MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INSPIRED BY LIFE

CERTIFICATE

This is to certify that Ms./Mr. …………………...…………………………………… Reg. No. …..…………………… Section: ……………… Roll No: ………………... has satisfactorily completed the lab exercises prescribed for Embedded systems lab [CSE 2263] of Second Year B. Tech. in Computer Science and Engineering Degree at MIT, Manipal, in the academic year 2020-2021.

Date: ……...................................

Signature
Faculty in Charge

CONTENTS

Course Objectives

- To gain knowledge about assembly language and Embedded C programming
- To implement the programs using ARM instruction set
- To understand various interfacing circuits necessary for various applications and programming using ARM.

Course Outcomes

On the completion of this laboratory course, the students will be able to:

- Experiment with Keil µVision simulator to write and execute ARM assembly programs using data transfer and arithmetic instructions.
- Develop the assembly language program for ARM cortex-M microcontroller using logical, branching and looping instructions.
- Create embedded C program for ARM cortex-M microcontroller by interfacing various modules to ARM kit

Evaluation plan

- Internal Assessment Marks : 60%

    - ✓ Continuous evaluation component (for each experiment):10 marks
    - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce
    - ✓ Total marks of the 12 experiments reduced to marks out of 60

- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Class notes, Lab Manual and the required stationery to every lab session
2. Be in time and follow the Instructions from Lab Instructors
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises given in Lab Manual
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Use meaningful names for variables and procedures.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty during evaluation.
- The exercises for each week are divided under three sets:
    - Solved exercise
    - Lab exercises - to be completed during lab hours
    - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed at students end or in a repetition class (if available) with the permission of the faculty concerned but credit will be given only to one day’s experiment(s).

- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

- A sample note preparation is given later in the manual as a model for observation.

Sample lab observation note preparation

LAB NO: **Date:**

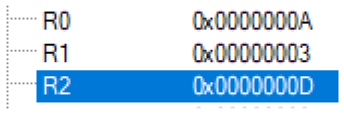
Title: INTRODUCTION TO KEIL μVISION-4 AND PROGRAMS ON DATA TRANSFER INSTRUCTIONS

Add two immediate values in the registers and store the result in the third register.

Program:

```
AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
                DCD 0X10001000
                DCD Reset_Handler
                ALIGN
                AREA mycode, CODE, READONLY
                ENTRY
                EXPORT Reset_Handler
Reset_Handler
        MOV R0, #10
        MOV R1, #3
        ADD R2, R0, R1
        END
```

Sample output:



```
R0        0x0000000A
R1        0x00000003
R2        0x0000000D
```

LAB NO: 1

INTRODUCTION TO KEIL µVISION-4 AND PROGRAMS ON DATA TRANSFER INSTRUCTIONS

Objectives:

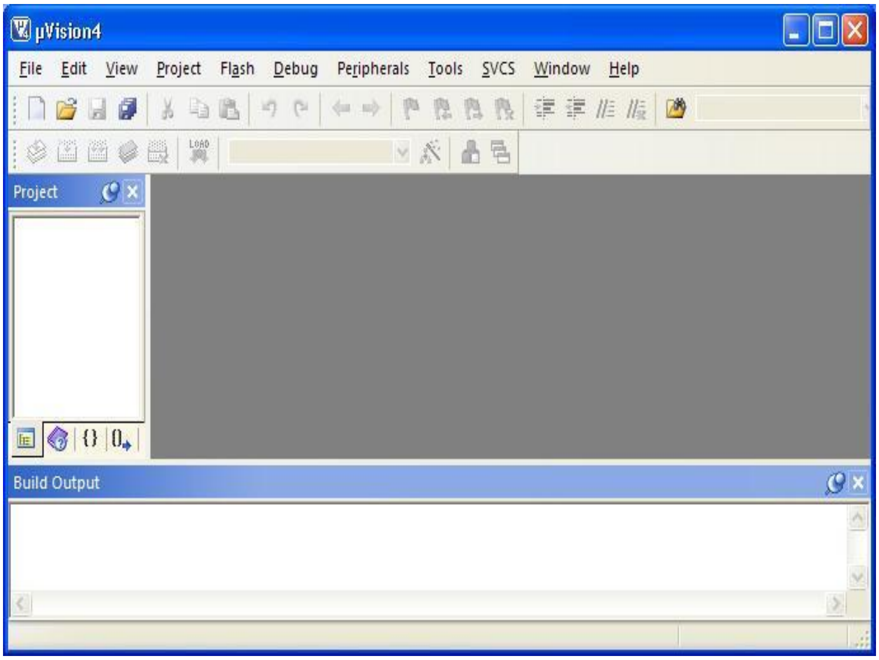
In this lab, students will be able to

- Understand the usage of Keil µVision 4 software for assembly language.
- Write, build and execute assembly language programs in Keil µVision 4.
- Use different data transfer instructions of ARM processor.

I. Running an assembly language program in Keil µVision 4

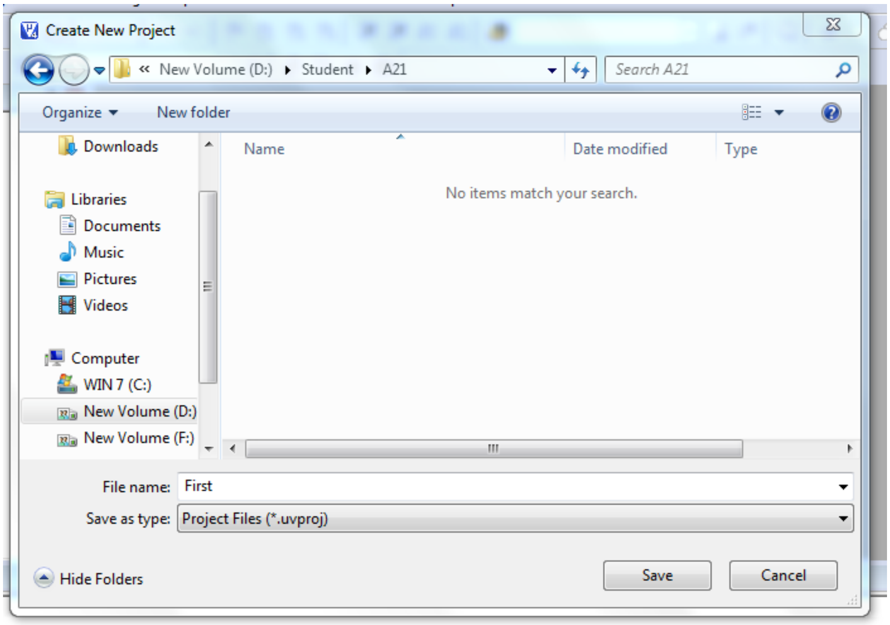
Step 1:

- Create a directory with section followed by roll number (to be unique); e.g. A21
- Start up µVision-4 by clicking on the icon from the desktop or from the "Start" menu or "All Programs". The following screen appears.


µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project
Build Output

Step 2: Create a project

To create a project, click on the "Project" menu from the µVision-4 screen and select "New µVision Project”. Then, select the folder you have created already, give project name and save.

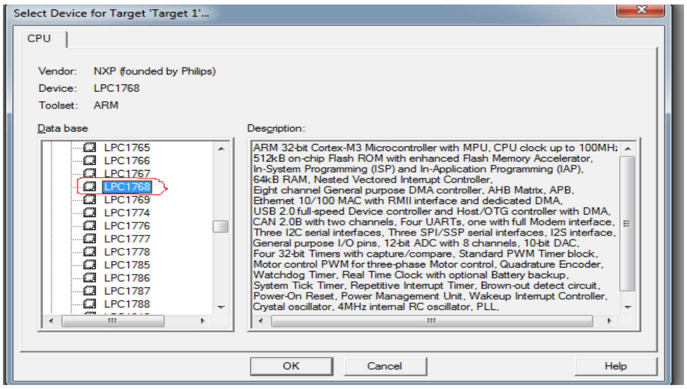


No items match your search.

File name: First

Save as type: Project Files (*.uvproj)

From the "Select Device for Target Target 1..." window, select "NXP" as the vendor. In that, select LPC1768 ARM controller, and then click on OK button. Some general information of the chip is shown in the description box.

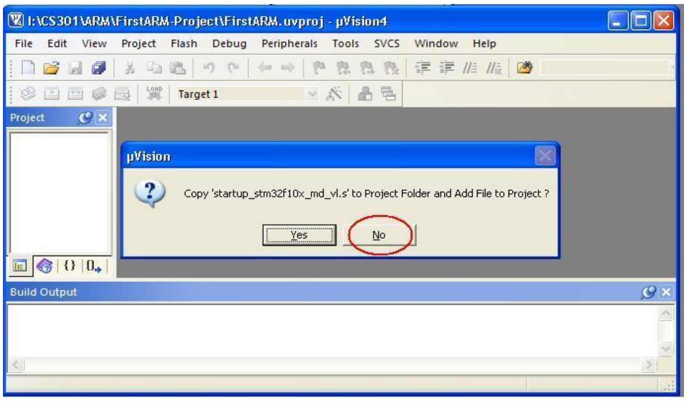


Vendor: NXP (founded by Philips)
Device: LPC1768
Toolset: ARM

Description:

```
ARM 32-bit Cortex-M3 Microcontroller with MPU, CPU clock up to 100MHz
512kB on-chip Flash ROM with enhanced Flash Memory Accelerator,
In-System Programming (ISP) and In-Application Programming (IAP),
64kB RAM, Nested Vectored Interrupt Controller,
Eight channel General purpose DMA controller, AHB Matrix, APB,
Ethernet 10/100 MAC with RMII interface and dedicated DMA,
USB 2.0 full-speed Device controller and Host/OTG controller with DMA,
CAN 2.0B with two channels, Four UARTs, one with full Modem interface,
Three I2C serial interfaces, Three SPI/SSP serial interfaces, I2S interface,
General purpose I/O pins, 12-bit ADC with 8 channels, 10-bit DAC,
Four 32-bit Timers with capture/compare, Standard PWM Timer block,
Motor control PWM for three-phase Motor control, Quadrature Encoder,
Watchdog Timer, Real Time Clock with optional Battery backup,
System Tick Timer, Repetitive Interrupt Timer, Brown-out detect circuit,
Power-On Reset, Power Management Unit, Wakeup Interrupt Controller,
Crystal oscillator, 4MHz internal RC oscillator, PLL,
```

Make sure you click on "NO" for the following pop up window.

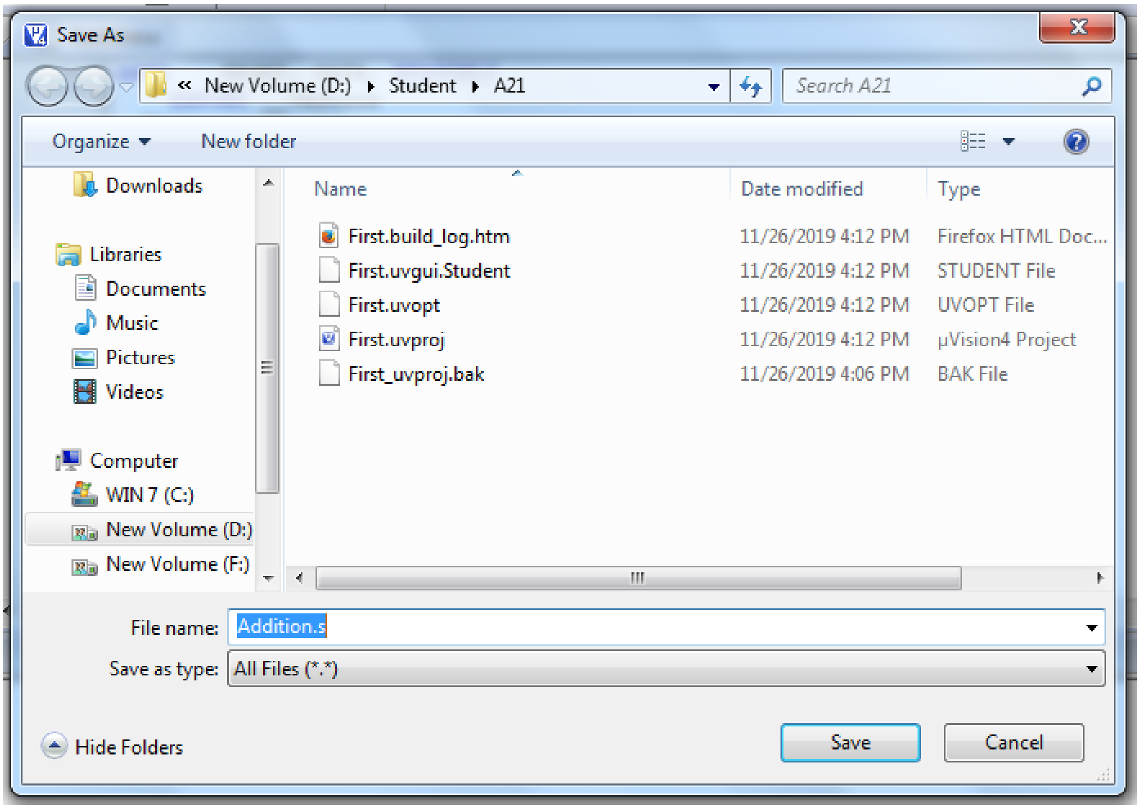

I:\CS301\ARM\FirstARM-Project\FirstARM.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Build Output
µVision

Copy 'startup_stm32f10x_md_vl.s' to Project Folder and Add File to Project ?

