ACS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MICROCONTROLLER LAB MANUAL

(BCS402)

(Effective from the academic year 2025-2026)

SEMESTER – IV

List of Program Outcomes

PO1	Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE
PO2	Apply mathematical foundations, algorithmic principles, and computer Science theory in the modelling and design of computer based systems in a way that demonstrates comprehension of trade-offs involved in design choices
PO3	Analyze a problem, and identify and define the computing requirements appropriate to its solution
PO4	Design and development principles in the construction of software systems of varying Complexity
PO5	Design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints such as memory, runtime efficiency, as well as appropriate constraints related to economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability considerations
PO6	Use the techniques, skills, and modern engineering tools necessary for practice as a CSE professional
PO7	Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary environment.
PO8	Demonstrate knowledge of contemporary issues and understand professional, ethical, legal, security and social issues and responsibilities
PO9	Analyze the local and global impact of computing on individuals, organizations, and society
PO10	Demonstrate knowledge and understanding of the engineering and management principles including financial implications and apply these to his/her work, as a member and leader in a team, and to manage project work as part of a multidisciplinary team.
PO11	Communicate effectively in both verbal and written forms
PO12	Recognize the need for, and be motivated to engage in life-long learning and continuing professional development.

List of Program Specific Outcomes

PSO1	Foundation of mathematical concepts: To use mathematical methodologies to crack problem using suitable mathematical analysis, data structure and suitable algorithm
PSO2	Foundation of Computer System: The ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.
PSO3	Foundations of Software development: the ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research

COURSE LEARNING OBJECTIVES

CLO 1: Understand the fundamentals of ARM-based systems and basic architecture of CISC and RISC.

CLO 2: Familiarize with ARM programming modules along with registers, CPSR and Flags.

CLO 3: Develop ALP using various instructions to program the ARM controller.

CLO 4: Understand the Exceptions and Interrupt handling mechanism in Microcontrollers.

CLO 5: Discuss the ARM Firmware packages and Cache memory polices.

COURSEOUTCOMES

At the end of the course the student will be able to:

CO1: Explain the ARM Architectural features and Instructions.

CO2: Develop programs using ARM instruction set for an ARM Microcontroller.

CO3: Explain C-Compiler Optimizations and portability issues in ARM Microcontroller.

CO4: Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.

CO5: Demonstrate the role of Cache management and Firmware in Microcontrollers.

PRACTICAL COMPONENT OF IPCC *(May cover all / major modules)*

Sl.No.	Experiments
Module – 1	
1.	Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).
Module – 2	
2.	Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program).
3.	Develop an ALP to multiply two 16-bit binary numbers.
4.	Develop an ALP to find the sum of first 10 integer numbers.
5.	Develop an ALP to find the largest/smallest number in an array of 32 numbers.
6.	Develop an ALP to count the number of ones and zeros in two consecutive memory locations.
Module – 3	
7.	Simulate a program in C for ARM microcontroller using KEIL to sort the numbers in ascending/descending order using bubble sort.
8.	Simulate a program in C for ARM microcontroller to find factorial of a number.
9.	Simulate a program in C for ARM microcontroller to demonstrate case conversion of characters from upper to lowercase and lower to uppercase.
Module – 4 and 5	
10.	Demonstrate enabling and disabling of Interrupts in ARM.
11.	Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM.

Sl. No.	Program List	CO	PO, PSO	RBT
1	Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).	CO1	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
2	Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program).	CO2	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
3	Develop an ALP to multiply two 16-bit binary numbers.	CO2	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
4	Develop an ALP to find the sum of first 10 integer number	CO2	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
5	Develop an ALP to find the largest/smallest number in an array of 32 number	CO2	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
6	Develop an ALP to count the number of ones and zeros in two consecutive memory locations	CO2	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
7	Simulate a program in C for ARM microcontroller using KEIL to sort the numbers in ascending/descending order using bubble sort.	CO3	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L2
8	Simulate a program in C for ARM microcontroller to find factorial of a number.	CO3	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L2
9	Simulate a program in C for ARM microcontroller to demonstrate case conversion of characters from upper to lowercase and lower to uppercase.	CO3	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L2
10	Demonstrate enabling and disabling of Interrupts in ARM	CO4	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3
11	Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM	CO4	PO1, PO2, PO3, PO4, PO5, PO12, PSO1,PSO2,PSO3	L3

