MULTI CORE ARCHITECTURE LAB
VI Sem, CE

20CEL67A

2022-23

MANUAL

FOR

MULTI CORE ARCHITECTURE LAB **(20CEL67A)**

NEW HORIZON COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

OUTER RING ROAD, NEAR MARATHAHALLI

BANGALORE

SYLLABUS

MULTI CORE ARCHITECTURE LAB

Course Code : 20CEL67A **Credits: 02**
L: T: P: S : 0: 0: 2: 0 **CIE Marks: 25**
Exam Hours: : 03 **SEE Marks: 25**

Course Outcomes:	**At the end of the Course, the Student will be able to**
CO#	COURSE OUTCOME
20CEL67A.1	Understand the instruction set of 32- bit microcontroller ARM Cortex M3 and the software tool required for programming in assembly and C language.
20CEL67A.2	Develop assembly language programs for different problem statements .
20CEL67A.3	Develop C language programs for different applications.
20CEL67A.4	Perform floating-point operations, Interface external hardware with ARM Cortex M3.

Mapping of Course Outcomes to Program Outcomes

CO#	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
20CEL67A.1	3	3	3	1	-	-	2	-	-	2	-	2	3	2
20CEL67A.2	3	3	3	1	-	-	2	-	2	2	-	2	3	2
20CEL67A.3	3	3	3	1	-	-	2	-	2	2	-	2	3	2
20CEL67A.4	3	3	3	1	-	-	2	-	2	2	-	2	3	2

Correlation levels: 1-Slight(Low) 2-Moderate(Medium) 3-Substantial(High)

S.NO	**LIST OF EXPERIMENTS**	**Hours**	**COS**	**RBT Levels**
1	Program to sort a given array of N elements is ascending / descending order using bubble sort	**4**	CO1	**L5**
2	Program to perform addition, multiplication and division operations	**4**	CO2	**L5**
3	Program to generate Fibonacci series of N numbers	**4**	CO1	**L5**
4	Program to compute factorial and n C r using recursion	**4**	CO1	**L5**
5	Program to find square and cube of a floating-point number	**4**	CO2	**L5**
6	Program to perform floating point addition and Subtraction	**4**	CO3	**L5**

7	Program to display a message using Internal UART	**4**	CO4	**L4**
8	Program to Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction	**4**	CO4	**L4**
9	Program to Interface a DAC and generate Sinusoidal and Triangular waveforms	**4**	CO3	**L4**
10	Program to display the given message on a 7-segment LED interface, with an appropriate delay in between	**4**	CO3	**L4**
11	Program to Interface a 4x4 keyboard and display the key pressed on an LCD	**4**	CO3	**L4**

CIE – Continuous Internal Evaluation: Lab (25 Marks)

Blooms Taxonomy	Tests	Conduction	Viva	Observation	Record
Marks (Out of 25)	**25**	10	5	5	5
L1: Remember	-	-	-	-	5
L2: Understand	5	5	-	-	-
L3: Apply	15	5	-	5	-
L4: Analyze	5	-	5	-	-
L5: Evaluate	-	-	-	-	-
L6: Create	-	-	-	-	-

SEE – Semester End Examination: Lab (25 Marks)

Blooms Taxonomy	Tests	Write-up	Conduction	Viva
L1: Remember	-	-	5	-
L2: Understand	5	5	-	-
L3: Apply	15	5	5	-
L4: Analyze	5	-	-	5
L5: Evaluate	-	-	-	-
L6: Create	-	-	-	-

Prepared By	**Approved By**
Dr.T.KAVITHA	**Dr.S.P.MANIKANDAN**
Lab Incharge	**HoD & Prof. CE Dept**

RUBRICS

CIE- Continuous Internal Evaluation: LAB (25 Marks)

Divided into two components:

a. **Continuous Assessment : 10 marks**

b. **Internal Test : 15 marks**

1. **Continuous Assessment:**
   i) Will be carried out in every lab (12 Experiments)
   ii) Each program will be evaluated for 10 marks
      - Write up : 3 Marks
      - Observation : 3 Marks
      - Record writing: 4 Marks
   iii) Totally for 12 Experiments it will be 120 marks. This will be scaled down to 10.

Break up of 10 marks (in every lab): Will be carried out in every lab (for 12 Experiments)

Attributes	Descriptors	Scores
Program Write-up(3)	Complete program with proper variable naming, proper commenting / Complete the experiment with proper connection	3
	Complete program with not so proper variable naming, poor commenting / Complete the experiment without proper connection	2
	Incomplete code / Incomplete circuit connection	1
	Not written	0
Execution & Results (3)	Passes all specified test cases efficiently	3
	Fails in some test cases	2
	Incomplete execution	1
Record Writing (4)	Complete program with proper variable naming, proper commenting / Complete the experiment with proper connection	4
	Complete program with not so proper variable naming, poor commenting / Complete the experiment without proper connection	3
	Incomplete code / Incomplete circuit connection	2
	Not written	1

2. Internal Test:

i. During the semester, Two internal tests will be conducted for 15 marks each. Then average of each internal test marks is taken.

ii. The 1st lab internal will comprise of the first 6 experiments and the 2nd lab internal will comprise of the next 5 experiments.

Break up of 15 marks (for each of the 2 internal tests):

Attributes	Descriptors	Scores
Program Write-up(5)	Complete the program with proper variable naming, proper commenting / Complete the experiment with proper connection	5
	Complete program with not so proper variable naming, poor commenting / Complete the experiment without proper connection	3-4
	Incomplete code / Incomplete circuit connection	1-2
	Not written	0
Execution & Results (5)	Passes all specified test cases efficiently	5
	Fails in some test cases	3-4
	Incomplete execution	1-2
Viva Voce(5)	Answers 100% questions correctly	5
	Answers 75% questions correctly	3-4
	Answers satisfactorily	1-2
	Does not answer any question	0

SEE- Semester End Examination: LAB (25 Marks)

SEE Assessment Marks: 25

Session End Examination is conducted for 50 marks

Attributes	Descriptors	Scores
Program Write-up(20)	Complete the program with proper variable naming, proper commenting / Complete the experiment with proper connection	20
	Complete program with not so proper variable naming, poor commenting / Complete the experiment without proper connection	10 - 15
	Incomplete code / Incomplete circuit connection	01- 09
	Not written	0
Execution & Results (20)	Passes all specified test cases efficiently	20
	Fails in some test cases	10-14
	Incomplete execution	01-09
Viva Voce(10)	Answers 100% questions correctly	10
	Answers 75% questions correctly	06-09
	Answers 50% satisfactorily	01-05
	Does not answer any question	0

