Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

10211CS305 – MICROPROCESSOR AND MICROCONTROLLER LABORATORY

Program Core

2023-2024 Winter Semester

Lab Manual

x
Associate Professor
Department of Computer Science and Engineering
School of Computing

COURSE CODE	COURSE TITLE	L	T	P	C
10211CS305	**MICROPROCESSOR AND MICROCONTROLLER LABORATORY**	**0**	**0**	**2**	**1**

I. Course Description (Max 200 words)

This laboratory course motivates the students to understand and develop assembly language programming skills by learning simple arithmetic, conditional programs. It teaches the students to realize the difference between microprocessor and microcontroller. Students are also exposed to interfacing techniques which has been the pathway for developing real time applications.

II. Course Prerequisite(s)

10211CS201 – Digital Electronics

III. Course Objectives

Learners are exposed to

- Understand 8086 Microprocessor architecture, instruction set.
- Develop assembly language programs using 8086 microprocessor.
- Implement processor interfacing with I/O devices.
- Develop applications using ARM processor.

IV. Course Outcomes

Upon the successful completion of the course, students will be able to:

Knowledge Level (Based on revised Bloom's Taxonomy)

K1-Remember; K2-Understand; K3-Apply; K4-Analyze; K5-Evaluate; K6-Create

CO No's	Course Outcomes	K - Level
CO1	Develop programming skills using 8086 assembly language instructions.	K3
CO2	Identify the list of ARM processor related instructions and develop simple programs using it.	K3
CO3	Establish ARM processor interfacing with keyboard and display unit using embedded C programs.	K3
CO4	Realize UART data transmission between two ARM processor boards.	K3
CO5	Measure the temperature of an object using an ARM processor.	K3

V. Mapping of Course Outcomes of this course with Program Outcomes (PO) and Program Specific Outcomes (PSO) (Mention the PO and PSO mapped to the course with levels: 1-Low, 2-Medium, 3-High) POs and PSOs are listed in VIII

Cos	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO2	PSO 3
CO1	3	3												2	

CO 2	3	3			3									3	
CO 3	3	3	3		3									3	
CO 4	3	3	3		3									3	
CO 5	3	3	3		3									3	3

VI. Course Content

Part 1

TASK 1: 16bit arithmetic and logical operations using 8086 instruction set.
Construct assembly language programs for addition, subtraction, multiplication and division.
Tools : 8086 Microprocessor kit development board

TASK 2: String operations using 8086 instruction set
Develop programs for string concatenation and conversion of uppercase letters into lowercase letters and vice versa.
Tools : 8086 Microprocessor kit development board

TASK 3: Smallest and largest number using 8086 instruction set
Realize a program that is able to identify the smallest and largest number from the given set of data.
Tools : 8086 Microprocessor kit development board

TASK 4: Ascending and descending order using 8086 instruction set
Write an assembly language program to arrange the given number from smallest to largest and vice versa.
Tools : 8086 Microprocessor kit development board

TASK 5: Data transfer using ARM processor
Employ an assembly language program which is able to transfer data from ARM processor register to memory or memory to ARM processor.
Tools : Keil software, ARM Processor kit development board

TASK 6: Count of negative numbers in an array using ARM processor instructions
Formulate a logic to identify the list of negative numbers from the given numbers and develop the program using the ARM processor instruction set.
Tools : Keil software, ARM Processor kit development board

TASK 7: Factorial of positive integer N using ARM processor instructions
Calculate the factorial of a given number using ARM processor instructions.
Tools : Keil software, ARM Processor kit development board

TASK 8: GCD algorithm using ARM processor instructions
Identify the greatest common divisor of given numbers by executing a program developed using ARM processor instructions.
Tools : Keil software, ARM Processor kit development board

TASK 9: Seven segment LED display
Control the display of an LED through an ARM processor.
Tools : Keil software, ARM Processor kit development board

TASK 10: Keyboard and display interfacing using ARM processor kit development board
Realize the display of the information typed using a keyboard and processed by an ARM processor.
Tools : Keil software, ARM Processor kit development board

TASK 11: UART device driver for data transmission
Experiment the process of data transfer that can be established between two ARM processor boards.
Tools : Keil software, ARM Processor kit development board

TASK 12: Temperature measurement using ARM processor kit development board
Construct a data acquisition mechanism of any physical parameter like temperature using an ADC which is in-built in an ARM processor.
Tools : Keil software, ARM Processor kit development board

Part-2

Use Cases:

Use Case 1 : Develop an application which counts the number of customers entering into a supermarket for purchasing using 8086 microprocessor. IR sensor shall be used to identify the customer entering into the supermarket. The 8086 Processor recognizes the output of IR sensor as a digital input and a suitable ALP is able to initiate the count as and when it receives a logic '1'.

Use Case 2 : Implement a scheme to measure the distance between two cars that helps to avoid collision due to over speed. GP2D120 sensor can be used to measure the distance between two objects. Output of the sensor is an analog signal. A/D converter in an ARM CORTEX processor recognizes this output as an input to it. Suitable Keil 'C, coding implemented in the ARM processor converts the analog signal into digital and displays the result through LCD display.

Use Case 3 : Interpret the given digital signal, develop a Keil 'C' program to implement in an ARM CORTEX processor for identifing the frequency of the signal and display its value in an LCD.

Use Case 4 : Design a Keil 'C' program to generate a PWM signal based on the input value given by the user. The program must be implemented in an ARM CORTEX processor and the output shall be displayed by changing the ON / OFF condition of an LED. Timer in the ARM processor can change the width of the pulse depending the input applied to it. Frequency of the timer is the frequency of the ARM processor.

Total: 30 Hours

VII. Required Textbook(s):

1. Doughlas V. Hall, "*Microprocessors and Interfacing*", TMH, Revised second edition, 2005. (Unit 1-2)
2. Muhammad Tahir, Kashif Javed, "*ARM Microprocessor Systems: Cortex-M Architecture, Programming, and Interfacing*", CRC Press, Taylor & Francis group, 2017. (Unit 3-5)

Suggested Reference Book(s):

1. Andrew N. Sloss, "*ARM system developer's guide – Designing and optimizing system software*", ELSEVIER, 2004.
2. Charles M. Gilmore, "*Microprocessors: Principles and applications*", TMH, Second edition, 1995.

3. Jonathan W. Valvano, "*Embedded microcomputer systems: Real time interfacing*", CENGAGE Learning, Third edition, 2012.
4. Steve Furber, "*ARM system-on-chip architecture*", Pearson education, Second edition, 2015.

Suggested Web Resource(s):

1. "Microprocessor interfacing", Accessed on Sep 20, 2022 [Online], Available: https://onlinecourses.nptel.ac.in/noc20_ee11/preview.
2. "Microprocessors and Digital systems", Accessed on Sep 21, 2022 [Online], Available: https://archive.org/details/microprocessorsd0000hall/page/n3/mode/2up.
3. "ARM processor architecture", Accessed on Sep 21, 2022 [Online], Available: https://www.cs.ccu.edu.tw/~pahsiung/courses/ese/notes/ESD_03_ARM_Architecture.pdf.

SCHOOL OF COMPUTING
DEPARTMENT OF CSE
VTU R21 B. TECH – CSE
WINTER 2023-24

10211CS305 & MICROPROCESSOR AND MICROCONTROLLER Lab Program Core

Rubrics

Continuous internal assessment (15)

Performance in conducting experiment (5)	Result and analysis (3)	Viva Voce (3)	Record (4)

Model practical examination (25)

Performance in conducting experiment (15)	Result and analysis (5)	Viva Voce (5)

Semester end examination (60)

Performance in conducting experiment (30)	Result and analysis (15)	Viva Voce (10)	Record (5)

LIST OF EXPERIMENTS

S. No	Name of the Experiment
1.	16bit arithmetic and logical operations using 8086 instruction set.
2.	String operations using 8086 instruction set
3.	Smallest and largest number using 8086 instruction set
4.	Ascending and descending order using 8086 instruction set
5.	Data transfer using ARM processor
6.	Count of negative numbers in an array using ARM processor instructions
7.	Factorial of positive integer N using ARM processor instructions
8.	GCD algorithm using ARM processor instructions
9.	Seven segment LED display
10.	Keyboard and display interfacing using ARM processor kit development board
11.	UART device driver for data transmission
12.	Temperature measurement using ARM processor kit development board
13.	Use Case 1,2
14.	Use Case 3,4

TASK 1: 16bit arithmetic and logical operations using 8086 instruction set

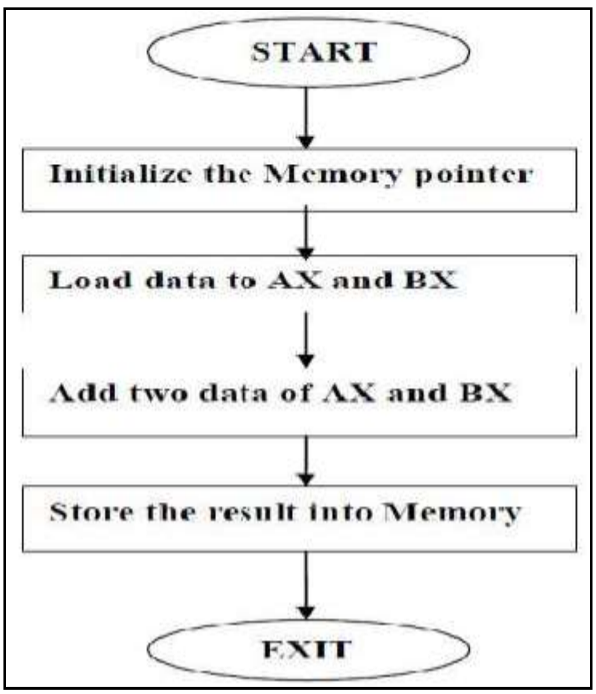
Aim

To perform addition, subtraction, multiplication, division and BCD – Hexadecimal code conversion using 8086 Microprocessor development kit.