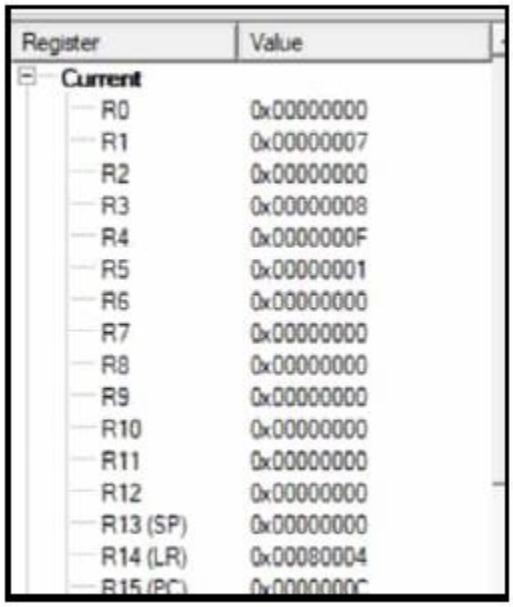
CO-PO MATRIX

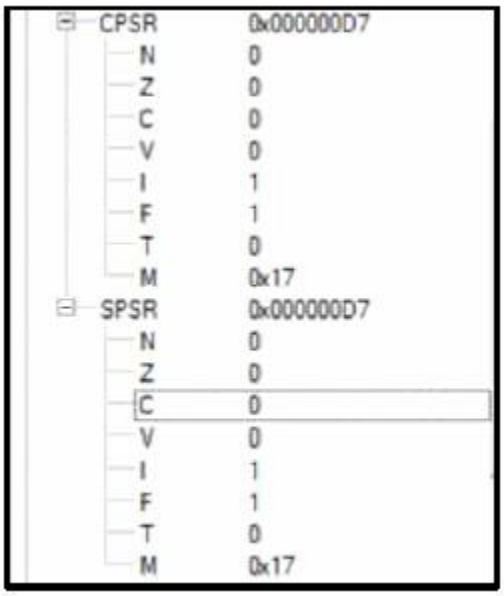
CO's	PO's												PSO's		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	2	1	3	-	-	-	-	-	-	1	1	3	2
CO2	3	3	3	1	3	-	-	-	-	-	-	1	1	3	2
CO3	3	3	3	1	3	-	-	-	-	-	-	1	1	3	2
CO4	3	3	3	1	3	-	-	-	-	-	-	1	1	3	2
CO5	3	3	3	1	3	-	-	-	-	-	-	1	1	3	2

3 - High Correlation 2 –MediumCorrelation 1– Low Correlation

1. **Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).**

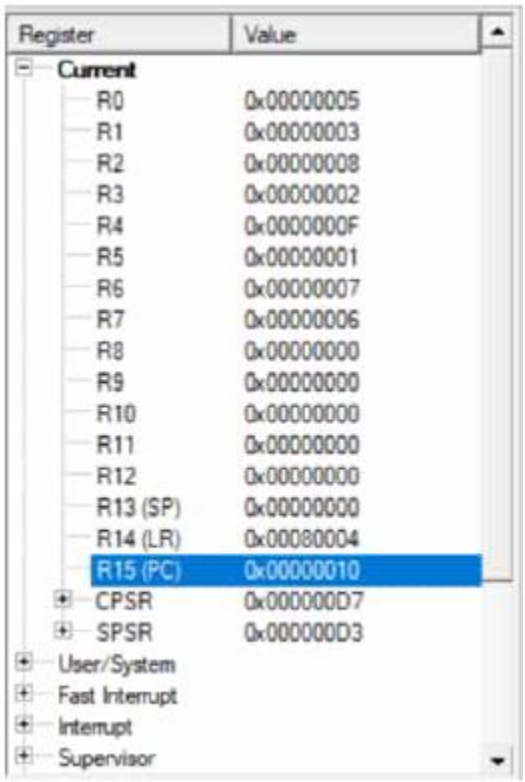
```
AREA Multiply, CODE, READONLY

ENTRY

MOV R1, #2

MOV R3, #8

END
```





2. Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program)

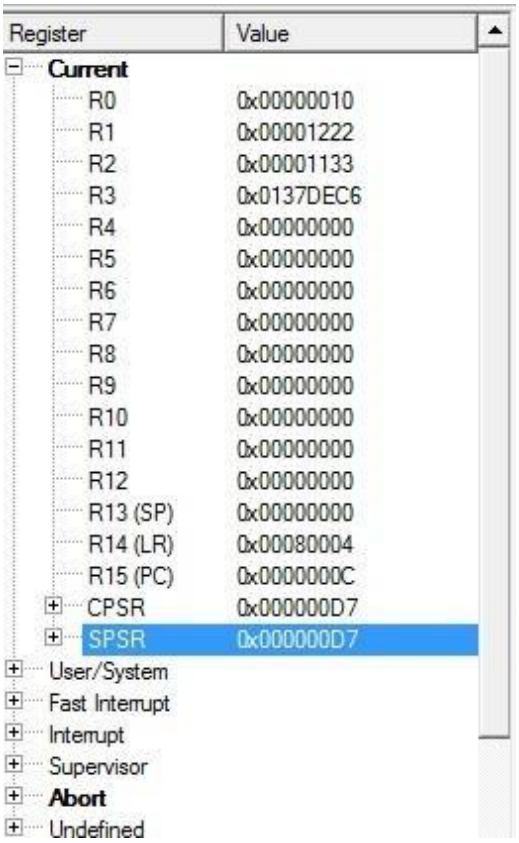
```
AREA PRG2, CODE, READONLY; Define a logical area named PRG2
ENTRY; Entry point where the code starts
LDR R0, =5; Data transfer – R0=5
LDR R1, =3; R1=3
ADD R2, R0, R1; Arithmetic: R2 = 8 (5 + 3)
SUB R3, R0, R1; SUB: R3 = 2 (5 - 3)
MUL R4, R0, R1; MUL: R4 = F (5 * 3 = 15 = F in hexadecimal)
AND R5, R0, R1 ; Logical AND: R5 = 1 (5 && 3)
ORR R6, R0, R1 ; Logical OR: R6 = 7 (5 || 3)
EOR R7, R0, R1 ; Logical XOR: R7 = 6 (5 ^ 3)
END ; End of the program
```