GENERALINSTRUCTIONS

1. Use a black/blue ballpoint pen for writing the record and observations.
2. Enter the page numbers on each page and date on the first page of each experiment
3. Fill the index page with dates and page numbers of the experiments.
4. Fill all the details on the certificate page including name, USN, semester,academic year, signature etc.
5. CIE 1: Experiments: 2 , 3, 4, 5, 6, 7
6. CIE 2: Experiments: 8, 9, 10, 11, 12
7. Observations and Record must be written as below:

Left Side	***Right Side***
Hardware Experiments	
Diagram if any	Aim
Program	Requirement
Sample Input & Output	Theory
	Procedure
	Algorithm
	Result

INDEX

S.no	Laboratory experiments	Signature
	PART A	
1	Study of ARM LPC2148	
2	ALP Program to perform Addition, Subtraction, Multiplication and Division	
3	ALP Program to generate Fibonacci series	
4	ALP Program to perform factorial and nCr	
5	ALP Program to perform soring of given N numbers in ascending / Descending	
6	ALP Program to perform square and cube of a number floating point number	
7	ALP Program to perform addition and subtraction of floating-point number	
	PART B	
8	Program to display a message using Internal UART	
9	Program to Interface a Stepper motor	
10	Program to Interface a DAC	
11	Program to Interface a 7-segment LED	
12	Program to Interface a 4x4 keyboard	

Open Ended Experiments

13	ALP program to find sum of first 10 integers	
14	ALP program to find the number of 0's and 1's in a 32 bit data	

1.Study of ARM LPC1768

Aim

To learn about the evolution, core features, general characteristics and applications of ARM processors.

Theory

The LPC1768/66/65/64 are ARM Cortex-M3 based microcontrollers for embedded applications featuring a high level of integration and low power consumption. The ARM Cortex-M3 is a next generation core that offers system enhancements such as enhanced debug features and a higher level of support block integration.

The LPC1768/66/65/64 operate at CPU frequencies of up to 100 MHz. The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals. The ARM Cortex-M3 CPU also includes an internal prefetch unit that supports speculative
branching.

The peripheral complement of the LPC1768/66/65/64 includes up to 512 kB of flash memory, up to 64 kB of data memory, Ethernet MAC, USB Device/Host/OTG interface, 8-channel general purpose DMA controller, 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface, 3 I2C-bus interfaces, 2-input plus 2-output I2S-bus interface, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, 4 general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock (RTC) with separate battery supply, and up to 70 general purpose I/O pins.

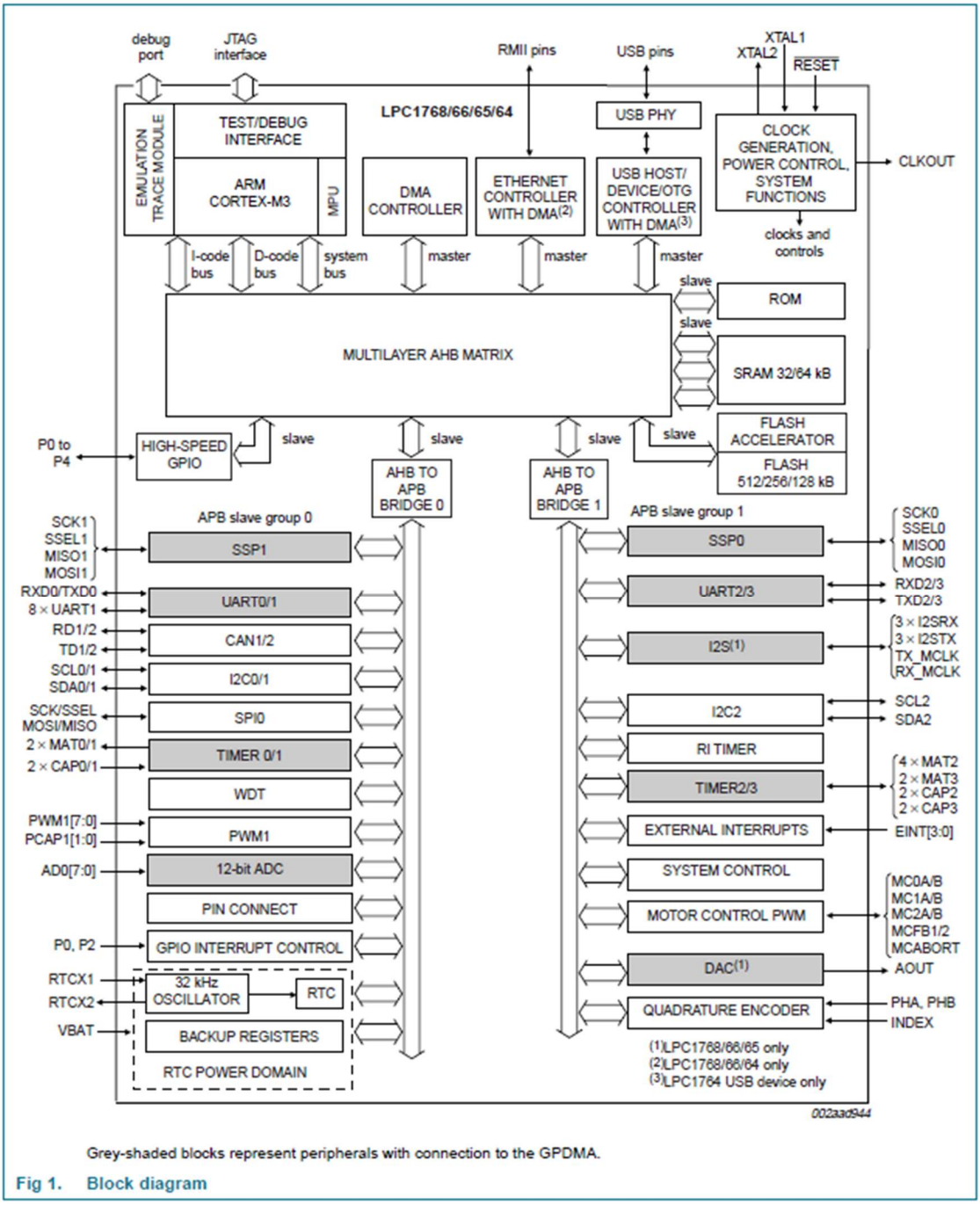
Features:

- ARM Cortex-M3 processor, running at frequencies of up to 100 MHz. A Memory Protection Unit (MPU) supporting eight regions is included.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash programming memory. Enhanced flash memory accelerator enables high-speed 100 MHz operation with zero wait states.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software.
- on-chip SRAM includes:

  u 32/16 kB of SRAM on the CPU with local code/data bus for high-performance CPU access.

  u Two/one 16 kB SRAM blocks with separate access paths for higher throughput. These SRAM blocks may be used for Ethernet (LPC1768/66/64 only), USB, and DMA memory, as well as for general purpose CPU instruction and data storage.

- Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with the SSP, I2S-bus, UART, the Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, and for memory-to-memory transfers.
- Multilayer AHB matrix interconnect provides a separate bus for each AHB master. AHB masters include the CPU, General Purpose DMA controller, Ethernet MAC (LPC1768/66/64 only), and the USB interface. This interconnect provides communication with no arbitration delays.
- Split APB bus allows high throughput with few stalls between the CPU and DMA.
- Serial interfaces:
  - Ethernet MAC with RMII interface and dedicated DMA controller (LPC1768/66/64 only).
  - USB 2.0 full-speed device/Host/OTG controller with dedicated DMA controller and on-chip PHY for device, Host, and OTG functions. The LPC1764 includes a device controller only.
  - Four UARTs with fractional baud rate generation, internal FIFO, DMA support, and RS-485 support. One UART has modem control I/O, and one UART has IrDA support.
    CAN 2.0B controller with two channels.
  - SPI controller with synchronous, serial, full duplex communication and programmable data length.
  - Two SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.
  - Two I2C-bus interfaces supporting fast mode with a data rate of 400 kbits/s with multiple address recognition and monitor mode.
  - One I2C-bus interface supporting full I2C-bus specification and fast mode plus with a data rate of 1 Mbit/s with multiple address recognition and monitor mode.
  - On the LPC1768/66/65 only, I2S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control. The I2S-bus interface can be used with the GPDMA. The I2S-bus interface supports 3-wire and 4-wire data transmit and receive as well as master clock input/output.



Grey-shaded blocks represent peripherals with connection to the GPDMA.

Fig 1. Block diagram

Result

The core features and general characteristics of ARM processors has been studied and is evaluated.

Experiment No : 2 **Date:**

Arithmetic operation of two - 32 bit numbers

Aim:

To write an assembly language program for performing addition, subtraction, Multiplication and division of two – 32 bit numbers.

Requirement:

a. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family ARM7 LPC1768 and click ok.
f. click "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Open the startup file and edit as:

   Line No 121: include "IMPORT __main"

   Line No 127 to 130: Comment by including ";" before the statement

   Line No 272 to 273: Comment by including ";" before the statement

i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Algorithm:

a. Start the program
b. Load the operand1 in a register (R1)
c. Load the operand2 in a register (R2)
d. Initialize R4 & R5 to zero.
e. Enter the option in a register (R3)

    1: Addition
    2: Subtraction
    3: Multiplication
    4: Division

f. Compare R3 with 1,
    i. If Zero flag is not set, jump to L2.
    ii. Else perform addition (R1 + R2) and store the result in a register R4.
    iii. Jump to L5.
g. L2: Compare R3 with 2,
    i. If Zero flag is not set, jump to L3.
    ii. Else perform subtraction (R1 - R2) and store the result in a register R4.
    iii. Jump to L5
h. L3: Compare R3 with 3,
    i. If Zero flag is not set, jump to L4.
    ii. Else perform multiplication (R1 * R2) and store the result in a register R4 & R5.
    iii. jump to L5.
i. L4: Compare R3 with 4,
    i. If Zero flag is not set, jump to L5.
    ii. Else perform division (R1 / R2) and store the result in a register R4.
    iii. Jump to L5
j. L5: Stop the program

Program:

```
AREA ARITH,CODE, READONLY
        ENTRY
        EXPORT __main
__main
        LDR R1, =0X10000000
        LDR R2,[R1]
        LDR R3,[R1,#4]
        LDR R4,[R1,#8]

        CMP R2,#01
        BNE CHK_SUB
        ADD R5,R3,R4
        B LAST

CHK_SUB CMP R2,#02
                BNE CHK_MUL
                SUB R5,R3,R4
                B LAST

CHK_MUL CMP R2,#03
                BNE CHK_DIV
                MUL R5,R3,R4
                B LAST

CHK_DIV CMP R2,#04
                BNE DEFAULT
                UDIV R5,R3,R4
                B LAST

DEFAULT MOV R5,#0

LAST        STR R5,[R1,#0XC]

                NOP
                END
```

Sample Input and Output:

Addition:

Subtraction:

Multiplication:

Division:

Result:

The assembly language program to perform addition, Subtraction, Multiplication and division is written and its output is verified using keil software.

Experiment No : 3 **Date:**

Fibonacci Series of N numbers

Aim:

To write an assembly language program to generate Fibonacci series with N number.

Requirement:

a. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family ARM7 LPC1768 and click ok.
f. click "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Open the startup file and edit as:

   Line No 121: include "IMPORT __main"

   Line No 127 to 130: Comment by including ";" before the statement

   Line No 272 to 273: Comment by including ";" before the statement

i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Algorithm:

a. Start the program
b. Load a address in hexadecimal (0x followed by 8 hexadecimal digit) in a register (R0) to store 1st element of Fibonacci series.
c. Move the number of elements of Fibonacci series to be generated to a register (R5)
d. Loop:
    i. Load the 1st element to a register (R1) from the address location (R0).
    ii. Load the 2nd element to a register (R2) from the address (R0) by incrementing by 4.
    iii. Generate the next element of Fibonacci Series by adding previous two element. (R1, R2).
    iv. Store the generated 3rd element in an address location by incrementing the address (R0) by 8.
    v. Update the address in R0 to point the next element by incrementing the address by 4.
    vi. Subtract the number of element to be generated (R5) by 1.
    vii. Check for Not equal, if true jump to Loop.
    viii. Else Stop the program

Note: *Force the first and second element value in the specified address location to '0'.*

Program:

```
   AREA FIBI, CODE,READONLY
   ENTRY
   EXPORT __main
__main

   LDR R0, =0x10000000
   MOV R5, #4                ; N = 4

L2  LDR R1, [R0]
   LDR R2, [R0,#4]
   ADD R3 , R1, R2
   STR R3, [R0, #8]

   ADD R0, R0, #4
   SUBS R5, #01
   BNE L2
```

```
NOP
END
```

Sample Input and Output:

Result:

The assembly language program to generate Fibonacci series with N numbers is written and its output is verified using keil software.