Yes [✓ No]

Step 3: Create Source File

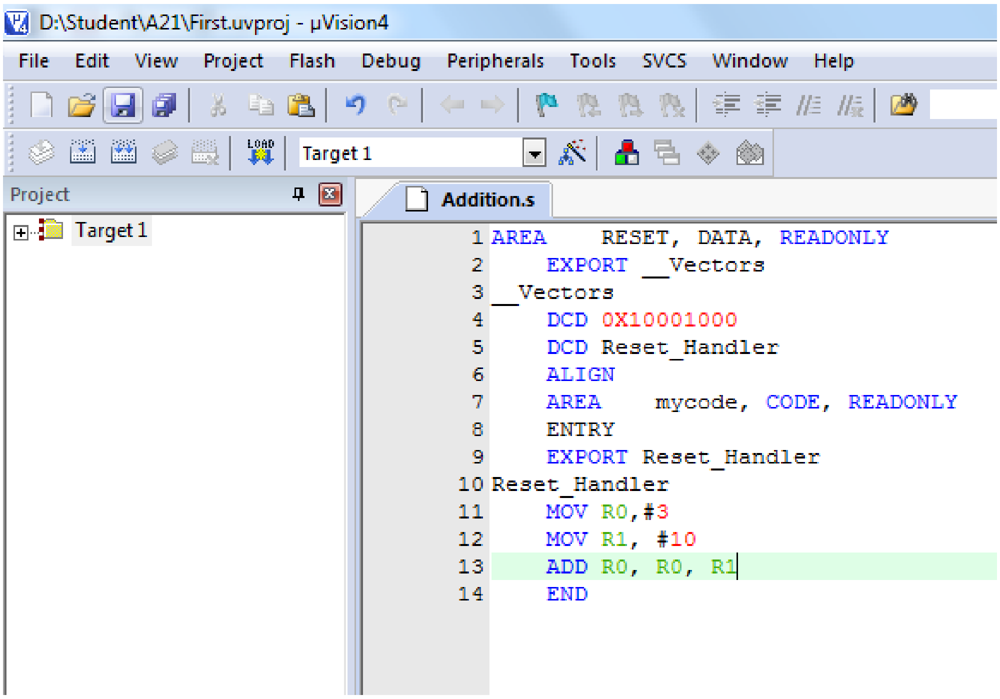
From the "File" menu, select "New", to get the editor window. Type the program here. (Note: give a tab space at the beginning). Save the program with .s extension in the directory.



Name	Date modified	Type
First.build_log.htm	11/26/2019 4:12 PM	Firefox HTML Doc...
First.uvgui.Student	11/26/2019 4:12 PM	STUDENT File
First.uvopt	11/26/2019 4:12 PM	UVOPT File
First.uvproj	11/26/2019 4:12 PM	µVision4 Project
First_uvproj.bak	11/26/2019 4:06 PM	BAK File

File name: Addition.s

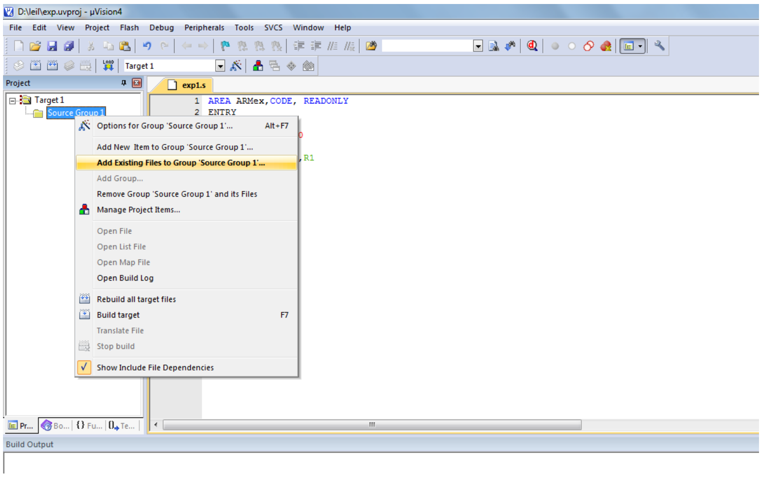
Save as type: All Files (*.*)


D:\Student\A21\First.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Target 1
Addition.s

```
AREA     RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0X10001000
    DCD Reset_Handler
    ALIGN
    AREA     mycode, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
    MOV R0,#3
    MOV R1, #10
    ADD R0, R0, R1
    END
```

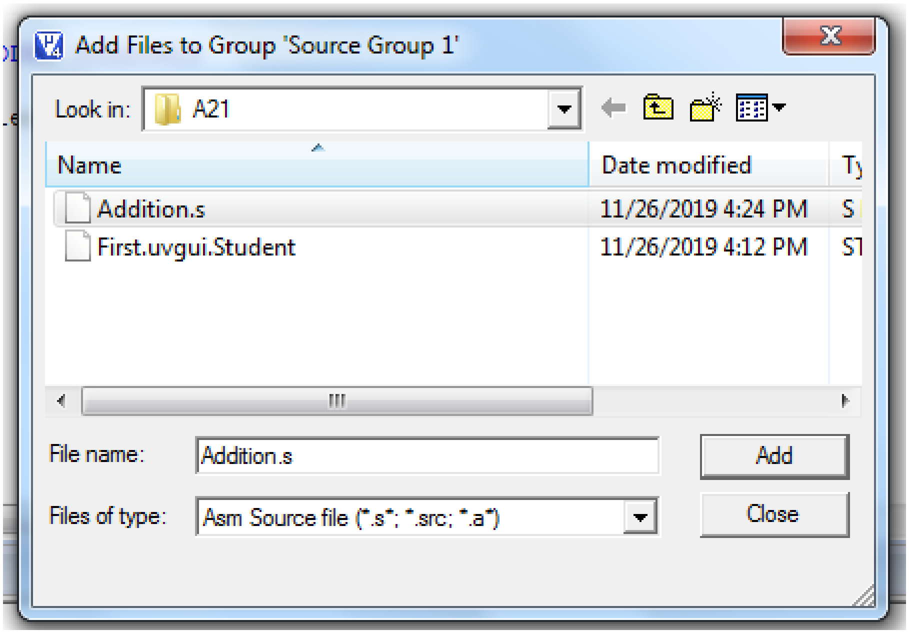
Step 4: Add Source File to the Project

Click on the + symbol near the Target 1 in the top left corner of the window. Right click on the "Source Group 1", select "Add Existing Files to Group 'Source Group 1'".


D:\leil\exp.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Target 1
Source Group 1
exp1.s
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Add Group...
Remove Group 'Source Group 1' and its Files
Manage Project Items...
Open File
Open List File
Open Map File
Open Build Log
Rebuild all target files
Build target F7
Translate File
Stop build
Show Include File Dependencies
Build Output

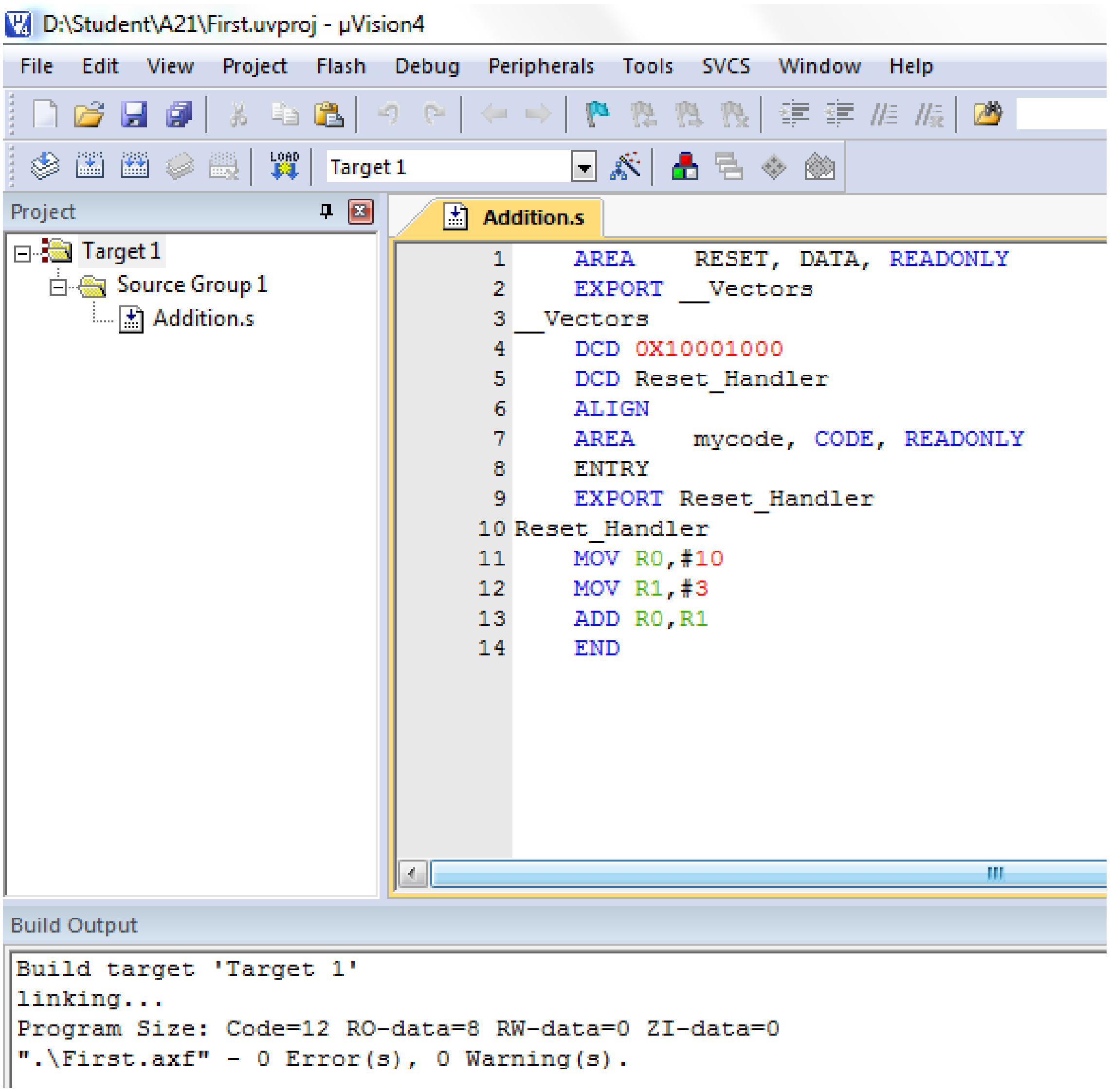
```
AREA ARMex, CODE, READONLY
ENTRY
```

Select "Files of type" as "asm Source file (*.s*;*.src*;*.a*), then select the file. Click on "Add", and then click on "Close".



Step 5: Build your project

Click on the "+" beside the "Source Group 1", you will see the program " Addition.s" Click on the "Build" button or from the "Project" menu, you will see the following screen.

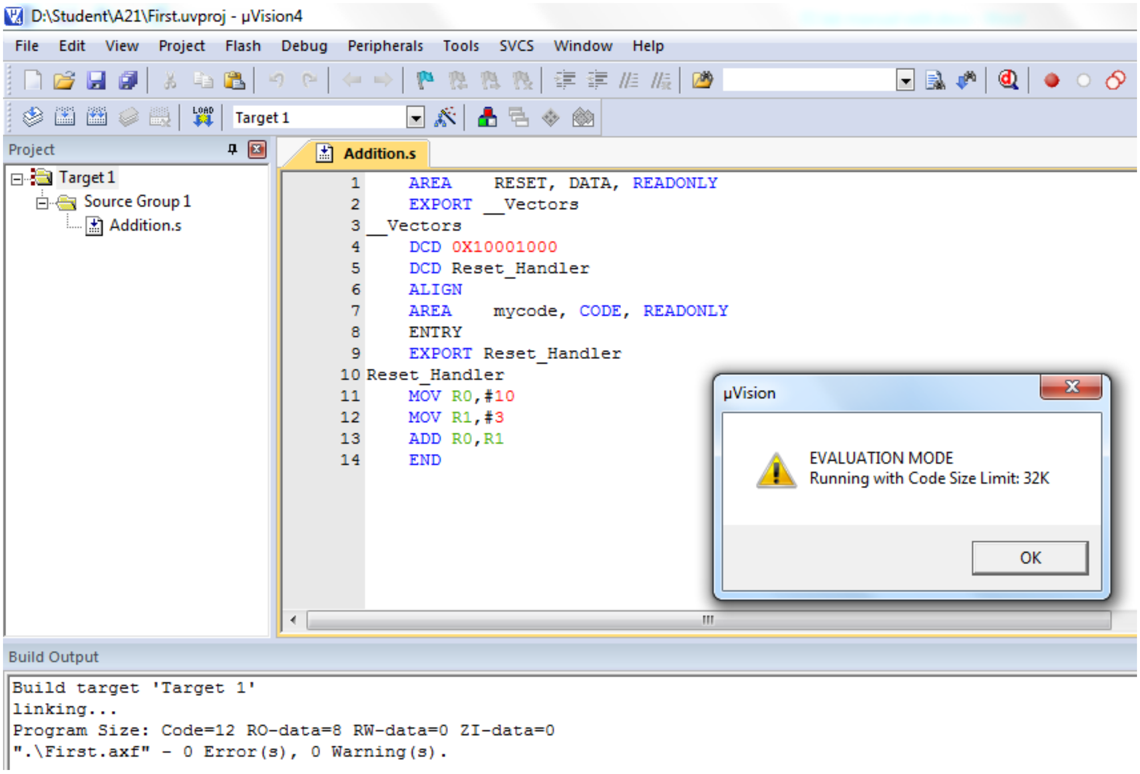

D:\Student\A21\First.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Target 1
Source Group 1
Addition.s
Build Output

```
        AREA    RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
        DCD 0X10001000
        DCD Reset_Handler
        ALIGN
        AREA    mycode, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
        MOV R0,#10
        MOV R1,#3
        ADD R0,R1
        END
```

```
Build target 'Target 1'
linking...
Program Size: Code=12 RO-data=8 RW-data=0 ZI-data=0
".\First.axf" - 0 Error(s), 0 Warning(s).
```

Step 6: Run the program

Run the program through the "Debug" menu.