3. Develop an ALP to multiply two 16-bit binary numbers.

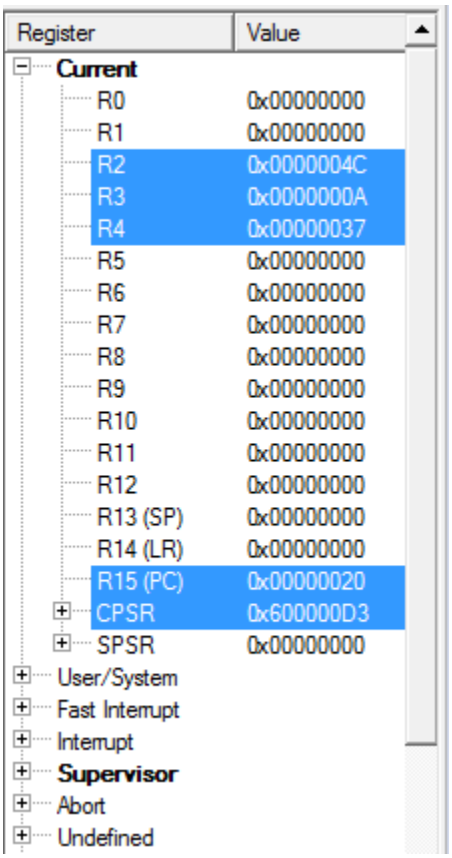
```
AREA Multiply, CODE, READONLY ENTRY

LDR   R0, =NUM     ; load address of multiplicand
LDRH R1, [R0]      ; load First number
LDRH R2, [R0,#2]   ; load Second number
MUL   R3, R1, R2   ; R3 = R1 x R2
STOP B STOP                ; all done

NUM DCW0X1222,0X1133 ; Declaration of no's to be multiply END
```

OUTPUT :



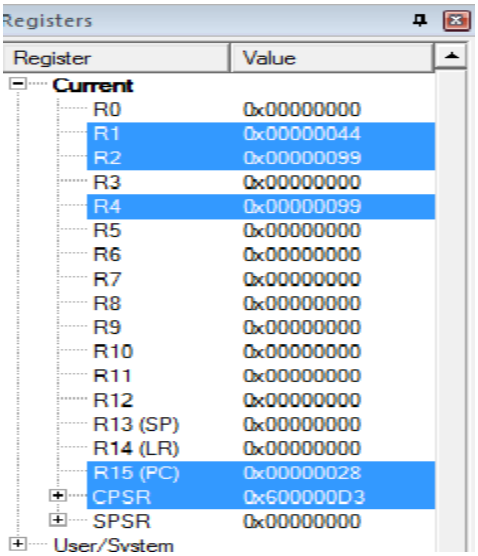
4. Develop an ALP to find the sum of first 10 integer number

```
        AREA ADD1TO10, CODE, READONLY
        ENTRY
        MOVR1,#10                   ;length of array
        LDRR2,=ARRAY                ;Load the starting address of thearray
        MOVR4,#0                    ;Initial sum
NEXTLDRR3,[R2],#4                   ;Load first integer of the array in R3
        ADDR4,R4,R3                 ;R4=sum of integers
        SUBS R1,R1,#1
        BNENEXT                     ;repeat until R1=0
STOP BSTOP

ARRAY DCD 1,2,3,4,5,6,7,8,9,10
        ;END
```



5. Develop an ALP to find the largest/smallest number in an array of 32 number

```
        AREA LARGE,CODE,READONLY
        ENTRY
        MOVR5,#5                ;R5 = length of array -1
        LDRR1,=ARRAY            ;load starting addressing of array
        LDRR2,[R1],#4           ;load 1st element of array
LOOPLDR R4,[R1],#4              ;load next element of array
        CMPR2,R4                ;compare 1st and 2nd element
        BHINEXT
        MOVR2,R4                ;R2=largest value
NEXTSUBSR5,R5,#1                ;decrement the counter after every comparison
        BNELOOP                 ; repeat until R5=0
STOP B STOP

ARRAY DCD 0X23,0X45,0X65,0X76,0X12,0X99

        END
```