Experiment No : 4 **Date:**

Factorial and Combinations of a number

Aim:

To write an assembly language program to find factorial and combinations of a number.

Requirement:

b. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family ARM7 LPC1768 and click ok.
f. click "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Open the startup file and edit as:

   Line No 121: include "IMPORT __main"

   Line No 127 to 130: Comment by including ";" before the statement

   Line No 272 to 273: Comment by including ";" before the statement

i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Algorithm to find Factorial of a number:

a. Start the program
b. Load the number whose factorial is required in a register (R2)
c. Initialize the register (R1) with one.
d. Fact:
    a. Multiply the value of R2 with R1 and store the answer in the register (R1).
    b. Subtract the value of R2 from the number 1 and store back to R2.
    c. Check for condition "Not Equal", If it is yes go to the label "Fact".
e. Stop: No operation.
f. Stop the program.

Program:

```
AREA FACTORIAL,CODE,READONLY
        ENTRY
        EXPORT __main
__main

        MOV R1,#1
        MOV R2,#5

L1      MUL R1,R2
        SUBS R2,#1
        BNE L1
        NOP
        END
```

Sample Input and Output:

Algorithm to find the combination of a number :

a. Start the program

b. Initialize the register (R1) with one.
c. Load the n1 value in a register (R2)
d. Update the Link Register and jump to Fact.
e. Move factorial of n1 to a register (R7)

f. Initialize the register (R1) with one.
g. Load the n2 value in a register (R2)
h. Update the Link Register and jump to Fact.
i. Move factorial of n2 to a register (R8)

j. Initialize the register (R1) with one.
k. Load the n3 value in a register (R2)
l. Update the Link Register and jump to Fact.
m. Move factorial of n3 to a register (R9)

n. Multiply factorial of n2(R8) and factorial of n3 (R9), store the result in a register (R10).
o. Divide the factorial of n1 (R7) by the value of R10 and store the result in a register (R11).
p. Stop: No operation.

q. Fact:
    i. Push the return address available in Link register to stack
    ii. Compare the value of R2 with the number 1.
    iii. Check for “equal”, if yes jump to L1.
    iv. Multiply the value of R2 with R1 and store the answer in the register (R1).
    v. Subtract the value of R2 from the number 1 and store back to R2.
    vi. Update the Link Register and jump to Fact.
r. L1: Pop the top of the stack to PC
s. No operation.
s. Stop the program

Program:

```
AREA FACTORIAL,CODE, READONLY
        ENTRY
    EXPORT __main
__main

    MOV R1, #1
    MOV R2, #5
    BL FUN
    MOV R7, R1

    MOV R1, #1
    MOV R2, #3
    BL FUN
    MOV R8, R1

    MOV R1, #1
    MOV R2, #2
    BL FUN
    MOV R9, R1

    MUL R10,R8,R9
    UDIV R11,R7,R10
    STOP


FUN     PUSH {LR}
    CMP R2,#1
    BEQ L1
    MUL R1,R1,R2
    SUB R2,#1
    BL FUN

L1  POP{PC}
```

```
NOP
END
```

Sample Input and Output:

Combination Formula

$$^{n}C_{r} = \frac{n!}{(n-r)!r!}$$

Let

N = n1

r = n2

N-r = n3

Result:

The assembly language program to find factorial and combination of a number are written and its output are verified using keil software.

Experiment No : 5 **Date:**

Ascending & Descending order of N numbers

Aim:

To write an assembly language program to sort a given array of N elements in ascending / descending order using bubble sort.

Requirement:

c. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family ARM7 LPC1768 and click ok.
f. click "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Open the startup file and edit as:

   Line No 121: include "IMPORT __main"

   Line No 127 to 130: Comment by including ";" before the statement

   Line No 272 to 273: Comment by including ";" before the statement

i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Algorithm:

a. Start the program
b. Enter the number of iterations to be done in a register(R5).
c. L1: Enter the number of comparison to be done in a register(R6).

d. Load the register (R0) with a starting memory address of the array.
e. L2: Load the register (R1) also with a starting memory address of the array.
f. Load the register (R2) with the address of 2nd element by incrementing starting address by 4.

g. Compare the register R1 with R2 and check for carry flag is set or not?
h. If carry flag is not set, swapping is not required. Jump to "SKIP" (Ascending)
i. If carry flag is set, swapping is not required. Jump to "SKIP" (Descending)
j. Else perform the swapping
    a. Store R2 value in a address location pointed by the register R0.
    b. Store the R1 value in the next address location by incrementing the address in R0 by 4.

k. SKIP: Update the address in register R0 to next address location by incrementing by 4.
l. Subtract the number of comparison (R6) by 1
m. Check for Zero Flag, if not set Jump to "L2"
n. Else Subtract the number of iteration (R5) by 1.
o. Check for zero flag, if not set Jump to "L1"
p. Else, Stop the program

Note: Force the memory with any N = 5 elements.

Number of iteration (N-1)= 4

Number of comparison(N-1) = 4

Program:

```
AREA FACTORIAL,CODE, READONLY
            ENTRY
      EXPORT __main
__main


      Mov R5, #4
L1    Mov R6, R5

      LDR R0, =0x10000000
L2    LDR R1, [R0]
      LDR R2, [R0,#4]

      CMP R1,R2
      BLO SKIP  // BHI SKIP
      STR R2, [R0]
      STR R1, [R0, #4]

 SKIP       ADD R0,R0,#4
      SUBS R6,#1
      BNE L2
      SUBS R5, #1
      BNE L1
      NOP
      END
```

Sample Input and Output:

Result:

The assembly language program to sort the given array of N numbers in ascending / Descending order using bubble sort is written and its output is verified using keil software.

Experiment No : 6 **Date:**

Square and Cube of a floating point number

Aim:

To write an assembly language program to find square and cube of a number.

Requirement:

d. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family **ARM STM32F401RE** and click ok.
f. "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Right click "Source Group 1" Add .C file by selecting "**system_stm32f4xx.c"** from the folder C:\Keil\ARM\Startup\ST\STM32F4xx
i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file with extension ".s"is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Note: error 65: access violation at 0x40023800 : no 'read' permission

- Project->Options for Target->Debug
- then added an Initialization file using the ... button (my.ini)

- create the file and click OK then in the my.ini file add **MAP 0x40000000, 0x47FFFFFF READ WRITE EXEC** // allow R/W access to IO space
- Save the file and restart the debugger. This will give the permission to read or modify any of the SYSCTL, GPIO, etc. registers.

