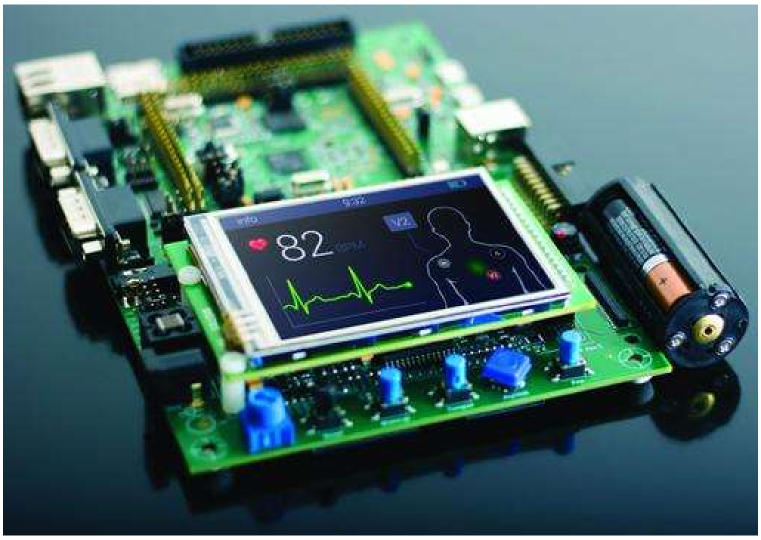
MAHARAJA INSTITUTE OF TECHNOLOGY THANDAVAPURA
NH 766, Nanjangud Taluk, Mysuru- 571 302
(An ISO 9001:2015 and ISO 21001:2018 Certified Institution)
(Affiliated to VTU, Belagavi and approved by AICTE, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Microcontroller & Embedded Systems Laboratory Manual

(2018 CBCS Scheme)

VII Semester B.E Computer Science & Engineering



DEPARTMENT OF MECHANICAL ENGINEERING

VISION OF THE DEPARTMENT

"To be frontier in producing globally competent graduates towards becoming significant part of industry or research and contribute to betterment of the nation"

MISSION OF THE DEPARTMENT

M1 Providing theoretical and practical knowledge through effective pedagogies that can be applied for betterment of the global competency.

M2 Strengthening technical skills needed to adapt to the changing scenario, through industry-academia interface & Alumni interaction.

M3 Developing innovative research capabilities through MoU's with national and international universities.

M4 Empowering Professional & Leadership skills to make competent software engineer.

PROGRAMME SPECIFIC OUTCOMES (PSOS):

PSO1 Carry out research in the advanced areas of Computer Science and address the basic needs of the society.

PSO2 Apply computational knowledge and skills to provide innovative solutions.

PSO3 Become an Entrepreneur and deliver a quality product for business success.

PSO4 Pursue Higher Education or qualify competitive examinations.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOS):

PEO1 Apply problem solving skills to empower the society.

PEO2 Function as a responsible member of society with willingness to mentor fellow employees and impart ethical, social and economic impact of their work in global context.ss

PEO3	Inculcate a conviction to believe in self, impart professional and ethical attitude, nurture to be an effective team member, infuse leadership qualities, build proficiency in soft skills and the abilities to relate engineering with the social issues.
PEO4	Provide in depth knowledge to design and develop novel products and innovative solution for real life problems.

PROGRAM OUTCOMES (POs)

The Department of Computer Science & Engineering has adopted the NBA Outcomes as its Program Outcomes. These are that our graduates have ability to:	
PO1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

SYLLABUS

MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY
(Effective from the academic year 2018 -2019)
SEMESTER – IV

Course Code	**18CSL48**	**CIE Marks**	40
Number of Contact Hours/Week	0:2:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03

Credits – 2

Course Learning Objectives: This course (18CSL48) will enable students to:

- Develop and test Program using ARM7TDMI/LPC2148
- Conduct the experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

Descriptions (if any):

Programs List:

PART A Conduct the following experiments by writing program using ARM7TDMI/LPC2148 using an evaluation board/simulator and the required software tool.

1.	Write a program to multiply two 16 bit binary numbers.
2.	Write a program to find the sum of first 10 integer numbers.
3.	Write a program to find factorial of a number.
4.	Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM
5.	Write a program to find the square of a number (1 to 10) using look-up table.
6.	Write a program to find the largest/smallest number in an array of 32 numbers .
7.	Write a program to arrange a series of 32 bit numbers in ascending/descending order.
8.	Write a program to count the number of ones and zeros in two consecutive memory locations.

PART –B Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

9.	Display "Hello World" message using Internal UART.
10.	Interface and Control a DC Motor.
11.	Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.
12.	Determine Digital output for a given Analog input using Internal ADC of ARM controller.
13.	Interface a DAC and generate Triangular and Square waveforms.
14.	Interface a 4x4 keyboard and display the key code on an LCD.
15.	Demonstrate the use of an external interrupt to toggle an LED On/Off.
16.	Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between

Laboratory Outcomes: The student should be able to:

- Develop and test program using ARM7TDMI/LPC2148
- Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  g) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 =