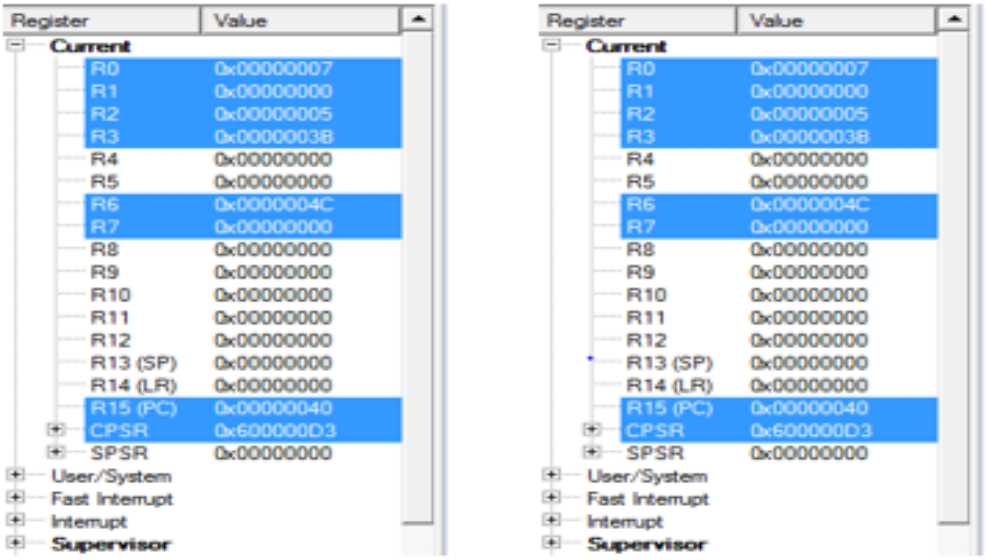


Note: Use BLS instead of BHI for finding the smallest number.

6. Develop an ALP to count the number of ones and zeros in two consecutive memory locations

```
            AREA ONEZERO, CODE, READONLY
            ENTRY
            MOVR2,#0                ;Counter forones
            MOVR3,#0                ;Counter forzeros
            MOVR7,#2                ;Counter of 2numbers
            LDR   R6,=LOOKUP        ;Load starting address of numbers
LOOP        MOVR1,#32               ;Number of bits in eachnumber
            LDRR0,[R6]              ;Load 1st number to r0
NEXTBIT     MOVSR0,R0,ROR #1        ;Check the bit is one orzero
            BHIONES                 ; IF CF=1 increment r2 else increment r3
ZEROS       ADDR3,R3,#1             ;R3 stores count of 0s
            B REPEAT
ONES        ADDR2,R2,#1             ;R2 stores count of 1s
REPEAT      SUBSR1,R1,#1            ; Decrement the bit counter by 1 till 0
            BNENEXTBIT              ; Repeat until r1=0
            ADDR6,R6,#4             ; Load r6=address of nextnumber
            SUBSR7,R7,#1            ; Decrement the number counter by 1 till 0
            BNE LOOP
STOP        B   STOP
            LOOKUP      DCD   0X5,0X7           ; Memory address of lookup table
            END
```

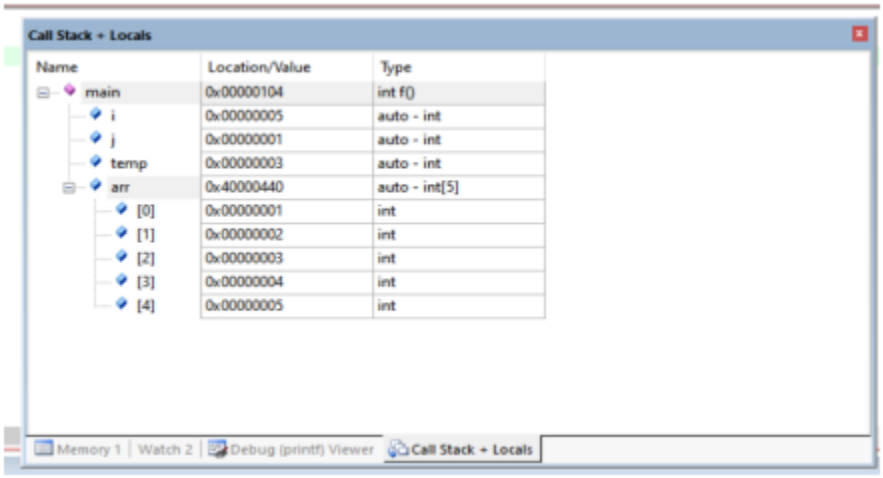
OUTPUT :



7. Simulate a program in C for ARM microcontroller using KEIL to sort the numbers in ascending/descending order using bubble sort

```
#include <lpc214x.h>
int main(void)
{
        volatile inti, j, temp;
        intarr[ ] = {4,1,3,5,2}; //the array to be sort
        //bubble sort
        for (i = 1; i< 5; i++)
        {
                for (j = 0; j < 5 - i; j++)
                {
                        if (arr[j] >arr[j + 1]) //Compares first and second element of the array
                        {
                                temp = arr[j]; // If first element is greater, swap elements of  array
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
                }
}
while(1);
}
```