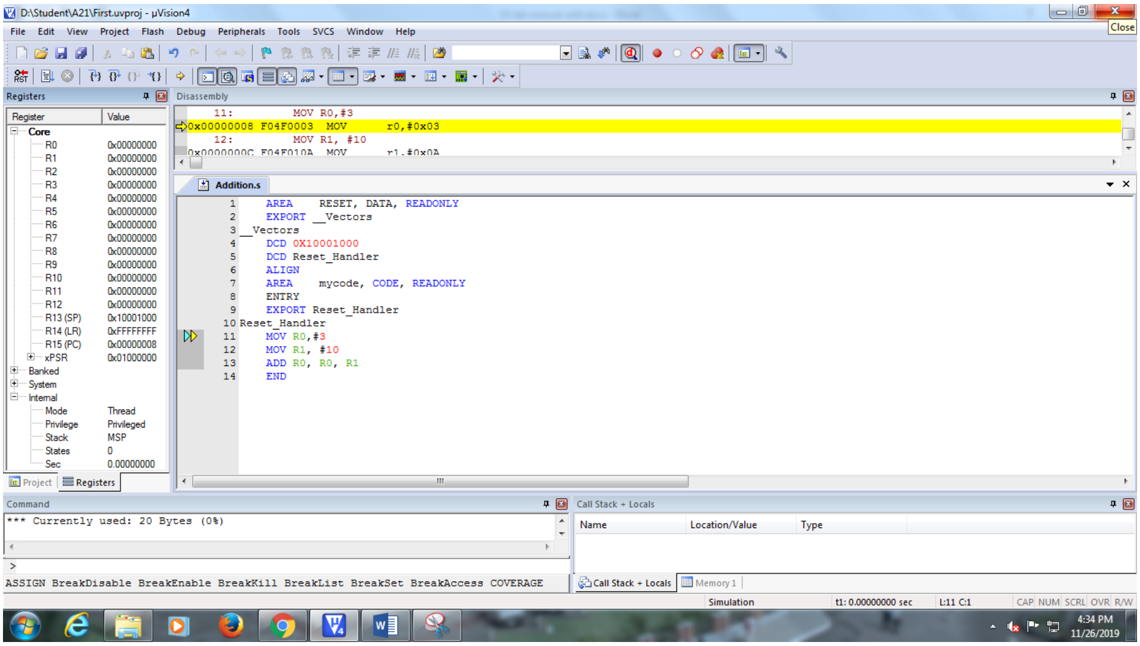

D:\Student\A21\First.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Target 1
Source Group 1
Addition.s
µVision
OK
Build Output

```
        AREA     RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
        DCD 0X10001000
        DCD Reset_Handler
        ALIGN
        AREA     mycode, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
        MOV R0,#10
        MOV R1,#3
        ADD R0,R1
        END
```

EVALUATION MODE
Running with Code Size Limit: 32K

```
Build target 'Target 1'
linking...
Program Size: Code=12 RO-data=8 RW-data=0 ZI-data=0
".\First.axf" - 0 Error(s), 0 Warning(s).
```

Click on "OK" for the pop up window showing "EVALUATION MODE, Running with Code Size Limit: 32K". You will see the following window.



Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000008
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	0
Sec	0.00000000

```
    11:          MOV R0,#3
0x00000008 F04F0003  MOV       r0,#0x03
    12:          MOV R1, #10
0x0000000C F04F010A  MOV       r1,#0x0A
```

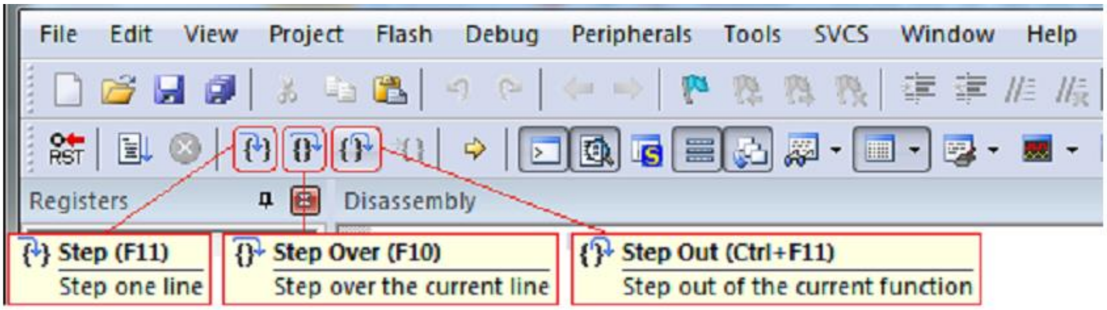
```
    AREA    RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0X10001000
    DCD Reset_Handler
    ALIGN
    AREA    mycode, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
    MOV R0,#3
    MOV R1, #10
    ADD R0, R0, R1
    END
```

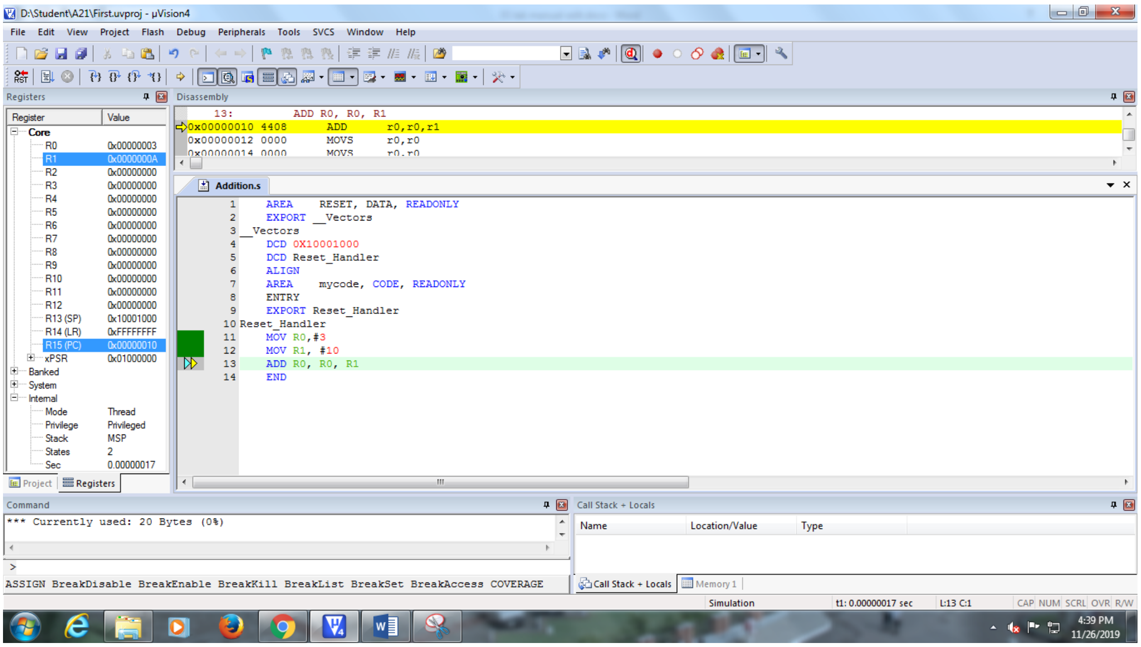
```
*** Currently used: 20 Bytes (0%)
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
```

Name	Location/Value	Type

Open µVision4 to full screen to have a better and complete view. The left hand side window shows the registers and the right side window shows the program code. There are some other windows open. Adjust the size of them to have a better view. Run the program step by step; observe the change of the values in the registers.

Run the program using the **Step Over** button or click on **Step Over** from the Debug menu. It executes the instructions of the program one after another. To trace the program one can use the **Step** button, as well. The difference between the **Step Over** and **Step** is in executing functions. While **Step** goes into the function and executes its instructions one by one, **Step Over** executes the function completely and goes to the instruction next to the function. To see the difference between them, trace the program once with **Step Over** and then with **Step**. When the PC is executing the function and wants the function to be executed completely one can use **Step Out**. In this case, the instructions of the function will be executed, it returns from the function, and goes to the instruction which is next to the function call.





Register	Value
Core	
R0	0x00000003
R1	0x0000000A
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000010
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	2
Sec	0.00000017

```
      13:          ADD R0, R0, R1
0x00000010 4408        ADD        r0,r0,r1
0x00000012 0000        MOVS       r0,r0
0x00000014 0000        MOVS       r0,r0
```

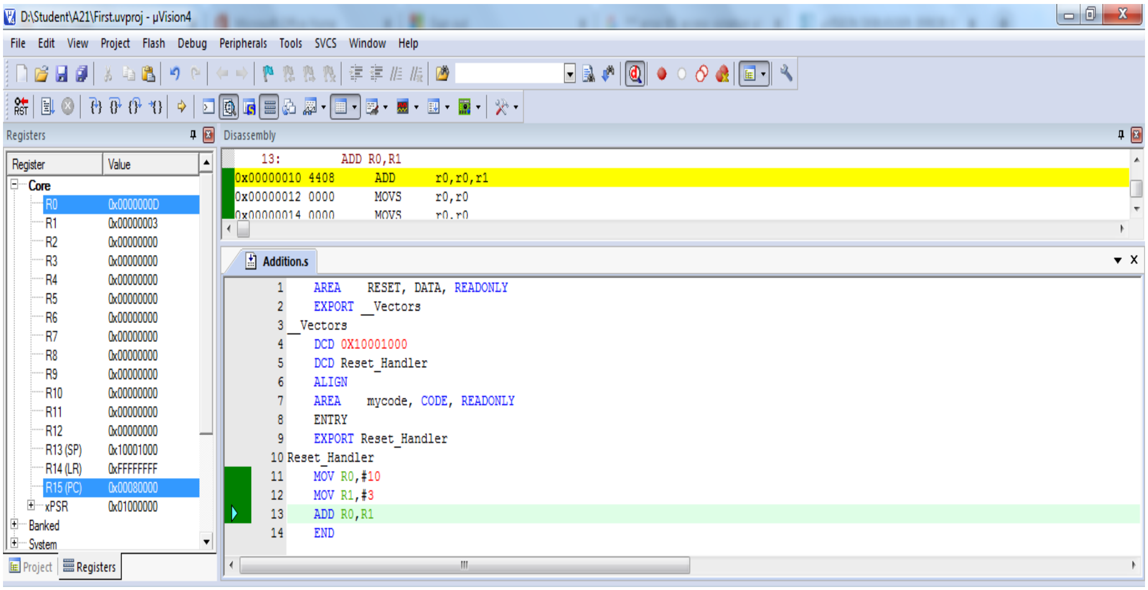
```
    AREA    RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0X10001000
    DCD Reset_Handler
    ALIGN
    AREA    mycode, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
    MOV R0,#3
    MOV R1, #10
    ADD R0, R0, R1
    END
```

```
*** Currently used: 20 Bytes (0%)
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
```

Name	Location/Value	Type

Simulation   t1: 0.00000017 sec   L:13 C:1   CAP NUM SCRL OVR R/W

4:39 PM 11/26/2019



D:\Student\A21\First.uvproj - µVision4

Register	Value
Core	
R0	0x0000000D
R1	0x00000003
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00080000
xPSR	0x01000000
Banked	
System	

```
    13:          ADD R0,R1
0x00000010 4408      ADD      r0,r0,r1
0x00000012 0000      MOVS     r0,r0
0x00000014 0000      MOVS     r0,r0
```

Addition.s

```
    AREA    RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0X10001000
    DCD Reset_Handler
    ALIGN
    AREA    mycode, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
    MOV R0,#10
    MOV R1,#3
    ADD R0,R1
    END
```

Click on the "**Start/Stop Debug** Session" again to stop execution of the program.

II. ARM assembly language module

An ARM assembly language module has several constituent parts.

These are:

- Extensible Linking Format (ELF) sections (defined by the AREA directive).
- Application entry (defined by the ENTRY directive).
- Program end (defined by the END directive).

Assembler Directives

- ➢ Assembler directives are the commands to the assembler that direct the assembly process.
- ➢ They do not generate any machine code i.e. they do not contribute to the final size of machine code and they are assembler specific

AREA:

The AREA directive tells the assembler to define a new section of memory. The memory can be code (instructions) or data and can have attributes such as READONLY, READWRITE and so on. This is used to define one or more blocks of indivisible memory for code or data to be used by the linker. The following is the format:

AREA sectionname attribute, attribute, …

The following line defines a new area named mycode which has CODE and READONLY attributes:

AREA mycode, CODE, READONLY

Commonly used attributes are CODE, DATA, READONLY, READWRITE, ALIGN and END.

READONLY:

It is an attribute given to an area of memory which can only be read from. It is by default for CODE. This area is used to write the instructions.

READWRITE:

It is an attribute given to an area of memory which can be read from and written to. It is by default for DATA.

CODE:

It is an attribute given to an area of memory used for executable machine instructions. It is by default READONLY memory.

DATA:

It is an attribute given to an area of memory used for data and no instructions can be placed in this area. It is by default READWRITE memory.

ALIGN:

It is an attribute given to an area of memory to indicate how memory should be allocated according to the addresses. When the ALIGN is used for CODE and READONLY, it is aligned in 4-bytes address boundary by default since the ARM instructions are 32 bit word. If it is written as ALIGN = 3, it indicates that the information should be placed in memory with addresses of 2^3, that is for example 0x50000, 0x50008, 0x50010, 0x50018 and so on.

EXPORT:

The EXPORT directive declares a symbol that can be used by the linker to resolve symbol references in separate object and library files.

DCD (Define constant word):

Allocates a word size memory and initializes the values. Allocates one or more words of memory, aligned on 4-byte boundaries and defines initial run time contents of the memory.

ENTRY:

The ENTRY directive declares an entry point to the program. It marks the first instruction to be executed. In applications using the C library, an entry point is also contained within the C library initialization code. Initialization code and exception handlers also contain entry points

END:

It indicates to the assembler the end of the source code. The END directive is the last line of the ARM assembly program and anything after the END directive in the source file is ignored by the assembler.