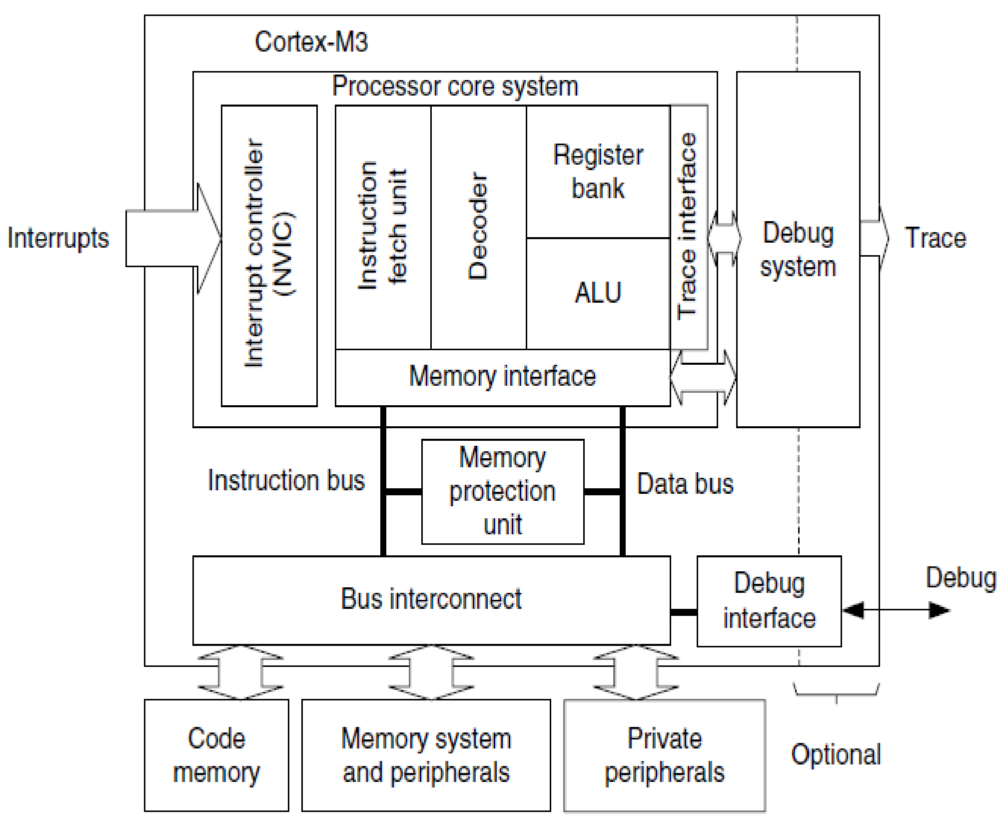
CONTENTS

INTRODUCTION TO ARM CORTEX M3

The Cortex™-M3 is a 32-bit microcontroller. It has a 32-bit data path, a 32-bit register bank, and 32-bit memory interfaces. The processor has a Harvard architecture, which means that it has a separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time, and as a result of this, the performance of the processor increases because data accesses do not affect the instruction pipeline.
This feature results in multiple bus interfaces on Cortex-M3, each with optimized usage and the ability to be used simultaneously. However, the instruction and data buses share the same memory space (a unified memory system).

For complex applications that require more memory system features, the Cortex-M3 processor has an optional Memory Protection Unit (MPU), and it is possible to use an external cache if it's required. Both little endian and big endian memory systems are supported.

The Cortex-M3 processor includes a number of fixed internal debugging components. These components provide debugging operation supports and features, such as breakpoints and watch points



Registers:

The Cortex-M3 processor has registers R0 through R15

Name		*Functions (and banked registers)*	
R0		General-purpose register	Low registers
R1		General-purpose register	
R2		General-purpose register	
R3		General-purpose register	
R4		General-purpose register	
R5		General-purpose register	
R6		General-purpose register	
R7		General-purpose register	
R8		General-purpose register	High registers
R9		General-purpose register	
R10		General-purpose register	
R11		General-purpose register	
R12		General-purpose register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	

R0–R12: General-Purpose Registers:

R0–R12 are 32-bit general-purpose registers for data operations. Some 16-bit Thumb® instructions can only access a subset of these registers (low registers, R0–R7).

R13: Stack Pointers

The Cortex-M3 contains two stack pointers (R13). They are banked so that only one is visible at a time.

The two stack pointers are as follows:

• *Main Stack Pointer (MSP)*: The default stack pointer, used by the operating system (OS) kernel and exception handlers

• *Process Stack Pointer (PSP)*: Used by user application code

The lowest 2 bits of the stack pointers are always 0, which means they are always word aligned.

R14: The Link Register

When a subroutine is called, the return address is stored in the link register.

R15: The Program Counter

The program counter is the current program address. This register can be written to control the program flow.

Special Registers

The Cortex-M3 processor also has a number of special registers (see Figure 2.3). They are as follows

- Program Status registers (PSRs)
- Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)
- Control register (CONTROL)

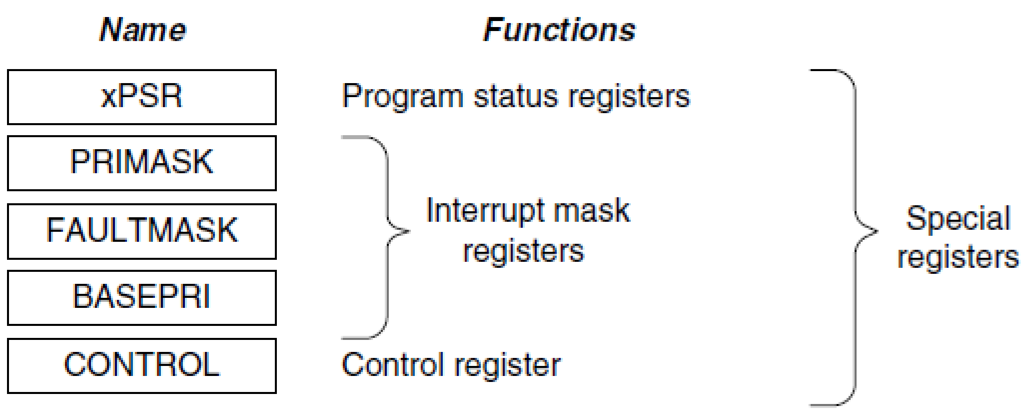


Table 2.1 Special Registers and Their Functions

Register	Function
xPSR	Provide arithmetic and logic processing flags (zero flag and carry flag), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and hard fault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

INTRODUCTION TO KEILµ VISION 4

PROCEDURE TO USE KEIL UVISION

Step1: open the **KEIL uvision4 window**.

Step2: click on **PROJECT** and select **NEW PROJECT**.

Step3: create new folder and give file name and select **SAVE**.

Step4: In the **SELECT DEVICE FOR TARGET "TARGET1'**window double click on **NXP**. Select the device as **LPC1768** and click **OK.**

Step5: A window is displayed asking whether to **add start up code**. click **NO**.

Step6: click on **FILE** and select **NEW**, type the program **IN EDIT WINDOW**.

Step7: Go to **FILE** and save the file with **.C** extension and another file with .s extension

Step8: After the file is saved, click on **TARGET1** in **PROJECT WORK SPACE**. right click on **source group1** and select **add files to source group1**.

Step9: select **type of file as c source file** and select the needed file then press **ADD & then click on close.**

Step10: Go to **PROJECT** and select **options for target "target1''**a dialog box is displayed assign the frequency as **12 MHZ** in **xtal.** in the same window select the output tab and select the **"CREAT HEX FILE"**option.then click **ok.**

Step11: Go to **PROJECT** and select **BULID TARGET** option. Before debugging check whether the project is built or not, that is whether there are any errors. if there is any error it should be eliminated before Debugging. if there is no error then only target is created.

Step12: Again in the **DEBUG menu** select **START/STOP debug session** in the disassembly window

Program Code will be displayed in the **register space**, Contents of registers are visible. Notedown the register content

Write an ALP to find the sum of first 10 integer numbers

.C FILE

```
#include <LPC17xx.H>
extern void sumten(void); //Name of assembly routine
int main(void)
{
SystemInit();
sumten(); //calling asm code
while(1);
}
```

.S FILE

```
AREA sum,CODE,READONLY
EXPORT sumten
ENTRY
sumten
MOV R1,#10 ; load 10 to regester
MOV R2,#0 ; empty the register to store result
loop
ADD R2,R2,R1 ; add the content of R1 with result at R2
SUBS R1,#0x01 ; Decreament R1 by 1
BNE loop ; repeat till r1 goes 0
BX LR ; jumps back to C code
END
```

OUTPUT:

R2:0X037

Write an ALP to multiply two 16 bit binary numbers

.C FILE

```
#include <LPC17xx.H>
extern void multiply(void); //Name of assembly routine
int main(void)
{
SystemInit();
multiply(); //calling asm code
while(1);
}
```