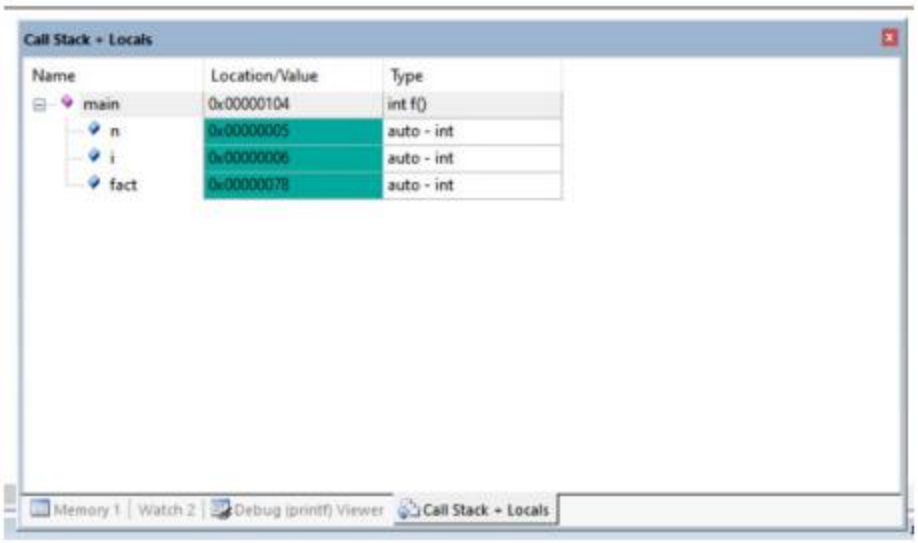
Output:



8. Simulate a program in C for ARM microcontroller to find factorial of a number.

```
#include <lpc214x.h>
intmain()
{
        volatile int n, I,fact =1;
        n=5; //initialize the number
       for (i = 1; i<= n; ++i)
       fact *= i; // factorial of the number

       while(1);
}
```

Output:

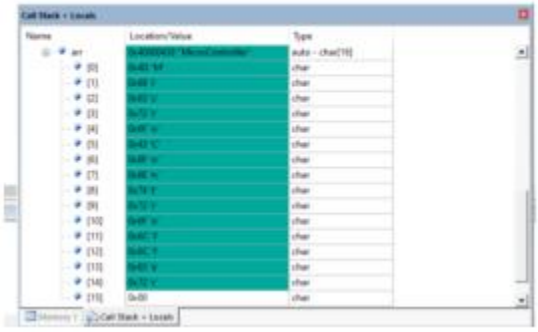


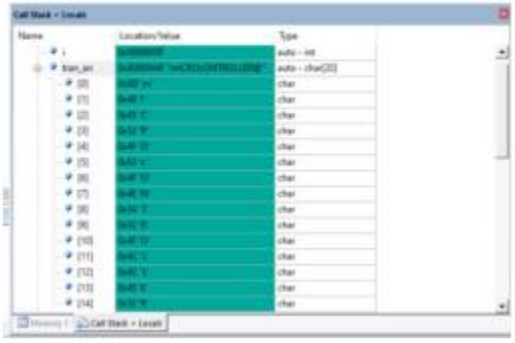
9. Simulate a program in C for ARM microcontroller to demonstrate case conversion of characters from upper to lowercase and lower to uppercase.

```
#include <lpc214x.h>
int main(void)
{
        volatile inti;
        char tran_arr[20]; // declare the variable to store translated string
        char arr[ ] ="MicroController" ; // Initialize Input String
        for(i=0; arr[i]!=0;i++)
        {
                if (arr[i] >= 'a' &&arr[i] <= 'z')
                {
                        tran_arr[i] = arr[i] - 32 ; // transalte lower to Upper case
```

```
		}
		else if(arr[i] >= 'A' &&arr[i] <= 'Z')
		{
			tran_arr[i] = arr[i]+ 32; // translate Upper to Lower case
		}
	}
	while(1);
}
```

