).
Syntax: SHR Register, Bits to be shifted
Example: SHR AX, 2
Working:

SHL: Shift Left
The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).
Syntax: SHL Register, Bits to be shifted
Example: SHL AX, 2
Working:

- **OUT:** Used to send out a byte or word from the accumulator to the provided port.

Basic I/O Interface

- Basic Output Interface:



Equipment:

x. 8086 microprocessor
xi. A PC
xii. DOSBox Software

Code:

```
                        ;*****************************************
                        ;   MDA-Win8086 EXPERIMENT PROGRAM    *
                        ;   FILENAME  : MATRIX_3.ASM
                        ;   PROCESSOR : I8086
                        ;   DOT MATRIX TEST
                        ;*****************************************
                        ;
0000                    CODE SEGMENT
                            ASSUME    CS:CODE,DS:CODE,ES:CODE,SS:CODE
                            ;
                            PPIC_C    EQU   1EH ; control register
= 001E                      PPIC      EQU   1CH
= 001C                      PPIB      EQU   1AH
= 001A                      PPIA      EQU   18H
= 0018                      ;
                            ORG   1000H
1000
```

```
1000 B0 80                      MOV  AL,10000000B
1002 E6 1E                      OUT  PPIC_C,AL
                                ;
1004 B0 FF                      MOV  AL,11111111B
1006 E6 18                      OUT  PPIA,AL
                                ;
1008 BE 1040 R          L1:     MOV  SI,OFFSET FONT1
                                ;
100B B3 08                      MOV  BL,8 ; font No.
100D B7 1E              L3:     MOV  BH,30 ; display time
                                ;
100F 56                 L2:     PUSH SI
1010 E8 1021 R                  CALL SCAN
1013 5E                         POP  SI
1014 FE CF                      DEC  BH
1016 75 F7                      JNZ  L2
                                ;
1018 83 C6 08                       ADD  SI,8
101B FE CB                      DEC  BL
101D 75 EE                      JNZ  L3
                            ;
101F EB E7                      JMP  L1
                                ;
                                ;
                                ;
1021                    SCAN    PROC      NEAR
1021 B4 01                      MOV  AH,00000001B
1023 2E: 8A 04          SCAN1:      MOV  AL,BYTE PTR CS:[SI]
1026 E6 1A                      OUT  PPIB.AL
                                ;
1028 8A C4                      MOV  AL,AH
102A E6 1C                      OUT  PPIC,AL
102C E8 1036 R                  CALL TIMER
102F 46                         INC  SI
1030 F8                         CLC
1031 D0 C4                      ROL  AH,1
1033 73 EE                      JNC  SCAN1
1035 C3                         RET
1036                    SCAN ENDP

                                ;
1036 B9 012C            TIMER:      MOV  CX,300
1039 90                 TIMER1:     NOP
103A 90                         NOP
103B 90                         NOP
103C 90                         NOP
103D E2 FA                      LOOP TIMER1
```