Addition Algorithm

- Initialize SI register to get data from a memory location (1200).
- Load the data in the registers AX, BX from memory.
- Add the data in the above two registers.
- Initialize DI register for memory location (1300).
- Move the result of addition from AX register to the specified memory location (1200).

Flow chart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 12	MOV SI, **1200H**	Move 1200 into SI pointer
1103	AD	LODSW	Load the first data into AX
1104	89 C3	MOV BX, AX	Move AX value into BX
1106	AD	LODSW	Load the second data into AX
1107	01 C3	ADD BX, AX	Add BX and AX registers
1109	BF 00 13	MOV DI, **1300H**	Load 1300 address location into DI
110C	89 1D	MOV [DI], BX	Store BX value into memory
110E	F4	HLT	Stop the program

Result

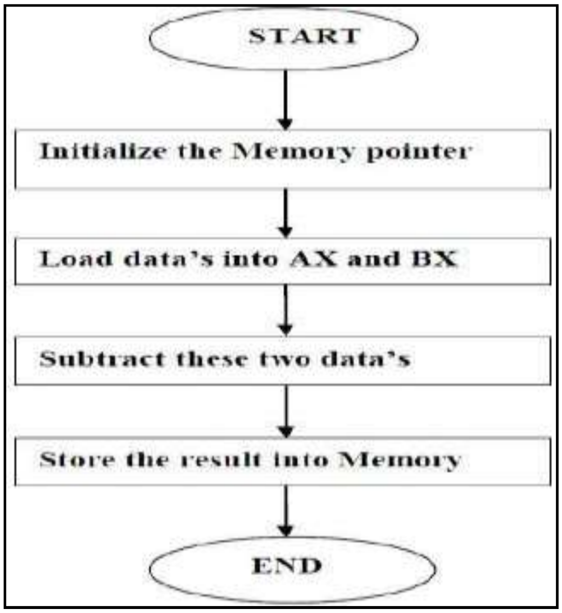
Memory location	Input data	Output data (result)
1200	13	---------
1201	12	---------
1202	00	---------
1203	34	---------
1213 + 3400 = 4613		
1300	---------	**13**
1301	---------	**46**

Subtraction

Algorithm

- Initialize SI register to get data from a memory location (1200).
- Load the data in the registers AX, BX from memory.
- Subtract (BX = BX – AX) the data in the above two registers.
- Initialize DI register for memory location (1300).
- Move the result of addition from AX register to the specified memory location (1200).

Flow chart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 12	MOV SI,**1200H**	Load 1200 into SI
1103	AD	LODSW	Load the first data
1104	89 C3	MOV BX, AX	Move AX value into BX
1106	AD	LODSW	Load the second data
1107	29 C3	SUB BX, AX	subtract **AX** from BX
1109	BF 00 13	MOV DI, **1300H**	Load 1300 address into DI
110C	89 1D	MOV [DI],BX	Load BX value into DI
110E	F4	HLT	Stop the program

Result

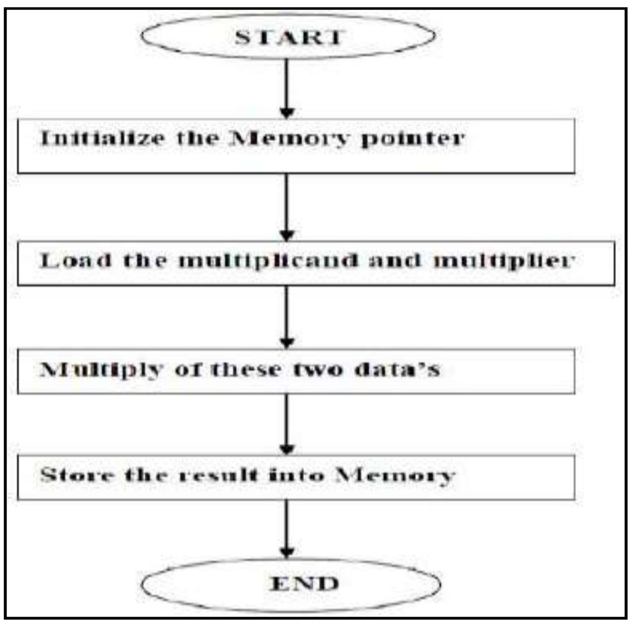
Memory location	**Input data**	**Output data (result)**
1200	34	---------
1201	12	---------
1202	22	---------
1203	11	---------
1234 – 1122 = 0111		
1300	---------	**11**
1301	---------	**01**

Multiplication

Algorithm

- Initialize DX register to zero.
- Load the multiplicand in AX register.
- Load the multiplier in BX register.
- Multiply (AX = AX * BX) the above two data. Carry data will be saved in the DX register.
- Initialize the DI register (1300) to store data in a memory location.
- Transfer the result from AX, DX registers to the above memory locations.

Flow chart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BA 00 00	MOV DX, 0000	Clear DX registers
1103	B8 06 00	MOV AX, **0404H**	Load the multiplicand in AX
1106	B9 02 00	MOV CX, **0202H**	Load the multiplier value in BX
1109	F7 F1	MUL CX	Multiply the two data's
110B	BF 00 13	MOV DI, **1300H**	Load 1300 address into DI
110E	88 05	MOV [DI], AL	Load AL value into DI
1110	47	INC DI	Increment DI
1111	88 25	MOV [DI], AH	Load AH value into DI
1113	47	INC DI	Increment DI
1114	89 15	MOV [DI], DX	Load DX value into DI
1116	F4	HLT	End

Result

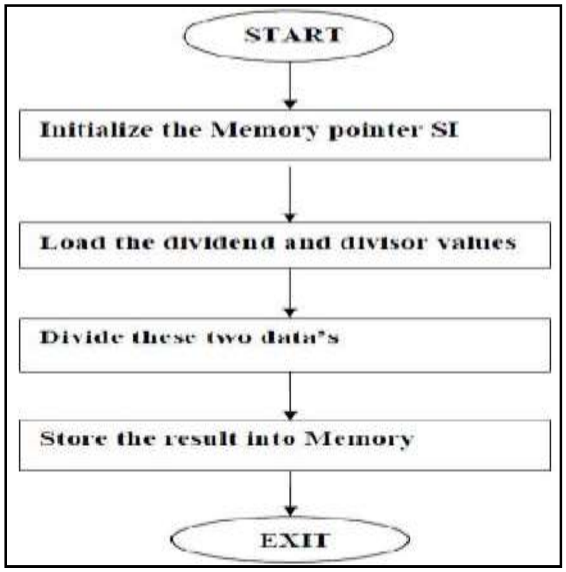
Memory location	Input data	Output data (result)
	AX = 0404	---------
	CX = 0202	---------
0404 * 0202 = 081008		
1300	---------	**08**
1301	---------	**10**
1302	---------	**08**

Division

Algorithm

- Initialize DX register to zero.
- Load the dividend in AX register.
- Load the divisor in CX register.
- Divide AX by CX (AX / CX). Quotient will be available in AX register. Remainder will be available in DX register.
- Initialize the DI register (1300) to store data in a memory location.
- Transfer the result from AX, DX registers to the above memory locations.

Flow chart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BA 00 00	MOV DX, 0000	Clear DX registers
1103	B8 06 00	MOV AX, **0006H**	Load the dividend in AX
1106	B9 04 00	MOV CX, **0004H**	Load the divisor value in BX
1109	F7 F1	DIV CX	Divide the two data's
110B	BF 00 13	MOV DI, **1300H**	Load 1300 address into DI
110E	88 05	MOV [DI], AL	Load AL value into DI
1110	47	INC DI	Increment DI
1111	88 25	MOV [DI], AH	Load AH value into DI
1113	47	INC DI	Increment DI
1114	89 15	MOV [DI], DX	Load DX value into DI
1116	F4	HLT	End

Result

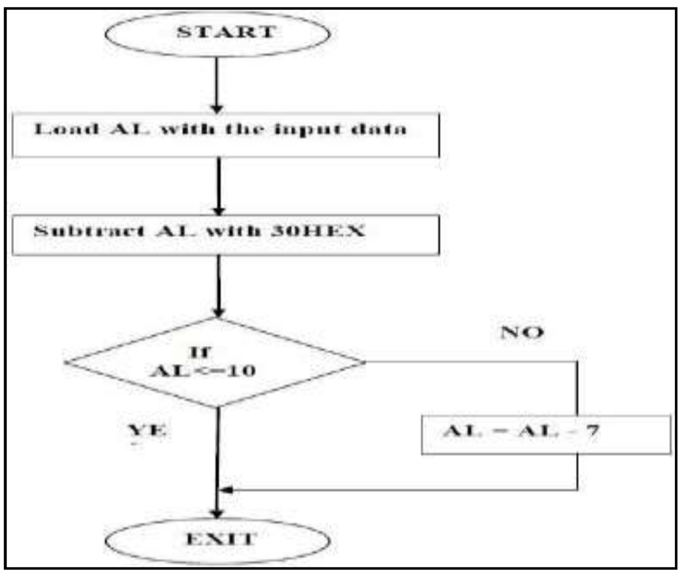
Memory location	Input data	Output data (result)
	AX = 0006	---------
	CX = 0004	---------
0006 / 0004 = quotient 1 in AX, Remainder 2 in DX		
1300	---------	**01**
1301	---------	**00**
1302	---------	**02**

Hexadecimal to ASCII Code conversion

Algorithm

- Get a data in AL register.
- Subtract the data in AL register by 30.
- If the result is less than or equal to 16 store the result in a memory location (1300).
- If the result is greater than 16 and not equal to then subtract the result again by 7 and store it in a memory location (1300).
- Move the content of AL register to the memory location (1300).