Input sting: **Translated String:**





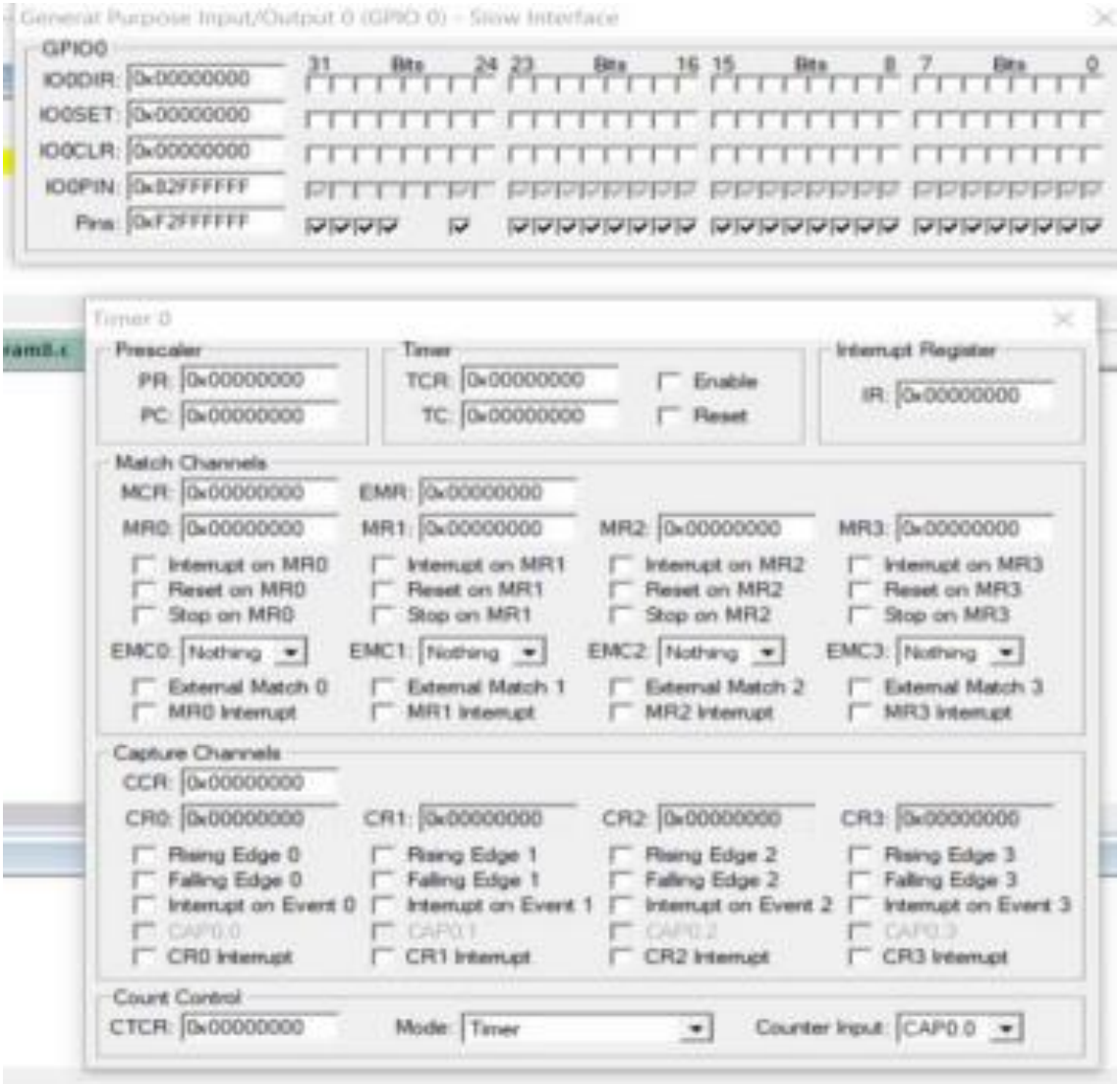
10. Demonstrate enabling and disabling of Interrupts in ARM.

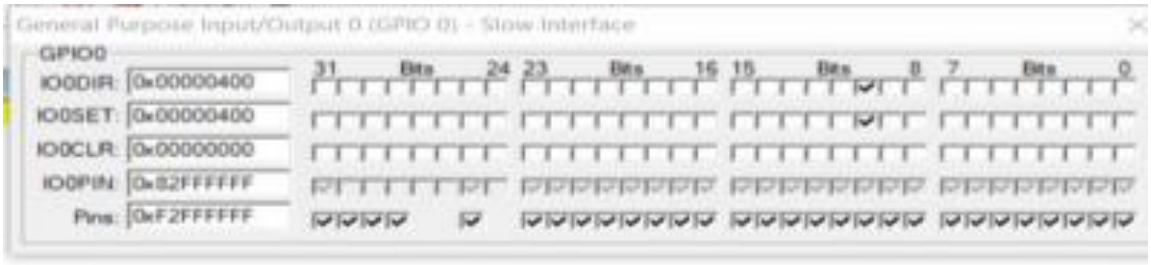
```
#include <lpc214x.h>
void initClocks(void);
void initTimer0(void);
__irq void timer0ISR(void);
int main(void)
{
	initClocks(); // Initialize PLL to setup clocks
	 initTimer0(); // Initialize Timer0
	 IO0DIR = (1<<10); // Configure pin P0.10 as Output
	IO0PIN = (1<<10);

	T0TCR = (1<<0); // Enable timer
```