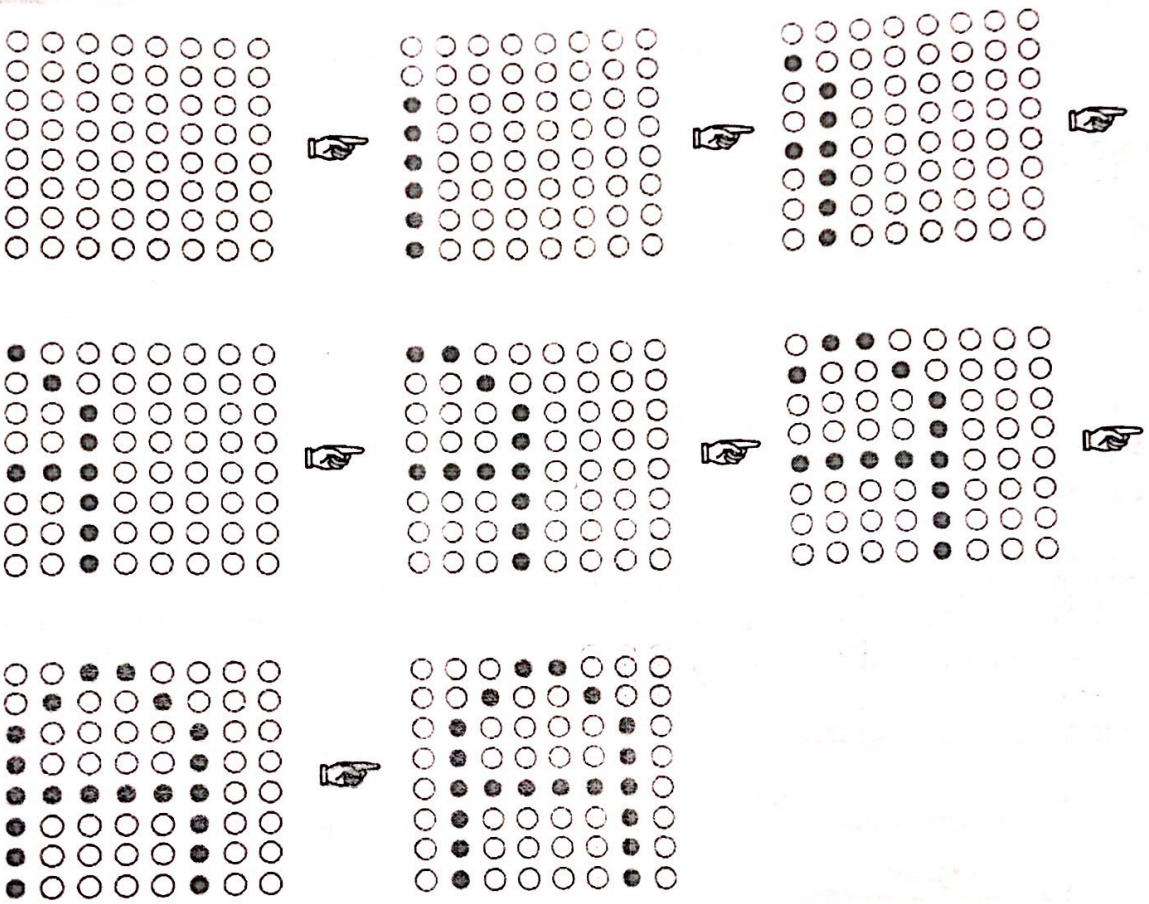
```
103F C3                 RET
                        ;
1040 FF         FONT1:  DB      11111111B
1041 FF                 DB      11111111B
1042 FF                 DB      11111111B
1043 FF                 DB      11111111B
1044 FF                 DB      11111111B
1045 FF                 DB      11111111B
1046 FF                 DB      11111111B
1047 FF                 DB      11111111B
                        ;
1048 C0         FONT2:  DB      11000000B
1049 FF                 DB      11111111B
104A FF                 DB      11111111B
104B FF                 DB      11111111B
104C FF                 DB      11111111B
104D FF                 DB      11111111B
104E FF                 DB      11111111B
104F FF                 DB      11111111B
                        ;
1050 B7         FONT3:  DB      10110111B
1051 C0                 DB      11000000B
1052 FF                 DB      11111111B
1053 FF                 DB      11111111B
1054 FF                 DB      11111111B
1055 FF                 DB      11111111B
1056 FF                 DB      11111111B
1057 FF                 DB      11111111B
                        ;
1058 77         FONT4:  DB      01110111B
1059 B7                 DB      10110111B
105A C0                 DB      11000000B
105B FF                 DB      11111111B
105C FF                 DB      11111111B
105D FF                 DB      11111111B
105E FF                 DB      11111111B
105F FF                 DB      11111111B
                        ;
1060 77         FONT5:  DB      01110111B
1061 77                 DB      01110111B
1062 B7                 DB      10110111B
1063 C0                 DB      11000000B
1064 FF                 DB      11111111B
1065 FF                 DB      11111111B
1066 FF                 DB      11111111B
1067 FF                 DB      11111111B
                        ;
```

```
1068 B7                 FONT6: DB    10110111B
1069 77                        DB    01110111B
106A 77                        DB    01110111B
106B B7                        DB    10110111B
106C C0                        DB    11000000B
106D FF                        DB    11111111B
106E FF                          DB    11111111B
106F FF                          DB    11111111B
                                ;
1070 C0                 FONT7: DB    11000000B
1071 B7                        DB    10110111B
1072 77                        DB    01110111B
1073 77                        DB    01110111B
1074 B7                        DB    10110111B
1075 C0                        DB    11000000B
1076 FF                        DB    11111111B
1077 FF                          DB    11111111B
                                ;
1078 FF                 FONT8: DB    11111111B
1079 C0                        DB    11000000B
107A B7                        DB    10110111B
107B 77                        DB    01110111B
107C 77                        DB    01110111B
107D B7                        DB    10110111B
107E C0                        DB    11000000B
107F FF                        DB    11111111B

                                ;
1080                    CODE ENDS
                               END
```

