

Experiment NO. 7(a)

Aim: Write an assembly language program using 8086-microprocessor to convert Packed BCD number to Unpacked BCD number.

Software Used: emu8086 Simulator

Program:

```
org 100h  
MOV AL,[2000H]  
MOV AH,AL  
AND AL,0FH  
MOV CL,04H  
SHR AH,CL  
MOV [2001H],AX  
Ret
```

Observation:

Input Value: 2000H = 26

Output Value: 2001H = 06

2002H = 02

Output:

The screenshot shows the emu8086 interface with the following details:

- Assembly Editor:** Displays the assembly code:

```
01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04 org 100h
05
06 MOV AL,[2000H]
07 MOV AH,AL
08 AND AL,0FH
09 MOV CL,04H
10 SHR AH,CL
11 MOV [2001H],AX
12
13 ret
14
15
16
17
18
19
```
- Registers View:** Shows register values:

	H	L
AX	02	06
BX	00	00
CX	00	04
DX	00	00
CS	F480	
IP	0154	
SS	0700	
SP	FFFA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	
- Memory Dump View:** Shows memory dump from 09000 to F400:

Address	Value	Content
09000:26	038	8
09001:06	006	6
09002:00	002	2
09003:00	000	NI
09004:00	000	NI
09005:00	000	NI
09006:00	000	NI
09007:00	000	NI
09008:00	000	NI
09009:00	000	NI
0900A:00	000	NI
0900B:00	000	NI
0900C:00	000	NI
0900D:00	000	NI
0900E:00	000	NI
0900F:00	000	NI
- Stack Dump View:** Shows stack dump from F400:

Address	Value	Content
F400:0154	INT	15
F400:0155	02	2
F400:0156	00	0
F400:0157	00	0
F400:0158	00	0
F400:0159	00	0
F400:015A	00	0
F400:015B	00	0
F400:015C	00	0
F400:015D	00	0
F400:015E	00	0
F400:015F	00	0
F400:0160	00	0
F400:0161	00	0
F400:0162	00	0
F400:0163	00	0
F400:0164	00	0
F400:0165	00	0
F400:0166	00	0
F400:0167	00	0
F400:0168	00	0
F400:0169	00	0
F400:016A	00	0
F400:016B	00	0
F400:016C	00	0
F400:016D	00	0
F400:016E	00	0
F400:016F	00	0
F400:0170	00	0
F400:0171	00	0
F400:0172	00	0
F400:0173	00	0
F400:0174	00	0
F400:0175	00	0
F400:0176	00	0
F400:0177	00	0
F400:0178	00	0
F400:0179	00	0
F400:017A	00	0
F400:017B	00	0
F400:017C	00	0
F400:017D	00	0
F400:017E	00	0
F400:017F	00	0
F400:0180	00	0
F400:0181	00	0
F400:0182	00	0
F400:0183	00	0
F400:0184	00	0
F400:0185	00	0
F400:0186	00	0
F400:0187	00	0
F400:0188	00	0
F400:0189	00	0
F400:018A	00	0
F400:018B	00	0
F400:018C	00	0
F400:018D	00	0
F400:018E	00	0
F400:018F	00	0
F400:0190	00	0
F400:0191	00	0
F400:0192	00	0
F400:0193	00	0
F400:0194	00	0
F400:0195	00	0
F400:0196	00	0
F400:0197	00	0
F400:0198	00	0
F400:0199	00	0
F400:019A	00	0
F400:019B	00	0
F400:019C	00	0
F400:019D	00	0
F400:019E	00	0
F400:019F	00	0
F400:01A0	00	0
F400:01A1	00	0
F400:01A2	00	0
F400:01A3	00	0
F400:01A4	00	0
F400:01A5	00	0
F400:01A6	00	0
F400:01A7	00	0
F400:01A8	00	0
F400:01A9	00	0
F400:01AA	00	0
F400:01AB	00	0
F400:01AC	00	0
F400:01AD	00	0
F400:01AE	00	0
F400:01AF	00	0
F400:01B0	00	0
F400:01B1	00	0
F400:01B2	00	0
F400:01B3	00	0
F400:01B4	00	0
F400:01B5	00	0
F400:01B6	00	0
F400:01B7	00	0
F400:01B8	00	0
F400:01B9	00	0
F400:01BA	00	0
F400:01BB	00	0
F400:01BC	00	0
F400:01BD	00	0
F400:01BE	00	0
F400:01BF	00	0
F400:01C0	00	0
F400:01C1	00	0
F400:01C2	00	0
F400:01C3	00	0
F400:01C4	00	0
F400:01C5	00	0
F400:01C6	00	0
F400:01C7	00	0
F400:01C8	00	0
F400:01C9	00	0
F400:01CA	00	0
F400:01CB	00	0
F400:01CC	00	0
F400:01CD	00	0
F400:01CE	00	0
F400:01CF	00	0
F400:01D0	00	0
F400:01D1	00	0
F400:01D2	00	0
F400:01D3	00	0
F400:01D4	00	0
F400:01D5	00	0
F400:01D6	00	0
F400:01D7	00	0
F400:01D8	00	0
F400:01D9	00	0
F400:01DA	00	0
F400:01DB	00	0
F400:01DC	00	0
F400:01DD	00	0
F400:01DE	00	0
F400:01DF	00	0
F400:01E0	00	0
F400:01E1	00	0
F400:01E2	00	0
F400:01E3	00	0
F400:01E4	00	0
F400:01E5	00	0
F400:01E6	00	0
F400:01E7	00	0
F400:01E8	00	0
F400:01E9	00	0
F400:01EA	00	0
F400:01EB	00	0
F400:01EC	00	0
F400:01ED	00	0
F400:01EE	00	0
F400:01EF	00	0
F400:01F0	00	0
F400:01F1	00	0
F400:01F2	00	0
F400:01F3	00	0
F400:01F4	00	0
F400:01F5	00	0
F400:01F6	00	0
F400:01F7	00	0
F400:01F8	00	0
F400:01F9	00	0
F400:01FA	00	0
F400:01FB	00	0
F400:01FC	00	0
F400:01FD	00	0
F400:01FE	00	0
F400:01FF	00	0
- Stack Dump View:** Shows stack dump from F400:

Address	Value	Content
F400:0154	INT	15
F400:0155	02	2
F400:0156	00	0
F400:0157	00	0
F400:0158	00	0
F400:0159	00	0
F400:015A	00	0
F400:015B	00	0
F400:015C	00	0
F400:015D	00	0
F400:015E	00	0
F400:015F	00	0
F400:0160	00	0
F400:0161	00	0
F400:0162	00	0
F400:0163	00	0
F400:0164	00	0
F400:0165	00	0
F400:0166	00	0
F400:0167	00	0
F400:0168	00	0
F400:0169	00	0
F400:016A	00	0
F400:016B	00	0
F400:016C	00	0
F400:016D	00	0
F400:016E	00	0
F400:016F	00	0
F400:0170	00	0
F400:0171	00	0
F400:0172	00	0
F400:0173	00	0
F400:0174	00	0
F400:0175	00	0
F400:0176	00	0
F400:0177	00	0
F400:0178	00	0
F400:0179	00	0
F400:017A	00	0
F400:017B	00	0
F400:017C	00	0
F400:017D	00	0
F400:017E	00	0
F400:017F	00	0
F400:0180	00	0
F400:0181	00	0
F400:0182	00	0
F400:0183	00	0
F400:0184	00	0
F400:0185	00	0
F400:0186	00	0
F400:0187	00	0
F400:0188	00	0
F400:0189	00	0
F400:018A	00	0
F400:018B	00	0
F400:018C	00	0
F400:018D	00	0
F400:018E	00	0
F400:018F	00	0
F400:0190	00	0
F400:0191	00	0
F400:0192	00	0
F400:0193	00	0
F400:0194	00	0
F400:0195	00	0
F400:0196	00	0
F400:0197	00	0
F400:0198	00	0
F400:0199	00	0
F400:019A	00	0
F400:019B	00	0
F400:019C	00	0
F400:019D	00	0
F400:019E	00	0
F400:019F	00	0
F400:01A0	00	0
F400:01A1	00	0
F400:01A2	00	0
F400:01A3	00	0
F400:01A4	00	0
F400:01A5	00	0
F400:01A6	00	0
F400:01A7	00	0
F400:01A8	00	0
F400:01A9	00	0
F400:01AA	00	0
F400:01AB	00	0
F400:01AC	00	0
F400:01AD	00	0
F400:01AE	00	0
F400:01AF	00	0
F400:01B0	00	0
F400:01B1	00	0
F400:01B2	00	0
F400:01B3	00	0
F400:01B4	00	0
F400:01B5	00	0
F400:01B6	00	0
F400:01B7	00	0
F400:01B8	00	0
F400:01B9	00	0
F400:01BA	00	0
F400:01BB	00	0
F400:01BC	00	0
F400:01BD	00	0
F400:01BE	00	0
F400:01BF	00	0
F400:01C0	00	0
F400:01C1	00	0
F400:01C2	00	0
F400:01C3	00	0
F400:01C4	00	0
F400:01C5	00	0
F400:01C6	00	0
F400:01C7	00	0
F400:01C8	00	0
F400:01C9	00	0
F400:01CA	00	0
F400:01CB	00	0
F400:01CC	00	0
F400:01CD	00	0
F400:01CE	00	0
F400:01CF	00	0
F400:01D0	00	0
F400:01D1	00	0
F400:01D2	00	0
F400:01D3	00	0
F400:01D4	00	0
F400:01D5	00	0
F400:01D6	00	0
F400:01D7	00	0
F400:01D8	00	0
F400:01D9	00	0
F400:01DA	00	0
F400:01DB	00	0
F400:01DC	00	0
F400:01DD	00	0
F400:01DE	00	0
F400:01DF	00	0
F400:01E0	00	0
F400:01E1	00	0
F400:01E2	00	0
F400:01E3	00	0
F400:01E4	00	0
F400:01E5	00	0
F400:01E6	00	0
F400:01E7	00	0
F400:01E8	00	0
F400:01E9	00	0
F400:01EA	00	0
F400:01EB	00	0
F400:01EC	00	0
F400:01ED	00	0
F400:01EE	00	0
F400:01EF	00	0
F400:01F0	00	0
F400:01F1	00	0
F400:01F2	00	0
F400:01F3	00	0
F400:		

Experiment NO. 7(b)

Aim: Write an assembly language program using 8086-microprocessor to convert Packed BCD number to ASCII code.