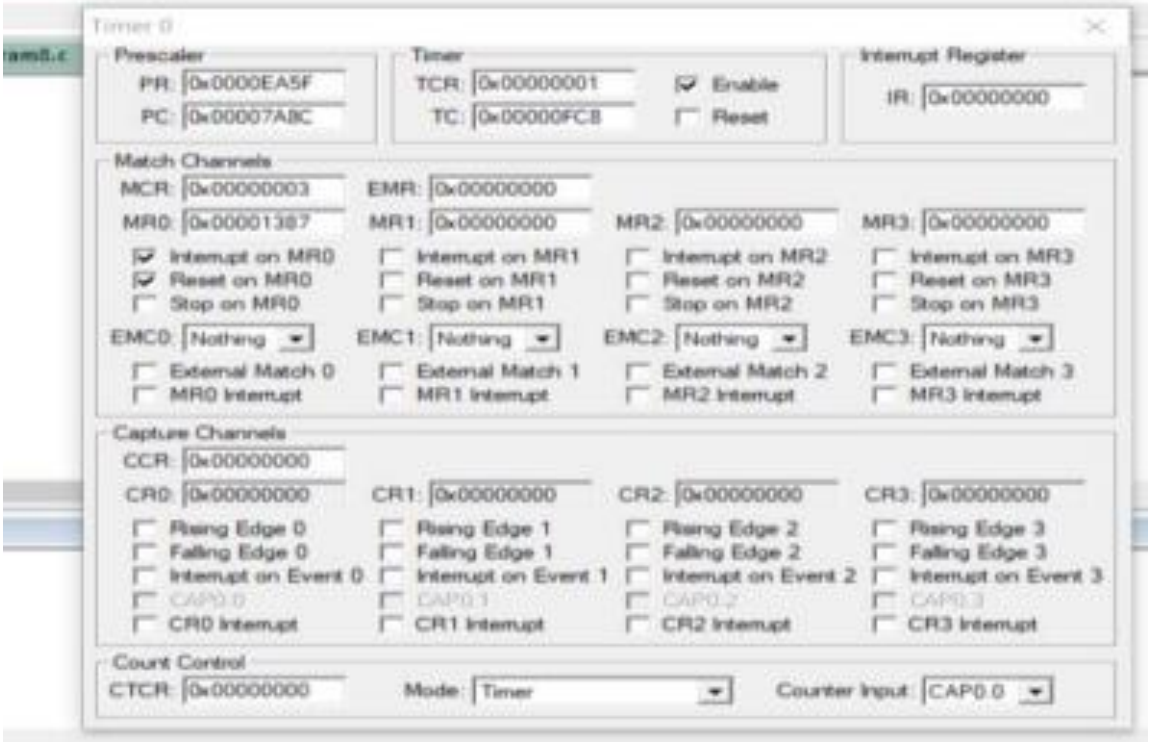
```
	while(1); // Infinite Idle Loop
}
void initTimer0(void)
{
	T0CTCR = 0x0; //Set Timer Mode
	T0PR = 60000-1; //Increment T0TC at every 60000 clock cycles
	//60000 clock cycles @60Mhz = 1 mS

	T0MR0 = 5000-1; //Zero Indexed Count-hence subtracting 1
	T0MCR = (1<<0) | (1<<1); //Set bit0 & bit1 to Interrupt & Reset TC on MR0
	 VICVectAddr4 = (unsigned )timer0ISR; //Pointer Interrupt Function (ISR)
	VICVectCntl4 = (1<<5) | 4; //(bit 5 = 1)->to enable Vectored IRQ slot
	//bit[4:0]) -> this the source number
	VICIntEnable = (1<<4); // Enable timer0 interrupt
	T0TCR = (1<<1); // Reset Timer
}
__irq void timer0ISR(void)
{
	long intreadVal;
	readVal = T0IR; // Read current IR value
	IO0PIN ^= (1<<10); // Toggle LED at Pin P0.10
	T0IR = readVal; // Write back to IR to clear Interrupt Flag
	VICVectAddr = 0x0; // End of interrupt execution
}


void initClocks(void)
{
	 PLL0CON = 0x01; //Enable PLL
	PLL0CFG = 0x24; //Multiplier and divider setup
	PLL0FEED = 0xAA; //Feed sequence
	PLL0FEED = 0x55;

	while(!(PLL0STAT & 0x00000400)); //is locked?

	 PLL0CON = 0x03; //Connect PLL after PLL is locked
	PLL0FEED = 0xAA; //Feed sequence
	PLL0FEED = 0x55;
	VPBDIV = 0x01; //PCLK is same as CCLK i.e.60 MHz
```

}

Output:


General Purpose Input/Output 0 (GPIO 0) - Slow Interface
GPIO0
IO0DIR: 0x00000000
IO0SET: 0x00000000
IO0CLR: 0x00000000
IO0PIN: 0x82FFFFFF
Pins: 0xF2FFFFFF
31 Bits 24 23 Bits 16 15 Bits 8 7 Bits 0
Timer 0
Prescaler
PR: 0x00000000
PC: 0x00000000
Timer
TCR: 0x00000000
TC: 0x00000000
Enable
Reset
Interrupt Register
IR: 0x00000000
Match Channels
MCR: 0x00000000
EMR: 0x00000000
MR0: 0x00000000
MR1: 0x00000000
MR2: 0x00000000
MR3: 0x00000000
Interrupt on MR0
Reset on MR0
Stop on MR0
Interrupt on MR1
Reset on MR1
Stop on MR1
Interrupt on MR2
Reset on MR2
Stop on MR2
Interrupt on MR3
Reset on MR3
Stop on MR3
EMC0: Nothing
EMC1: Nothing
EMC2: Nothing
EMC3: Nothing
External Match 0
MR0 Interrupt
External Match 1
MR1 Interrupt
External Match 2
MR2 Interrupt
External Match 3
MR3 Interrupt
Capture Channels
CCR: 0x00000000
CR0: 0x00000000
CR1: 0x00000000
CR2: 0x00000000
CR3: 0x00000000
Rising Edge 0
Falling Edge 0
Interrupt on Event 0
CAP0.0
CR0 Interrupt
Rising Edge 1
Falling Edge 1
Interrupt on Event 1
CAP0.1
CR1 Interrupt
Rising Edge 2
Falling Edge 2
Interrupt on Event 2
CAP0.2
CR2 Interrupt
Rising Edge 3
Falling Edge 3
Interrupt on Event 3
CAP0.3
CR3 Interrupt
Count Control
CTCR: 0x00000000
Mode: Timer
Counter Input: CAP0.0