Figure

4. Matrix - Scroll 'A' left to right

Purpose



Task:

1.Shift the matrix from third row

2.Change the color of row red to green and orange

3.Show two row with different color

Experiment No: 12

Experiment Name: Study on how to scroll a letter from top to bottom in dot matrix display using MDA 8086 microprocessor.

Objective:

i. To familiar with dot matrix and work with that
ii. Generating different shapes in dot matrix display

Theory:

A dot-matrix display is an electronic digital display device that displays information on machines, clocks and watches, public transport departure indicators and many other devices requiring a simple alphanumeric (and/or graphic) display device of limited resolution.

The display consists of a dot matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot matrix controller converts instructions from a processor into signals which turns on or off indicator elements in the matrix so that the required display is produced.

Command OR Instruction: Here use many type of command:

- **MOV:** The MOV instruction is the most important command in the 8086 because it moves data from one location to another.

  format:

  MOV destination,source

destination	source	example
register	register	mov ax,bx
register	immediate	mov ax,10h
register	memory	mov ax,es:[bx]
memory	immediate	mov aNumber,10h
memory	register	mov aDigit,ax

  The MOV instruction has a few limitations:
  - an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
  - segment registers cannot be copied directly (i.e. mov es,ds)
  - a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
  - CS cannot be copied to (i.e. mov cs,ax)
- JMP: Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.
- There are two types of Jump instructions:

9. Unconditional Jump Instructions
10. Conditional Jump Instructions

1.Conditional Jump Instruction:

In these types of instructions, the processor must check for the particular condition. If it is true,only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements tests the flag and jump is the flag is set.

There are following types of conditional jump instructions:

i) JC : Stands for 'Jump if Carry'

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

ii) JNC : Stands for 'Jump if Not Carry'

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.

iii) JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.

iv) JNE / JNZ : Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, then jump.

v) JP / JPE : Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, then jump.

vi) JNP / JPO : Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = 0, then jump.

vii) STC : Stands for 'Set Carry Flag'

Used to set carry flag CF to 1. Operation is the same in all modes.

- **Rotate and Shift Instruction:**

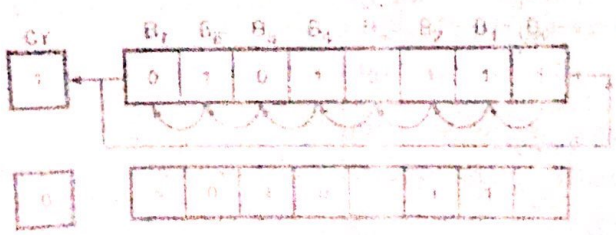
The 8086 Rotate Instructions are namely

- **ROL**
- **ROR**
- **RCL**
- **RCR**

ROL Instruction: ROL destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Diagram shows ROL instruction for byte rotation.