Example:

```
        AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
```

```
        DCD 0X10001000   ;stack pointer value when stack is empty
                          ;The processor uses a full descending stack.
                          ;This means the stack pointer holds the address of the last
                        ;stacked item in memory. When the processor pushes a new item
                          ;onto the stack, it decrements the stack pointer and then
                          ;writes the item to the new memory location.
        DCD Reset_Handler  ; reset vector. The program linker requires Reset_Handler

        ALIGN

        AREA mycode, CODE, READONLY

        ENTRY

        EXPORT Reset_Handler

Reset_Handler

;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;;

        MOV R0, #10

        MOV R1, #3

        ADD R0, R1

        END   ;End of the program
```

III. Introduction to ARM addressing modes

Data can be transferred into and out of the ARM controller using different addressing modes. There are different ways to specify the address of the operands for any given operations such as load, add or branch. The different ways of determining the address of the operands are called addressing modes. Different addressing modes used in ARM are listed in Appendix A.

Solved Exercise:

Write an ARM assembly language program to copy 32 bit data from code memory to data memory.

Source: SRC= 0X00000008 at location pointed by R0

Destination: DST = 0X00000008 at location pointed by R1 after the execution

Program:

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

   __Vectors
       DCD  0x10001000     ; stack pointer value when stack is empty
       DCD  Reset_Handler  ; reset vector
       ALIGN
       AREA mycode, CODE, READONLY
       ENTRY
       EXPORT Reset_Handler

   Reset_Handler
        LDR R0, =SRC        ; Load address of SRC into R0
        LDR R1, =DST        ;  Load the address of DST onto R1
        LDR R3, [R0]        ;  Load data pointed by R0 into R3
        STR R3,[R1]         ;  Store data from R3 into the address pointed by R1
   STOP
        B STOP          ; Be there
   SRC DCD 8          ;SRC  location in code memory
        AREA mydata, DATA, READWRITE
   DST DCD 0                    ;DST location in data memory
        END
```

Observations to be made

1. **Data storage into the memory:** Click on Memory window and go to Memory1 option. Type address pointed by R0 in address space and observe how the data are stored into the memory.
2. **Data movement from one memory to another memory:** Click on Memory window and go to Memory2 option. Type address pointed by R1 in address space and observe data movement to another location before execution and after execution.

Lab Exercises:

1. Write an ARM assembly language program to store data into general purpose registers.
2. Write an ARM assembly language program to transfer a 32 bit number from one location in the data memory to another location in the data memory.
3. Write an ARM assembly language program to transfer block of ten 32 bit numbers from code memory to data memory when the source and destination blocks are non-overlapping.
4. Reverse an array of ten 32 bit numbers in the memory.

Additional Exercises:

1. Repeat Q3 above using pre indexing mode.
2. Repeat Q3 above when the source and destination blocks are overlapping

LAB NO: 2 **Date:**

PROGRAMS ON ARITHMETIC INSTRUCTIONS

Objectives:

In this lab, students will be able to

- Identify and use the instructions required to perform addition and subtraction
- Debug and trace the programs.

Refer Appendix A for instruction details.

Solved Exercise:

Write a program to add two 32 bit numbers available in the code memory. Store the result in the data memory

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

  __Vectors
       DCD  0x40001000     ; stack pointer value when stack is empty
       DCD  Reset_Handler  ; reset vector

        ALIGN
       AREA mycode, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
        LDR R0, =VALUE1   ;pointer to the first value1
        LDR R1, [R0]      ;load the first value into R1
        LDR R0, =VALUE2   ;pointer to the second value
        LDR R3, [R0]      ;load  second number into r3
        ADDS R6, R1, R3   ;add two numbers and store the result in r6
        LDR R2, =RESULT
        STR R6, [R2]
  STOP
```

```
          B STOP
VALUE1 DCD 0X12345678          ; First 32 bit number
VALUE2 DCD 0XABCDEF12          ; Second 32 bit number
          AREA data, DATA, READWRITE
RESULT DCD 0
          END
```

Lab Exercises:

1. Write a program to add ten 32 bit numbers available in code memory and store the result in data memory.
2. Write a program to add two 128 bit numbers available in code memory and store the result in data memory.
   Hint: Use indexed addressing mode.
3. Write a program to subtract two 32 bit numbers available in the code memory and store the result in the data memory.
4. Write a program to subtract two 128 bit numbers available in the code memory and store the result in the data memory.

Additional Exercises:

1. Write a program to find the 2’s complement of 64 bit data in R0 and R1 registers. The R0 holds the lower 32 bit.
2. Add and subtract two 32 bit numbers and check all the flags. Take appropriate data to check all the flags.

LAB NO: 3 **Date:**

PROGRAMS ON ARITHMETIC AND LOGICAL INSTRUCTIONS

Objectives:

In this lab, students will be able to

- Identify and use the instructions required to perform multiplication, division and logical operations
- Debug and trace the programs.

Refer Appendix A for instruction details.

Solved Exercise:

Write an assembly program to multiply two 32 bit numbers

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

  __Vectors
        DCD  0x40001000     ; stack pointer value when stack is empty
       DCD  Reset_Handler  ; reset vector
       ALIGN
       AREA mycode, CODE, READONLY
       ENTRY
       EXPORT Reset_Handler
  Reset_Handler
        LDR R1, =VALUE1           ;pointer to the first value1
        LDR R2,=VALUE2            ;pointer to the second value
        UMULL R3, R4, R2, R1      ;Multiply the values from R1 and R2 and store
                                   ;least significant  32 bit number into R3 and most
                                   ;significant  32 bit number into R4.
        LDR R2, =RESULT
        STR R4, [R2]
        ADD R2, #4
        STR R3, [R2]  ; store result in memory
STOP
        B STOP
        VALUE1 DCD 0X54000000 ; First 32 bit number
        VALUE2 DCD 0X10000002 ; Second 32 bit number
        AREA data, DATA, READWRITE
        RESULT DCD 0
```

Note: If the result is within 32 bits, use MUL instruction.

Lab Exercises:

1. Write an assembly language program to implement division by repetitive subtraction.
2. Find the sum of ‘n’ natural numbers using MLA instruction.
3. Write an assembly language program to find GCD and LCM of two 8 bit numbers
4. Write an ARM assembly language program to convert 2-digit hexadecimal number into ascii format.
5. Write an ARM assembly language program to convert a 32 bit BCD number in the unpacked form into packed form.

Additional Exercises:

1. Write an assembly language program to generate Fibonacci series.
2. Check whether a given number is even or odd.

LAB NO: 4 **Date:**

BRANCHING AND LOOPING

Objectives:

In this lab, students will be able to

- ➢ Learn different kinds of branching instructions.
- ➢ Understand looping code conversion programs

Solved Exercise:

Write an assembly language program to unpack a 32 bit BCD number into eight 32-bit ASCII numbers.

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

   __Vectors
       DCD  0x40001000     ; stack pointer value when stack is empty
       DCD  Reset_Handler  ; reset vector
        ALIGN
        AREA mycode, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
   Reset_Handler
        MOV R5, #4
        LDR R0,=NUM
        LDR R3,=RESULT
UP      LDRB R1,[R0], #1              ; load BCD number into register R1
        AND R2,R1,#0x0F     ; mask upper 4 bits
        ADD R2,#0x30        ; Add 30H to the number, Ascii value of first digit
        STR R2,[R3], #4
        AND R4,R1,#0xF0     ; Mask the second digit
        MOV R4,R4,LSR#04 ; Shift right by 4 bits
        ADD R4,#0x30        ; Ascii value of second digit
        STR R4,[R3], #4
        SUBS R5, #1
        BNE UP              ;Repeat 4 times
STOP B STOP
        NUM DCD 0x12345678
        AREA data, DATA, READWRITE
        RESULT DCD 0
        END
```

Lab Exercises:

1. Convert a 32 bit packed BCD number into its equivalent hexadecimal number.
2. Convert a 16 bit hex number into its equivalent packed BCD
3. Add two 32 bit packed BCD numbers and store the result in packed BCD form.
4. Multiply two 16 bit packed BCD and store the result in packed BCD form.

Additional Exercises:

1. Write an assembly language program to unpack a 32 bit BCD number into eight 32-bit numbers
2. Write an assembly language program to find the sum of bits (no. of 1’s) of a 32 bit number available in the memory.

LAB NO: 5 **Date:**

SORTING AND SEARCHING PROGRAMS

Objectives:

In this lab, students will be able to

- Perform advanced list operations in a given list or array.
- Use different branch instructions.

Solved Exercise:

Write an ARM ALP to sort a list using bubble sort.

```
            AREA    RESET, DATA, READONLY
            EXPORT  __Vectors
  __Vectors
            DCD  0x40001000     ; stack pointer value when stack is empty
            DCD  Reset_Handler  ; reset vector
            ALIGN
            AREA ascend, code, readonly
            ENTRY
 Reset_Handler
            mov r4,#0
            mov r1,#10
            ldr r0, =list
            ldr r2, =result
    up      ldr r3, [r0,r4]
            str r3, [r2,r4]
            add r4, #04
            sub r1,#01
            cmp r1,#00
            bhi up
            ldr r0, =result
            mov r3, #10                 ; inner loop counter
            sub r3, r3, #1
            mov r9, r3                  ; R9 contain no of passes
                                        ; outer loop counter
outer_loop
            mov r5, r0
            mov r4, r3                  ; R4 contains no of comparison in a pass
inner_loop
            ldr r6, [r5], #4
            ldr r7, [r5]
            cmp r7, r6
                                        ; swap without swap instruction
            strls r6, [r5]
            strls r7, [r5, #-4]
            subs r4, r4, #1
            bne inner_loop
            sub r3, #1
            subs r9, r9, #1
            bne outer_loop
list dcd 0x10,0x05,0x33,0x24,0x56,0x77,0x21,0x04,0x87,0x01
        AREA data1, data, readwrite
result DCW 0,0,0,0,0,0,0,0,0,0
            end
```

Lab Exercises:

1. Write an assembly program to sort an array using selection sort
2. Write an assembly program to find the factorial of an unsigned number using recursion
3. Write an assembly program to search an element in an array of ten 32 bit numbers using linear search.
4. Assume that ten 32 bit numbers are stored in registers R1-R10. Sort these numbers in the empty ascending stack using selection sort and store the sorted array back into the registers. Use STM and LDMDB instructions wherever necessary.

Additional Exercises:

LAB NO 5:

1. Repeat question 4 for a fully descending stack using STMDB and LDM instruction wherever necessary.
2. Write an ARM ALP that contains a list of numbers and makes a count of
   a) Even and Odd numbers. b) Numbers greater than 10
3. Implement Full ascending and empty descending stack using LDR, STR, ADD and SUB instructions.

LAB NO: 6 **Date:**

INTERFACING LED TO ARM MICROCONTROLLER.

Objectives:

In this lab, students will be able to

- Interface LEDs to the ARM cortex LPC1768 microcontroller using ALS interfacing board.

Steps to be followed

Project Creation in Keil uvision4 IDE:

- Create a project folder before creating a NEW project.
- Use separate folder for each project
- Open Keil uVision4 IDE software by double clicking on “Keil Uvision4” icon.
- Select “Project” then to “New Project” and save it with a name in the respective Project folder, which is already created.
- Select the device as “NXP (founded by Philips)” Select “LPC1768” then Press “OK” and then press “YES” button to add “system_LPC17xx.s” file.
- Go to “File” select “New” to open an editor window. Create a source file and use the header file “LPC17xx.h” in the source file and save the file. Color syntax highlighting will be enabled once the file is saved with a Recognized extension such as “.C “.
- Right click on “Source Group 1” and select the option “Add Files to Group 'Source Group 1' “add the. C source file(s) to the group.
- Again right click on Source Group 1 and select the option “Add Files to Group 'Source Group 1' “add the file - C:Keil\ARM\startup\NXP\LPC17xx\system_LPC17xx.c
- Any changes made to this file at the current project will directly change the source system_LPC17xx.C file. As a result other project settings may get altered. So it is recommended to copy the file C:Keil\ARM\startup\NXP\LPC17xx\system_LPC17xx.c to the project folder and add to the source group.
- **Important: This file should be added during each project creation.**
- Select “Project” then select “Translate” to compile the File (s).

- Select "Project" , select "Build Target" for building all source files such as ".C",".ASM", ".h", files, etc…This will create the hex file if there are no warnings & no errors.

Solved Exercise:

Write a program to turn on/off the LEDs serially.

Note: Before writing the program please check GPIO port pins available in the kit (Refer Appendix C.)

```
#include <LPC17xx.h>

unsigned int i,j;
unsigned long LED = 0x00000010;

int main(void)
{
        SystemInit();           //Add these two function for its internal operation
        SystemCoreClockUpdate();

        LPC_PINCON->PINSEL0 &= 0xFF0000FF;
                        //Configure  Port0 PINS  P0.4-P0.11 as GPIO function
        LPC_GPIO0->FIODIR |= 0x00000FF0;
                                    //Configure P0.4-P0.11 as output port
        while(1)
        {
              LED = 0x00000010; Initial value on LED
              for(i=1;i<9;i++)          //On the LED's serially
              {
                     LPC_GPIO0->FIOSET = LED;
                          // Turn ON LED at LSB (LED connected to p0.4)
                     for(j=0;j<10000;j++);a random delay
                     LED <<= 1; Shift the LED to the left by one unit
              }      //loop for 8 times

              LED = 0x00000010;
```

```
                for(i=1;i<9;i++)        //Off the LED's serially
                {
                        LPC_GPIO0->FIOCLR = LED;
                                //Turn OFF LED at LSB (LED connected to p0.4)
                        for(j=0;j<10000;j++);
                        LED <<= 1;
                }
        }
}
```

Some Settings to be done in KEIL µV-4 for Executing C programs :

- In Project Window Right click "TARGET1" and select "options for target 'TARGET1' select to option "Target" in that select
  1. XTAL 12.0MHz
  2. Select IROM1 (starting 0×0 size 0×8000).
  3. Select IRAM1 (starting 0×10000000 size 0×8000).
- Then go to option "Output"
  Select "Create Hex file".
- Then go to option "Linker"
  Select use memory layout from target dialog

Settings to be done at configuration wizard of system_LPC17xx.c file

- There are three clock sources for the CPU. Select Oscillator clock out of three. This selection is done by CLKSRCSEL register.
- If we disable the PLL0 System clock will be bypassed directly into the CPU clock divider register.
- Use CCLKCFG register for choosing the division factor of 4 to get 3MHz out of 12 MHz Oscillator frequency
- For any other peripherals use the PCLK same as CCLK.