Flow chart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	B0 31	MOV AL,**31H**	Get data 31 into AL
1102	2C 30	SUB AL,30	Subtract 30 with the AL
1104	3C 10	CMP AL,10	If data is less than or equal to 16 go to 110C
1106	72 04	JB **LOOP**	If 1^{st} operand is below the 2^{nd} operand then short jump into 110C
1108	74 02	JZ **LOOP**	If count zero then jump into to 110C
110A	2C 07	SUB AL,07	Else subtract 7 from AL register value
110C	BE 00 13	**LOOP**: MOV SI,**1300H**	Load 1300 memory location
110F	88 04	MOV [SI],AL	Store the result
1111	F4	HLT	END

Result

Memory location	Input data	Output data (result)
	AL = 31	---------
AL = AL – 30 = 1		
1300	**---------**	**01**

Result

Thus the 16 bit arithmetic and logical operations was done using 8086 instruction set in an 8086 Microprocessor development kit.

<u>TASK 2: String operations using 8086 instruction set</u>

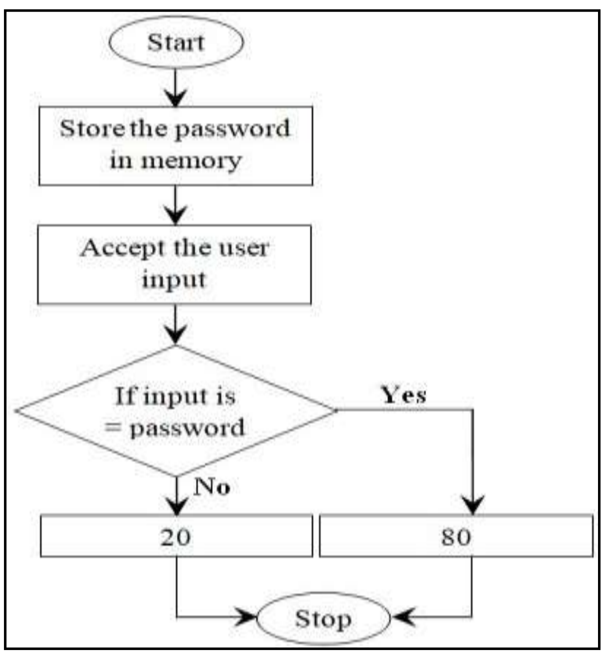
Aim

To develop an assembly language program to verify that the given input is equivalent to the preset password.

Algorithm

- Store the password in memory locations.
- Initialize DS, ES registers to calculate effective address.
- Calculate effective address using SI, DI registers.
- Input the password.
- Compare the received input string with the stored password.
- Display ‘80’ if the given input is equivalent to the password, otherwise display ‘20’.

Flowchart



Program

ADDRESS	Mnemonics	Comments
1100	MOV SI,1200	Store the password in 8 memory locations.
1103	MOV DI, 1208	Input the string for comparison.
1106	MOV CX, 0008	Initialize the count as ‘8’.
1109	CLD	Clear directional flag.
110A	REPE CMPSB	Repeat the comparison till all the strings are compared.
110C	JNE D1	Display ‘20’ if the input string is not equivalent to the
		password.
110E	JMP OK	Display ‘20’ if the input string is not equivalent to the password.
1110	D1 : MOV AL,20	Store AL ‘20’
1112	JMP E1	Store the result.
1114	OK : MOV AL,80	Store AL ‘80’
1116	E1 : MOV SI, 1210	Move the result to the memory location 1210.
1119	MOV AL,[SI]	
111B	HLT	End the program.

Output

Memory location	Input data	Output data (result)
1200 – 07	46H, 41H, 49H, 4CH, 53H, 41H, 46H, 45H	---------
1208 – 0F	46H, 41H, 49H, 4CH, 53H, 12H, 46H, 45H	---------
1210	---------	80H
1208 – 0F	46H, 41H, 49H, 4CH, 53H, 41H, 46H, 48H	---------
1210	---------	20H

Result

Thus the string instruction based assembly language program was able to verify the given input is equivalent to password.

TASK 3: Smallest and largest number using 8086 instruction set

Aim

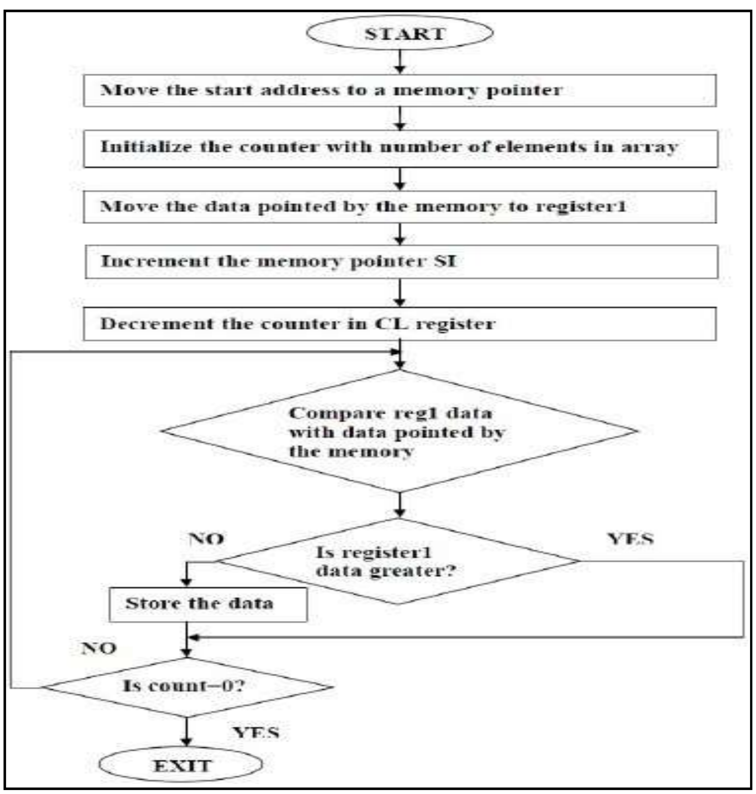
To develop an assembly language program to identify the smallest and largest number from the given set of numbers.

Largest number :

Algorithm

- Calculate Get the first number of the array.
- Get the second number and compare with the first one.
- Retain the bigger number in AL register.
- Decrement the count in CL register.
- If the count is not zero, repeat the above procedure after getting the next number from memory.
- Transfer the content of AL register to a memory location after all the data compared.

Flowchart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 12	MOV SI,**1200H**	Load 1200 address into SI
1103	8A 0C	MOV CL,[SI]	Load SI value into CL
1105	46	INC SI	Increment SI
1106	8A 04	MOV AL,[SI]	Move the first data in AL
1108	FE C9	DEC CL	Reduce the count
110A	46	**LOOP:** INC SI	Increment SI
110B	3A 04	CMP AL,[SI]	if AL> [SI] then go to jump1 (no swap)
110D	73 02	JNB **LOOP1**	If count is zero then jump into 1111
110F	8A 04	MOV AL,[SI]	Else store large no in to AL
1111	FE C9	**LOOP1**: DEC CL	Decrement the count
1113	75 F5	JNZ **LOOP**	If count is not zero then jump into 110A
1115	BF 00 13	MOV DI,**1300H**	Else store the biggest number at 1300
1118	88 05	MOV [DI],AL	Store the AL value into DI
111A	F4	HLT	End

Output

Memory location	Input data	Output data (result)
1200 – 05	05H, 09H, 01H, 02H, 07H, 08H	---------
1300	---------	09H

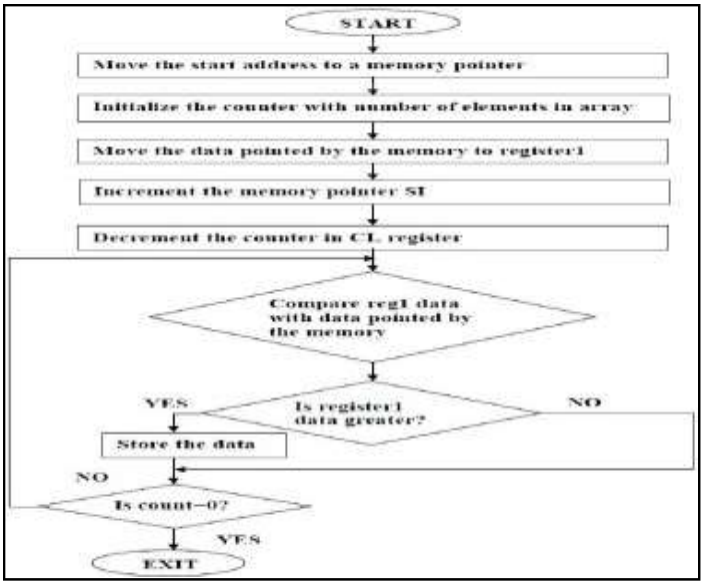
<u>Smallest number :</u>

Algorithm

Algorithm

- Calculate Get the first number of the array.
- Get the second number and compare with the first one.
- Retain the smaller number in AL register.
- Decrement the count in CL register.
- If the count is not zero, repeat the above procedure after getting the next number from memory.
- Transfer the content of AL register to a memory location after all the data compared.