Program:

```
        AREA SQUARE,CODE,READONLY
        ENTRY
        EXPORT __main
__main
        LDR R0,=0XE000ED88
        MOV R1,#0X00F00000
        STR R1,[R0]

        VMOV.F32 S0,#0X40200000
        VMUL.F32 S1,S0,S0
        VMUL.F32 S2,S1,S0
        NOP
        END
```

Sample Input and Output:

CPACR (Co-Processor Access Control Register : Address = 0xE000ED88
Single precision Floating Point Registers are: S0 – S31
Double precision Floating Point Registers are: D0 – D15
Floating Point Status and Control Register.
To enable the floating point unit, make 23^{rd}- 20^{th} bit as 1. So store the value 00F00000 in the address location of CPACR: 0xE000ED88.

Eg: +2.5 = 0010.10000

Normalization: $1.010000 * 2^{-1}$

Exponent = 7F+power of 2 = 7F+1 = 80 = 1000 0000

Sign bit is 31

Exponent bits is 30 to 23

Mantissa bits = 22 to 0

+2.5 = 0 1000 0000 01000000000000000000000 = 0100 0000 0010 0000 0000 0000 0000 0000

= (40200000)h

Result:

The assembly language program to find square and cube of a floating point number is written and its output is verified in keil software.

Experiment No : 7 **Date:**

Addition and Subtraction of a floating point number

Aim:

To write an assembly language program to find addition and subtraction of a floating point number.

Requirement:

e. Keil µVision 5 software.

Procedure:

a. Open Keil µVision 5 software.
b. Go to "Project", select new µVision project.
c. Create a project folder by entering a name.
d. In the project folder create a project by giving project name and save it.
e. Select a microcontroller family **ARM STM32F401RE** and click ok.
f. "yes" for "copy startup code to project folder and add to project".
g. Check "Target" folder created in the project window along with "Source Group 1" folder.
h. Right click "Source Group 1" Add .C file by selecting "**system_stm32f4xx.c"** from the folder C:\Keil\ARM\Startup\ST\STM32F4xx
i. Now select the target folder present in project window. Right click "Source Group1", select "Add New item to group "Source Group1".
j. Select "Asm.s", then enter file name and click "Add".
k. Verify that asm file with extension ".s"is added in the "Source Group 1" present in the project window.
l. Now Enter the program in the empty file.
m. Now "compile" the program, check for error and clear the errors present in the program.
n. Now "build" the program and then "debug".
o. Check the output.

Program:

```
        AREA FSA,CODE,READONLY
        ENTRY
        EXPORT __main
__main
        LDR R0,=0XE000ED88
        MOV R1,#0X00F00000
        STR R1,[R0]

        VMOV.F32 S0,#0X40C80000
        VMOV.F32 S1,#0X40200000
        VADD.F32 S2,S0,S1
        VSUB.F32 S3,S0,S1
        VMUL.F32 S4,S0,S1
        NOP
        END
```

Sample Input and Output:

CPACR (Co-Processor Access Control Register : Address = 0xE000ED88
Single precision Floating Point Registers are: S0 – S31
Double precision Floating Point Registers are: D0 – D15
Floating Point Status and Control Register.
To enable the floating point unit, make 23^{rd}- 20^{th} bit as 1. So store the value 00F00000 in the address location of CPACR: 0xE000ED88.

Eg: $(+6.25)_{10}$= 0110.010000

Normalization: $1.10010000 * 2^{-2}$

Exponent = 7F+power of 2 = 7F+2 = 81 = 1000 0001

Sign bit is 31

Exponent bits is 30 to 23

Mantissa bits = 22 to 0

+6.25 = 0 1000 0001 10010000000000000000 = 0100 0000 1100 1000 0000 0000 0000 0000

= (40C80000)h

+2.5 = 0 1000 0000 01000000000000000000000 = 0100 0000 0010 0000 0000 0000 0000 0000

= (40200000)h

Result:

The assembly language program to find addition and subtraction of a floating point number is written and its output is verified in keil software.

Procedure to do Interface Program

A. INSTALL KEIL UVISION4 IDE

B. PROJECT CREATION IN KEILUV4 IDE:

1. Create a project folder before creating NEW project.

2. Open Keil uVision4 IDE software by double clicking on "Keil Uvision4" icon.

3. Go to "Project" then to "New uVision Project" and save it with a name in the respective project folder, already you created.

4. Select the device as "NXP" In that "LPC2148" then press OK and then press "YES" button to add "startup.s" file.

5. In startup file go to Configuration Wizard. In Configuration Wizard window uncheck PLL Setup and check VPBDIV Setup.

6. Go to "File" In that "New" to open an editor window. Create your source file and use the header file "lpc21xx.h" in the source file and save the file. Colour syntax highlighting will be enabled once the file is saved with a extension such as ".C ".

7. Right click on "Source Group 1" and select the option "Add Existing Files to Group Source Group 1"add the *.C source file(s) to the group.

8. After adding the source file you can see the file in Project Window.

9. Then go to "Project" in that "Translate" to compile the File (s). Check out the Build output window.

10. Right click on Target1 and select options for Target Target1.

Then go to option "Target" in that

1. Xtal 12.0MHz
2. Select "Use MicroLIB".
3. Select IROM1 (starting 0x0 size 0x80000).
4. Select IRAM1 (starting 0x40000000 size 0x8000).

1. Then go to option "Output"
   1. Select "Create Hex file".
2. Then go to option "Linker"

Select "Use Memory Layout for Target Dialog". To come out of this window press OK.

11. Go to "Project" in that "Build Target" for building all source files such as ".C",".ASM", ".h", files, etc…This will create the *.HEX file if no warnings & no Errors. Check out the Build output window.

C. INSTALL FLASH MAGIC VERSION 6.1

D. ISP PROGRAMMING

FLASH MAGIC software can be used to download the HEX files to the Flash memory of controller.

How to Download?

Connect the serial cross cable from 9-pin DSUB Female connector (DB2) to the PC COM port. Push both SW11/1, 2 to ON position, JP7 should be shorted while using ISP programming. Connect DC +5V Power, through the 9-pin DSUB Male connector (DB1) applied from an external source. Switch ON the power. Open JP13 while downloading the software.