.S FILE

```
AREA sum,CODE,READONLY
EXPORT multiply
ENTRY
multiply
MOV R1,#03 ; load 10 to register
MOV R2,#02 ; empty the register to store result
loop
MUL R2,R2,R1 ; add the content of R1 with result at R2
BX LR ; jumps back to C code
END
```

Output:

R2=0x06

PART - A

Program 1: Write a program to multiply two 16 bit binary numbers.

```
        AREA BINMUL,CODE,READONLY
ENTRY
        MOV R1, #0X40000000     ; Move the address to the Register R1
        LDR R2, [R1]            ; Load the Data which is stored in Register R1 to the Register R2
        MOV R3, #0X40000004     ; Move the address to the Register R3
        LDR R4, [R3]            ; Load the Data which is stored in Register R3 to the Register R4
        MUL R5, R2, R4          ; Multiply the data which is stored in R2 and R4. Store it in R5
        MOV R6, #0X40000008     ; Move the address to the Register R6
        STR R5, [R6]            ; Store the resultant data to the address of Register R6
        NOP
        END
```

Input:

Enter the First 16 bit number onto 0x40000000 (Eg: AAAA)

Enter the Second 16 bit number onto 0x40000004 (Eg: BBBB)

Output:

Result can be viewed in Register R5 or on the memory location 0x40000008 (2ED8267D) in hex decimal values.

Program 2: Write a program to find the sum of first 10 integer numbers.

```
        AREA SUM_INTEGERS, CODE, READONLY
ENTRY
        MOV R1, #0X00000001     ; Move the first value to the register R0
        MOV R2, #0X0000000B     ; Move the last value to the register R1
        MOV R3, #0X00000000     ; Move Zero to the register R3
LOOP
        ADD R3, R3, R1          ; Add the content of R1 with R3 and store the result back to R3
        ADD R1, R1, #01         ; Add one to the data of R1 and store the result back to R1
        CMP R1, R2              ; Compare R1 and R2
        BNE LOOP                ; Repeat till R1 has the value 10
        MOV R4, R3              ; Move the resultant data to the register to R4
        MOV R5, #0X40000000     ; Move the address to the Register R5 to store the result
        STR R4, [R5]            ; Store the result to the address of register R5
        NOP
        END
```

Output:

Result can be viewed in Register R4 and on the memory location 0X40000000 (37) in hex decimal value.

Program 3. Write a program to find factorial of a number.

```
        AREA FACT,CODE,READONLY
ENTRY
        MOV R0,#0X00000005          ; Store factorial number in register R0
        MOV R1,R0                   ; Move the same number in R1
LOOP
        SUB R1, R1, #01             ; Subtract R1 with 1 and store the result back to R1
        CMP R1, #01                 ; Comparison
        BEQ STOP
        MUL R2, R0, R1              ; Multiply R0 and R1 and store the result to R2
        MOV R0, R2                  ; Move the result to the register R0
        MOV R3, #0X40000000         ; Move the address to the register R3 to store the result
        STR R0, [R3]                ; Store the result on the address of R3
        BNE LOOP
STOP
        NOP
        END
```

Output:

The result can be viewed in register R0 and on the memory location 0X40000000 (78) in hex decimal value.

Note: The factorial of 5 is 120 and hex decimal value is 78.

Program 4: Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM

```
        AREA ARRAY_SUM, CODE, READONLY
ENTRY
        MOV R1, #0X00000003     ; Initialize the counter to 3 (i.e N=4) and move that to R1
        MOV R3, #0X40000000     ; Move the first address to the register R3
        LDR R4, [R3]            ; Load the first 16 bit data to the register R4
LOOP
        ADD R3, R3, #4          ; Add 4 to the address in R3 to point out to the next address
        LDR R5, [R3]            ; Load data which is stored on the address of R3 to the register R5
        ADD R4, R5              ; Add R4 and R5 and store that in register R4
        SUB R1, R1, #01         ; Decrement the counter
        CMP R1, #00             ; Compare the counter with 0 to come out of the loop
        BNE LOOP
        MOV R6, #0X4000001C     ; Move the address to register to R6 to store the result
        STR R4, [R6]            ; Store the result to the address of R6
        NOP
        END
```

INPUT:

0X40000000 = 1111

0X40000004 = 2222

0X40000008 = 3333

0X4000000C = 4444

OUTPUT:

The result can be viewed in register R4 and on the memory location 0X4000001C (AAAA) in hex decimal value.

Program 5: Write a program to find the square of a number (1 to 10) using look-up table.

```
        AREA SQUARE, CODE, READONLY
ENTRY           ; Mark first instruction to execute
START
        LDR R0, = LOOKUP            ; Load start address of Lookup table
        LDR R1, =2                  ; Load no whose square is to be find
        MOV R1, R1, LSL #0x2        ; Generate address corresponding to square of given no
        MOV R4, #0x40000000         ;Move the address to the register R4 to store result
        ADD R0, R0, R1              ; Load address of element in Lookup table
        LDR R3, [R0]                ; Get square of given no in R3
        STR R3, [R4]                ; Store the result to the address of register R4
        NOP
        NOP
JMP B JMP
;Lookup table contains Squares of nos from 0 to 10 (in hex)
LOOKUP          DCD 0X00000000      ;SQUARE OF 0=0
                DCD 0X00000001      ;SQUARE OF 1=1
                DCD 0X00000004      ;SQUARE OF 2=4
                DCD 0X00000009      ;SQUARE OF 3=9
                DCD 0X00000010      ;SQUARE OF 4=16
                DCD 0X00000019      ;SQUARE OF 5=25
                DCD 0X00000024      ;SQUARE OF 6=36
                DCD 0X00000031      ;SQUARE OF 7=49
                DCD 0X00000040      ;SQUARE OF 8=64
                DCD 0X00000051      ;SQUARE OF 9=81
                DCD 0X00000064      ;SQUARE OF 10=100
        END             ; Mark end of file
```

OUTPUT:

The result can be viewed in register R3 and on the memory location 0X40000000 in hex decimal value. (eg: In this program 2 is loaded onto the register R1, so the output at 0X40000000 memory location is 4)

Program 6: Write a program to find the largest/smallest number in an array of 32 bit numbers.

A. To find Largest number in an array

```
        AREA LARGEST, CODE, READONLY
ENTRY
        MOV R1, #0x00000004     ; Initialize the counter to 4 (i.e N=5)
        MOV R2, #0x40000000     ; Move the first address to the register to R2
        MOV R3, #0x4000001C     ; Move the resultant address to register R3
        LDR R4, [R2]            ; Load the first 32 bit data to the register R4
LOOP
        ADD R2, R2, #04     ;Increment the address to 4 to point to the next address
        LDR R5, [R2]        ; Store the next data to the register R5
        CMP R4, R5          ; Compare both numbers which is stored in register R4 and R5
        BHI LOOP1           ; IF THE 1st NUMBER IS > THEN GOTO LOOP1
        MOV R4, R5          ; IF THE 1st NUMBER IS < THEN MOVE CONTENT R4 TO R2
LOOP1
        SUB R1, R1, #01     ; Decrement the counter
        CMP R1, #00         ; Compare counter to 0 to exit from LOOP
        BNE LOOP
        STR R4, [R3]        ; Store the resultant to the address of register R3
        NOP
        NOP
        NOP
        END
```

INPUT:

0X40000000 = 11111111

0X40000004 = 22222222

0X40000008 = 33333333

0X4000000C = 44444444

0X40000010 = 55555555

OUTPUT:

The result can be viewed in register R4 and on the memory location 0X4000001C (55555555) in hex decimal value.