Flowchart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 12	MOV SI,**1200H**	Load 1200 address into SI
1103	8A 0C	MOV CL,[SI]	Load SI value into CL
1105	46	INC SI	Increment SI
1106	8A 04	MOV AL,[SI]	Move the first data in AL
1108	FE C9	DEC CL	Reduce the count
110A	46	**LOOP:** INC SI	Increment SI
110B	3A 04	CMP AL,[SI]	if AL> [SI] then go to jump1 (no swap)
110D	73 02	JB **LOOP1**	If count is zero then jump into 1111
110F	8A 04	MOV AL,[SI]	Else store large no in to AL
1111	FE C9	**LOOP1**: DEC CL	Decrement the count
1113	75 F5	JNZ **LOOP**	If count is not zero then jump into 110A
1115	BF 00 13	MOV DI,**1300H**	Else store the biggest number at 1300
1118	88 05	MOV [DI],AL	Store the AL value into DI
111A	F4	HLT	End

Output

Memory location	Input data	Output data (result)
1200 – 05	05H, 09H, 01H, 02H, 07H, 08H	---------
1300	---------	01H

Result

Thus an assembly language program was developed to identify the smallest and largest number from the given set of numbers.

TASK 4: Ascending and descending order using 8086 instruction set

Aim

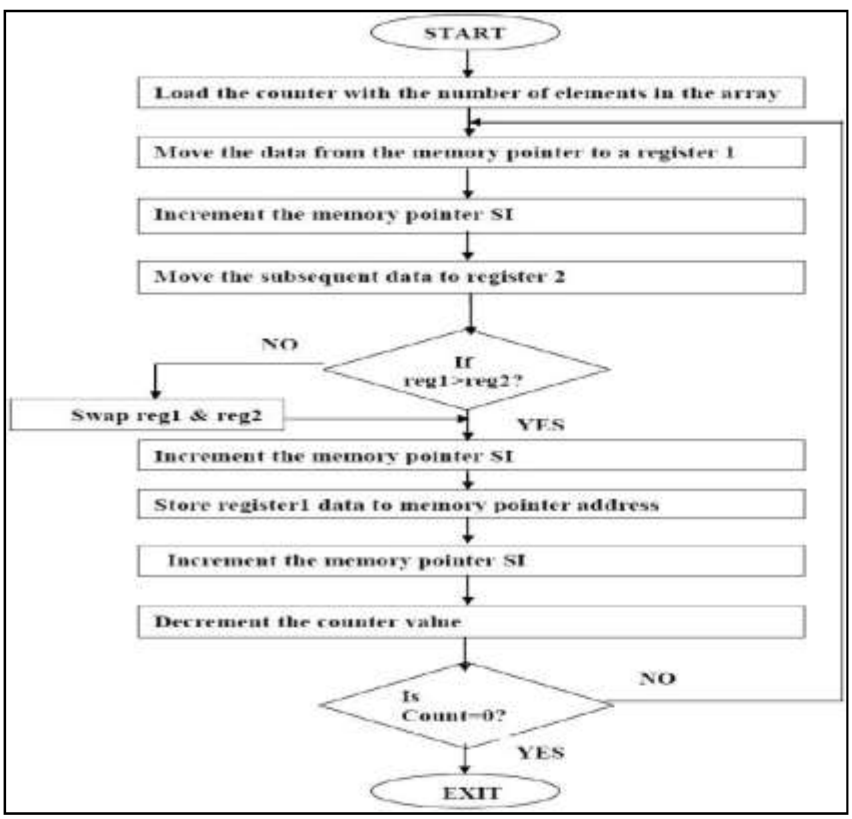
Write an assembly language program to arrange the given number from smallest to largest and vice versa.

Ascending order:

Algorithm

- Enter the total number of data to be arranged in ascending order.
- Get the first number in AL register.
- Get the second number in BL register.
- Compare the content of both AL, BL.
- Swap the content if AL > BL.
- Store the content of AL in memory.
- Get the next number in BL register and perform the comparison.
- Continue the comparison till the count becomes zero.

Flowchart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 13	MOV SI, **1300H**	Load 1300 into SI
1103	8A 0C	MOV CL, [SI]	Load SI value into CL
1105	BE 00 13	**LOOP:** MOV SI, **1300H**	Get second data
1108	8A 14	MOV DL, [SI]	Load SI second data into DL
110A	46	INC SI	Increment SI
110B	8A 04	MOV AL, [SI]	Load SI value into AL
110D	FE CA	DEC DL	Decrement DL
110F	74 16	JZ **LOOP4**	If count is zero then go to 1127
1111	46	**LOOP1:** INC SI	Increment SI
1112	8A 1C	MOV BL, [SI]	Load SI value into BL
1114	38 D8	CMP AL, BL	if AL > BL go to (jump1)
1116	73 07	JNB **LOOP2**	
1118	4E	DEC SI	Decrement SI
1119	88 04	MOV [SI],AL	Load AL value into SI
111B	88 D8	MOV AL, BL	Load BL value into AL
111D	EB 03	JMP **LOOP3**	
111F	4E	**LOOP2:** DEC SI	Decrement SI
1120	88 1C	MOV [SI], BL	Load BL value into SI
1122	46	**LOOP3:** INC SI	Increment SI
1123	FE CA	DEC DL	Decrement DL
1125	75 EA	JNZ **LOOP1**	If count is not zero then go to 1111
1127	88 04	**LOOP4:** MOV [SI], AL	Load AL value into SI
1129	FE C9	DEC CL	Decrement CL
112B	75 D8	JNZ **LOOP**	If count is not zero then go to 1105
112D	F4	HLT	

Output

Memory location	Input data	Output data (result)
1300	05H (Count value)	---------
1301 – 05	05, 04, 09, 06, 00	---------
1301 – 05		00, 04, 05, 06, 09

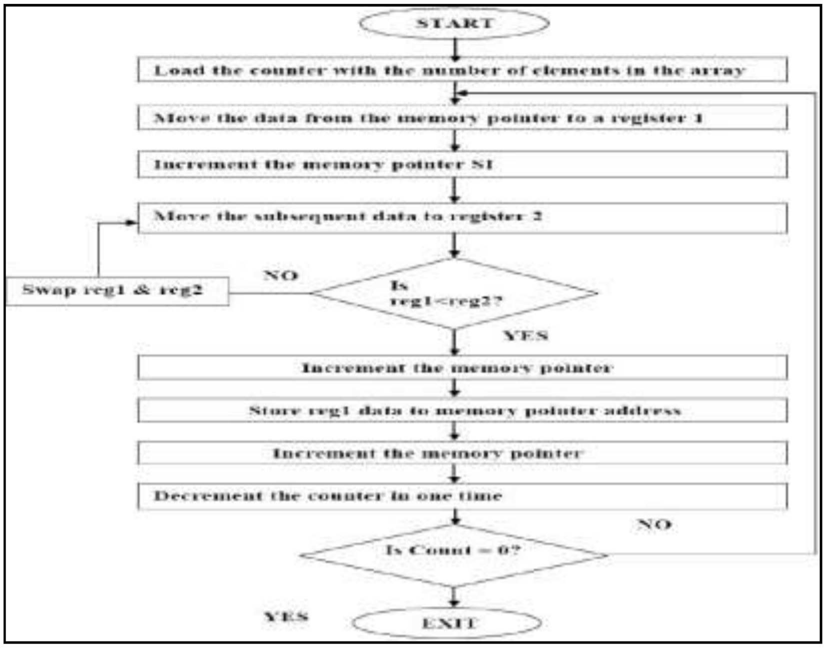
<u>Desscending order:</u>

Algorithm

- Enter the total number of data to be arranged in ascending order.
- Get the first number in AL register.
- Get the second number in BL register.
- Compare the content of both AL, BL.
- Swap the content if AL < BL.

- Store the content of AL in memory.
- Get the next number in BL register and perform the comparison.
- Continue the comparison till the count becomes zero.

Flowchart



Program

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1100	BE 00 13	MOV SI, **1300H**	Load 1300 into SI
1103	8A 0C	MOV CL, [SI]	Load SI value into CL
1105	BE 00 13	**LOOP:** MOV SI, **1300H**	Get second data
1108	8A 14	MOV DL, [SI]	Load SI second data into DL
110A	46	INC SI	Increment SI
110B	8A 04	MOV AL, [SI]	Load SI value into AL
110D	FE CA	DEC DL	Decrement DL
110F	74 16	JZ **LOOP4**	If count is zero then go to 1127
1111	46	**LOOP1:** INC SI	Increment SI
1112	8A 1C	MOV BL, [SI]	Load SI value into BL
1114	38 D8	CMP AL, BL	if AL > BL go to (jump1)
1116	72 07	JB **LOOP2**	
1118	4E	DEC SI	Decrement SI
1119	88 04	MOV [SI],AL	Load AL value into SI
111B	88 D8	MOV AL, BL	Load BL value into AL
111D	EB 03	JMP **LOOP3**	
111F	4E	**LOOP2:** DEC SI	Decrement SI
1120	88 1C	MOV [SI], BL	Load BL value into SI
1122	46	**LOOP3:** INC SI	Increment SI
1123	FE CA	DEC DL	Decrement DL
1125	75 EA	JNZ **LOOP1**	If count is not zero then go to 1111
1127	88 04	**LOOP4:** MOV [SI], AL	Load AL value into SI
1129	FE C9	DEC CL	Decrement CL
112B	75 D8	JNZ **LOOP**	If count is not zero then go to 1105
112D	F4	HLT	

Output

Memory location	Input data	Output data (result)
1300	05H (Count value)	---------
1301 – 05	05, 04, 09, 06, 00	---------
1301 – 05		09, 06, 05, 04, 00

Result

Thus an assembly language program was written to arrange the given number from smallest to largest and vice versa.