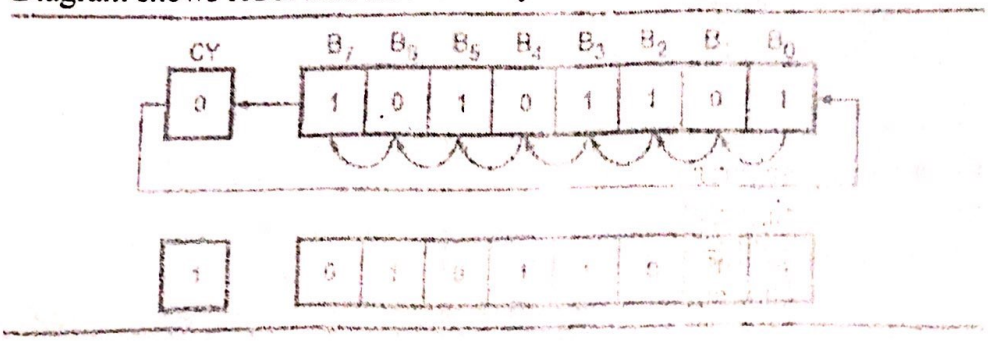
The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROR Instruction: ROR destination, count.

This 8086 Rotate Instructions all bits in a specified byte or word to the left some number of bit positions. LSB is placed as a new MSB and a new CF.

The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

Diagram shows ROR instruction for byte rotation.



Shift:

SHR: Shift Right

The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHR Register, Bits to be shifted

Example: SHR AX, 2

Working:

SHL: Shift Left

The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHL Register, Bits to be shifted

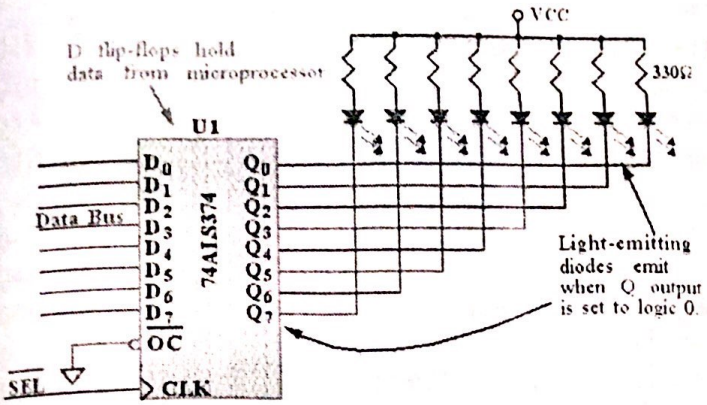
Example: SHL AX, 2

Working:

➢ **OUT:** Used to send out a byte or word from the accumulator to the provided port.

Basic I/O Interface

- Basic Output Interface:



Equipment:

xiii. 8086 microprocessor
xiv. A PC
xv. DOSBox Software

Code:

```
                    ;*****************************************
                    ;   MDA-Win8086 EXPERIMENT PROGRAM    *
                    ;   FILENAME  : MATRIX_4.ASM
                    ;   PROCESSOR : I8086
                    ;   DOT MATRIX TEST
                    ;*****************************************
                    CODE SEGMENT
0000                      ASSUME    CS:CODE,DS:CODE,ES:CODE,SS:CODE
                          ;
                          PPIC_C    EQU   1EH ; control register
= 001E                    PPIC      EQU   1CH ; c port
= 001C                    PPIB      EQU   1AH
= 001A                    PPIA      EQU   18H
= 0018                    ;
                          ORG   1000H
```