Follow the steps specified below to carry out the settings.

- Double click on system_LPC17xx.c file at project window
- Select the configuration wizard at the bottom
- Expand the icons

- Select Clock configuration
- Under System controls and Status registers
  OSCRANGE: Main Oscillator range select 1MHz to 20MHz
  OSCEN: Main oscillator enable √
- Under Clock source select register (CLKSRCSEL)
  CLKSRC: PLL clock source selection Main oscillator
- Disable PLL0 configuration and PLL1 configuration
- Under CPU Clock Configuration register(CCLKCFG)
  CCLKSEL: Divide value for CPU clock for PLL0 4
- Under USB Clock configuration register (USBCLKCFG)
  USBSEL: Divide value for USB clock for PLL0 4
- Under Peripheral clock selection register 0 (PCLKSEL0) and 1 (PCLKSEL1) select Pclk = Cclk for all.
- Under Power control for peripherals (PCONP)
  Enable the power for required peripherals
- If CLKOUT to be studied configure the Clock output configuration register as below
  CLKOUTSEL : Main Oscillator
  CLKOUTDIV : 1
  CLKOUT_EN : √
- Call the functions
  SystemInit();
  SystemCoreClockUpdate(); at the beginning of the main function without missing. These functions are defined in system_LPC17xx.c where the actual clock and other system control registers are configured.
- A small change is required in the file system_LPC17xx.c after installation. Go to text editor:

```
#define PLL0_SETUP      0
#define PLL1_SETUP      0
```

  if the above #defines are 1 then make 0

Components required

• ALS-SDA-ARMCTXM3-01 :	1 No.
• Power supply (+5V) :	1 No.

• Cross cable for programming and serial communication: 1 No

• One working USB port in the host computer system and PC for downloading the software.

• 10 core FRC cables of 8 inch length 2 No

• USB to B type cable 1 No

Some Settings for downloading the program in FLASH MAGIC:

Step1. Connect 9 pin DSUB cross cable from PC to CN9 at the board.

Step2. On the 2 way dip switch SW21. Short jumper JP3

Step3. Open flash magic 6.01

Step4. Make following setting in Flash magic(Only once)

a. Communications:

Device:	LPC1768
Com Port:	COM1
Baud Rate:	9600
Interface:	None(ISP)
Oscillator:	12MHz

b. ERASE:

Select “Erase Blocks Used By Hex File”.

c. Hex file:

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

d. Options:

Select “Verify After Programming”.

Go to Options -> Advanced Options->communications

Do not select High Speed Communications, keep baud rate 115200.

Options -> Advanced Options->Hardware config

Select Use DTR & RTS to control RST & ISP Pin.

Select Keep RTS asserted while COM Port open.

T1 = 50ms. T2 = 100ms.

Step5. Start:

Click “Start” to download the hex file to the controller.

Step6. Connect one end of 10 pin FRC cable to CNA1, Short other end to CNA

Step7. Press reset controller switch SW1 and Check output on the LEDs connected to CNA1.

Lab Exercises:

1. Write a C program to display an 8-bit binary up counter on the LEDs.
2. Write a C program to read a key and display an 8-bit up/down counter on the LEDs.
   Hint: Use key SW2 (if SW2=1, up counter else down counter), which is available at CNB1 pin 7. Connect CNB1 to any controller connector like CNB, CNC, etc. Configure the corresponding port pin as GPIO using corresponding PINSEL register and as input pin using corresponding FIODIR register.
3. Write a program to simulate an 8- bit ring counter with key press (SW2).

LAB NO: 7. **Date:**

PROGRAMS ON MULTIPLEXED SEVEN SEGMENT DISPLAY

Objectives

In this lab students will be able to

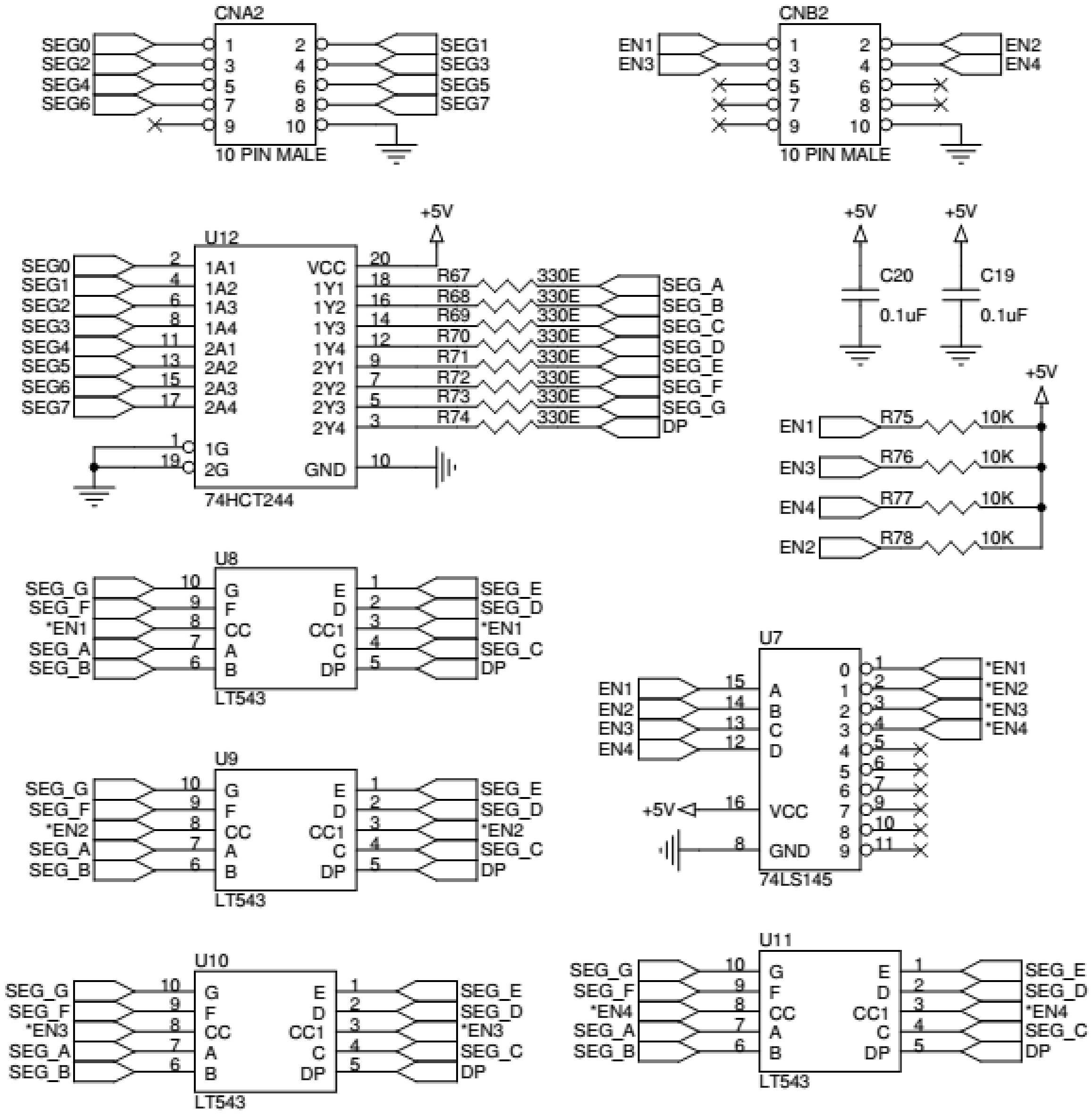
- Interface and understand the working of multiplexed seven segments display

Introduction:

There are four multiplexed 7-segment display units (U8, U9, U10 and U11) on the board. Each display has 8-inputs SEG_A (Pin-7), SEG_B (Pin-6), SEG_C (Pin-4), SEG_D (Pin-2), SEG_E (Pin-1), SEG_F (Pin-9), SEG_G (Pin-10) and SEG_H (Pin-5) and the remaining pins pin-3 & pin-8 are Common Cathode CC. These segments are common cathode type hence active high devices.

At power on all the segments are pulled up. A four bits input through CNB2 is used for multiplexing operation. A 1-of-10 Decoder/Driver U7 is used to accept BCD inputs and provide appropriate outputs for enabling the required display.

8 bits data is provided in this block using CNA2. All the data lines are taken buffered at U12 before giving to the displays.



SEVEN SEGMENT DISPLAY

At the controller end, any 2 connectors are required for interfacing this block.

Lookup Table for displaying 0,1,2,3 to 9

```
  value = h g f e d c b a          On 7-SEG U8,U9,U10 & U11.
 * 0x3F = 0 0 1 1 1 1 1 1   -> Displaying '0'
 * 0x06 = 0 0 0 0 0 1 1 0   -> Displaying '1'
 * 0x5B = 0 1 0 1 1 0 1 1   -> Displaying '2'
 * 0x4F = 0 1 0 0 1 1 1 1   -> Displaying '3'
 * 0x66 = 0 1 1 0 0 1 1 0   -> Displaying '4'
 * 0x6D = 0 1 1 0 1 1 0 1   -> Displaying '5'
 * 0x7D = 0 1 1 1 1 1 0 1   -> Displaying '6'
 * 0x07 = 0 0 0 0 0 1 1 1   -> Displaying '7'
 * 0x7F = 0 1 1 1 1 1 1 1   -> Displaying '8'
 * 0x6F = 0 1 1 0 1 1 1 1   -> Displaying '9'
```

Solved Exercise:

WAP to simulate 4-digit BCD up counter on the multiplexed seven segment display.

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

unsigned int seg_select[4] = {0<<23, 1<<23, 2<<23, 3<<23};
unsigned int dig1=0x00, dig2=0x00, dig3=0x00, dig4=0x00;
unsigned int seg_count=0x00, temp1=0x00;
unsigned char
array_dec[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
unsigned long int i=0;
void delay(void);
void display(void);

 int main(void)
 {
       SystemInit();
       SystemCoreClockUpdate();

       LPC_PINCON->PINSEL0 &= 0xFF0000FF;  //P0.4 to P0.11 GPIO data lines
       LPC_PINCON->PINSEL3 &= 0xFFC03FFF; //P1.23 to P1.26 GPIO enable lines

       LPC_GPIO0->FIODIR |= 0x00000FF0;           //P0.4 to P0.11 output
```

```
LPC_GPIO1->FIODIR |= 0x07800000;          //P1.23 to P1.26 output

  while(1)
  {
        delay();
        display();
        seg_count +=1;
        if(seg_count == 0x04)
        {
              seg_count = 0x00;
              dig1 +=1;
              if(dig1 == 0x0A)
              {
                    dig1 = 0;
                    dig2 +=1;

                    if(dig2 == 0x0A)
                    {
                          dig2 = 0;
                          dig3+=1;

                          if(dig3 == 0x0A)
                          {
                                dig3 = 0;
                                dig4 += 1;

                                if(dig4 == 0x0A)
                                {
                                      dig4 = 0;
                                } //end of dig4

                          } //end of dig3

                    } //end of dig2

              }  //end of dig1

        } //end of seg_count

  } //end of while(1)
```

```
}//end of main

void display(void)      //To Display on 7-segments
{
        LPC_GPIO1->FIOPIN = seg_select[seg_count];
        if(seg_count == 0x00) // For Segment U8
        {
                temp1 = dig1;
        }
        else if(seg_count == 0x01) // For Segment U9
        {
                temp1 = dig2;
        }
        else if(seg_count == 0x02) // For Segment U10
        {
                temp1 = dig3;
        }
        else if(seg_count == 0x03) // For Segment U11
        {
                temp1 = dig4;
        }

        LPC_GPIO0->FIOPIN = array_dec[temp1]<<4; // Taking Data Lines for 7-Seg
        for(i=0;i<500;i++);
}

void delay(void)
{       unsigned int i;
        for(i=0;i<10000;i++);
}
```

Components required

- ALS-SDA-ARMCTXM3-01 : 1 No.
- Power supply (+5V) : 1 No.
- Cross cable for programming and serial communication : 1 No
- One working USB in the host computer system and PC for downloading the software.
- 10 core FRC cables of 8 inch length 2 No

• USB to B type cable 1 No

Hardware setup: Connect a 10 core FRC cable from CNA to CNA2 and CNB to CNB2.

Working procedure: After software download and hardware setup, press the reset, Observe the count from 0000 to 9999 on the display.

Lab Exercises:

1. Write a C program to display the number "1234" serially in the seven segment display.
2. Write a C program to simulate a 4 digit BCD down counter. Use a timer for delay.
3. Write a C program for 4 digit BCD up/down counters on seven segment using a switch and timer with a delay of 1-second between each count.
4. Write a program for 4 digit Hexadecimal up/down counters on seven segment using a switch and timer with a delay of 1-second between each count.

LAB NO: 8: **Date:**

LIQUID CRYSTAL DISPLAY (LCD) AND KEYBOARD INTERFACING

Objectives:

In this lab students will be able to

- Interface and understand the working of LCD and matrix keyboard

Introduction:

LCD: A 16×2 alphanumeric LCD can be used to display the message from the controller. 16 pin small LCD has to be mounted to the connector CN11. 10 pin connector CNAD is used to interface this LCD from the controller. Only higher 4 data lines are used among the 8 LCD data lines. Use POT3 for contrast adjustment and Short the jumper JP16 to use this LCD. LCD connector CN11 is described in this table. CN11 is a single row 16 pin female berg.

Pin no CN11	Description
1	Ground
2	+5V
3	LCD contrast
4	RS
5,7,8,9,10	NC
6	En
11 to 14	Data 4 to 7
15	Back light anode
16	Back light cathode

Connection from CNAD to LCD connector CN11 is shown below

Pin no at CNAD	Description	Pin no at CN11
1	L0 – Data line 4 of LCD	11
2	L1 – Data line 5 of LCD	12
3	L2 – Data line 6 of LCD	13
4	L3 – Data line 7 of LCD	14
5	L5 – Command line of LCD	4
6	L5 – Enable line of LCD	6

	Code											
Instruction	RS	R/$\overline{\text{W}}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

	Code											
Instruction	RS	R/$\overline{\text{W}}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μs

Solved Exercise:

WAP to display message on LCD