TASK 5: Data transfer using ARM processor

Aim

Write an assembly language program to establish data transfer using ARM processor instruction set.

Algorithm / Procedure

- Develop any simple program using Embedded C program using Keil software.
- Compile and build the program.
- Click on “Debug” option.
- the Keil software window will show the assembly language program
- Click on “Step one line” option.
- Analyse the data transfer established in the program.

Program

```
extern int count = 0;
void RamFunc (void) {
        int a[5] = {2,4,6,8,1};
        int i = 0;
        for (i=0;i<=4;i++)
        {
                a[i] = count + 4;
                count = count + 3;
        }
}

int main (void)  {

  while (1) {
    RamFunc ();                    /* execute function in RAM */
  }

}

void RamFunc (void) {
PUSH     {r4,lr}
SUB      sp,sp,#0x18
int a[5] = {2,4,6,8,1};
MOVS     r2,#0x14
```

```
LDR      r1,[pc,#44] ; @0x20004034
ADD      r0,sp,#0x04
BL.W     $Ven$TT$L$$__aeabi_memcpy4 (0x2000403C)
int i = 0;
MOVS     r4,#0x00
for (i=0;i<=4;i++)
{
NOP
B        0x2000402C
a[i] = count + 4;
LDR      r0,[pc,#32] ; @0x20004038
LDR      r0,[r0,#0x00]
ADDS     r0,r0,#4
ADD      r1,sp,#0x04
STR      r0,[r1,r4,LSL #2]
count = count + 3;
}
LDR      r0,[pc,#20] ; @0x20004038
LDR      r0,[r0,#0x00]
ADDS     r0,r0,#3
LDR      r1,[pc,#16] ; @0x20004038
STR      r0,[r1,#0x00]
for (i=0;i<=4;i++)
{
        a[i] = count + 4;
        count = count + 3;
}
ADDS     r4,r4,#1
CMP      r4,#0x04
BLE      0x20004014
}
```

Output

Memory location	**Input data**	**Output data (result)**
0x1000024C, 0x10000250, 0x10000254, 0x10000258, 0x1000025C	a[5] = {2,4,6,8,1}	--------------
0x1000024C, 0x10000250, 0x10000254, 0x10000258, 0x1000025C	--------------	a[0] = 04, a[1] = 07, a[2] = 0A, a[3] = 0D, a[4] = 10

Result

Thus an assembly language program for data transfer was realized using Embedded C program.

TASK 6: Count of negative numbers in an array using ARM processor instructions

Aim

Write an assembly language program to count the total negative numbers from the given set of numbers.

Algorithm

- Initialize specified number of elements in an array.
- Initialize the counter.
- Access the numbers in the array one by one and verify as whether it is less than zero.
- Increment the counter value if the verified number is negative.
- End the program if all the numbers are verified.

Program

```
# include <stdio.h>
# include <lpc17xx.h>

int main()
{
        int i=0,count=0;
        int a[10] = {1,2,-3,4,-5,6,-7,-89,76,-98};
        while(1)
        {
                for (i=0;i<=9;i++)
                {
                        if (a[i]<0)
                        {
                                count++;
                        }
                }
        }
}
```

Equivalent assembly instructions for every individual 'Embedded C' instruction

```
0x00000260 B08A     SUB      sp,sp,#0x28
```
int i=0,count=0;
```
0x00000262 2400     MOVS     r4,#0x00
0x00000264 2500     MOVS     r5,#0x00
```
int a[10] = {1,2,-3,4,-5,6,-7,-89,76,-98};

```
        0x00000266 2228      MOVS     r2,#0x28
        0x00000268 4907      LDR      r1,[pc,#28]  ; @0x00000288
        0x0000026A 4668      MOV      r0,sp
        0x0000026C F7FFFF8C  BL.W     aeabi_memcpy4 (0x00000188)
while(1)
        {
        0x00000270 E009      B        0x00000286
for (i=0;i<=9;i++)
        {
        0x00000272 2400      MOVS     r4,#0x00
        0x00000274 E005      B        0x00000282
if (a[i]<0)
        {
        0x00000276 F85D0024  LDR      r0,[sp,r4,LSL #2]
        0x0000027A 2800      CMP      r0,#0x00
        0x0000027C DA00      BGE      0x00000280
count++;
        0x0000027E 1C6D      ADDS     r5,r5,#1
for (i=0;i<=9;i++)
        0x00000280 1C64      ADDS     r4,r4,#1
        0x00000282 2C09      CMP      r4,#0x09
        0x00000284 DDF7      BLE      0x00000276
        0x00000286 E7F4      B        0x00000272
```

Output

Input data	Output data (result)
a[10] = {1,2,-3,4,-5,6,-7,-89,76,-98}	Count = 5

Result

Thus an assembly language program to identify the total negative numbers in the given set of number using Embedded C program was verified.

TASK 7: Factorial of positive integer N using ARM processor instructions

Aim

Write an assembly language program to find the factorial of a given number.

Algorithm

- Initialize the number whose factorial is to be found.
- Initialize the counter.
- Access the numbers in the array one by one and verify as whether it is less than zero.
- Increment the counter value if the verified number is negative.
- End the program if all the numbers are verified.

Program

```
# include <stdio.h>
# include <lpc17xx.h>

int main()
{
        int i=1,count=5,fact=count,k;
        while(1)
        {
                for (k=count;k>1;k--)
                {
                        fact = fact * (k-1);
                        count = k;
                }
        }
}
```

Equivalent assembly instructions for every individual 'Embedded C' instruction

int i=1,count=5,fact=count,k;

```
0x000001FA    BF00    NOP
0x000001FC    2105    MOVS   r1,#0x05
0x000001FE    460A    MOV    r2,r1
```

while(1)

{

```
0x00000200    E007    B      0x00000212
```

for (k=count;k>1;k--)

{

```
0x00000202    4608    MOV    r0,r1
0x00000204    E003    B      0x0000020E
```

fact = fact * (k-1);

```
0x00000206    1E43    SUBS   r3,r0,#1
0x00000208    435A    MULS   r2,r3,r2
```

count = k;

```
0x0000020A    4601    MOV    r1,r0
0x0000020C    1E40    SUBS   r0,r0,#1
0x0000020E    2801    CMP    r0,#0x01
0x00000210    DCF9    BGT    0x00000206
```

while(1)

```
0x00000212    E7F6    B      0x00000202
```

Output

Input data	Output data (result)
5H	78H

Result

Thus an assembly language program to find the factorial of a given number using Embedded C program was verified.

TASK 8: GCD of two numbers using ARM processor instructions

Aim

Write an assembly language program to find the GCD of two given numbers.

Algorithm

- Initialize the two numbers whose GCD is to be found.
- Consider the largest number.
- Increment count value from 1.
- Verify the number is divisible by both the numbers.
- Repeat from second step.
- Repeat the above step until count value is equal to the largest number.

Program

```
# include <stdio.h>
# include <lpc17xx.h>

int main()
{
        int n1=36, n2=60, i, gcd;
        for(i=1; i <= n1 && i <= n2; ++i)
        {
            if(n1%i == 0 && n2%i == 0)
            gcd = i;
        }
}
```

Equivalent assembly instructions for every individual ‘Embedded C’ instruction

```
        0x000001FA B510      PUSH     {r4,lr}
int n1=36, n2=60, i, gcd;
        0x000001FC 2224      MOVS     r2,#0x24
        0x000001FE 233C      MOVS     r3,#0x3C
for(i=1; i <= n1 && i <= n2; ++i)
        {
        0x00000200 2101      MOVS     r1,#0x01
        0x00000202 E00B      B        0x0000021C
if(n1%i==0 && n2%i==0)
        0x00000204 FB92F0F1  SDIV     r0,r2,r1
        0x00000208 FB012010  MLS      r0,r1,r0,r2
```

```
0x0000020C B928      CBNZ     r0,0x0000021A
0x0000020E FB93F0F1  SDIV     r0,r3,r1
0x00000212 FB013010  MLS      r0,r1,r0,r3
0x00000216 B900      CBNZ     r0,0x0000021A
gcd = i;
}
0x00000218 460C      MOV      r4,r1
0x0000021A 1C49      ADDS     r1,r1,#1
0x0000021C 4291      CMP      r1,r2
0x0000021E DC01      BGT      0x00000224
0x00000220 4299      CMP      r1,r3
0x00000222 DDEF      BLE      0x00000204
}
0x00000224 2000      MOVS     r0,#0x00
0x00000226 BD10      POP      {r4,pc}
```

Output

Input data	Output data (result)
36, 60	12

Result

Thus an assembly language program to find the GCD of a given two numbers using Embedded C program was verified.