Software Used: emu8086 Simulator

Program:

```
org 100h  
MOV AL,[2000H]  
MOV AH,AL  
AND AL,0FH  
MOV CL,04H  
SHR AH,CL  
OR AX, 3030H  
MOV [2001H],AX  
Ret
```

Observation:

Input Value: 2000H = 26

Output Value: 2001H = 36

2002H = 32

Output:

The screenshot shows the emu8086 software interface. The main window displays assembly code for an 8086 microprocessor. The code initializes AL to 2000H, sets AH to AL, performs an AND operation with 0FH, moves CL to 04H, shifts AH right by CL, ors AX with 3030H, moves the result to 2001H, and finally retires. A second window titled "original source code" shows the same assembly code. The registers window shows initial values: AX=3236, BX=0000, CX=0004, DX=0000, CS=F400, IP=0154, SS=0700, SP=FFFA, BP=0000, SI=0000, DI=0000, DS=0700, and ES=0700. The memory dump window shows the memory starting at address 09000, where the value 26038 is present.

```
01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04 org 100h
05
06 MOV AL,[2000H]
07 MOV AH,AL
08 AND AL,0FH
09 MOV CL,04H
10 SHR AH,CL
11 OR AX,3030H
12 MOV [2001H],AX
13
14 ret
15
16
17
18
19
20
```

Registers:

	H	L
AX	32	36
BX	00	00
CX	00	04
DX	00	00
CS	F400	
IP	0154	
SS	0700	
SP	FFFA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

Memory Dump:

	0700:2000	F400:0154
09000:	26 038 & -	BIOS DI
09001:	36 054 6	INT 020h
09002:	32 050 2	IRET
09003:	00 000 NI	ADD [BX + SI], A
09004:	00 000 NI	ADD [BX + SI], A
09005:	00 000 NI	ADD [BX + SI], A
09006:	00 000 NI	ADD [BX + SI], A
09007:	00 000 NI	ADD [BX + SI], A
09008:	00 000 NI	ADD BH, BH
09009:	00 000 NI	DEC BP
0900A:	00 000 NI	SBB CL, BH
0900B:	00 000 NI	ADD [BX + SI], A
0900C:	00 000 NI	ADD [BX + SI], A
0900D:	00 000 NI	ADD [BX + SI], A
0900E:	00 000 NI	ADD [BX + SI], A
0900F:	00 000 NI	...

Result: Thus, we have converted Packed BCD number to ASCII code using 8086-microprocessor.

Experiment NO. 8

Aim: Write an Assembly Language Program to interface LCD with 8051 and display Subject Name and Class of Student on it.

Software Used: Keil IDE uvision4

Program:

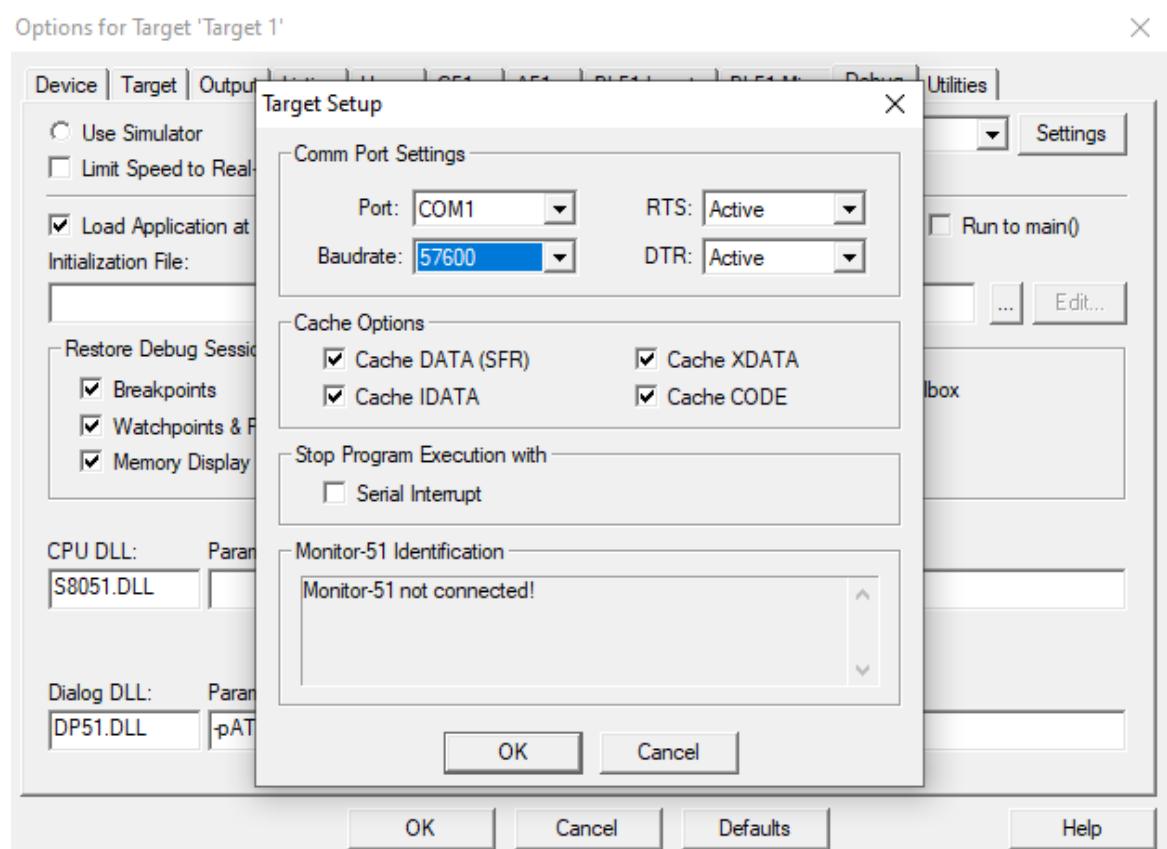
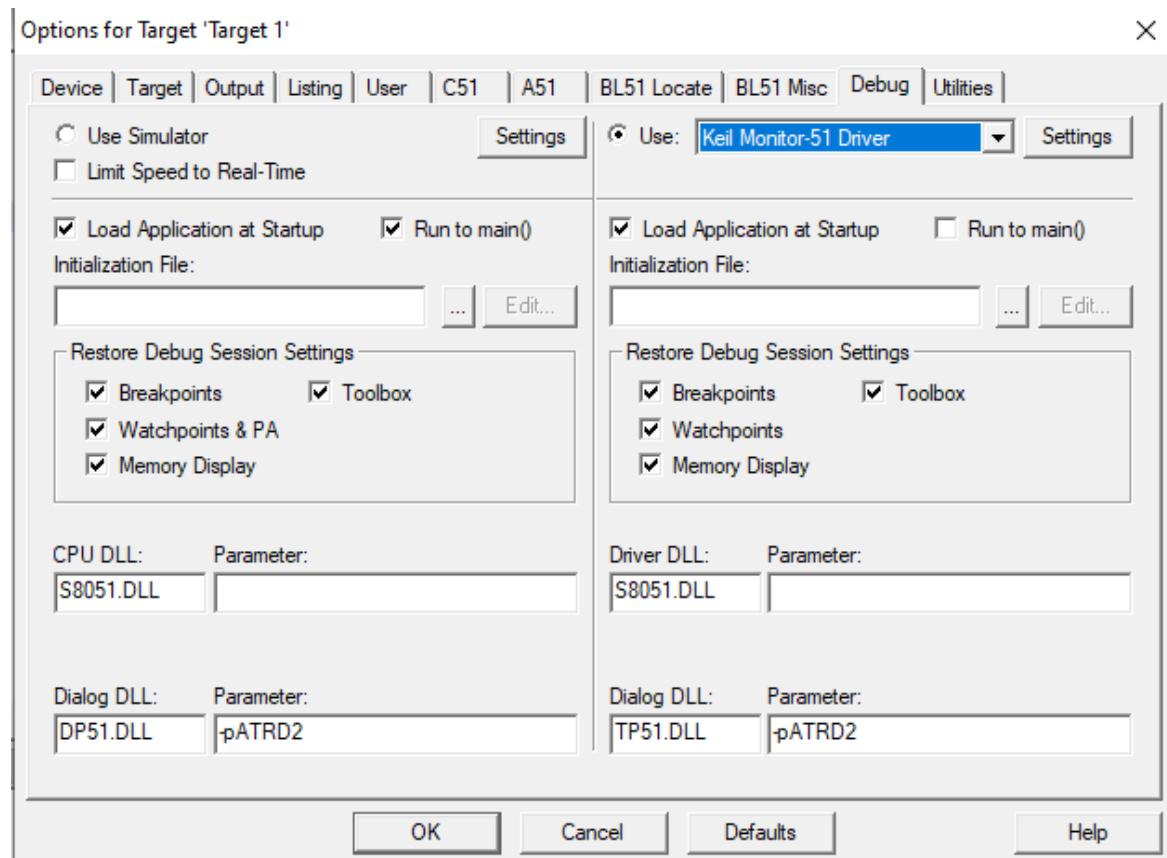
```
PORT_A EQU 0E000H
PORT_B EQU 0E001H
PORT_C EQU 0E002H
CWR EQU 0E003H
ORG 00H
SJMP 30H
ORG 30H
W2:ACALL INIT_LCD
ACALL CLR_LCD
START:MOV DPTR,#MSG
LCALL STRING
MOV A,#0C0H
CALL CMD
MOV DPTR,#MSG1
LCALL STRING
SJMP $
INIT_LCD:MOV DPTR,#CWR
MOV A,#80H
MOVX @DPTR,A
MOV R1,#1FH
ACALL DELAY
MOV R3,#03H
INIT:MOV A,#38H
```