B. To find Smallest number in an array

```
        AREA SMALLEST,CODE,READONLY
ENTRY
        MOV R1, #0x00000004; Initialize the counter to 4 (i.e N=5)
        MOV R2,#0x40000000; Move the first address to the register to R2
        MOV R3,#0x4000001C; Move the resultant address to register R3
        LDR R4,[R2]; Load the first 32 bit data to the register R4
LOOP
        ADD R2,R2,#04 ; Increment the address to 4 to point to the next address
        LDR R5,[R2] ; Store the next data to the register R5
        CMP R4,R5 ; Compare both numbers which is stored in register R4 and R5
        BLS LOOP1 ; IF THE 1st NUMBER IS < THEN GOTO LOOP1
        MOV R4,R5 ; IF THE 1st NUMBER IS > THEN MOVE CONTENT R4 TO R2
LOOP1
        SUB R1,R1,#01 ; Decrement the counter
        CMP R1,#00 ; Compare counter to 0 to exit from LOOP
        BNE LOOP
        STR R4,[R3] ; Store the resultant to the address of register R3
        NOP
        NOP
        NOP
        END
```

INPUT:

0X40000000 = 11111111
0X40000004 = 22222222
0X40000008 = 33333333
0X4000000C = 44444444
0X40000010 = 55555555

OUTPUT:

The result can be viewed in register R4 and on the memory location 0X4000001C (11111111) in hex decimal value.

Program 7: Write a program to arrange a series of 32 bit numbers in ascending/descending order.

A. Arrange a series of 32 bit numbers in Ascending order

```
        AREA ASCENDING, CODE, READONLY
ENTRY
        MOV R0, #0x00000004     ; Initialize the counter to 4 (i.e N=5)
LOOP2
        MOV R1, #04             ; Initialize another counter to 4 (i.e N=5)
        MOV R2, #0x40000000     ; Move the first address to the register to R2
LOOP1
        LDR R3, [R2]        ; Load the first 32 bit data to the register R3
        ADD R2, R2, #04     ; Increment the address to 4 to point to the next address
        LDR R4, [R2]        ; Store the next data to the register R4
        CMP R3, R4          ; Compare two values
        BLT LOOP            ; IF THE 1st NUMBER IS < THEN GOTO LOOP2
        STR R3, [R2]        ; Interchange the numbers stored in register R4 and R3
        SUB R2, R2, #04     ; Decrement the address with 4 to point out to the previous value
        STR R4, [R2]        ; Interchange the numbers stored in register R4 and R3
        ADD R2, R2, #04     ; After interchange then add 4 to the register R4 to point to the next Data
LOOP
        SUB R1, R1, #01     ; Decrement the first counter
        CMP R1,#00          ;Compare the counter with 0 to exit LOOP1
        BNE LOOP1
        SUB R0,R0,#01       ; Decrement the second counter
        CMP R0,#00          ; Compare the counter with 0 to exit LOOP2
        BNE LOOP2
        NOP
        END
```

INPUT:

0X40000000 = 22222222

0X40000004 = 11111111

0X40000008 = 55555555

0X4000000C = 44444444

0X40000010 = 33333333

OUTPUT:

The result can be viewed in the five different memory location starting from 0X40000000 to 0x40000010

0X40000000 = 11111111

0X40000004 = 22222222

0X40000008 = 33333333

0X4000000C = 44444444

0X40000010 = 55555555

B. Arrange a series of 32 bit numbers in Descending order

```
        AREA DESCENDING,CODE,READONLY
ENTRY
        MOV R0, #0x00000004     ; Initialize the counter to 4 (i.e N=5)
LOOP2
        MOV R1, #04             ; Initialize another counter to 4 (i.e N=5)
        MOV R2, #0x40000000     ; Move the first address to the register to R2
LOOP1
        LDR R3, [R2]        ; Load the first 32 bit data to the register R3
        ADD R2, R2, #04     ; Increment the address to 4 to point to the next address
        LDR R4, [R2]        ; Store the next data to the register R4
        CMP R3, R4          ; Compare two values
        BGT LOOP            ; IF THE 1st NUMBER IS > THEN GOTO LOOP2
        STR R3, [R2]        ; Interchange the numbers stored in register R4 and R3
        SUB R2, R2, #04     ; Decrement the address with 4 to point out to the previous value
        STR R4, [R2]        ; Interchange the numbers stored in register R4 and R3
        ADD R2, R2, #04     ; After interchange then add 4 to the register R4 to point to the next Data
LOOP
        SUB R1, R1, #01     ; Decrement the first counter
        CMP R1, #00         ;Compare the counter with 0 to exit LOOP1
        BNE LOOP1
        SUB R0, R0, #01     ; Decrement the second counter
        CMP R0, #00         ; Compare the counter with 0 to exit LOOP2
        BNE LOOP2
        NOP
        END
```

INPUT:

0X40000000 = 22222222

0X40000004 = 11111111

0X40000008 = 55555555

0X4000000C = 44444444

0X40000010 = 33333333

OUTPUT:

The result can be viewed in the five different memory location starting from 0X40000000 to 0x40000010

0X40000000 = 55555555

0X40000004 = 44444444

0X40000008 = 33333333

0X4000000C = 22222222

0X40000010 = 11111111

Program 8: Write a program to count the number of ones and zeros in two consecutive memory locations.

```
        AREA ONEZERO , CODE, READONLY
ENTRY           ; Mark first instruction to execute
START
        MOV R2, #0          ; COUNTER FOR ONES
        MOV R3, #0          ; COUNTER FOR ZEROS
        MOV R7, #2          ; COUNTER TO GET TWO WORDS
        LDR R6, =VALUE      ; LOADS THE ADDRESS OF VALUE
LOOP MOV R1, #32            ; 32 BITS COUNTER
        LDR R0, [R6], #4    ; GET THE 32 BIT VALUE
LOOP0 MOVS R0, R0, ROR #1       ; RIGHT SHIFT TO CHECK CARRY BIT (1's/0's)
        BHI ONES            ; IF CARRY BIT IS 1 GOTO ONES BRANCH OTHERWISE NEXT
ZEROS ADD R3, R3, #1        ; IF CARRY BIT IS 0 THEN INCREMENT THE COUNTER BY 1(R3)
        B LOOP1             ; BRANCH TO LOOP1
ONES ADD R2, R2, #1         ; IF CARRY BIT IS 1 THEN INCREMENT THE COUNTER BY 1(R2)
LOOP1 SUBS R1, R1, #1       ; COUNTER VALUE DECREMENTED BY 1
        BNE LOOP0           ; IF NOT EQUAL GOTO TO LOOP0 CHECKS 32BIT
        SUBS R7, R7, #1     ; COUNTER VALUE DECREMENTED BY 1
        CMP R7, #0          ; COMPARE COUNTER R7 TO 0
        BNE LOOP            ; IF NOT EQUAL GOTO TO LOOP
        NOP
        NOP
        NOP
JMP B JMP
VALUE DCD 0X11111111, 0XAA55AA55        ; TWO VALUES IN AN ARRAY
END                                     ; Mark end of file
```

Output:

Result can be viewed in hex decimal values on register R2 and R3.