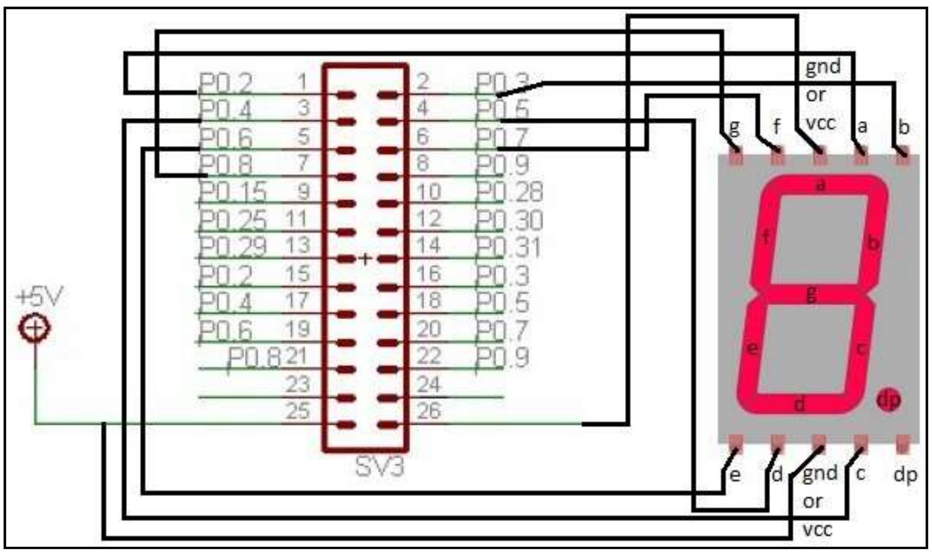
TASK 9 Seven segment LED display

Aim

Write an assembly language program to create a 7 Segment display using ARM processor interfacing with 7 Segment LED..

Algorithm

- Establish the connection between the ARM processor board and the 7 segment LED as shown below.
- Refer the table, identify the equivalent hexadecimal number for the display to be created.
- Include the value in the program.



GPIO	P0.8	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.0 - 1 Not used	Equivalent Hexadecimal number
Seven segment	**g**	**f**	**e**	**d**	**c**	**b**	**a**	**Not used**	
2	1	0	1	1	0	1	1	00	**0x0000016C**
3	1	0	0	1	1	1	1	00	**0x0000017C**
4	1	1	0	0	1	1	0	00	**0x00000198**

Program

```
# include <stdio.h>
# include <lpc17xx.h>

#include "LPC17xx.h"
```

```
volatile unsigned int delay;
int main (void)
{
	LPC_PINCON->PINSEL0 = (LPC_PINCON->PINSEL0 &~ 0x00FFFF00) ;
	LPC_GPIO0->FIODIR = (LPC_GPIO0->FIODIR | 0x00000FFF) ;
	while(1)
		{
			LPC_GPIO0->FIOCLR = (LPC_GPIO0->FIOCLR | 0x00000198);
		for(delay=0; delay<5000; delay++);
			LPC_GPIO0->FIOSET = (LPC_GPIO0->FIOSET | 0x00000198);
		for(delay=0; delay<5000; delay++);
		}
}
```

Result

Thus a program for 7 segment LED interfacing with ARM processor was developed and verified.

TASK 10 Interfacing Keyboard and display

Aim

Write an assembly language program to interfacing keyboard and display using ARM processor.

Algorithm

- Initialize the target device and reset peripherals.
- Initialize UART0 for serial communication.
- Send a message "Keypad Test" over UART0.
- Enter an infinite loop:

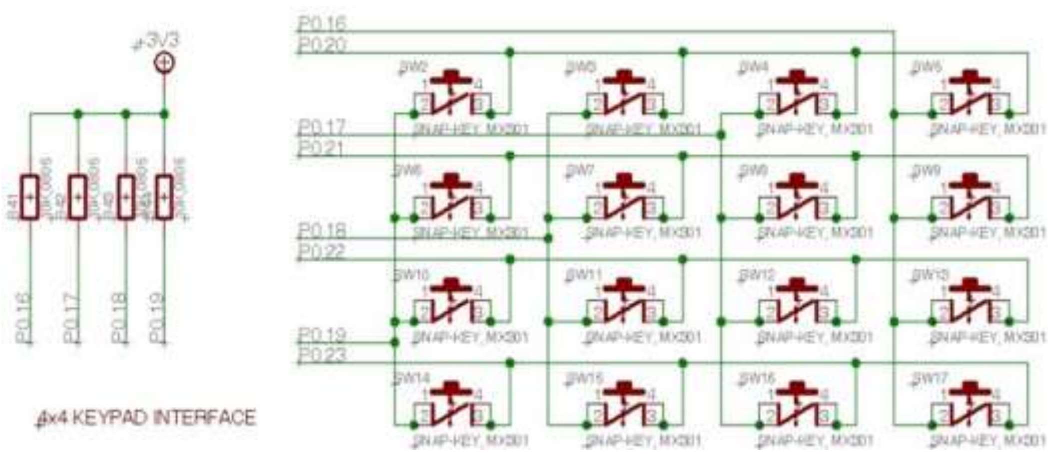
  Read a key press from the keypad using KBD_rdkbd() function.

  Convert the key code into ASCII format using sprintf() and store it in the szTemp array.

  Send the ASCII representation of the key code over UART0 for display.
- Repeat steps 4a to 4c indefinitely.

Connections

16 Keys (SW2 to SW17) present in 4x4 Matrix Keypad region on ADT are connected to P0.16 to P0.23.



Program

```
#include "LPC17xx.h"
#include <stdio.h>
#include <string.h>
#include "target.h"
#include "UART0.h"
#include "KBD.h"
int i8ch ;
char szTemp[16] ;
int main (void)
{
    TargetResetInit();
  InitUart0();
    UART0Puts("Keypad Test \n");
    while(1)
    {
        i8ch = KBD_rdkbd() ;                    // Read
Keyboard
        sprintf(szTemp,"\nKeyCode = %02X",i8ch); // Convert keycode
into ASCII to display it on LCD
        UART0Puts(szTemp) ;                              //
Display keycode on 2nd line of LCD
    }
}
```

Result

Thus a program for interfacing keyboard and display using ARM processor was developed and verified

TASK 11 UART for Driver Data Transmission

Aim

Write an assembly language program to demonstrate basic UART (serial communication) functionality on an LPC17xx microcontroller.

Algorithm

- Initialize the target device and reset peripherals.
- Initialize UART0 for serial communication.
- Send "HELLO World!" message over UART0.
- Enter an infinite loop:
- Read a character from UART0 using UART0getchar().
- Send the read character back over UART0 using UART0putchar().
- Repeat steps 4a to 4b indefinitely.

Program

```
#include "LPC17xx.h"

#include <stdio.h>

#include "target.h"

#include "UART0.h"


int main(void)

{

    TargetResetInit();

    InitUart0();

    UART0Puts("HELLO World! \n");

    while(1)

    {

        UART0putchar(UART0getchar());

    }

}
```

Result

Thus a program for program to demonstrate basic UART (serial communication) functionality on an LPC17xx microcontroller was developed and verified

TASK 12 Temperature measurement using ARM processor Kitdevelopment board

Aim

Write a program for temperature measurement using ARM processor Kit development board.

Algorithm

1. **Initialize ADC (ADC_Init):**

   - Enable power to ADC module.

   - Configure pin P0.4 as ADC input.

2. **Read ADC Value (ADC_GetAdcReading):**

   - Configure ADC control register (ADCR) to select AD0.7, select clock for ADC, and start conversion.

   - Wait until the conversion is complete (done bit set).

   - Read ADC global data register (ADGDR) to obtain the converted ADC value.

   - Extract the 10-bit ADC result and return it.

3. **Main Function:**

   - Initialize target and UART0.

   - Call ADC_Init to initialize ADC settings.

   - Send "ADC LM35 Test" message over UART0.

   - Enter an infinite loop.

     - Read ADC value from AD0.7.

     - Calculate temperature from ADC value.

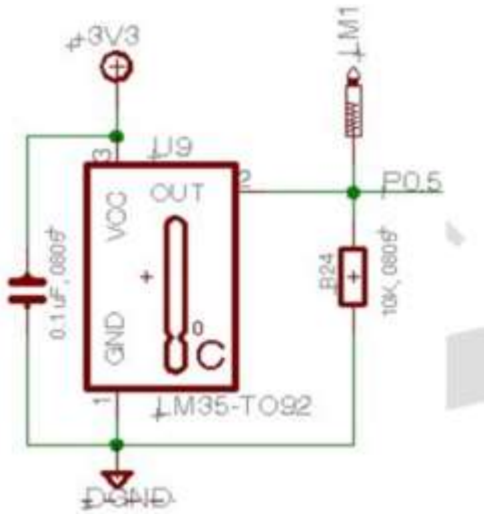
     - Convert temperature to ASCII format.

     - Send temperature value over UART0 for display.

     - Delay for a short period to slow down the loop.

Connections:

To interface LM35 (present in Analog Input region on ADT Board) with P0.5 .



Program

```
#include "LPC17xx.h"

#include "stdio.h"

#include "UART0.h"

#include "target.h"

#define PCADC 0x00001000


char adcreading[16] ;                                              //
Variable to hold the ADC converted Value


void ADC_Init (void)
                    // ADC initialation function

{

      LPC_SC->PCONP |= PCADC ;

      LPC_PINCON->PINSEL1 = (LPC_PINCON->PINSEL1 |= 0x00010000
);                        // P0.4 is configured as ADC Input

}
```

```
unsigned int ADC_GetAdcReading ()
{
	unsigned int adcdata;

	LPC_ADC->ADCR = 0x01200302 ;
							// Select AD0.7, Select clock for ADC,
Start of conversion,
	while(!((adcdata = LPC_ADC->ADGDR) & 0x80000000))		//
Check end of conversion (Done bit) and read result
	{
	}
	return((adcdata >> 6) & 0x3ff) ;				// Return 10 bit result
}
int main (void)
{
	unsigned int delay,adc;
	float Temp, Temperature;
	//float adc;
	TargetResetInit();
		InitUart0();
		ADC_Init() ;
				// Initiate ADC Setting
		UART0Puts("ADC LM35 Test");
		// Display ADC Test

	while(1)
	{
		adc = ADC_GetAdcReading();
				// Read AD0.7 reading
```

```
// Calculate Temperature

Temp = (adc * 3.3);

Temperature = (Temp / 1023) * 100;


// Display Temperature on LCD


sprintf(adcreading,"\nTemp =%0.03f *C",Temperature) ;  //
Convert result into ASCII to display it on LCD

UART0Puts(adcreading) ;
// Display Temperature on LCD


for(delay=0;delay<60000;delay++);

}

}
```

Result

Thus a program for temperature measurement using ARM processor Kit development board was developed and verified.