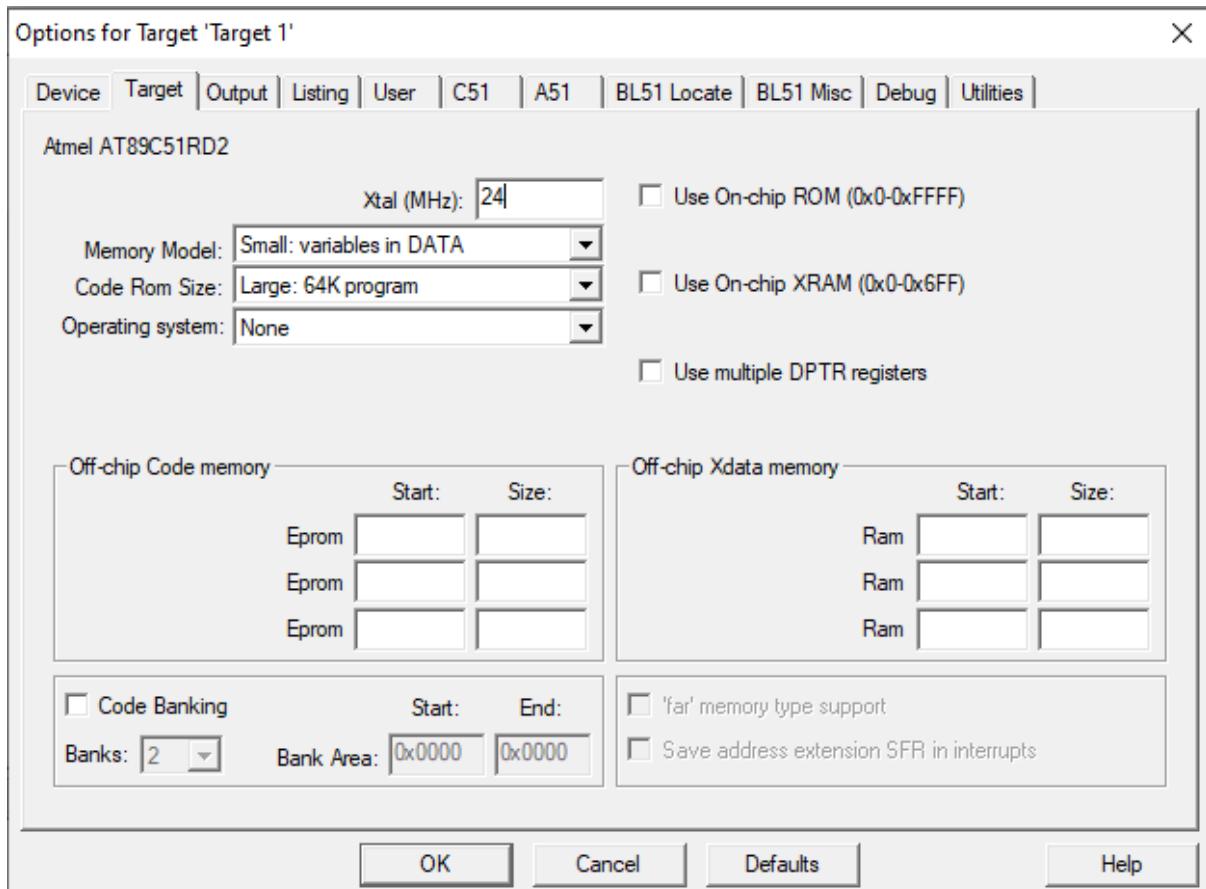
LCALL CMD
MOV R1,#3FH
LCALL DELAY
DJNZ R3,INIT
MOV A,#38H
LCALL CMD
MOV A,#0CH
LCALL CMD
RET
CMD:MOV DPTR,#PORT_A
MOVX @DPTR,A
MOV A,#0FBH
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0F8H
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0FCH
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0F8H
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV R1,#10H
LCALL DELAY
RET
DWRITE:PUSH DPL

PUSH DPH
MOV DPTR,#PORT_A
MOVX @DPTR,A
MOV A,#0FAH
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0F9H
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0FDH
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV A,#0F9H
MOV DPTR,#PORT_B
MOVX @DPTR,A
MOV R1,#10H
LCALL DELAY
POP DPH
POP DPL
RET
CLR_LCD:MOV A,#01H
CALL CMD
MOV A,#80H
CALL CMD
RET
DELAY:MOV R2,#02H
DLY:DJNZ R2,DLY

DJNZ R1,DLY
RET
STRING: CLR A
MOVC A,@A+DPTR
CJNE A,#00,CONTINUE
RET
CONTINUE:ACALL DWRITE
INC DPTR
JMP STRING
ORG 100H
MSG:DB'ECE',00H
MSG1:DB'MPMC',00H
END

Output:





Result: Thus, Subject Name and Class is displayed on LCD using 8051.

Experiment NO. 9

Aim: Write an Assembly Language Program to generate a square wave of 10 KHz using Timer 1 in mode 1 using 8051.

Software Used: Keil IDE uvision4

Program:

ORG 00H

MOV TMOD,#10H

MOV TH1,#0D2H

RPT:SETB TR1

WAIT:JNB TF1,WAIT

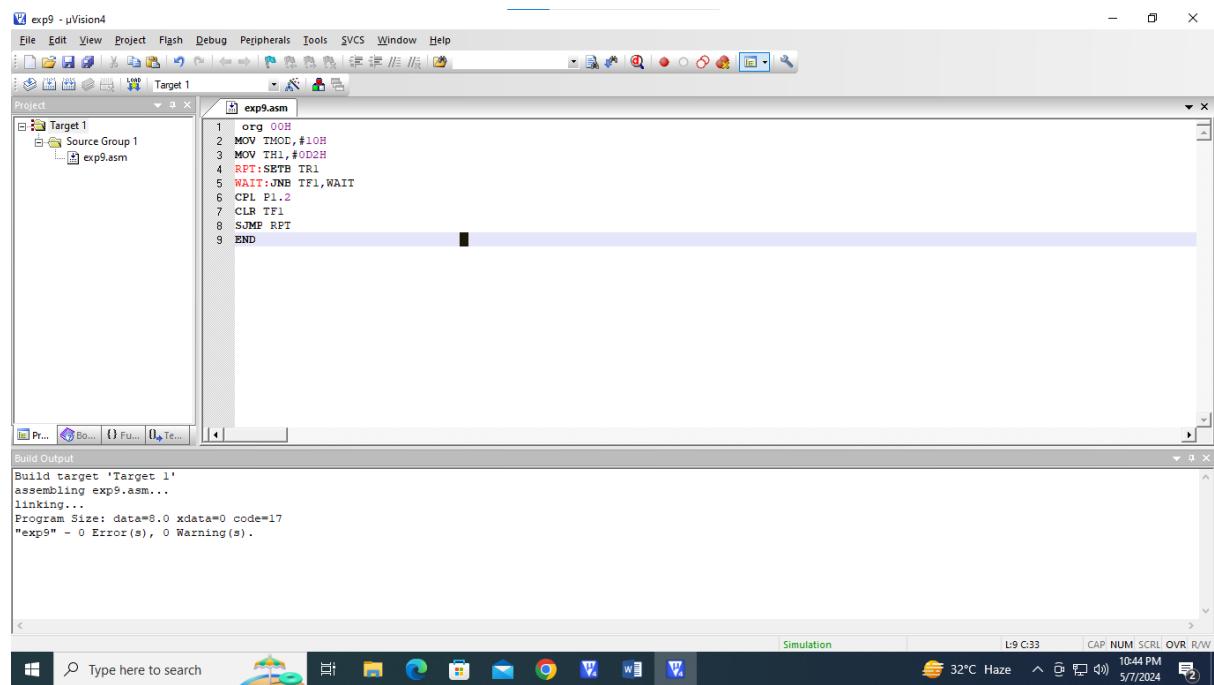
CPL P1.2

CLR TF1

SJMP RPT

END

Output:

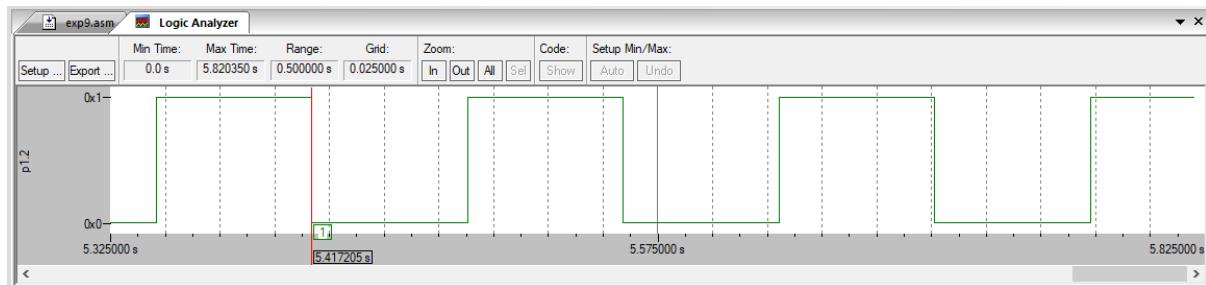
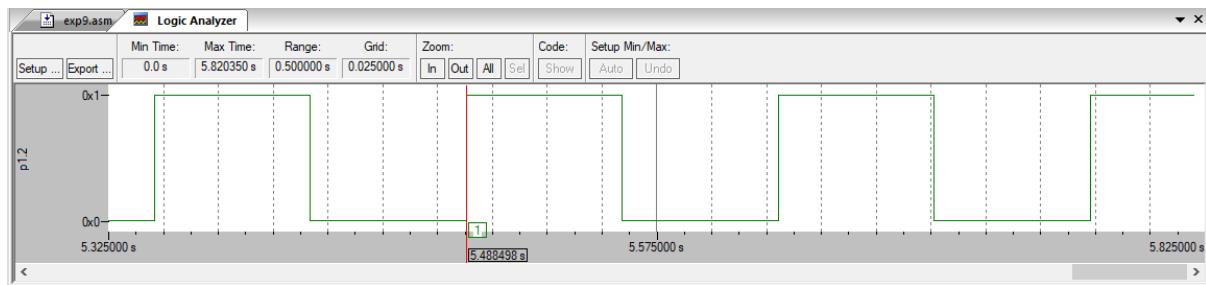


The screenshot shows the Keil IDE interface with the project 'exp9' open. The assembly code for the program is displayed in the main editor window. The code initializes the timer mode, sets up Timer 1, and creates a loop that toggles port P1.2 while monitoring the TF1 flag. The build output window at the bottom shows a successful build with no errors or warnings.

```
1 org 00H
2 MOV TMOD,#10H
3 MOV TH1,#0D2H
4 RPT:SETB TR1
5 WAIT:JNB TF1,WAIT
6 CPL P1.2
7 CLR TF1
8 SJMP RPT
9 END
```

Build Output

```
Build target 'Target 1'
assembling exp9.asm...
linking...
Program Size: data=8.0 xdata=0 code=17
"exp9" - 0 Error(s), 0 Warning(s).
```



Result: Thus, a square waveform is generated using Timer 1 of 8051.

Experiment NO. 10

Aim: Design a Minor project using 8051 Microcontroller (Diwali lights with 8 different combinations).

Software Used: Keil IDE uvision4

Program:

ORG 0000H

MAIN: MOV P0, #0F0H

ACALL DELAY

MOV P0, #0FH

ACALL DELAY

MOV P0, #AAH

ACALL DELAY

MOV P0, #55H

ACALL DELAY

ACALL DELAY

MOV P0, #00H

ACALL DELAY

MOV P0, #0FFH

ACALL DELAY

ACALL ROT_RGT

ACALL DELAY

ACALL ROT_LFT

ACALL DELAY

LJMP MAIN

ROT_RGT: MOV A, #00H

MOV P0, A

SETB C

MOV R3,#08

RR1: RRC A MOV P0,A

ACALL DELAY

DJNZ R3, RR1

RET

ROT_LFT: MOV A,#00H

MOV P0, A

SETB C

MOV R3, #08H

RL1: RLC A

MOV P0, A

ACALL DELAY

DJNZ R3, RL1

RET

DELAY: MOV R2, #150H

L3: MOV R1, #255

L2: MOV R0, #255

L1: DJNZ R0, L1

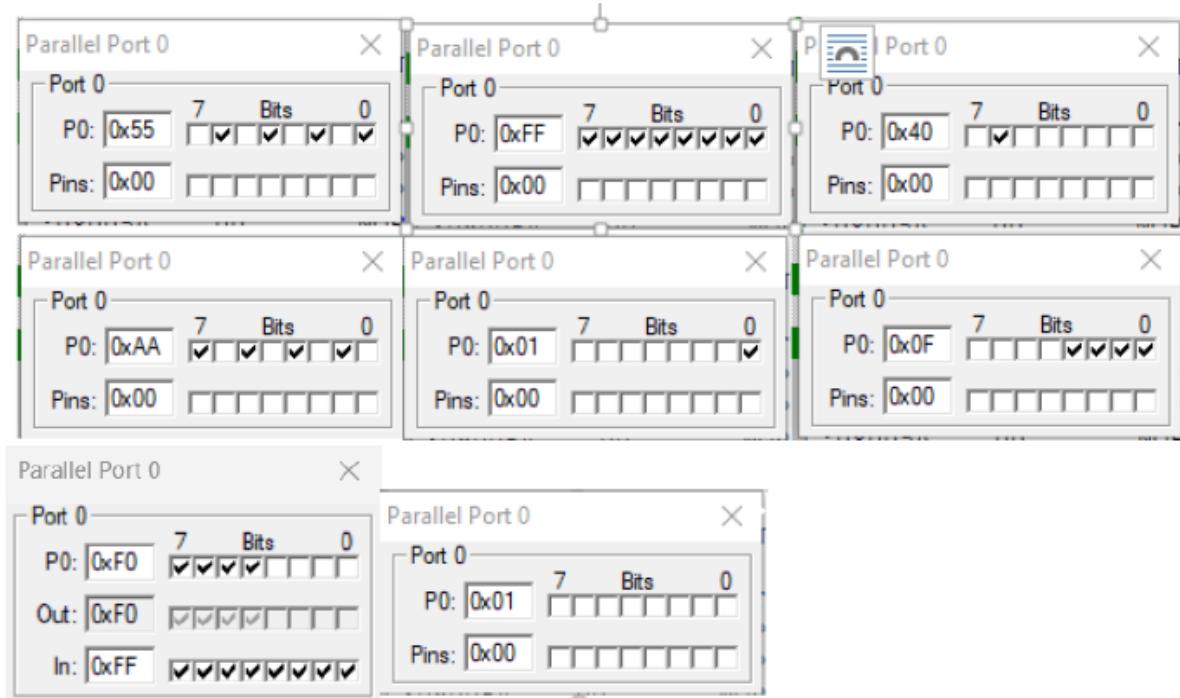
DJNZ R1,L2

DJNZ R2, L3

RET

END

Output:



Result: Thus, Diwali lights with 8 different combinations have been shown on port 0.

Experiment NO. 11

Aim: Write a program in Assembly Language to find a number from an array using string instruction.

Software Used: emu8086 Simulator

Program:

```
ORG 100H
```

```
MOV DI,1005H
```

```
MOV AL,03
```

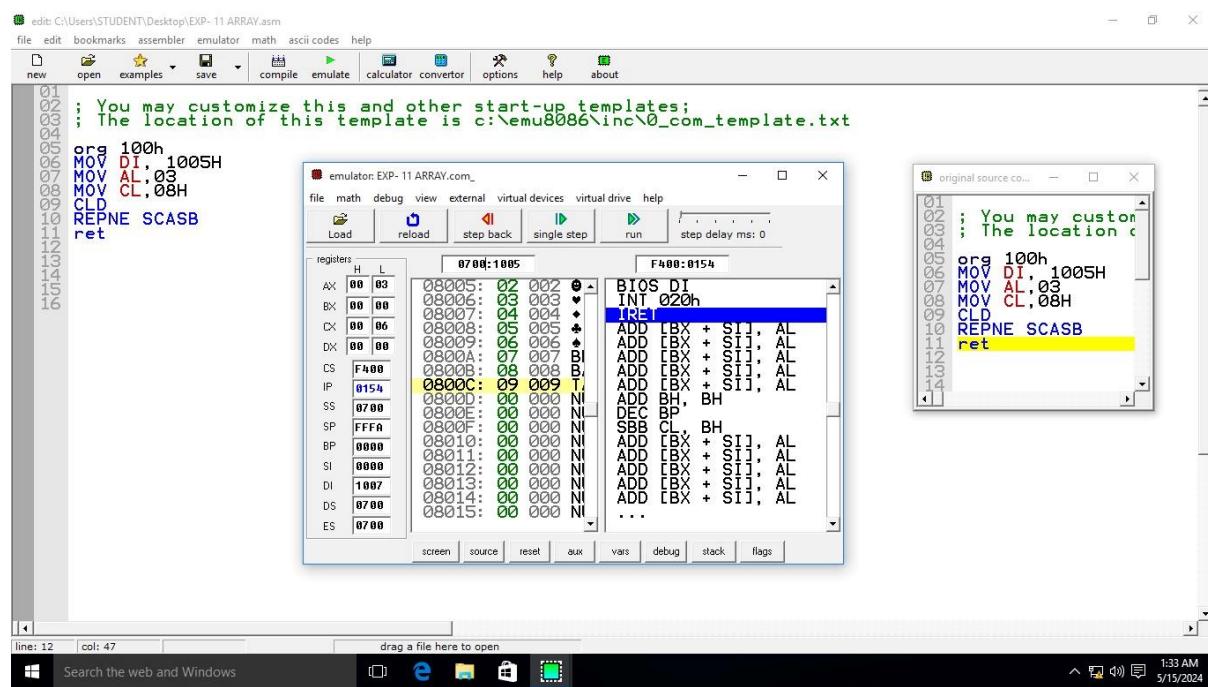
```
MOV CL,08H
```

```
CLD
```

```
REPNE SCASB
```

```
RET
```

Output:



Result: Thus, the number is found in the array.