Settings in FLASH MAGIC:

Options -> Advanced options -> Hardware Config

Enable these options only

Use DTR and RTS to control RST and ISP pin

Keep RTS asserted while COM port open

Press OK then do the below settings

Step1. Communications:

1. Device : LPC2148

2. Com Port : COM1

3. Baud Rate : 9600

4. Interface : None(ISP)

5. Oscillator : 12MHz

Step2. ERASE:

1. Select "Erase Blocks Used By Hex File".

Step3. Hex file:

1. Browse and select the Hex file which you want to download.

Step4. Options

1. Select "Verify after programming".

Step5. Start:

1. Click Start to download the hex file to the controller.

After downloading the code the program starts executing in the hardware, then remove the ISP jumper JP7.

DEMO PROGRAMS IN KEIL uVISION4 IDE:

For all the demo programs make sure that the corresponding settings have to be made:

1. Both the pins of SW11 should be in ON position for ISP programming.
2. Short JP7 for ISP programming.
3. Short JP4 to connect +5v to Interface Board.
4. Short JP13 while testing the on-board stepper motor interface.
5. Inter connect CN7 to CN8 by means of 26 core FRC cable to use On board interfaces.
6. Use only CN7 connector for External NIFC's. Use 26 core 2 feet flat cable

Note:

1) Do not short any pins of JP14, JP15, JP16, JP17, JP18 and JP19 when you make connection from CN7 to CN8.

2) While using SPI and I2C do not make connection from CN7 to CN8.

Experiment No: 8 **Date:**

UART

Aim:

To write an C program to display message using Internal UART.

Requirement:

a. Keil µVision 5 software.
b. ARM 7 LPC2148 kit
c. Flash Magic Software

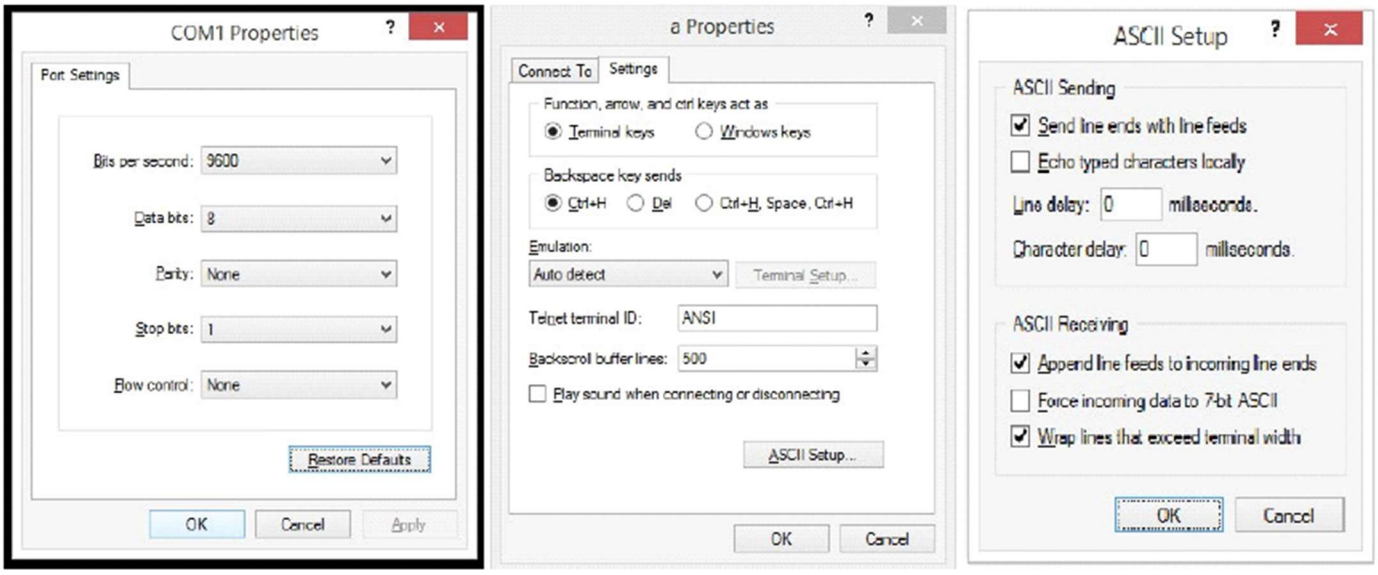
Theory:

The board has an RS-232 serial communication port. The RS-232 transmits and receives signals that appear on the female 9-pin DB connectors (DB2). Use a standard RS-232 cross cable to connect the board to the computer's serial port. The controller U1 provides serial I/O data at TTL levels to the MAX3232 (U2) device, which in turn converts the logic value to the appropriate RS-232 voltage level.

Procedure:

Refer Page No 35 and complete the procedure then do:

After downloading is completed, Push both pins of dip-switch SW11 to OFF position. Unshort the jumper JP7. Press RESET (SW9). Open the hyper terminal, set the Com port, baud rate and other settings as shown below. Pres any key on the PC keyboard the same will be displayed on the monitor.



Program:

```
#include<lpc214x.h>

void uart_init(void);
unsigned int delay;
unsigned char *ptr;
unsigned char arr[]="HELLO WORLD\r";

int main()
{
      while(1)
      {
            uart_init();
            ptr = arr;
            while(*ptr!='\0')
            {
                  U0THR=*ptr++;
                  while(!(U0LSR & 0x20)== 0x20);
                  for(delay=0;delay<=600;delay++);
            }
            for(delay=0;delay<=60000;delay++);
      }
}
void uart_init(void)
{
  PINSEL0=0X0000005;                        //select TXD0 and RXD0 lines
      U0LCR = 0X00000083;                         //enable baud rate divisor
loading and
      U0DLM = 0X00;                               //select the data format
  U0DLL = 0x13;                             //select baud rate 9600 bps
  U0LCR = 0X00000003;

}
```

Result:

The C program to display message using Internal UART is written and verified using LPC2148 kit.

Experiment No: 9 **Date:**

Stepper Motor

Aim:

To write an C program to interface a stepper motor with ARM7 LPC2148.

Requirement:

a. Keil µVision 5 software.
b. ARM 7 LPC2148 kit
c. Flash Magic Software

Theory:

CN1 can be used for High Current applications where a stepper motor, a DC motor, Buzzer and a relay are interfaced through the high current driver ULN2803. These lines will have high current (max 300 mA) with low voltage level of 0.7V.
The Stepper motor can be interfaced to the board by connecting it into the Power Mate PM1. The rotating direction of the stepper motor can be changed through software. Port lines used for Stepper motor are P0.12 – P0.15.