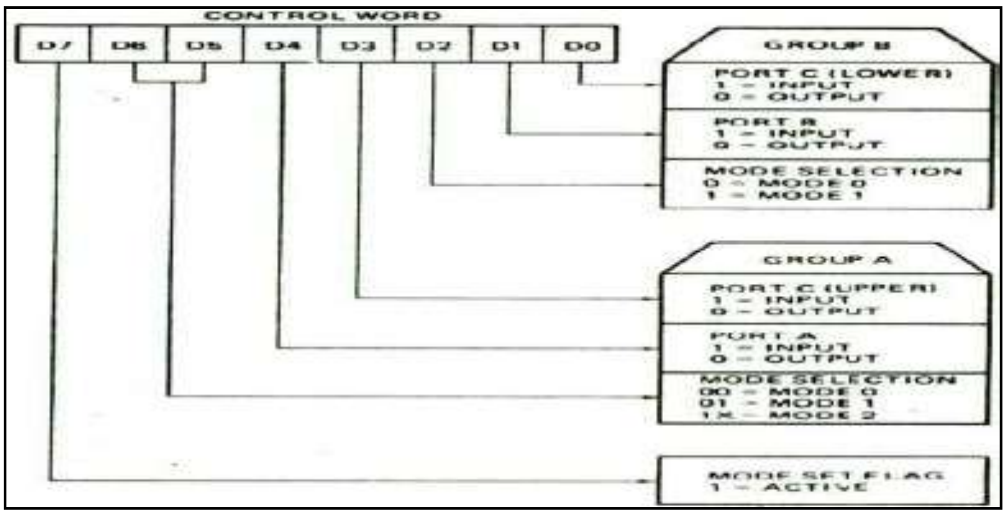
<u>Use case 1 – count the number of customers using 8086 Processor</u>

Aim

To develop an application which count the number of customers entering into a supermarket for purchasing using 8086 Microprocessor.

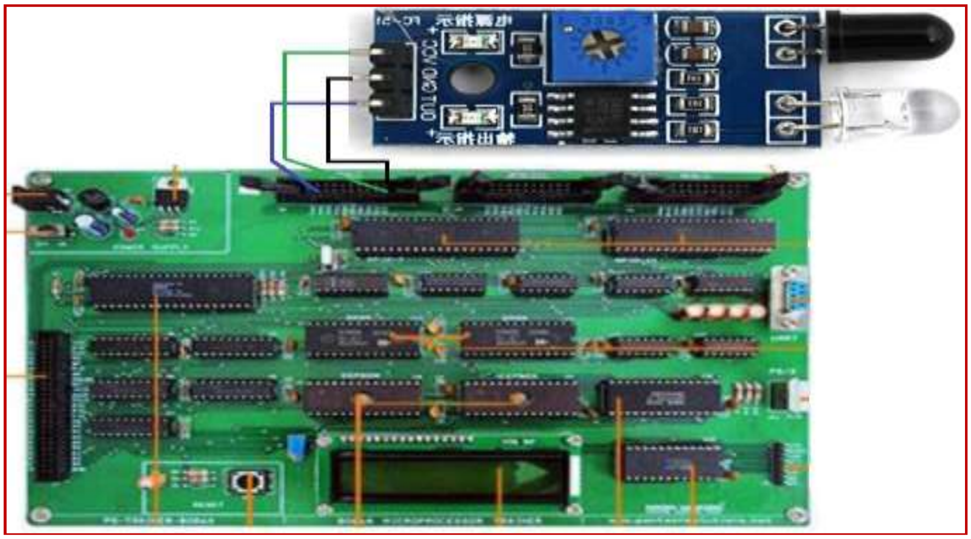
Theory

An IR sensor placed at the entrance of the supermarket senses the people who enter inside the market. It is connected to the 8086 microprocessor. It is actually connected to the PortA – 0 pin of parallel port in the 8086 microprocessor board. The control word shown in the below diagram explains the procedure for generating control word.



D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Hexa decimal value**
1	0	0	1	0	0	0	0	90
Mode set flag	Mode 0		Port A – Input	Port C –Output	Mode 0	Port B – Output	Port C – Output	

The logic written in the microprocessor will read the IR sensor output through port A. It increments the count value whenever the IR sensor output is high. Updated count value is stored in a memory location. Therefore, the present count shall be always read by referring to the memory location. The connection between the IR sensor and the microprocessor is shown in the below diagram.



Assembly program for counting

```
          MOV CX, 0000
          MOV SI, 1200
          MOV [SI], CX
read:     MOV DX, FF26
          MOV AL, 0090
          OUT DX, AL
          MOV DX, FF20
          IN AL, DX
          MOV AH, 00
          AND AL, AH
          JZ next
          INC CX
          MOV [SI], CX
next:     JMP read
```

In the above program, counter (CX register) is initialized to zero. The control word (90H) developed using the above configuration is released to the port A (control reg address – FF26). Now, the processor reads the output of IR sensor by reading the content of Port A (Port A address – FF20). This is logically OR with ‘0’ to verify the entry of a person. Processorincrements the counter value when the ‘0’th bit of Port A is set. Otherwise, the control will be transferred for reading the next entry. Updated count value is stored in the memory location ‘1200’.

Result

Thus this case study was able to count the number people who enter into a supermarket.

Use case 2 – Collision avoidance using over speed

Aim

To implement a scheme that avoids collision between two cars due to over speed.

Theory

Ultrasonic sensor located in the car A senses the distance of the car B speeding in front of it. It detects the decrease in the distance whenever the speed of the car A is greater than that of the car B. A microcontroller which reads the ultrasonic sensor output recognizes the difference and initiates action accordingly. Figure 1 shows the pin diagram as well as the ultrasonic sensor.

Figure 1 – Ultrasonic sensor

It is able to detect the distance in the range of 2cm – 400cm. Output voltage range of the sensor is 0 – 5V. If the the ADC output is 8bit then its resolution is 19.53mV (5V/256). Now, the resolution of the ultrasonic sensor is 79.6cm/V (398/5V). Therefore, an 8-bit ADC is able to sense the distance with the resolution of 1.56cm (19.53mV * 79.6cm/V)). However, it shall be improved when increasing the output data size of ADC.

Program – ADC

```
#include "LPC17xx.h"
#include "stdio.h"
#define PCADC 0x00001000

char adcreading[16] ;
void ADC_Init (void)
{
	LPC_SC->PCONP |= PCADC ;
```

```
        LPC_PINCON->PINSEL1 = (LPC_PINCON->PINSEL1 |= 0x00004000 );
}
unsigned int ADC_GetAdcReading ()
{
        unsigned int adcdata;
        LPC_ADC->ADCR = 0x01200301 ;
        while(!((adcdata = LPC_ADC->ADGDR) & 0x80000000))
        {
        }
        return((adcdata >> 4) & 0x3ff) ;
}
int main (void)
{
        unsigned int delay,adc=25;
        int Temp, Temperature;
                LPC_GPIO0->FIODIR  = 0x00000FF0;
                ADC_Init() ;
        while(1)
        {
                adc = ADC_GetAdcReading();
                LPC_GPIO0->FIOSET = adc;
        }
}
```

Above program displays the equivalent digital output for the applied analog input. Analog input is applied by changing the potentiometer in the ARM kit. Similarly the digital output shall be read by observing those LEDs in the ARM kit. If the potentiometer is replaced by the ultrasonic sensor, then the above code shall be used to monitor the over speed of a car and initiate a suitableaction.

Result

Thus this case study was able to monitor the over speed of a car with the help of an ultrasonic sensor and hence initiate a suitable action.

Use case 3 – Identify the frequency of the given signal and display in LCD

Aim

To develop an Embedded ‘C’ program that identifies the frequency of the signal generated in the program and display it in LCD.

Theory

The operating frequency of the LPC1768 Processor is 100MHz. Timer in the processor creates a delay by referring to this clock signal. This generates a new square waveform of the frequency which will be always lesser than the processor clock frequency. Embedded ‘C’program written using this creates a new signal of specific and display the value in LCD display.