- Number of 0’s in R3 is 0X00000028 (40 Zeros)
- Number of 1’s in R2 is 0X00000018 (24 Ones)

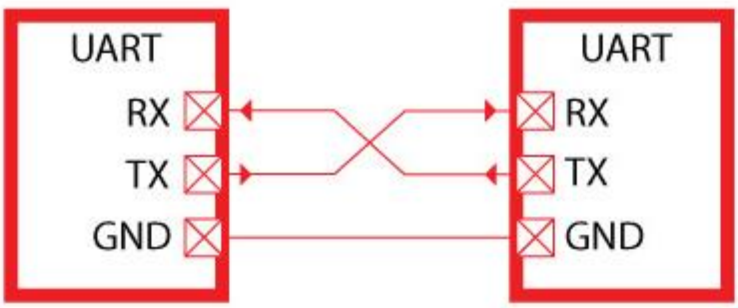
MCES LAB – PART B – UNRULED SIDE DIAGRAMS

(18CSL48)

✪ INSTRUCTIONS:

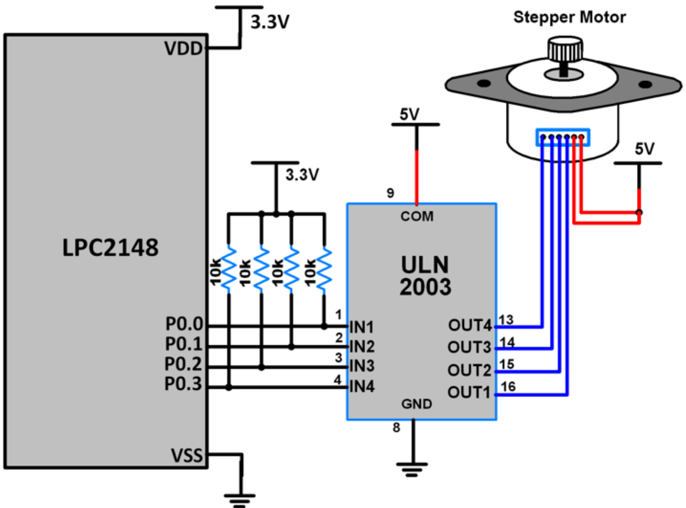
- Include all these diagrams neatly according to the respective programs.
- Write these diagrams on the unruled (left side) part of the record – beginning page of the program.
- For example, "UART" diagram should be included on the left side of the record, where the right side contains the beginning of the UART (Hello World) program.