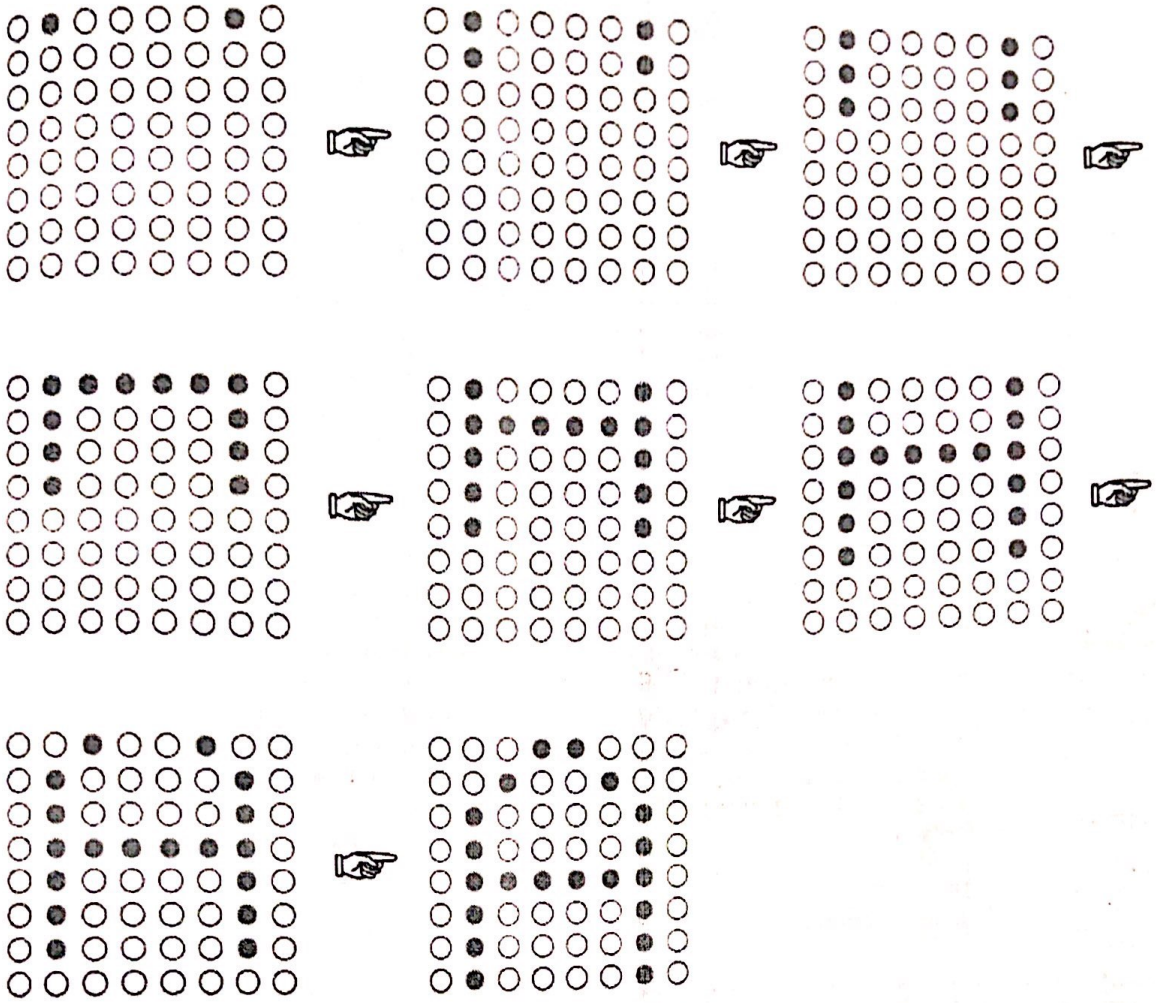


```
1000
```

```
1000 B0 80              MOV  AL,10000000B
1002 E6 1E              OUT  PPIC_C,AL
                        ;
1004 B0 FF              MOV  AL,11111111B
1006 E6 18              OUT  PPIA,AL
                        ;
1008 BE 1040 R      L1: MOV  SI,OFFSET FONT1
                        ;
100B B3 08              MOV  BL,8 ; font No.
100D B7 1E          L3: MOV  BH,30 ; display time
                        ;
100F 56             L2: PUSH SI
1010 E8 1021 R          CALL SCAN
1013 5E                 POP  SI
1014 FE CF              DEC  BH
1016 75 F7              JNZ  L2
                        ;
1018 83 C6 08           ADD  SI,8
101B FE CB              DEC  BL
101D 75 EE              JNZ  L3
                    ;
101F EB E7              JMP  L1
                        ;
                        ;
                        ;
1021                SCAN PROC   NEAR
1021 B4 01              MOV  AH,00000001B
1023 2E: 8A 04      SCAN1:  MOV  AL,BYTE PTR CS:[SI]
1026 E6 1A              OUT  PPIB,AL
                        ;
1028 8A C4              MOV  AL,AH
102A E6 1C              OUT  PPIC,AL
102C E8 1036 R          CALL TIMER
102F 46                 INC  SI
1030 F8                 CLC
1031 D0 C4              ROL  AH,1
1033 73 EE              JNC  SCAN1
1035 C3                 RET
1036                SCAN ENDP

                        ;
1036 B9 012C        TIMER:  MOV  CX,300
1039 90             TIMER1: NOP
103A 90                 NOP
103B 90                 NOP
103C 90                 NOP
103D E2 FA              LOOP TIMER1
```

```
103F C3
                    RET
1040 FF             ;
1041 7F     FONT1: DB   11111111B
1042 FF          DB   01111111B
1043 FF            DB   11111111B
1044 FF            DB   11111111B
1045 FF          DB   11111111B
1046 7F            DB   11111111B
1047 FF          DB   01111111B
                   DB   11111111B
1048 FF            ;
1049 3F     FONT2: DB   11111111B
104A FF          DB   00111111B
104B FF            DB   11111111B
104C FF            DB   11111111B
104D FF          DB   11111111B
104E 3F            DB   11111111B
104F FF          DB   00111111B
                   DB   11111111B
1050 FF            ;
1051 1F     FONT3: DB   11111111B
1052 FF          DB   00011111B
1053 FF            DB   11111111B
1054 FF            DB   11111111B
1055 FF          DB   11111111B
1056 1F            DB   11111111B
1057 FF          DB   00011111B
                   DB   11111111B
1058 FF            ;
1059 0F     FONT4: DB   11111111B
105A 7F          DB   00001111B
105B 7F          DB   01111111B
105C 7F          DB   01111111B
105D 7F          DB   01111111B
105E 0F          DB   01111111B
105F FF          DB   00001111B
                   DB   11111111B
1060 FF            ;
1061 07     FONT5: DB   11111111B
1062 BF          DB   00000111B
1063 BF          DB   10111111B
1064 BF          DB   10111111B
1065 BF          DB   10111111B
1066 07          DB   10111111B
1067 FF          DB   00000111B
                   DB   11111111B
                   ;
```

```
1068 FF                 FONT6: DB    11111111B
1069 03                        DB    00000011B
106A DF                        DB    11011111B
106B DF                        DB    11011111B
106C DF                        DB    11011111B
106D DF                        DB    11011111B
106E 03                        DB    00000011B
106F FF                        DB    11111111B
                               ;
1070 FF                 FONT7: DB    11111111B
1071 81                        DB    10000001B
1072 6F                        DB    01101111B
1073 EF                        DB    11101111B
1074 EF                        DB    11101111B
1075 6F                        DB    01101111B
1076 81                        DB    10000001B
1077 FF                        DB    11111111B
                               ;
1078 FF                 FONT8: DB    11111111B
1079 C0                        DB    11000000B
107A B7                        DB    10110111B
107B 77                        DB    01110111B
107C 77                        DB    01110111B
107D B7                        DB    10110111B
107E C0                        DB    11000000B
107F FF                        DB    11111111B

                               ;
1080                    CODE ENDS
                               END
```