Program

```
#include "LPC17xx.h"
#include <stdio.h>

#define PRESCALE (25000-1) //25000 PCLK clock cycles to increment TC by 1

void initTimer0(void)
{
	LPC_SC->PCONP |= (1<<1);
	LPC_SC->PCLKSEL0 &= ~(0x3<<3);
	LPC_TIM0->CTCR = 0x0;
	LPC_TIM0->PR = PRESCALE;
	LPC_TIM0->TCR = 0x02; //Reset Timer
}

void SmallDelay (void)
{
	volatile int i,j; j = 0;
	for(i=0;i<800;i++)
	{
		j++;
	}
}

void Delay250 (void)
{
```

```
	 volatile int k,j ;
	j =300;
	for(k = 0 ; k <200 ; k ++)
	{
		j-- ;
	}
}

void DelayMs (int n)
{
	volatile int k ;
	for(k = 0 ; k < n ; k ++)
	{
		Delay250() ;
	}
}

void LcdCmd(unsigned char cmd)
{
	if (cmd & 0x01)
		LPC_GPIO0->FIOSET = (1<<4);
	else
		LPC_GPIO0->FIOCLR = (1<<4);
	if (cmd & 0x02)
		LPC_GPIO0->FIOSET = (1<<5);
	else
		LPC_GPIO0->FIOCLR = (1<<5);
	if (cmd & 0x04)
		LPC_GPIO0->FIOSET = (1<<6);
	else
		LPC_GPIO0->FIOCLR = (1<<6);
	if (cmd & 0x08)
		LPC_GPIO0->FIOSET = (1<<7);
	else
		LPC_GPIO0->FIOCLR = (1<<7);
	if (cmd & 0x10)
		LPC_GPIO0->FIOSET = (1<<8);
	else
		LPC_GPIO0->FIOCLR = (1<<8);
	if (cmd & 0x20)
		LPC_GPIO0->FIOSET = (1<<9);
	else
		LPC_GPIO0->FIOCLR = (1<<9);
	if (cmd & 0x40)
		LPC_GPIO0->FIOSET = (1<<10);
	else
```

```
		LPC_GPIO0->FIOCLR = (1<<10);
	if (cmd & 0x80)
		LPC_GPIO0->FIOSET = (1<<11);
	else
		LPC_GPIO0->FIOCLR = (1<<11);
	LPC_GPIO1->FIOCLR = (1<<27) | (1<<26); SmallDelay() ;
	LPC_GPIO1->FIOSET = (1<<27) ; SmallDelay() ;
	LPC_GPIO1->FIOCLR = (1<<27)  ;  SmallDelay() ;
}

void LcdDat(unsigned char dat)
{
	if (dat & 0x01)
		LPC_GPIO0->FIOSET = (1<<4);
	else
		LPC_GPIO0->FIOCLR = (1<<4);
	if (dat & 0x02)
		LPC_GPIO0->FIOSET = (1<<5);
	else
		LPC_GPIO0->FIOCLR = (1<<5);
	if (dat & 0x04)
		LPC_GPIO0->FIOSET = (1<<6);
	else
		LPC_GPIO0->FIOCLR = (1<<6);
	if (dat & 0x08)
		LPC_GPIO0->FIOSET = (1<<7);
	else
		LPC_GPIO0->FIOCLR = (1<<7);
	if (dat & 0x10)
		LPC_GPIO0->FIOSET = (1<<8);
	else
		LPC_GPIO0->FIOCLR = (1<<8);
	if (dat & 0x20)
		LPC_GPIO0->FIOSET = (1<<9);
	else
		LPC_GPIO0->FIOCLR = (1<<9);
	if (dat & 0x40)
		LPC_GPIO0->FIOSET = (1<<10);
	else
		LPC_GPIO0->FIOCLR = (1<<10);
	if (dat & 0x80)
		LPC_GPIO0->FIOSET = (1<<11);
	else
		LPC_GPIO0->FIOCLR = (1<<11);
	LPC_GPIO1->FIOSET = (1<<26)  ;  SmallDelay() ;
	LPC_GPIO1->FIOCLR = (1<<27) ;  SmallDelay() ;
```

```
        LPC_GPIO1->FIOSET = (1<<27)  ;  SmallDelay() ;
        LPC_GPIO1->FIOCLR = (1<<27)  ; SmallDelay() ;
}

void LcdInit (void)
{

        LPC_GPIO0->FIODIR |= 0x00000FF0  ;      LPC_GPIO0->FIOCLR |= 0x00000FF0  ;
        LPC_GPIO1->FIODIR |= 0x0C000000  ;      LPC_GPIO1->FIOCLR |= 0x0C000000;
        DelayMs(6)  ;  LcdCmd(0x03)  ;
        DelayMs(6)  ;  LcdCmd(0x03)  ;
        Delay250() ;
        LcdCmd(0x03)    ;    Delay250()    ;
        LcdCmd(0x02)    ;    Delay250()    ;
        LcdCmd(0x38) ;
        LcdCmd(0x01);      LcdCmd(0x08)      ;
        LcdCmd(0x0c) ; LcdCmd(0x06) ;

void DisplayRow (int row, char *str)
{
        int k ;
        if (row == 1)
                LcdCmd(0x80) ;
        else
                LcdCmd(0xc0) ;
        for(k = 0 ; k < 16 ; k ++)
        {
                if (str[k])
                        LcdDat(str[k]) ;
                else
                        break ;
        }

        while(k < 16)
        {
                LcdDat(' ') ;
                k ++ ;
        }
}

void display_char(int row, char *str)
{
                LcdCmd(0x80) ;
                LcdDat(*str) ;
}

void delayMS(unsigned int milliseconds)
```

```
{
        LPC_TIM0->TCR = 0x02;
        LPC_TIM0->TCR = 0x01;
        DisplayRow (2,"32");
        while(LPC_TIM0->TC < milliseconds);
        {
                LPC_TIM0->TCR = 0x00;
        }
}

int main ()
{
        volatile int l;
        LcdInit();
        while(1)
                delayMS(500);

}
```

Result

Thus this case study was able to recognize the frequency of the signal derived in the program and display the same in LCD display.

Use case 4 – Embedded ‘C’ program to develop PWM signal

Aim

To develop an Embedded ‘C’ program to generate PWM signal from the given input value.

Theory

PWM generates square wave pulses of different widths. The width of the pulse, some times called pulse width offset, is commonly proportional to the amplitude of the input signal. PWM is used to control motors or to power LEDs. The main reason for using PWM is that it allows for controlling the average amount of power delivered to a load or the output. They are also used for voltage regulation and modulation in communications.

A PWM system normally has several parameters for tuning, including the PWM period (reciprocal of the pulse frequency), the pulse width offset, and the pulse delay time. Pulse delay time represents the shift in time compared to the pulse starting at 0 s. The values can be given in percentages of PWM period or seconds.

Program

```
#include "LPC17xx.h"
#include "PWM.h"
#include "lpc_types.h"

volatile unsigned int *MR_Reg ;
unsigned char flag=0,flag1 = 0; int i = 0;

void RedLedGlow(void);
void GreenLedGlow(void);
void BlueLedGlow(void);
void ALLLedGlow(void);

int main (void)
{
        while(1)
        {
                RedLedGlow();
                GreenLedGlow();
                BlueLedGlow();
                ALLLedGlow();
        }
}
```

```
void RedLedGlow(void)
{
	PWM_Init( RED, PWM_CYCLE );
	MR_Reg = (volatile unsigned int *)&(LPC_PWM1->MR1);
	updateCount(MR_Reg,LER1_EN);
}

void GreenLedGlow(void)
{
	PWM_Init( GREEN, PWM_CYCLE );
	MR_Reg = (volatile unsigned int *)&(LPC_PWM1->MR2);
	updateCount(MR_Reg,LER2_EN);
}

void BlueLedGlow(void)
{
	PWM_Init( BLUE, PWM_CYCLE );
	MR_Reg = (volatile unsigned int *)&(LPC_PWM1->MR3);
	updateCount(MR_Reg,LER3_EN);
}

void ALLLedGlow(void)
{
	int complete = 0, count = 0;                                              //
Define 2 integer variable
	flag = 0x00;
	flag1  = 0x00;
	PWM_Init( ALL, PWM_CYCLE );
	while(complete == 0)
	{
		for(i=0;i<=PWM_OFFSET;i++);
		if(flag == 0x00)
		{
			LPC_PWM1->MR1 = LPC_PWM1->MR1 + 10;
			LPC_PWM1->MR2 = LPC_PWM1->MR2 + 10;
			LPC_PWM1->MR3 = LPC_PWM1->MR3 + 10;
			LPC_PWM1->LER = LER1_EN | LER2_EN | LER3_EN ;
			if(LPC_PWM1->MR2 >= 49000)
			{
				flag = 0xff;
				flag1 = 0xff;
				if(count == 1)
					complete = 1;
				LPC_PWM1->LER = LER1_EN | LER2_EN | LER3_EN ;
			}
```

```
                }
                else if(flag1 == 0xff)
                {
                        LPC_PWM1->MR1 = LPC_PWM1->MR1 - 10;
                        LPC_PWM1->MR2 = LPC_PWM1->MR2 - 10;
                        LPC_PWM1->MR3 = LPC_PWM1->MR3 - 10;
                        LPC_PWM1->LER = LER2_EN ;
                        if(LPC_PWM1->MR2 <= 500)
                        {
                                flag1  = 0x00;
                                flag = 0x00;
                                count = 1;
                                LPC_PWM1->LER = LER1_EN | LER2_EN | LER3_EN ;
                        }
                }
        }
}

void updateCount(volatile unsigned int *MR_Register, int val )
{
        int complete = 0, count = 0;
        flag = 0x00;
        flag1 = 0x00;
        while(complete == 0)
        {
                for(i=0;i<=PWM_OFFSET;i++);
                if(flag == 0x00)
                {
                        *MR_Register = *MR_Register + 10;
                        LPC_PWM1->LER = val ;
                        if(*MR_Register >= 49000)
                        {
                                flag1 = 0xff;
                                flag = 0xff;
                                if(count == 1)
                                        complete = 1;
                                LPC_PWM1->LER = val ;
                        }
                }
                else if(flag1 == 0xff)
                {
                        *MR_Register = *MR_Register - 10;
                        LPC_PWM1->LER = val ;
                        if(*MR_Register <= 500)
                        {
                                flag  = 0x00;
```

```
                        flag1 = 0x00;
                        count = 1;
                        LPC_PWM1->LER =  val ;
                    }
                }
            }
        }
```

In the above program, indicating status of LEDs of different colours like Red, Green and Blue are changed according to the value of PWM.

Result

Thus an Embedded ‘C’ program was developed to generate PWM waveform and displayed using LED of different colours.