```
#include <lpc17xx.h>

void lcd_init(void);
void write(int, int);
void delay_lcd(unsigned int);
void lcd_comdata(int, int);
void clear_ports(void);
void lcd_puts(unsigned char *);

int main(void)
{
        unsigned char Msg1[4] = {"MIT"};
        unsigned char Msg2[19] = {"Department of CSE:"};

        SystemInit();
        SystemCoreClockUpdate();
        lcd_init();
        lcd_comdata(0x80, 0);
        delay_lcd(800);
        lcd_puts(&Msg1[0]);

        lcd_comdata(0xC0, 0);
        delay_lcd(800);
        lcd_puts(&Msg2[0]);
}
//lcd initialization
void lcd_init()
{
        /*Ports initialized as GPIO */
        LPC_PINCON->PINSEL3 &= 0xFC003FFF;  //P0.23 to P0.28

        /*Setting the directions as output */
        LPC_GPIO0->FIODIR |= 0x0F<<23 | 1<<27 | 1<<28;

        clear_ports();
        delay_lcd(3200);
        lcd_comdata(0x33, 0);
        delay_lcd(30000);

        lcd_comdata(0x32, 0);
        delay_lcd(30000);

        lcd_comdata(0x28, 0);           //function set
        delay_lcd(30000);

        lcd_comdata(0x0c, 0);//display on cursor off
        delay_lcd(800);

        lcd_comdata(0x06, 0);           //entry mode set increment cursor right
        delay_lcd(800);

        lcd_comdata(0x01, 0);           //display clear
        delay_lcd(10000);

        return;
 }

void lcd_comdata(int temp1, int type)
{
        int temp2 = temp1 & 0xf0; //move data (26-8+1) times : 26 - HN  place, 4 - Bits
        temp2 = temp2 << 19;            //data lines from 23 to 26
        write(temp2, type);
        temp2 = temp1 & 0x0f;           //26-4+1
        temp2 = temp2 << 23;
        write(temp2, type);
        delay_lcd(1000);
        return;
}

void write(int temp2, int type)                 //write to command/data reg
 {
        clear_ports();
        LPC_GPIO0->FIOPIN = temp2; // Assign the value to the data lines
        if(type==0)
                LPC_GPIO0->FIOCLR = 1<<27;      // clear bit RS for Command
	else
		LPC_GPIO0->FIOSET = 1<<27;	// set bit RS for Data

	LPC_GPIO0->FIOSET = 1<<28;	// EN=1
	delay_lcd(25);
	LPC_GPIO0->FIOCLR  = 1<<28;	// EN =0
	return;
}

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

void clear_ports(void)
{
	/* Clearing the lines at power on */
	LPC_GPIO0->FIOCLR = 0x0F<<23; //Clearing data lines
	LPC_GPIO0->FIOCLR = 1<<27;  //Clearing RS line
	LPC_GPIO0->FIOCLR = 1<<28; //Clearing Enable line

	return;
}

void lcd_puts(unsigned char *buf1)
{
	unsigned int i=0;
	unsigned int temp3;
	while(buf1[i]!='\0')
	{
		temp3 = buf1[i];
		lcd_comdata(temp3, 1);
		i++;
		if(i==16)
		{
			lcd_comdata(0xc0, 0);
		}
```

```
        }
        return;
}
```

Components required

• ALS-SDA-ARMCTXM3-01 :	1 No.
• Power supply (+5V) :	1 No.
• Cross cable for programming and serial communication:	1 No
• One working USB port in the host computer system and PC for downloading the software.	
• 10 core FRC cables of 8 inch length	2 No
• USB to B type cable	1 No

Hardware setup:

Connect 10 pin FRC cable from CND to CNAD. Short the jumper JP16 & JP5. Use POT3 for contrast adjustment.

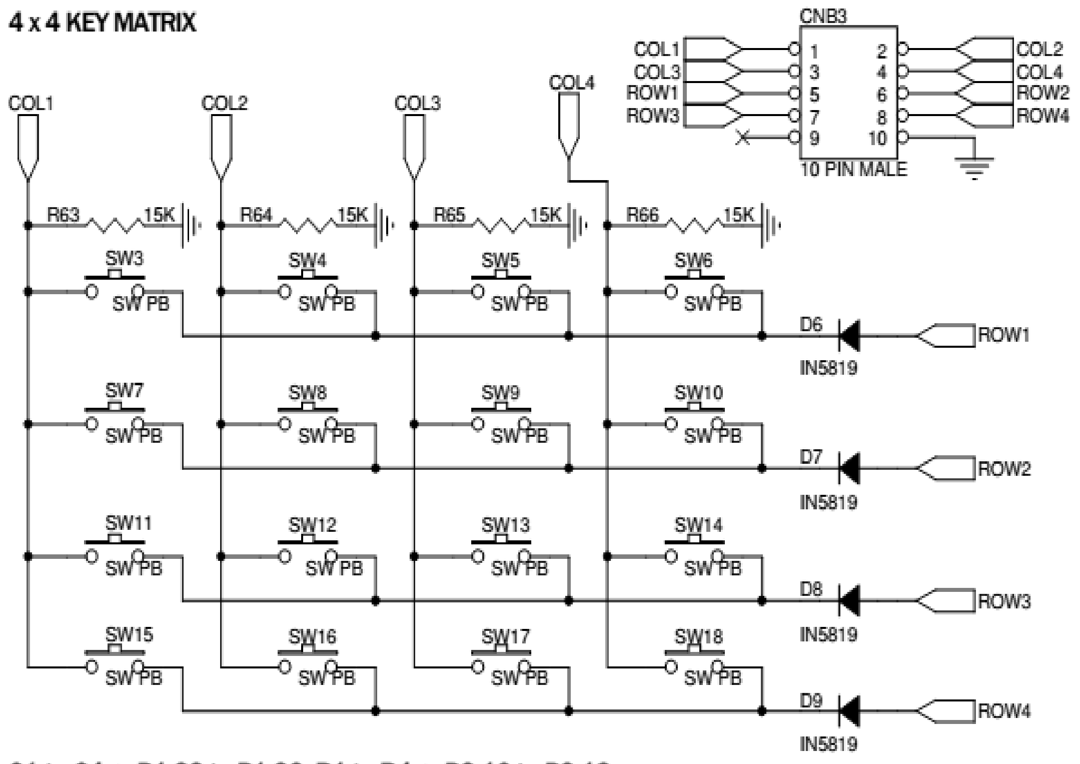
Working procedure: After software download and hardware setup, press the reset. A fixed message will display on the LCD.

Exercise Questions:

1. Simulate DIE tossing on LCD.
   Hint: Program reads the external interrupt using the key SW2. A random number between 0-6 should be displayed on the LCD upon keypress.

Keyboard connection: The switches SW3 to SW18 are organized as 4 rows X 4 columns matrix. One end of all the switches are configured as columns. The other end of the matrix is configured as rows. A row line will always be an output from the controller. Column lines are pulled to ground. A high level sent from the row will appear at column end if the switch is pressed.

Connector CNB3 is used for interfacing this block with the controller. At the controller end any connector can be used to interact with this connector CNB3.



C1 to C4 -> P1.23 to P1.26. R1 to R4 -> P2.10 to P2.13

Solved Exercise:
WAP to read a key from the matrix keyboard and display its key code on the LCD.

```
#include <LPC17xx.h>
#include "lcd-disp.c"     // use all the functions of lcd program
 void scan(void);
 unsigned char Msg1[13] = "KEY PRESSED=";
 unsigned char row, var, flag, key;
 unsigned long int i, var1, temp, temp1, temp2, temp3;
 unsigned char SCAN_CODE[16] = {0x11,0x21,0x41,0x81,
                                0x12,0x22,0x42,0x82,
                                0x14,0x24,0x44,0x84,
                                0x18,0x28,0x48,0x88};
```

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

int main(void)
{
        LPC_GPIO2->FIODIR |= 0x00003C00; //made output P2.10 to P2.13 (rows)
        LPC_GPIO1->FIODIR &= 0xF87FFFFF; //made input P1.23 to P1.26(cols)
                                         ;not required since it is by default
        LPC_GPIO0->FIODIR |= 0x0F<<23 | 1<<27 | 1<<28;
        clear_ports();
        delay_lcd(3200);

        lcd_init();

        lcd_com(0x80); //point to first line of LCD
        delay_lcd(800);

        lcd_puts(&Msg1[0]);   //display the message

        while(1)
        {
                while(1)
                {
                        for(row=1;row<5;row++)
                        {
                                if(row == 1)
                                        var1 = 0x00000400;
                                else if(row == 2)
                                        var1 = 0x00000800;
                                else if(row == 3)
                                        var1 = 0x00001000;
                                else if(row == 4)
                                        var1 = 0x00002000;

                                temp = var1;

                                LPC_GPIO2->FIOCLR = 0x00003C00;   //first clear the
port and send appropriate value for
                                LPC_GPIO2->FIOSET = var1;      //enabling the row
                                flag = 0;
                                scan();    //scan if any key pressed in the enabled row
                                if(flag == 1)
                                break;
                        } //end for
                        if(flag == 1)
                        break;
                } //2nd while(1)
                for(i=0;i<16;i++)        //get the ascii code for display
                {
                        if(key == SCAN_CODE[i])
                        {
                                key = ASCII_CODE[i];
                                break;
                        } //end if(key == SCAN_CODE[i])
                }//end for(i=0;i<16;i++)

                lcd_com(0xc0);//display in the second line
                delay_lcd(800);
                lcd_puts(&key);
        }//end while 1
}//end main

void scan(void)
{
        temp3 = LPC_GPIO1->FIOPIN;
        temp3 &= 0x07800000;   //check if any key pressed in the enabled row
        if(temp3 != 0x00000000)
        {
                        flag = 1;
                        temp3 >>= 19;//Shifted to come at HN of byte
                        temp >>= 10;  //shifted to come at LN of byte
                        key = temp3|temp;      //get SCAN_CODE
        }//if(temp3 != 0x00000000)
    }//end scan
```

Components required

• ALS-SDA-ARMCTXM3-01 :	1 No.
• Power supply (+5V) :	1 No.
• Cross cable for programming and serial communication :	1 No
• One working USB port in the host computer system and PC for downloading the software.	
• 10 core FRC cables of 8 inch length	2 No
• USB to B type cable	1 No

Hardware setup: Connect 10 core FRC cable from CNB to CNB3, short JP4(1, 2)

Connect another 10 core FRC cable from CND to CNAD, Short the jumper JP16 & JP5. Use POT3 for contrast.

Working procedure: After software download and hardware setup, use the reset. Identity of key pressed (0 to F) will be displayed on LCD.

Lab Exercise:

1. Write a program to input an expression of the type A operator B =, from the keyboard, where A and B are the single digit BCD numbers and operator may be + or -. Display the result on the LCD.

LAB NO: 9 **Date:**

ANALOG TO DIGITAL CONVERTOR PROGRAM

Objectives:

In this lab students will be able to

- Understand the working of a 12 bit internal Analog-to-Digital Converter (ADC)

Introduction: The LPC1768 contains a single 12-bit successive approximation ADC with eight channels and DMA support. 12-bit ADC with input multiplexing among eight pins, conversion rates up to 200 kHz, and multiple result registers. The 12-bit ADC can be used with the GPDMA controller. On board there are two interfaces for internal ADC's. AD0.5 (pin P1.31) of the controller is used to convert the analog input voltage varied using POT1 to digital value. AD0.4(Pin 1.30) used to convert the analog voltage varied using POT4. An input voltage range of 0 to 3.3V is accepted. 000 to FFF is the converted digital voltage range here. Short JP18 (2, 3) to use AD0.4.

Solved Exercise:

WAP to configure and read analog data from ADC channel no 5, and display the digital data on the LCD.

```
#include<LPC17xx.h>
#include<stdio.h>
#include"AN_LCD.h"
#define Ref_Vtg         3.300
#define Full_Scale      0xFFF                   //12 bit ADC

int main(void)
{
        unsigned long adc_temp;
        unsigned int i;
        float in_vtg;
        unsigned char vtg[7], dval[7];
        unsigned char Msg3[11] = {"ANALOG IP:"};
        unsigned char Msg4[12] = {"ADC OUTPUT:"};

        SystemInit();
        SystemCoreClockUpdate();
```

```
LPC_SC->PCONP |= (1<<15);                    //Power for GPIO block
lcd_init();
LPC_PINCON->PINSEL3 |= 0xC0000000;           //P1.31 as AD0.5
LPC_SC->PCONP |= (1<<12);                    //enable the peripheral ADC

SystemCoreClockUpdate();

lcd_comdata(0x80, 0);
delay_lcd(800);
lcd_puts(&Msg3[0]);

lcd_comdata(0xC0, 0);
delay_lcd(800);
lcd_puts(&Msg4[0]);

while(1)
{
        LPC_ADC->ADCR = (1<<5)|(1<<21)|(1<<24);      //0x01200001;
                              //ADC0.5, start conversion and operational
        while(!(LPC_ADC->ADDR5 & 0x80000000));
                //wait till 'done' bit is 1, indicates conversion complete
        adc_temp = LPC_ADC->ADDR5;
        adc_temp >>= 4;
        adc_temp &= 0x00000FFF;                      //12 bit ADC
        in_vtg = (((float)adc_temp * (float)Ref_Vtg))/((float)Full_Scale);
                                  //calculating input analog voltage
        sprintf(vtg, "%3.2fV", in_vtg);
                    //convert the readings into string to display on LCD
        sprintf(dval, "%x", adc_temp);
        for(i=0; i<2000; i++);

        lcd_comdata(0x89, 0);
        delay_lcd(800);
        lcd_puts(&vtg[0]);

        lcd_comdata(0xC8, 0);
        delay_lcd(800);
        lcd_puts(&dval[0]);
```

```
            for(i=0;i<200000;i++);
            for(i=0;i<7;i++)
                vtg[i] = dval[i] = 0x00;
            adc_temp = 0;
            in_vtg = 0;
        }
}
```

Components required

● ALS-SDA-ARMCTXM3-01 :	1 No.
● Power supply (+5V) :	1 No.
● Cross cable for programming and serial communication :	1 No
● One working COM port (Ex: COM1) in the host computer system and PC for downloading the software.	
● 10 core FRC cables of 8 inch length	2 No
● USB to B type cable	1 No

Hardware Setup: Do the setup related to LCD

Working procedure: Vary POT1 and observe the corresponding analog and digital voltage values on LCD.