Experiment NO. 12

Aim: Design a minor project using 8086 microprocessors (Stepper Motor Execution)

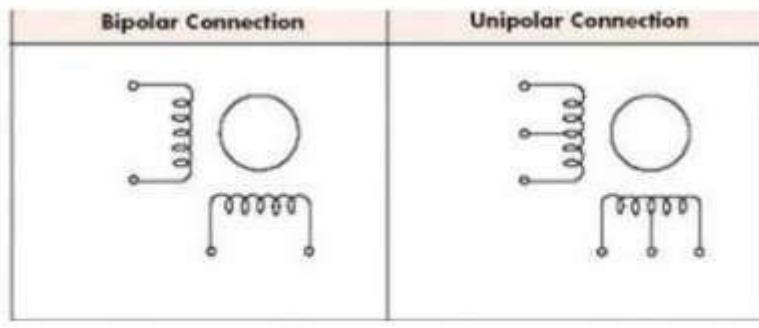
Software Used: emu8086 Simulator

Theory:

Stepper motor is interfaced to 8086 with the help of 8255 IC (Programmable Peripheral Interface). 8255 has two modes of operation BSR (Bit Set Reset) mode & IO (Input Output) mode. The mode is determined by the Control Word (CW) register; (here the CW address is 26). A stepper motor is an electric motor that rotates in discrete step increments. The movement of each step is precise and repeatable; therefore, the motor's position can be controlled precisely without any feedback mechanism, as long as the motor is carefully sized to the application. This type of control eliminates the need for expensive sensing and feedback devices such as optical encoders. The position is known simply by keeping track of the input step pulses. It is one of the most versatile forms of positioning systems. They are typically digitally controlled as part of an open loop system, and are simpler and more rugged than closed loop servo systems. Industrial applications include high speed pick and place equipment and multi-axis CNC machines, often directly driving lead screws or ball screws. In the field of optics, they are frequently used in precision positioning equipment such as linear actuators, linear stages, rotation stages, goniometers, and mirror mounts. Other uses are in packaging machinery, and positioning of valve pilot stages for fluid control systems. Commercially, stepper motors are used in floppy disk drives, flatbed scanners, computer printers, plotters, slot machines, image scanners, compact disc drives and many more devices.

Coil Windings and Stepping Mechanism:

There are two common winding arrangements for the electromagnetic coils: bipolar and unipolar (Fig 4). The described stepping sequence utilizes the bipolar winding. Each phase consists of a single winding. By reversing the current in the windings, electromagnetic polarity is reversed. A unipolar stepper motor has one winding with centre tap per phase. Each section of windings is switched on for each direction of magnetic field. Since in this arrangement a magnetic pole can be reversed without switching the direction of current, the commutation circuit can be made very simple for each winding.



Two common winding arrangements

Program:

```

ORG 100H
#START=STEPPER_MOTOR.EXE#
JMP START
DATCW DB 0000_0011B
DB 0000_0110B
DB 0000_1100B
DB 0000_1001B
START: MOV BX,OFFSET DATCW
MOV SI,0
NEXT_STEP:
WAIT: IN AL,07h
TEST AL,10000000B
JZ WAIT
MOV AL,[BX][SI]
OUT 7, AL
INC SI
CMP SI,4
JC NEXT_STEP
MOV SI,0

```

JMP NEXT_STEP

RET

Output:

The screenshot shows a Windows desktop environment with several windows open. The main window is a debugger titled "emulator: EXP-8 STEPPER MOTOR.com" showing assembly code. The code initializes a variable START to 100H, sets up a loop starting at address 001B, and performs a series of operations including moving BX to offset 0, reading AL via IN, testing AL against 10000000B via TEST, jumping if zero to WAIT, and incrementing SI. It then moves AL to BX[SI], increments SI, compares it to 4, and loops back if less than or equal to 4. Finally, it jumps to the next step and returns. A second window titled "stepper-m..." shows a circular stepper motor diagram with pins numbered 1 through 8, and a status indicator showing "ready". The taskbar at the bottom includes the Start button, a search bar, and icons for File Explorer, Edge browser, Task View, and File Explorer again. The date and time shown are 5/15/2024 at 1:36 AM.

```
01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04
05 org 100h
06
07 #START=STEPPER_MOTOR
08 JMP START
09 DATCH DB 0000_0011B
10 DB 0000_0110B
11 DB 0000_1100B
12 DB 0000_1001B
13 START: MOV BX, OFFSET
14 MOV SI, 0
15 NEXT_STEP:
16 WAIT: IN AL, 07H
17 TEST AL, 10000000B
18 JZ WAIT
19 MOV AL, [BX + SI]
20 OUT 7, AL
21 INC SI
22 CMP SI, 4
23 JC NEXT_STEP
24 MOV SI, 0
25 JMP NEXT_STEP
26 ret
27
28
29
30
31
```

Registers window:

	H	L
AX	00 03	
BX	01 02	
CX	00 22	
DX	00 00	
CS	0700	
IP	010E	
SS	0700	
SP	FFFE	
BP	0000	
SI	0001	
DI	0000	
DS	0700	
ES	0700	

Memory dump window:

	0700:010E	0700:010E
0710C:	E4 228	Σ
0710D:	07 007	BI
0710E:	A8 168	z
0710F:	80 128	c
07110:	74 116	t
07111:	FA 250	.
07112:	8A 138	è
07113:	00 000	NI
07114:	E6 230	
07115:	07 007	BI
07116:	46 070	F
07117:	83 131	â
07118:	FE 254	•
07119:	04 004	♦
0711A:	72 114	r
0711B:	F0 240	
0711C:	BE 190	
	...	

Stepper motor window:

Result: Thus, the Stepper Motor runs clockwise in direction.