✪ PROGRAM – 9: **Hello World – UART**

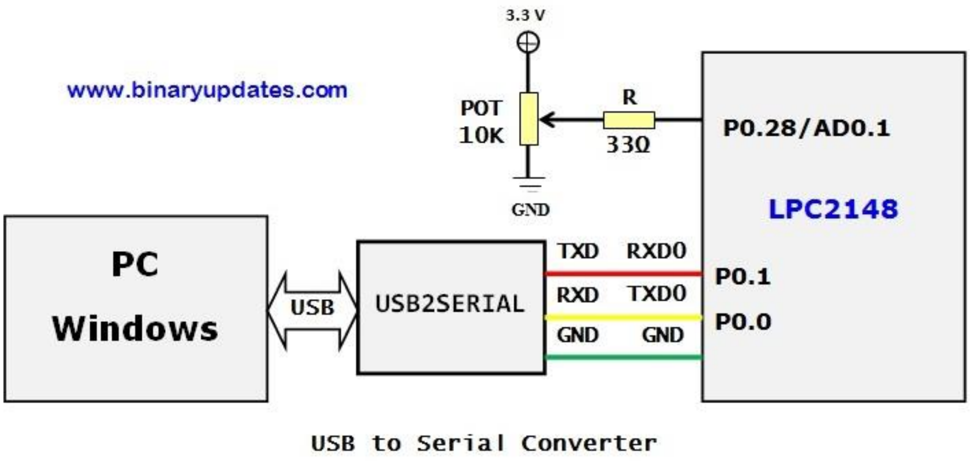


✪ PROGRAM – 10: **DC Motor – No diagram**

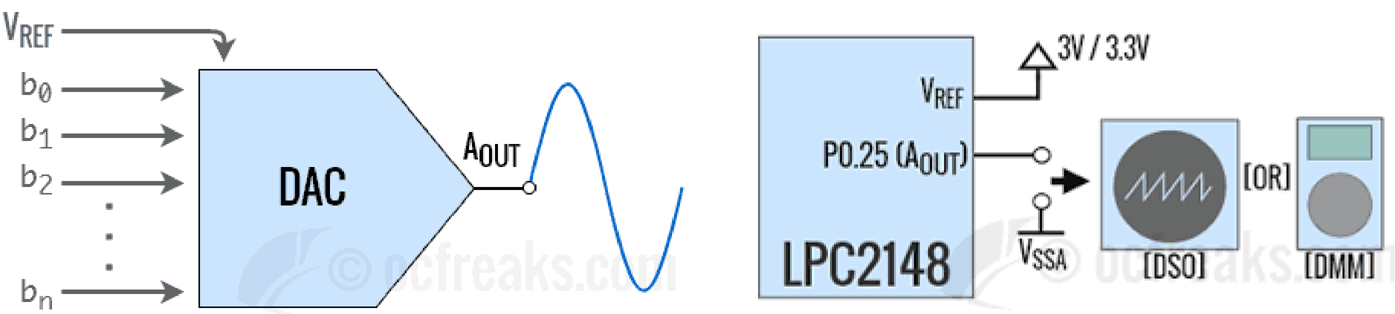
✪ PROGRAM – 11: **Stepper Motor**



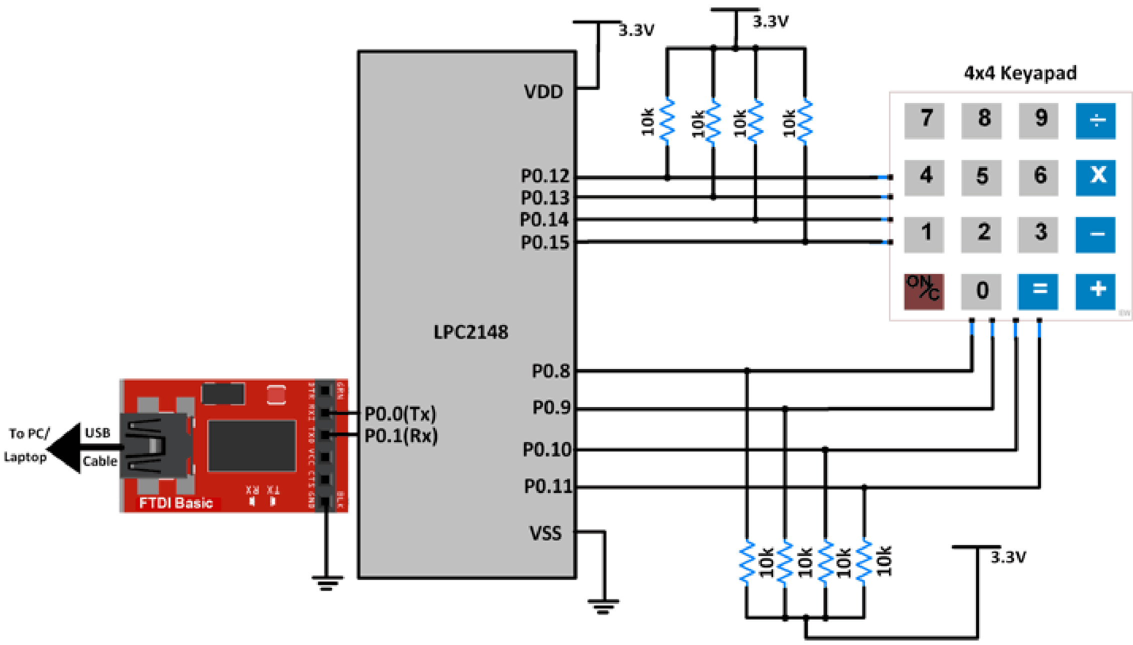
✪ PROGRAM – 12: **ADC Interfacing**



✪ PROGRAM – 13: **DAC – Triangle Wave**



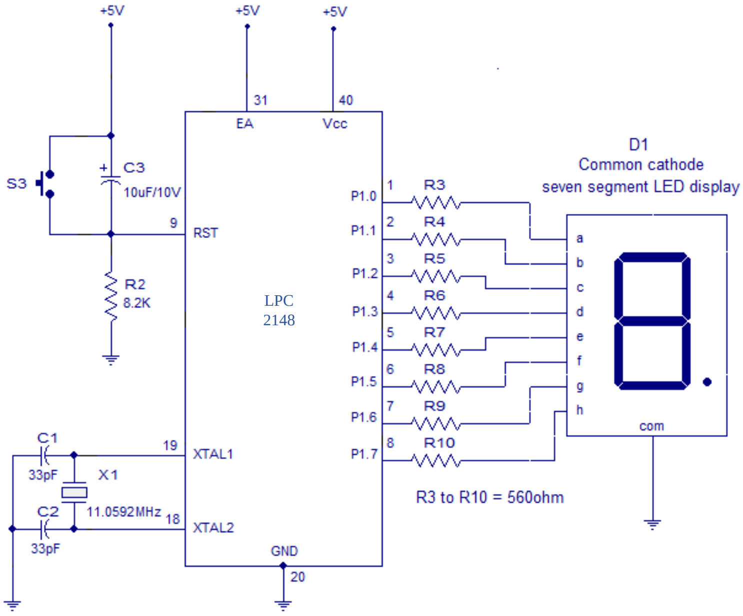
✪ PROGRAM – 14: **4x4 Keypad interfacing**



MCES
ASN

✪ PROGRAM – 15: **Ext. INT – No diagram**

✪ PROGRAM – 16: **Seven Segment Display interfacing**



PART - B

9. Display "Hello World" message using Internal UART.

//Serial.c

```
#include <LPC21xx.H>                    /* LPC21xx definitions              */
#include "Serial.h"
#define CR     0x0D
int sendchar (int ch)
{               /* Write character to Serial Port    */
  if (ch == '\n')  {
    while (!(U1LSR & 0x20));
    U1THR = CR;                        /* output CR */
  }
  while (!(U1LSR & 0x20));
  return (U1THR = ch);
}
int uart0_getkey (void)
{                   /* Read character from Serial Port   */
   while (!(U0LSR & 0x01));
   return (U0RBR);
}
void uart0_init()
{
  PINSEL0 = 0x00000005;          /* Enable RxD0 and TxD0                 */
  U0LCR = 0x83;                 /* 8 bits, no Parity, 1 Stop bit          */
  U0DLL = 97;                   /* 9600 Baud Rate @ 15MHz VPB Clock        */
  U0LCR = 0x03;                 /* DLAB = 0                               */
}
void uart0_putc(char c)
{
        while(!(U0LSR & 0x20)); // Wait until UART0 ready to send character
        U0THR = c; // Send character
}
void uart0_puts(char *p)
{
        while(*p) // Point to character
        {
                uart0_putc(*p++);  // Send character then point to next character
        }
}
```

```
//main.c

#include <LPC21xx.H>          /* LPC21xx definitions */
#include "Serial.h"
void delay_ms(int count)
{
 int j=0,i=0;

 for(j=0;j<count;j++)
 {
  for(i=0;i<35;i++);
 }
}
int main (void)
{
 uart0_init();                          // Initialize UART0
 delay_ms(100000);

 while (1)
  {
 uart0_puts ("\n\rHello World\n\r");
 delay_ms(1000000);
 }
}
```

10. Interface and Control a DC Motor.

```
#include <LPC214x.H>
void delay_led(unsigned long int); // Delay Time Function
int main(void)
{

IO1DIR = 0xC0000000;
IO0DIR = 0x00200000;
while(1) // Loop Continue
{
IO0SET = 0x00200000;
delay_led(15000);
IO1SET = 0x80000000;
IO1CLR = 0x40000000;      // Clear Pin P0.7,6,5,4 (ON LED)
delay_led(1500000);       // Display LED Delay
IO1SET = 0x40000000;
IO1CLR = 0x80000000;      // Set Pin P0.7,6,5,4 (OFF LED)
delay_led(1500000);       // Display LED Delay
}
}

/***********************/
/* Delay Time Function */
/***********************/
void delay_led(unsigned long int count1)
{
while(count1 > 0) {count1--;} // Loop Decrease Counter
}
```

11. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.

```
#include <lpc214x.h>
void  delay();

void delay()
{
 int i,j;
 for(i=0;i<0xff;i++)
    for(j=0;j<0x25;j++);
}

void main()
{
 int num=0x08,val=0x00;
 char rotate=0;
 IO0DIR=0x000F0000;
 while(1)
 {
  if(rotate==1)
  {
   IO0CLR=val;
   val=(num<<16);
   num=num*2;
   IO0SET=val;
   if(num>8)
     num=1;
   delay();
  }
  else
  {
   IO0CLR=val;
   val=(num<<16);
   num=num/2;
   IO0SET=val;
   if(num<1)
     num=0x08;
   delay();
  }
```

```
            if(!(IO0PIN & 0x00008000))
            {
              while(!(IO0PIN & 0x00008000));

              rotate=1;
            }
            else if(!(IO0PIN & 0x00100000))
            {
              while(!(IO0PIN & 0x00100000));

              rotate=0;
            }
           }
}
```

12. Determine Digital output for a given Analog input using Internal ADC of ARM controller.

```
#include<LPC214X.H>
/*-----------------------------------------------------
MACRO FOR ADC
----------------------------------------------------- */
   #define ch (1 << 3)
  #define clk_div (3 << 8)
#define bst_on (1 << 16)
//#define bst_off (0 << 16)
#define clk_res (0 << 17)
#define operational (1 << 21)
#define start (0 << 24)
#define adc_init_macro ch | clk_div | bst_on | clk_res | operational | start
/*----------------------------------------------------------
MACRO FOR LCD
----------------------------------------------------------*/
#define EN (1 << 28)
#define RW (1 << 29)
#define RS (1 << 22)
#define DATA  (0Xff << 6)
#define port  EN | RW | RS | DATA

/*-------------------------------------------------------------------
FUNCTION DECLARATIONS
-------------------------------------------------------------------*/
        void adc_init(void);
        void delay(int count);
        void cmd(int c);
        void data(char d);
        void lcd_string(char *str);
        void display(unsigned int n);

/*------------------------------------------------------------------------
        GLOBAL VARIABLES
------------------------------------------------------------------------*/
        unsigned int result;
    float voltage;
        char volt[18];
```

```
/*-------------------------------------------------------------------------------
                FUNCTION DEFINITIONS
-------------------------------------------------------------------------------*/
                void adc_init(void)
                {
                        AD0CR = adc_init_macro;
                }
                void cmd(int c)
                {
                        IOPIN0 = c << 6;
                        IOCLR0 = RW;
                        IOCLR0 = RS;
                        IOSET0 = EN;
                        delay(100);
                        IOCLR0 = EN;
                }
                void data(char d)
                {
                        IOPIN0 = d << 6;
                        IOCLR0 = RW;
                        IOSET0 = RS;
                        IOSET0 = EN;
                        delay(100);
                        IOCLR0 = EN;
                }

                void lcd_string(char *str)
                {
                        while(*str)
                        {
                                data(*str);
                                str++;
                                delay(20);
                        }
                }
                void display(unsigned int n)
                {
                        if(n == 0)
                                data(n+0x30);
                        if(n)
                        {
                                display(n / 10);
```

```
                data((n % 10) + 0x30);
            }
        }

        void delay(int count)
        {
            int i,j;
            for(i = 0;i < count;i++)
                for(j = 0;j < 5000;j++);
        }
/*----------------------------------------------------------------
MAIN
-----------------------------------------------------------------*/
            int main()
            {
                int c = 0;
                IODIR0 |= port ;
                PINSEL1|=0x10000000;
                cmd(0x38);
                cmd(0x0E);
                cmd(0X80);
                cmd(0X01);
                adc_init();
                lcd_string("ADC PROGRAM");
                cmd(0X01);
                while(1)
                {
                    cmd(0x01);
                    while((AD0DR3 & (0x80000000)==0));
                    result = (AD0DR3 & (0X3FF << 6));
                    result = result >> 6;
                    lcd_string("ADC:");
                    cmd(0x86);
                    display(result);
                    voltage = ( (result/1023.0) * 3.3 );
                    cmd(0xc0);
                    sprintf(volt, "Voltage=%.2f V  ", voltage);
                    lcd_string(volt);
                    //delay(1000);
                }

            }
```

13. Interface a DAC and generate Triangular and Square waveforms.

```
//Triangle.c
#include "LPC214X.h"

unsigned int  value;

int main()
{

 PINSEL1|=0x00080000;


 while(1)
 {
   value = 0;
			while ( value != 1023 )
			{
				DACR = ( (1<<16) | (value<<6) );
				value++;
			}
			while ( value != 0 )
			{
				DACR = ( (1<<16) | (value<<6) );
				value--;
			}
 }

}
```

```
//Square.c

#include "LPC214X.h"

unsigned int  result=0x00000040,val;

int main()
{

 PINSEL1|=0x00080000;


 while(1)
 {
   while(1)
   {

     val =0xFFFFFFFF;
     DACR=val;

     {
      break;
     }
   }
   while(1)
   {

     val =0x00000000;
     DACR=val;

     {
     break;
     }
   }
 }

}
```