Procedure:

Refer Page No 35 and complete the procedure then do:

Connect the Female Power mate of the stepper motor to the male Power mate PM1 present on the board. Short JP13. Now, the stepper motor rotates one rotation Clockwise & other rotation Anti Clockwise direction. This process is continuously in loop.

Program:

```
#include <LPC21xx.H>

 void clock_wise(void);
 void anti_clock_wise(void);

 unsigned long int var1,var2;
 unsigned int i=0,j=0,k=0;

 int main(void)
 {
      PINSEL0 = 0x00FFFFFF;       //P0.12 to P0.15 GPIo
      IO0DIR |= 0x0000F000;       //P0.12 to P0.15 output

      while(1)
      {
            for(j=0;j<50;j++)        // 20 times in Clock wise Rotation
                  clock_wise();
```

```
            for(k=0;k<65000;k++);   // Delay to show  anti_clock Rotation

            for(j=0;j<50;j++)       // 20 times in  Anti Clock wise Rotation
                 anti_clock_wise();

            for(k=0;k<65000;k++);   // Delay to show clock Rotation

     }                                          // End of while(1)

}                                               // End of main

void clock_wise(void)
{
     var1 = 0x00000800;          //For Clockwise
    for(i=0;i<=3;i++)            // for A B C D Stepping
     {
           var1 = var1<<1;             //For Clockwise
       var2 = ~var1;
       var2 = var2 & 0x0000F000;

          IO0PIN = ~var2;

       for(k=0;k<3000;k++);     //for step speed variation
      }

}

void anti_clock_wise(void)
{
     var1 = 0x00010000;          //For Anticlockwise
     for(i=0;i<=3;i++)           // for A B C D Stepping
      {
       var1 = var1>>1;           //For Anticlockwise
       var2 = ~var1;
       var2 = var2 & 0x0000F000;

          IO0PIN = ~var2;
       for(k=0;k<3000;k++);     //for step speed variation

      }
}
```

Result:

The C program to interface a stepper motor with ARM7 LPC2148 is written and stepper motor is rotated in clock wise and anti-clock wise direction .

Experiment No: 10 **Date:**

DAC

Aim:

To write an C program to interface a DAC with ARM7 LPC2148.

Requirement:

a. Keil µVision 5 software.
b. ARM 7 LPC2148 kit
c. Flash Magic Software

Theory:

DAC0800 is used to convert the digital data into analog signals. Digital data from specified port lines is given to DAC input. Amplitude of output waveform can be varied by varying POT3 (5K Pot) that is by varying the reference voltage of DAC0800 when JP3 is closed. For Bipolar mode open jumper JP11. For Unipolar mode short jumper JP11. Port lines used for DAC are P0.16 – P0.23.

Procedure:

Refer Page No 35 and complete the procedure.

Program:

```
#include <lpc21xx.h>

void delay(void);

int main ()
{
       PINSEL0 = 0x00000000 ;      // Configure P0.0 to P0.15 as GPIO
       PINSEL1 = 0x00000000 ;      // Configure P0.16 to P0.31 as GPIO
       IO0DIR  = 0x00FF0000 ;

       while(1)
    {
       IO0PIN = 0x00000000;
        delay();
        IO0PIN = 0x00FF0000;
        delay();
    }
}

void delay(void)
{
       unsigned int i=0;
       for(i=0;i<=95000;i++);
}
```

Result:

The C program to interface a DAC with ARM7 LPC2148 is written and square wave is generated.

Experiment No: 11 **Date:**

7 – Segment LED

Aim:

To write an C program to interface a 7-segment LED with ARM7 LPC2148.

Requirement:

a. Keil µVision 5 software.
b. ARM 7 LPC2148 kit
c. Flash Magic Software

Procedure:

Refer Page No 35 and complete the procedure.

Keep all pins of SW13 in ON position and unshort the jumper JP21. Press SW1 and observe display changing from 0000 to FFFF. For every press of SW1 the display will change ex: 0000, 1111, 2222, 3333 etc upto FFFF and then back to 0000. This process is continuously in loop.

Program:

```
unsigned int delay;
unsigned int Switchcount=0;
unsigned int Disp[16]={0x003F0000, 0x00060000, 0x005B0000, 0x004F0000, 0x00660000,0x006D0000,
                                   0x007D0000, 0x00070000, 0x007F0000, 0x006F0000,
0x00770000,0x007C0000,
                                   0x00390000, 0x005E0000, 0x00790000, 0x00710000 };

#define SELDISP1 0x10000000     //P0.28
#define SELDISP2 0x20000000       //P0.29
#define SELDISP3 0x40000000       //P0.30
#define SELDISP4 0x80000000       //P0.31
#define ALLDISP  0xF0000000       //Select all display
#define DATAPORT 0x00FF0000       //P0.16 to P0.23 Data lines connected to drive Seven Segments

int main (void)
{
       PINSEL0 = 0x00000000;
       PINSEL1 = 0x00000000;
       IO0DIR  = 0xF0FF0000;
       IO1DIR  = 0x00000000;

       while(1)
       {
              IO0SET |= ALLDISP;                      // select all digits
              IO0CLR = 0x00FF0000;              // clear the data lines to 7-segment displays
              IO0SET = Disp[Switchcount];       // get the 7-segment display value from the array

              if(!(IO1PIN & 0x00800000))        // if the key is pressed
              {
                     for(delay=0;delay<100000;delay++)   // delay
```

```
                    {}

                if((IO1PIN & 0x00800000))   // check to see if key has been released
                   {
                          Switchcount++;
                          if(Switchcount == 0x10)     // 0 to F has been displayed ? go back to 0
                       {
                           Switchcount = 0;
                           IO0CLR  =  0xF0FF0000;
                       }
                   }
            }
      }
}
```

Result:

The C program to interface a 7 – segment display with ARM7 LPC2148 is written and output is verified.

Experiment No: 12 **Date:**

4x4 keyboard

Aim:

To write an C program to interface a 4x4 keyboard with ARM7 LPC2148.

Requirement:

a. Keil µVision 5 software.
b. ARM 7 LPC2148 kit
c. Flash Magic Software

Procedure:

Refer Page No 35 and complete the procedure.

When keys SW14 to SW29 are pressed the corresponding outputs '0 to F' will be displayed on the LCD.