Exercise

1. Write a C program to display the digital value representing the difference in analog voltages at ADC channel 4 and channel 5 on LCD using BURST and Software mode.

LAB NO: 10 **Date:**

PROGRAM ON DIGITAL TO ANALOG CONVERTOR (DAC)

Objectives:

In this lab students will be able to

- Understand the working of a 10 bit DAC and check the waveform on Cathode Ray Oscilloscope (CRO).

Introduction: LPC1768 has 10 bit internal DAC with dedicated conversion timer and DMA support. The DAC allows to generate a variable analog output. The maximum output value of the DAC is VREFP. The equation to calculate output voltage value is given as below.

AOUT = DACR value x ((VREFP - VREFN)/1024) + VREFN

An analog output from the controller can be observed in this block at TP8. Open JP5 to use this feature and use CRO to watch analog output value.

Solved Exercise:

WAP to generate a sawtooth waveform using DAC and display it on CRO.

```
#include <lpc17xx.h>
void delayMS(unsigned int milliseconds);
int main(void)
{
unsigned int value=0; //Binary value used for Digital to Analog Conversion
LPC_PINCON->PINSEL1 |= (1<<21); //Select AOUT function for P0.26 , bits[21:20]
= [10]
while(1)
      {
            if(value > 1023) value=0; //For 10-bit DAC max-value is 2^10 - 1 = 1023
            LPC_DAC->DACR = (value<<6);
            delayMS(9999); //for 10 millisecond delay
            value++;
      }
}
void delayMS(unsigned int count) //Using Timer0
{
	LPC_TIM0->CTCR = 0x0; //Timer mode
	LPC_TIM0->PR = 2; //Increment TC at every 3 pclk
	LPC_TIM0->TCR = 0x02; //Reset Timer
	LPC_TIM0->TCR = 0x01; //Enable timer
	while(LPC_TIM0->TC < count); //wait until timer counter reaches the desired
//delay
	LPC_TIM0->TCR = 0x00; //Disable timer
}
```

Components required

• ALS-SDA-ARMCTXM3-01 :	1 No.
• Power supply (+5V) :	1 No.
• Cross cable for programming and serial communication :	1 No
• One working USB port in the host computer system and PC for downloading the software.	
• 10 core FRC cables of 8 inch length	2 No
• USB to B type cable	1 No

Hardware setup:

Open the jumper JP5

Connect TP8 pin to CRO positive wire and TP3 to CRO negative wire. Scale the CRO to the proper display

Working procedure: Reset the controller and observe the analog output waveform on CRO.

Lab Exercises:

1. Using DAC, generate a triangular waveform with maximum possible peak-peak amplitude.
2. Using DAC, generate a ramp waveform with maximum possible peak-peak amplitude.

Additional Exercise:

Using DAC, generate a variable frequency sine waveform. Use ROW-0 of keyboard for frequency variation

LAB NO: 11 **Date:**

PROGRAM ON PULSE WIDTH MODULATION (PWM)

Objectives:

In this lab students will be able to

- Interface and understand the working of PWM

Introduction: The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on theLPC1768. The Timer is designed to count cycles of the system derived clock and optionally switch pins, generate interrupts or perform other actions when the specified timer values occur, based on seven match registers. The PWM function is in addition to these features, and is based on match register events. A PWM output from the controller can be observed as an intensity variation of the LED LD10.

Solved Exercise:

WAP to vary the intensity of an LED using PWM.

```
#include <LPC17xx.h>

void initPWM(void);
void updatePulseWidth(unsigned int pulseWidth);
void delayMS(unsigned int milliseconds);

int main(void)
{
	int pulseWidths[] = {0, 3000, 6000, 9000, 12000, 15000, 18000, 21000, 24000, 
27000}; //Pulse Widths for varying LED Brightness
	const int numPulseWidths = 10;
	int count=1;
	int dir=0; //direction, 0 = Increasing, 1 = Decreasing
	initPWM(); //Initialize PWM
	while(1)
	{
		updatePulseWidth(pulseWidths[count]); //Update LED Pulse Width
		delayMS(10000);
```

```
			if(count == (numPulseWidths-1) || count == 0)
			{
				dir = !dir; //Toggle direction if we have reached count limit
			}

			if(dir) count--;
			else count++;
		}
}

void initPWM(void)
{
		LPC_PINCON->PINSEL3 |= 0x8000; //Select PWM1.4 output for Pin1.23,
function 2
		LPC_PWM1->PCR = 0x1000; //enable PWM1.4, by default it is single Edged
		LPC_PWM1->PR = 0;
		LPC_PWM1->MR0 = 30000; //period=10ms if pclk=cclk/4
		LPC_PWM1->MCR = (1<<1); //Reset PWM TC on PWM1MR0 match
		LPC_PWM1->LER = 0xff; //update values in MR0 and MR1
  LPC_PWM1->TCR = 0x00000002; //RESET COUNTER TC and PC
		LPC_PWM1->TCR = 0x00000009; //enable TC and PC
}

void updatePulseWidth(unsigned int pulseWidth)
{
		LPC_PWM1->MR4 = pulseWidth; //Update MR4 with new value
		LPC_PWM1->LER = 0xff; //Load the MR4 new value at start of next cycle
}

void delayMS(unsigned int milliseconds) //Using Timer0
{
		LPC_TIM0->CTCR = 0x0; //Timer mode
		LPC_TIM0->PR = 2; //Increment TC at every 3 pclk
```

```
	LPC_TIM0->TCR = 0x02; //Reset Timer
	LPC_TIM0->TCR = 0x01; //Enable timer
	while(LPC_TIM0->TC < milliseconds); //wait until timer counter reaches the desired delay
	LPC_TIM0->TCR = 0x00; //Disable timer
}
```

Hardware setup: Connect 10 pin FRC cable from CNB to CNB1.

Working procedure: As the pulse width varies, intensity of LED LD10 varies. Observe the pulses at TP5. Observe the amplitude level at TP6.

Lab Exercises:

Write a program to set the following intensity levels to the LED connected to PWM output. Use ROW-0 of keyboard for intensity variation

Intensity level	Key pressed
10%	0
25%	1
50%	2
75%	3

LAB NO: 12 **Date:**

PROGRAM ON STEPPER MOTOR

Objectives

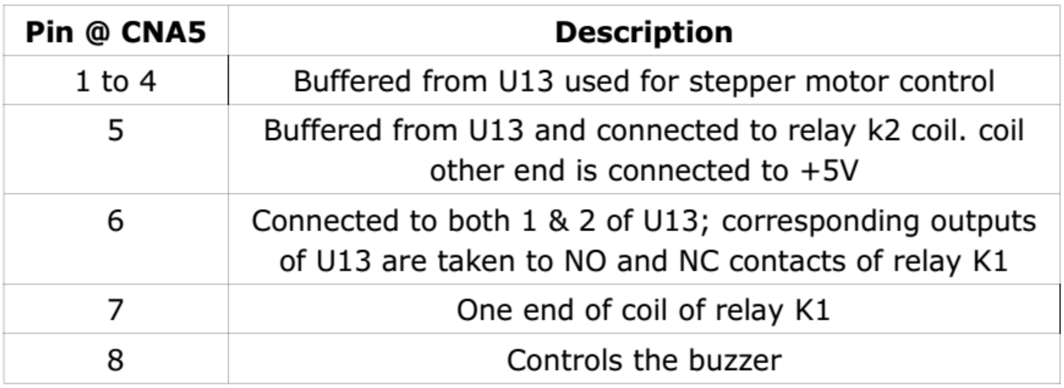
In this lab students will be able to

- Interface and understand the working of stepper motor

Introduction: The Stepper motor can be interfaced to the board by connecting it to the Power Mate PM1. The direction of the rotation can be changed through software. The DC Motor can also be interfaced to the board by connecting it to the Reliamate RM5. The direction of the rotation can be changed through software.

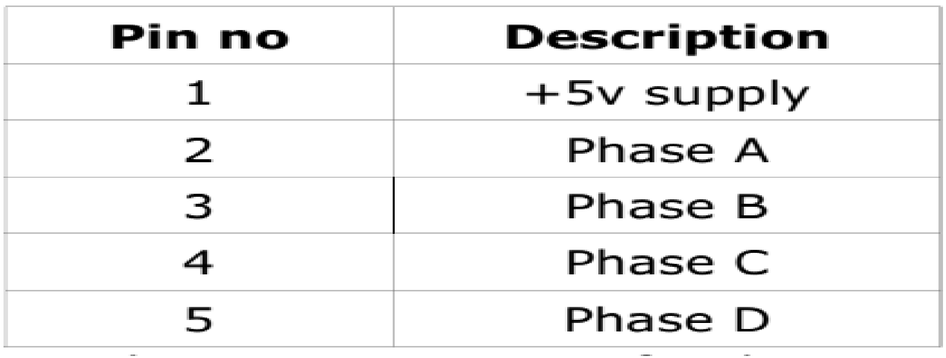
The Relay K2 is switched between ON and OFF state. The LED L12 will toggle for every relay switch over. The contact of NO & NC of the relay can be checked at the MKDSN connector CN12 pins 1 & 2 using a CRO– these contacts can be connected to external devices. Using connector CNA5 micro controller can interface with this block.

Description of the connector pins are given in below table.



Pin @ CNA5	Description
1 to 4	Buffered from U13 used for stepper motor control
5	Buffered from U13 and connected to relay k2 coil. coil other end is connected to +5V
6	Connected to both 1 & 2 of U13; corresponding outputs of U13 are taken to NO and NC contacts of relay K1
7	One end of coil of relay K1
8	Controls the buzzer

PM1– it's a 5 pin straight male power mate. PIN descriptions are as given below.



Pin no	Description
1	+5v supply
2	Phase A
3	Phase B
4	Phase C
5	Phase D

Pin 2 to 5 are phase A to D output for the stepper motor respectively.

Sample program: To rotate the stepper motor in clockwise and anticlockwise direction at a particular speed continuously.

```
#include <LPC17xx.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)
 {
       SystemInit();
       SystemCoreClockUpdate();

       LPC_PINCON->PINSEL0 = 0xFFFF00FF;  //P0.4 to P0.7 GPIO
       LPC_GPIO0->FIODIR = 0x000000F0;        //P0.4 to P0.7 output

       while(1)
       {
             for(j=0;j<50;j++)
                   clock_wise();

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

```
            for(j=0;j<50;j++)
                    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 = 0x00000008;        //For Clockwise
  for(i=0;i<=3;i++)        // for A B C D Stepping
    {
            var1 = var1<<1;       //For Clockwise
            LPC_GPIO0->FIOPIN = var1;

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

}

void anti_clock_wise(void)
{
    var1 = 0x00000100;     //For Anticlockwise
   for(i=0;i<=3;i++)       // for A B C D Stepping
    {
     var1 = var1>>1;      //For Anticlockwise
      LPC_GPIO0->FIOPIN = var1;

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

```
}
```

Components required

• ALS-SDA-ARMCTXM3-01 :	1 No.
• Power supply (+5V) :	1 No.
• Cross cable for programming and serial communication:	1 No
• Stepper motor	1 No
• One working USB port in the host computer system and PC for downloading the software.	
• 10 core FRC cables of 8 inch length	2 No
• USB to B type cable	1 No

Hardware setup: Connect 10 pin FRC cable from CNA to CNA5. Connect the stepper motor to PM1.

Working procedure: Stepper motor will rotate clockwise and in anti-clock wise direction automatically after reset.

Lab Exercise:

Write a C program to rotate the stepper motor in the clockwise direction when SW2 is high and anticlockwise direction when SW2 is low.

Appendix A: Instructions

Instruction Set Summary

Mnemonic	Operation	Description
ADC	Rd := Rn + Op2 + C	Add with carry
ADD	Rd := Rn + Op2	Add
ADDS	Rd: = Rn+ Op2	Add and update falgs
ADR	Rd: = Rn, label	Load register with adress
AND	Rd := Rn AND Op2	AND
ANDS	Rd := Rn AND Op2	AND update flags
ASR	Rd: = Rn,#LSB,#width	Arithmetic shift right
B	R15 := address	Branch
BIC	Rd := Rn AND NOT Op2	Bit Clear
BL	R14 := address of next instruction, R15 :=address	Branch with Link
BX	R15 := Rn, change to Thumb if address bit 0 is 1	Branch and Exchange
CLZ	Rd := number of leading zeroes in Rm	Count Leading Zeroes
CMN	CPSR flags := Rn + Op2	Compare Negative
CMP	CPSR flags := Rn - Op2	Compare
EOR	Rd:= Rn EOR Op2	Exclusive OR
LDM	Stack manipulation (Pop)	Load multiple Registers (refer last paragraph of this appendix)
LDMIA	LDMIA Rn!, {reglist}	Load multiple registers from memory
LDR	Rd := [address][31:0]	Load 32-bit word from memory.
LDRB	Rd := ZeroExtend ([address][7:0])	Load register byte value to Memory.
LDRH	Rd := ZeroExtend ([address][15:0])	Load register 16-bit halfword value to Memory.