14. Interface a 4x4 keyboard and display the key code on an LCD.

```
//lcd.c
#include <LPC214x.H>                    /* LPC214x definitions */
#include "lcd.h"
void lcd_command_write(unsigned char command);
void lcd_data_write(unsigned char data);

#define LCD_DATA_DIR      IO0DIR
#define LCD_DATA_SET      IO0SET
#define LCD_DATA_CLR      IO0CLR

#define LCD_CTRL_DIR      IO0DIR
#define LCD_CTRL_SET      IO0SET
#define LCD_CTRL_CLR      IO0CLR

#define LCDEN             (1 << 2)
#define LCDRS             (1 << 3)

//scale
31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,09,08,07,06,05,04,03,02,
01,00
#define LCD_DATA_MASK         0x007F8000

/*************************************************************************
*
 Function Name : delay()
 Description :
 Input :
 Output : void
*************************************************************************
*****/
void delay(unsigned int count)
{
 int j=0,i=0;

 for(j=0;j<count;j++)
 {
  for(i=0;i<120;i++);
 }
}
```

```
/*****************************************************************************
*****************
 Function Name :     lcd_command_write()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void lcd_command_write( unsigned char command )

{
int a=0;
a = command | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET =(IO0SET | 0x00003c00)&a;
delay(1000);
IO0CLR = 0x10000000;

a=0x0;
command=command<<4;
a = command | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET = (IO0SET | 0x00003C00)&a;
delay(1000);
IO0CLR = 0x10000000;
}

/*****************************************************************************
*******************
 Function Name :     lcd_data_write()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void lcd_data_write( unsigned char data )
```

```
{
int b=0;
b = data|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00)&b;
delay(1000);
IO0CLR = 0x10000000;

b=0x0;
data=data<<4;
b=data|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00)&b;
delay(1000);
IO0CLR = 0x10000000;
}

/*******************************************************************************************
 Function Name :     lcd_clear()
 Description   :
 Input         :
 Output        :Void
*******************************************************************************************/
void lcd_clear( void)
{
 lcd_command_write( 0x01 );
}

/*******************************************************************************************
 Function Name :     lcd_gotoxy()
 Description   :
 Input         :
 Output        :Void
```

```
*****************************************************************************
******************/
int lcd_gotoxy( unsigned char x, unsigned char y)
{
 unsigned char retval = TRUE;

 if( (x > 1) && (y > 15) )
  {
   retval = FALSE;
  }
 else
  {
   if( x == 0 ) lcd_command_write( 0x80 + y );
        else if( x==1 ) lcd_command_write( 0xC0 + y );
  }
  return retval;
}

/*****************************************************************************
*******************
 Function Name :     lcd_putchar()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void lcd_putchar( unsigned char c )
{
 lcd_data_write( c );
}

/*****************************************************************************
*******************
 Function Name :     lcd_putstring()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void lcd_putstring( char *string )
{
  while(*string != '\0')
```

```
  {
   lcd_putchar( *string );
   string++;
  }
}

/*****************************************************************************
*******************
 Function Name :     lcd_putstring16()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void lcd_putstring16( unsigned char line, char *string )
{
 unsigned char len = 16;

 lcd_gotoxy( line, 0 );
 while(*string != '\0' && len--)
 {
   lcd_putchar( *string );
   string++;
 }
}

/*****************************************************************************
*******************
 Function Name :     init_lcd()
 Description   :
 Input         :
 Output        :Void
*****************************************************************************
******************/
void init_lcd( void )
{
 IO0DIR=0x30403C00;
delay(100);

lcd_command_write(0x02);                                          //cursor
home command
```

```
        lcd_command_write(0x01);                                          //clear
        display command
        lcd_command_write(0x28);                                          //4-bit
        mode entry command(0x38 for 8 bit mode)
        lcd_command_write(0x06);                                          //entry
        mode command
        lcd_command_write(0x0C);                                          //display
        on cursor off command
        //cmd(0xC0);
         delay(1000);
}

//main.c

        #include <LPC214x.H>                    /* LPC214x definitions */
        #include "lcd.h"

        /////////////////////////////////////////////
        // Matrix Keypad Scanning Routine
        //
        // COL1 COL2 COL3 COL4
        // 0    1    2    3    ROW 1
        // 4    5    6    7    ROW 2
        // 8    9    A    B    ROW 3
        // C    D    E    F    ROW 4
        /////////////////////////////////////////////

        #define SEG7_CTRL_DIR      IO0DIR
        #define SEG7_CTRL_SET      IO0SET
        #define SEG7_CTRL_CLR      IO0CLR

        #define COL1               (1 << 16)
        #define COL2               (1 << 17)
        #define COL3               (1 << 18)
        #define COL4               (1 << 19)

        #define ROW1               (1 << 20)
        #define ROW2               (1 << 21)
        #define ROW3               (1 << 22)
        #define ROW4               (1 << 23)

        #define COLMASK                   (COL1 | COL2 | COL3 | COL4)
```

```
#define ROWMASK                    (ROW1 | ROW2 | ROW3 | ROW4)

#define KEY_CTRL_DIR      IO1DIR
#define KEY_CTRL_SET      IO1SET
#define KEY_CTRL_CLR      IO1CLR
#define KEY_CTRL_PIN      IO1PIN

/////////////// COLUMN WRITE /////////////////////
void col_write( unsigned char data )
{
  unsigned int temp=0;

  temp=(data << 16) & COLMASK;

  KEY_CTRL_CLR |= COLMASK;
  KEY_CTRL_SET |= temp;
}


////////////////////////////////// MAIN  ///////////////////////////////////////
int main (void)
{
unsigned char key, i;
unsigned char rval[] = {0x7,0xB,0xD,0xE,0x0};
unsigned char keyPadMatrix[] =
{
    '4','8','B','F',
    '3','7','A','E',
    '2','6','0','D',
    '1','5','9','C'
};

  init_lcd();

  KEY_CTRL_DIR |= COLMASK;           //Set COLs as Outputs
  KEY_CTRL_DIR &= ~(ROWMASK); //  Set ROW lines as Inputs

  lcd_putstring16(0,"Press HEX Keys..");
  lcd_putstring16(1,"Key Pressed =   ");

while (1)
 {
```

```
        key = 0;
        for( i = 0; i < 4; i++ )
        {
          // turn on COL output one by one
                  col_write(rval[i]);

          // read rows - break when key press detected
          if (!(KEY_CTRL_PIN & ROW1))
            break;

          key++;
          if (!(KEY_CTRL_PIN & ROW2))
            break;

          key++;
          if (!(KEY_CTRL_PIN & ROW3))
            break;

          key++;
                  if (!(KEY_CTRL_PIN & ROW4))
            break;

          key++;
        }

            if (key == 0x10)
                  lcd_putstring16(1,"Key Pressed =   ");
            else
                  {
                        lcd_gotoxy(1,14);
                        lcd_putchar(keyPadMatrix[key]);
                  }
      }

}
```

15. Demonstrate the use of an external interrupt to toggle an LED On/Off.

```
#include <LPC214x.H>
int i;
void init_ext_interrupt(void);
__irq void Ext_ISR(void);
int main (void)
{         init_ext_interrupt();   // initialize the external interrupt
  while (1)
  {
          }
}
void init_ext_interrupt()  // Initialize Interrupt
{
  EXTMODE = 0x4;             //Edge sensitive mode on EINT2
  EXTPOLAR &= ~(0x4);    //Falling Edge Sensitive
  PINSEL0 = 0x80000000; //Select Pin function P0.15 as EINT2
  /* initialize the interrupt vector */
  VICIntSelect &= ~ (1<<16);          // EINT2 selected as IRQ 16
  VICVectAddr5 = (unsigned int)Ext_ISR; // address of the ISR
  VICVectCntl5 = (1<<5) | 16;                              //
  VICIntEnable = (1<<16);             // EINT2 interrupt enabled
  EXTINT &= (0x4);
}
__irq void Ext_ISR(void) // Interrupt Service Routine-ISR
{
          IO1DIR |= (1<<16);
          IO1SET |= (1<<16);   // Turn OFF Buzzer
          for(i=0; i<2000000;i++);
          IO1CLR |= (1<<16);  // Turn ON Buzzer
          EXTINT |= 0x4;                   //clear interrupt
          VICVectAddr = 0;     // End of interrupt execution
}
```

16. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.

```
#include <LPC214x.H>
void delay_led(unsigned long int);
int main(void)
{
IO0DIR = 0x000007FC;
while(1)
{
IO0CLR = 0x00000FFF;
IO0SET = 0x00000604;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x000007E4;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000648;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000618;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000730;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000690;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000680;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x0000063C;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000600;
delay_led(150000);
IO0CLR = 0x00000FFF;
IO0SET = 0x00000630;
delay_led(150000);
IO0CLR = 0x00000FFF;
```

```
        IO0SET = 0x00000620;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        IO0SET = 0x00000780;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        IO0SET = 0x000006C4;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        IO0SET = 0x00000708;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        IO0SET = 0x000006C0;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        IO0SET = 0x000006E0;
        delay_led(150000);
        IO0CLR = 0x00000FFF;
        }
        }
        void delay_led(unsigned long int count1)
        {
        while(count1 > 0) {count1--;}
}
```