Task:

1.Shift the matrix from third row

2.Change the color of row red to green and orange

3.Show two row with different color

Table of Content

Summary of Lab Report

Ex. No.	Experiment Title	Date of Exp.	Date of Subm	Marks	Teacher's Signature	Remarks
01	Study on the familiarization with MDA-8086 microprocessor kit and its operation in "Machine Code" mode					
02	Study on the observations of MNEMONIC operations with machine code in MDA-8086 microprocessor kit.					
03	Study on how to load the machine codes of a sample program to MDA-8086 then execute it and verify the results.					
04	Study on different types of arithmetic and logic operation in assembly language					
05	Study on the different jump condition of MDA 8086 microprocessor					
06	Study on Seven segment display operation to display different characters using MDA8086 microprocessor					
07	Study on the bit shifting of seven segment display in MDA 8086 using different conditional instruction.					
08	Study on row shifting of dot matrix display using MDA 8086 microprocessor.					
09	Study on column shifting of dot matrix display using MDA 8086 microprocessor.					
10	Study on printing different letters in dot matrix display using MDA 8086 microprocessor.					
11	Study on how to scroll a letter from left to right in dot matrix display using MDA 8086 microprocessor.					
12	Study on how to scroll a letter from top to bottom in dot matrix display using MDA 8086 microprocessor.					
	Average marks in lab Report (out of 20)					