MCR	cRn:=rRn {<op>cRm}	Move CPU register to coprocessor register
MLA	Rd := (Rm * Rs) + Rn	Multiply Accumulate
MOV	Rd := Op2	Move register or constant
MRS	Rn := PSR	Move PSR status flags to register
MSR	PSR := Rm	Move register to PSR status flags
MUL	Rd := Rm * Rs	Multiply
MVN	Rd := NOT Rm	Move inverted register or constant
NOP	None	No operation
ORR	Rd:=Rn OR Op2	OR
PUSH	PUSH {reg list}	Push registers on to the stack pointed by R13
POP	POP{reg. list}	Pop registers from the stack pointed by R13
RSB	Rd := Op2 – Rn	Reverse Subtract
RSC	Rd := Op2 - Rn - 1+Carry	Reverse Subtract with Carry
RBIT	RBIT Rd, Rn	Reverse the bit order in a 32-bit word
REV	REV Rd, Rn	converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data
ROR	Rd: = Rd, Rs	Rotate Rd register by Rs bits
RRX	Rd: = Rd, Rm	Rotate Right with Extend
SBC	Rd := Rn - Op2 - 1+Carry	Subtract with Carry
STM	stack manipulation (Push)	Store Multiple (refer last paragraph of this appendix
STR	<address>:=Rd	Store register to memory
STRB	[address][7:0] := Rd[7:0]	Store register byte value to Memory.
STRH	[address][15:0] :=Rd[15:0]	Store register 16-bit halfword value to Memory
SUB	Rd := Rn - Op2	Subtract

TEQ	CPSR flags:= Rn EOR Op2	Test bitwise equality
TST	CPSR flags:= Rn AND Op2	Test bits
UMULL	UMULL r0, r4, r5, r6	Unsigned Long Multiply

A conditional instruction is only executed on match of the condition flags in the Program Status Register. For example, the BEQ (B instruction with EQ condition) branches only if the Z flag is set. If the {cond} field is empty the instruction is always executed.

{cond} Suffix	Tested Status Flags	Description
EQ	Z set	equal
NE	Z clear	not equal
CS/HS	C set	unsigned higher or same
CC/LO	C clear	unsigned lower
MI	N set	negative
PL	N clear	positive or zero
VS	V set	overflow
VC	V clear	no overflow
HI	C set and Z clear	unsigned higher
LS	C clear or Z set	unsigned lower or same
GE	N equals V	signed greater or equal
LT	N not equal to V	signed less than
GT	Z clear AND (N equals V)	signed greater than
LE	Z set OR (N not equal to V)	signed less than or equal
AL	(ignored)	always (usually omitted)

Addressing Mode for LDM and STM

The instructions LDM and STM provide four different addressing modes. The addressing mode specifies the behavior of the base register and is explained in the following table.

Addressing Mode	Description

IA	Increment base register after instruction execution.
IB	Increment base register before instruction execution.
DA	Decrement base register after instruction execution.
DB	Decrement base register before instruction execution.

Examples: STMDB R2!,{R4,R5,LR}

```
LDMIA   R0!,{R1-R5}

STMDB   R6!,{R0,R1,R5}
```

Appendix B: Addressing modes

Name	Alternative Name	ARM Examples
Register to register	Register direct	MOV R0, R1
Absolute	Direct	LDR R0, MEM
Literal	Immediate	MOV R0, #15 ADD R1, R2, #12
Indexed, base	Register indirect	LDR R0, [R1]
Pre-indexed, base with displacement	Register indirect with offset	LDR R0, [R1, #4]
Pre-indexed, autoindexing	Register indirect pre-incrementing	LDR R0, [R1, #4]!
Post-indexing, autoindexed	Register indirect post-increment	LDR R0, [R1], #4
Double Reg indirect	Register indirect Register indexed	LDR R0, [R1, R2]
Double Reg indirect with scaling	Register indirect indexed with scaling	LDR R0, [R1, r2, LSL #2]
Program counter relative		LDR R0, [PC, #offset]

Literal Addressing

In this addressing mode data is a part of instruction. '#' symbol is used to indiacate the data. ARM and Thumb instructions can only be 32 bits wide. You can use a MOV or MVN instruction to load a register with an immediate value from a range that depends on the instruction set. Certain 32-bit values cannot be represented as an immediate operand to a single 32-bit instruction, although you can load these values from memory in a single instruction. you can load any 32-bit immediate value into a register with two instructions, a MOV followed by a MOVT. Or, you can use a pseudo-instruction, MOV32, to construct the instruction sequence for you. You can also use the LDR pseudo-instruction to load immediate values into a register

Examples	Meaning
CMP R0, #22	;Compare Register content R0 with 22
ADD R1, R2, #18	;Add the content of R2 and 18 then store ;the result in R1
MOV R1, #30	;copy the data 30 into register R1
MOV R1, #0Xff	;copy the data ff in hexadecimal into R1
MOV R2, #0xFF0000FF	
AND R0, R1, #0xFF000000	
CMN R0, #6400	; update the N, Z, C and V flags
CMPGT SP, R7, LSL #2	; update the N, Z, C and V flags

- MOV can load any 8-bit immediate value, giving a range of 0x0-0xFF (0-255). It can also rotate these values by any even number. These values are also available as immediate operands in many data processing operations, without being loaded in a separate instruction.
- MVN can load the bitwise complements of these values. The numerical values are -(n+1), where n is the value available in MOV.
- A MOVT instruction that can load any value in the range 0x0000 to 0xFFFF into the most significant half of a register, without altering the contents of the least significant half.
- The LDR Rd,=const pseudo-instruction generates the most efficient single instruction to load any 32-bit number

Introduction to Register Indirect Addressing : Register indirect addressing means that the location of an operand is held in a register. It is also called indexed addressing or base addressing.

Register indirect addressing mode requires three read operations to access an operand. It is very important because the content of the register containing the pointer to the operand can be modified at runtime. Therefore, the address is a vaiable that allows the access to the data structure like arrays.

- Read the instruction to find the pointer register
- Read the pointer register to find the oprand address
- Read memory at the operand address to find the operand

Some examples of using register indirect addressing mode:

```
          LDR      R2, [R0] ; Load R2 with the word pointed by R0
---------------------------------------------------------------------
          STR      R2, [R3] ; Store the word in R2 in the location
                                        ; pointed by R3
---------------------------------------------------------------------
```

LDR Rd,=label can load any 32-bit numeric value into a register. It also accepts PC-relative expressions such as labels, and labels with offsets

Register Indirect Addressing with an Offset

ARM supports a memory-addressing mode where the effective address of an operand is computed by adding the content of a register and a literal offset coded into load/store instruction. For example,

```
          Instruction                    Effective Address
--------------------------------------------------------------------
          LDR R0, [R1, #20]              R1 + 20    ; loads R0 with the word
                             ; pointed at by R1+20
--------------------------------------------------------------------
```

ARM's Autoindexing Pre-indexed Addressing Mode

This is used to facilitate the reading of sequential data in structures such as arrays, tables, and vectors. A pointer register is used to hold the base address. An offset can be added to achieve the effective address. For example,

```
          Instruction                    Effective Address
-------------------------------------------------------------------
          LDR R0, [R1, #4]!       R1 + 4          ; loads R0 with the word
                               ;pointed at by R1+4 then
                               ;update the pointer
                               ;by adding 4 to R1
-------------------------------------------------------------------
```

ARM's Autoindexing Post-indexing Addressing Mode

This is similar to the above, but it first accesses the operand at the location pointed by the base register, then increments the base register. For example,

```
          Instruction                    Effective Address
-------------------------------------------------------------------
          LDR R0, [R1], #4       R1        ;loads R0 with the word
                             ;pointed at by R1 then
                             ;update the pointed by
                                                        ;adding 4 to R1
-------------------------------------------------------------------
```

Program Counter Relative (PC Relative) Addressing Mode

Register R15 is the program counter. If you use R15 as a pointer register to access operand, the resulting addressing mode is called PC relative addressing. The operand is specified with respect to the current code location. Please look at this example,

```
          Instruction                    Effective Address
-------------------------------------------------------------------
          LDR R0, [R15, #24]      R15 + 24  ;loads R0 with the word
                                                       ;pointed at by R1+24
-------------------------------------------------------------------
```

APPENDIX C

GPIO extension connectors:

There are four 10 pin FRC type male connectors, they extends the controllers general purpose port lines for the use of user requirements. Details on each connector is given below:

CNA –10 pin male box type FRC connector. Port lines P0.4 to P0.11 from controller are terminated in this connector. They can be extended to interface few on board or external peripherals. The pins mentioned in the above table are configured to work as a GPIO's at power on. Other alternate functions on those pins needs to be selected using respective PINSEL registers.

Pin CNA	PIN LPC1768	Description
1	81	P0.4/I2SRX_CLK/RD2/CAP2.0
2	80	P0.5/I2SRX_WS/TD2/CAP2.1
3	79	P0.6/I2SRX_SDA/SSEL1/MAT2.0
4	78	P0.7/I2STX_CLK/SCK1//MAT2.1
5	77	P0.8/I2STX_WS/MISO1/MAT2.2
6	76	P0.9/I2STX_SDA/MOSI1/MAT2.3
7	48	P0.10/TXD2/SDA2/MAT3.0
8	49	P0.11/RXD2/SCL2/MAT3.1
9	-	No connection
10	-	Ground

CNB – 10 pin male box type FRC connector. Port lines fromP1.23 to P1.26 and P2.10 to P2.13 are terminated in this connector.

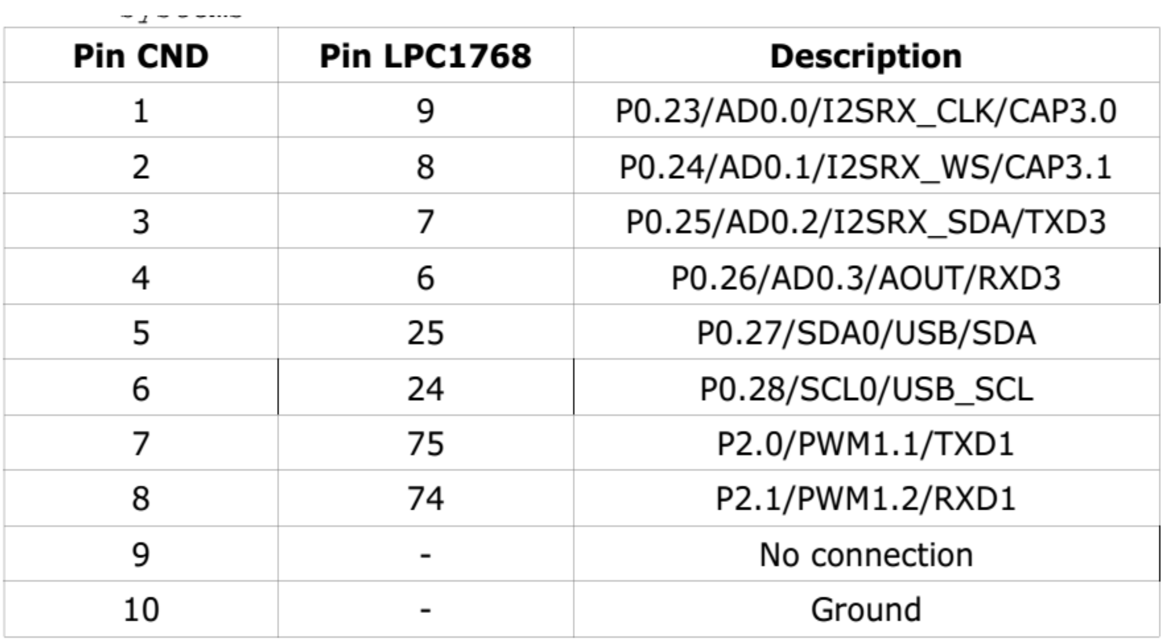
Description of the connector **CNB:**

Pin CNB	Pin LPC1768	Description
1	37	P1.23/MCI1/PWM1.4/MISO0
2	38	P1.24/MCI2/PWM1.5/MOSI0
3	39	P1.25/MCOA1/MAT1.1
4	40	P1.26/MCOB1/PWM1.6/CAP0.0
5	53	P2.10/EINT0/NMI
6	52	P2.11/EINT1/I2STX_CLK
7	51	P2.12/EINT2/I2STX_WS
8	50	P2.13/EINT3/I2STX_SDA
9	-	No connection
10	-	Ground

CNC – 10 pin male box type FRC connector. Port lines fromP0.15 to P0.22 and P2.13 are terminated in this connector.

Pin CNC	Pin LPC1768	Description
1	62	P0.15/TXD1/SCK0/SCK
2	63	P0.16/RXD1/SSEL0/SSEL
3	61	P0.17/CTS1/MISO0/MISO
4	60	P0.18/DCD1/MOSI0/MOSI
5	59	P0.19/DSR1/SDA1
6	58	P0.20/DTR1/SCL1
7	57	P0.21/RI1/RD1
8	56	P0.22/RTS1/TD1
9	50	P2.13/I2STX_SDA
10	-	Ground

CND – 10 pin male box type FRC connector. Port lines fromP0.23 to P0.28 and P2.0 to P2.1 are terminated in this connector.



Pin CND	Pin LPC1768	Description
1	9	P0.23/AD0.0/I2SRX_CLK/CAP3.0
2	8	P0.24/AD0.1/I2SRX_WS/CAP3.1
3	7	P0.25/AD0.2/I2SRX_SDA/TXD3
4	6	P0.26/AD0.3/AOUT/RXD3
5	25	P0.27/SDA0/USB/SDA
6	24	P0.28/SCL0/USB_SCL
7	75	P2.0/PWM1.1/TXD1
8	74	P2.1/PWM1.2/RXD1
9	-	No connection
10	-	Ground