Program:

```
#include<lpc21xx.h>
#include<stdio.h>

/******* FUNCTION PROTOTYPE*******/

void lcd_init(void);
void clr_disp(void);
void lcd_com(void);
void lcd_data(void);
void wr_cn(void);
void wr_dn(void);
void scan(void);
void get_key(void);
void display(void);
void delay(unsigned int);
void init_port(void);

unsigned long int scan_code[16]= {0x00EE0000,0x00ED0000,0x00EB0000,0x00E70000,
                                  0x00DE0000,0x00DD0000,0x00DB0000,0x00D70000,
                                  0x00BE0000,0x00BD0000,0x00BB0000,0x00B70000,
                                  0x007E0000,0x007D0000,0x007B0000,0x00770000};


unsigned char ASCII_CODE[16]= {'0','1','2','3',
                               '4','5','6','7',
                               '8','9','A','B',
                               'C','D','E','F'};

unsigned char  row,col;
unsigned char temp,flag,i,result,temp1;
unsigned int r,r1;
unsigned long int var,var1,var2,res1,temp2,temp3,temp4;
```

```
unsigned char *ptr,disp[] = "4X4 KEYPAD";
unsigned char disp0[] = "KEYPAD TESTING";
unsigned char disp1[] = "KEY = ";
int main()
{
    //   __ARMLIB_enableIRQ();
       init_port();            //port intialisation
       delay(3200);               //delay
       lcd_init();             //lcd intialisation
       delay(3200);               //delay
    clr_disp();                          //clear display
       delay(500);             //delay

       //........LCD DISPLAY TEST.........//
       ptr = disp;
       temp1 = 0x81;               // Display starting address
       lcd_com();
       delay(800);

       while(*ptr!='\0')
    {
       temp1 = *ptr;
        lcd_data();
          ptr ++;
    }

       //........KEYPAD Working.........//
       while(1)
       {
       get_key();
        display();
       }

}  //end of main()

void get_key(void)                //get the key from the keyboard
{
       unsigned int  i;
       flag = 0x00;
    IO1PIN=0x000f0000;
       while(1)
       {
       for(row=0X00;row<0X04;row++)      //Writing one for col's
        {
              if( row == 0X00)
            {
              temp3=0x00700000;
                  }
            else if(row == 0X01)
            {
              temp3=0x00B00000;
                  }
                  else if(row == 0X02)
                  {
              temp3=0x00D00000;
                  }
            else if(row == 0X03)
                  {
```

```
                temp3=0x00E00000;
                    }
                       var1 = temp3;
                IO1PIN = var1;              // each time var1 value is put to port1
                IO1CLR =~var1;              // Once again Conforming (clearing all other bits)
                scan();
                delay(100);                 //delay
                if(flag == 0xff)
                break;
                } // end of for
                if(flag == 0xff)
                break;
          }  // end of while

       for(i=0;i<16;i++)
       {
          if(scan_code[i] == res1)      //equate the scan_code with res1
          {
                        result =  ASCII_CODE[i];    //same position value of ascii code
                        break;                      //is assigned to result
          }
       }
}// end of get_key();

void scan(void)
{
       unsigned long int t;
       temp2 = IO1PIN;                            // status of port1
       temp2 = temp2 & 0x000F0000;                // Verifying column key
       if(temp2 != 0x000F0000)                    // Check for Key Press or Not
       {
          delay(1000);                             //delay(100)//give debounce delay check again
          temp2 = IO1PIN;
          temp2 = temp2 & 0x000F0000;             //changed condition is same

           if(temp2 != 0x000F0000)                // store the value in res1
           {
                 flag = 0xff;
                 res1 = temp2;
                 t = (temp3 & 0x00F00000);          //Verfying Row Write
                 res1 = res1 | t;                   //final scan value is stored in res1
           }
           else
           {
                 flag = 0x00;
           }
       }
}  // end of scan()

void display(void)
{
       ptr = disp0;
          temp1 = 0x80;              // Display starting address of first line
          lcd_com();

          while(*ptr!='\0')
       {
          temp1 = *ptr;
```

```
       lcd_data();
              ptr ++;
    }

    ptr = disp1;
       temp1 = 0xC0;              // Display starting address of second line
       lcd_com();

       while(*ptr!='\0')
    {
       temp1 = *ptr;
        lcd_data();
              ptr ++;
    }
    temp1 = 0xC6;                 //display  address for key value
       lcd_com();
    temp1 = result;
    lcd_data();
}

void lcd_init (void)
{
       temp = 0x30;
       wr_cn();
       delay(3200);

       temp = 0x30;
       wr_cn();
       delay(3200);

       temp = 0x30;
       wr_cn();
       delay(3200);

       temp = 0x20;
       wr_cn();
       delay(3200);

// load command for lcd function setting with lcd in 4 bit mode,
// 2 line and 5x7 matrix display

       temp = 0x28;
       lcd_com();
       delay(3200);

// load a command for display on, cursor on and blinking off
       temp1 = 0x0C;
       lcd_com();
       delay(800);

// command for cursor increment after data dump
       temp1 = 0x06;
       lcd_com();
       delay(800);

       temp1 = 0x80;
       lcd_com();
       delay(800);
```

```
}

void lcd_data(void)
{
    temp = temp1 & 0xf0;
    wr_dn();
    temp= temp1 & 0x0f;
    temp= temp << 4;
    wr_dn();
    delay(100);
}

void wr_dn(void)                    ////write data reg
{
       IO0CLR  = 0x000000FC;        // clear the port lines.
       IO0SET = temp;                       // Assign the value to the PORT lines
       IO0SET = 0x00000004;         // set bit  RS = 1
       IO0SET = 0x00000008;         // E=1
       delay(10);
       IO0CLR = 0x00000008;
}

void lcd_com(void)
{
    temp = temp1 & 0xf0;
    wr_cn();
    temp = temp1 & 0x0f;
    temp = temp << 4;
    wr_cn();
    delay(500);
}

void wr_cn(void)                //write command reg
{
       IO0CLR  = 0x000000FC;            // clear the port lines.
       IO0SET        = temp;                            // Assign the value to the PORT lines
       IO0CLR  = 0x00000004;            // clear bit  RS = 0
       IO0SET        = 0x00000008;      // E=1
       delay(10);
       IO0CLR  = 0x00000008;
}

void clr_disp(void)
{
// command to clear lcd display
    temp1 = 0x01;
    lcd_com();
    delay(500);
}

void delay(unsigned int r1)
{
       for(r=0;r<r1;r++);
}

void init_port()
{
    IO0DIR = 0x000000FC;    //configure o/p lines for lcd
```

```
	IO1DIR = 0XFFF0FFFF;
}
```

Result:

The C program to interface a keyboard with ARM7 LPC2148 is written and output is verified.