** Daffodil International University (DIU)**

**Department of Electrical and Electronic Engineering**

**EEE 324: Microprocessors and Interfacing Laboratory**

**Experiment No: 10**

**Name of the Experiment:   
Writing a C language program to turn ON/OFF 8x8 Matrix LED present in microprocessor 8086 kit and load it to memory using Wincom software.**

## Dot Matrix LED Display:

KMD D1288C is 8x8 dotmatrix display. Both orange and green LEDs are present in each point. In the same column 8 orange and 8 green leds’ anodes(+ terminals) are all connected to a bit of port C of an 8255A(named U30, see board). In the same row 8 orange leds’ cathodes (- terminals) are all connected to a bit of port A. In the same row 8 green leds’ cathodes (- terminals) are all connected to a bit of port B.

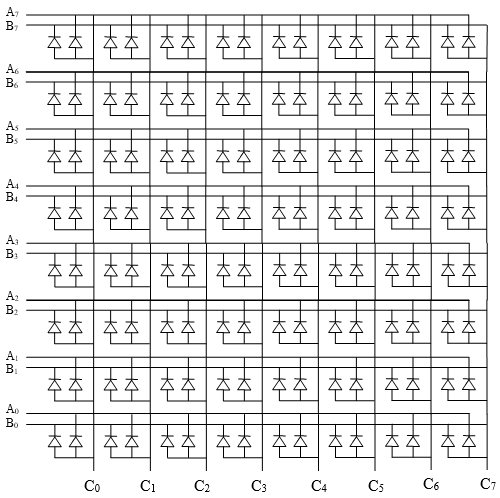


Figure: C. A. arrangement of 8x8 bi-color LED

## Some syntax rules in C language:

In C, hexadecimal numbers are written in form of 0x(Num)

Whatever is written after // is considered as comment

Whatever is written between /\* and \*/ also is considered as comment

#include “mde8086.h” adds a file mde8086.h from current/header file directory during compile time

Outportb(address,data) sends one byte data to output port located at particular address.

C-8086 is a batch file………….

## Procedure:

1. Create a source file using editor program: Open: C:\mda\8086\8086c>edit ↵
2. We use the built-in program.
3. From File menu Open Matrix file. Or write the code

# void main(void)

# {

# outportb( 0x1e, 0x80 ); //0x1e is address of control register of 8255

# outportb( 0x18, 0xff ); //0x18 is address of port A of 8255

# outportb( 0x1a, 0x55 ); //0x1a is address of port B of 8255

# outportb( 0x1c, 0xff ); //0x1c is address of port C of 8255

# }

1. Save the file as matrix.c

## Creating .abs file and .obj file from .c file

1. Make sure that folder containing your file also have TCC.EXE , MEXE2ABS.EXE, LINK.EXE, CS.LIB, RAMSTART.OBJ, TASM.EXE
2. In command prompt browse to your folder.
3. Write command **c-8086 matrix**↵
4. C:\mda\8086\8086c>c-8086 matrix ↵
5. You will see

C:\mda\my>c-8086 matrix

Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International

matrix.c:

Turbo Assembler Version 1.0 Copyright (c) 1988 by Borland International

Assembling file: MATRIX.ASM

Error messages: None

Warning messages: None

Remaining memory: 302k

Available memory 439410

C:\mda\my>link ramstart+matrix, matrix, matrix, cs.lib

Microsoft (R) Overlay Linker Version 3.60

Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

LINK : warning L4021: no stack segment

Press any key to continue . . .

Then

Press Enter

Press Enter

Press Enter

# >>Change Execution file name (\*.exe): matrix.exe↵<<

# Program of Segment: 0000

# Program of Offset: 1000

# Are you want to set up a jump instruction address of FFFF0H (Y/N)? N ↵

# It is all done making HEX (\*.ABS)!

# Press any key to return DOS......

1. Then see there is matrix.abs filein that folder.
2. Start Wincomm software and reset the MDE-8086(which must be in serial monitor mode).
3. Give L command “8086> L ↵” you will see “Down load start!!”
4. Press F3 key and open the Matrix.ABS file.

# 8086 >L

# Down load start !!

# :061000000B155B030FEC145

# :00

# OK Completed !!

# 8086>

1. Run the program By G command
2. 8086>G
3. To terminate program press MON in 8086 kit.

APPENDIX

Exp 1: Flag bits

Flag Register

The 8086 contains a 16-bit flag register that indicates the salient states and status of the processor after execution of an instruction. The flag register is also called Status Register (**SR**) or Program Status Word (**PSW**). Each bit of the flag register is called a **flag** and can take a value of **1** or **0**. There are altogether nine flags in 8086. The bit assignment of the flag register is shown in **Table 0.1**.

**Table 0.1 Flag register in 8086.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | **OF** | **DF** | **IF** | **TF** | **SF** | **ZF** |  | **AF** |  | **PF** |  | **CF** |

**Carry Flag (CF)** - this flag is set to **1** when there is an **unsigned overflow**. For example when you add bytes **255 + 1** (result is not in range 0...255). When there is no overflow this flag is set to **0**.

**Zero Flag (ZF)** - set to **1** when result is **zero**. For none zero result this flag is set to **0**.

**Sign Flag (SF)** - set to **1** when result is **negative**. When result is **positive** it is set to **0**. Actually this flag take the value of the most significant bit.

**Overflow Flag (OF)** - set to **1** when there is a **signed overflow**. For example, when you add bytes **100 + 50** (result is not in range -128...127).

**Parity Flag (PF)** - this flag is set to **1** when there is even number of one bits in the result, and to **0** when there is odd number of one bits. Even if result is a word (16-bit wide) only 8 low bits are analyzed.

**Auxiliary Flag (AF)** - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).

**Interrupt enable Flag (IF)** - when this flag is set to **1,** the processor reacts to interrupts from external devices.

**Direction Flag (DF)** - this flag is used by string instructions to process data chains, when this flag is set to **0** - the processing is done forward, when this flag is set to **1** the processing is done backward.

Exp 5 : make bangladesh flag example

### CODE SEGMENT

### ASSUME CS:CODE

### ORG 1000H

### MOV AL, 10000000B

### OUT 1EH, AL

### START:

### MOV AL, 00011000B

### OUT 1CH, AL

### MOV AL, 10111101B

### OUT 18H, AL

### MOV AL, 11000011B

### OUT 1AH, AL

### CALL DELAY

### MOV AL, 11000011B

### OUT 1CH, AL

### MOV AL, 10000001B

### OUT 18H, AL

### MOV AL, 11111111B

### OUT 1AH, AL

### CALL DELAY

### MOV AL, 00100100B

### OUT 1CH, AL

### MOV AL, 11100111B

### OUT 1AH, AL

### MOV AL, 10011001B

### OUT 18H, AL

### CALL DELAY

### JMP START

### 

### DELAY: MOV CX, 1000;

### L1: NOP

### NOP

### LOOP L1

### RET

### 

### CODE ENDS

### END