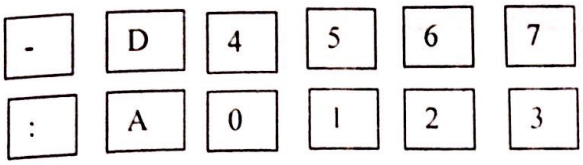
Format of Lab Report

All lab reports should have to follow the common format as below

a. Experiment No
b. Experiment Title
c. Objectives
d. Introduction
e. Theory Overview (with formula/equations and/or figure if required)
f. Equipment's
g. Diagram (with adequate labeling and figure caption)
h. Results
i. Discussion
j. Conclusion
k. References

Instruction for Lab Report Writing

a. Lab report must be hand written without copying from other works.
b. Writing should be neat and clean with proper caption and labeling in figure and table.
c. The title page of report should contain all the basic information such as experiment no & title, course code & title, student's information, teacher's information, experiment date, submission date.
d. Result should include calculated and/or simulated and/or measured data with proper unit.
e. Table and/or graph of result should be neat and clear with axis label and units where applicable.
f. The discussion should present your findings from the experiment. Evaluate the outcome objectively, taking a candid and unbiased point of view. Suppose that the outcome is not close to what you expected. Even then, after checking your results, give reasons why you believe that outcome is not consistent with the expected.



RE system reset ST execute user's program, a single step

AD set memory address GO go to user's program or execute monitor function

D update segment & offset MO Immediately break user's and input data to memory program and Non markable interrupt

: Offset set. RE Register Display.

\+ Segment & Offset +1 increment

Register display increment.

\- Segment & Offset -1 increment

Register display decrement.

Figure 4: Keys and Functions

Experiment No: 02

Experiment Name: Study on the observations of MNEMONIC operations with machine code in MDA-8086 microprocessor kit.

Objectives:

1. To observe the result MNEMONIC operations with machine code in MDA-8086 microprocessor kit.

Theory:

MNEMONIC: A system such as a pattern of letters, ideas or associations which assists in remembering something. In other words, A combination of letters to suggest the operation of an instruction.

Machine code: A computer programming language consisting of binary and hexadecimal instructions which a computer can respond to directly. It uses the instruction set of particular computer architecture.

Assembly language: A medium of communication with a computer in which programs are written in mnemonics.

Instructions:

MOV: MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)
- a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
- CS cannot be copied to (i.e. mov cs,ax)

SHAF: Loads the SF, ZF, AF, PF, and CF flags of the EFLAGS register with values from the corresponding bits in the AH register (bits 7, 6, 4, 2, and 0, respectively). Bits 1, 3, and 5 of register AH are ignored; the corresponding reserved bits (1, 3, and 5) in the EFLAGS register

ADD AX, 4789: This adds the value 4789 to the contents of AX and leaves the result in AX.

ADC: Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a carry from a previous addition.

SUB: Subtract the destination operand (first operand), the source operand (second operand) stores the result in the destination operand. Used to subtract the byte from byte/word from word.

SBB: Subtract with borrow instruction subtract the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Used to perform subtraction with borrow.

DEC: The decrement instruction subtracts one from the contents of the specified register or memory location .Used to decrement the provided byte/word by 1.

INC: The instruction increases the contents of the specified register or memory location by 1.Immediate data cannot be operand of this instruction .Used to increment the provided byte/word by 1.

CBW: Used to fill the upper byte of the word with the copies of sign bit of the lower byte.

MUL: Used to multiply unsigned byte by byte/word by word.

IMUL: Used to multiply signed byte by byte/word by word.

DIV: Used to divide the unsigned word by byte or unsigned double word by word.

IDIV: Used to divide the signed word by byte or signed double word by word.

ADDRESS	MACHINE CODE	MNEMONIC
1000	B8 0000	MOV AX, 0
1003	9E	SAHF