General Purpose Input/Output 0 (GPIO 0) - Slow Interface
GPIO0
IO0DIR: 0x00000400
IO0SET: 0x00000400
IO0CLR: 0x00000000
IO0PIN: 0x82FFFFFF
Pins: 0xF2FFFFFF
31 Bits 24 23 Bits 16 15 Bits 8 7 Bits 0


Timer 0
Prescaler
PR: 0x0000EA5F
PC: 0x00007ABC
Timer
TCR: 0x00000001
TC: 0x00000FC8
Enable
Reset
Interrupt Register
IR: 0x00000000
Match Channels
MCR: 0x00000003
EMR: 0x00000000
MR0: 0x00001387
MR1: 0x00000000
MR2: 0x00000000
MR3: 0x00000000
Interrupt on MR0
Reset on MR0
Stop on MR0
Interrupt on MR1
Reset on MR1
Stop on MR1
Interrupt on MR2
Reset on MR2
Stop on MR2
Interrupt on MR3
Reset on MR3
Stop on MR3
EMC0: Nothing
EMC1: Nothing
EMC2: Nothing
EMC3: Nothing
External Match 0
MR0 Interrupt
External Match 1
MR1 Interrupt
External Match 2
MR2 Interrupt
External Match 3
MR3 Interrupt
Capture Channels
CCR: 0x00000000
CR0: 0x00000000
CR1: 0x00000000
CR2: 0x00000000
CR3: 0x00000000
Rising Edge 0
Falling Edge 0
Interrupt on Event 0
CAP0.0
CR0 Interrupt
Rising Edge 1
Falling Edge 1
Interrupt on Event 1
CAP0.1
CR1 Interrupt
Rising Edge 2
Falling Edge 2
Interrupt on Event 2
CAP0.2
CR2 Interrupt
Rising Edge 3
Falling Edge 3
Interrupt on Event 3
CAP0.3
CR3 Interrupt
Count Control
CTCR: 0x00000000
Mode: Timer
Counter Input: CAP0.0

INTRODUCTION

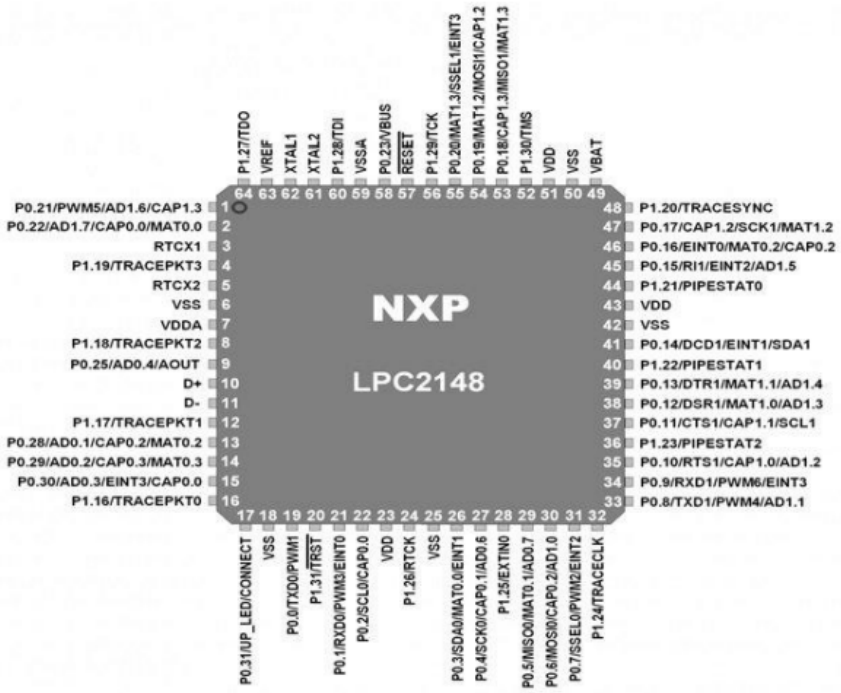
- The µVision IDE combines project management, run-time environment, build facilities, source code editing, and program debugging in a single powerful environment. µVision is easy-to-use and accelerates your embedded software development. µVision supports multiple screens and allows you to create individual window layouts anywhere on the visual surface.

- The µVision Debugger provides a single environment in which you may test, verify, and optimize your application code. The debugger includes traditional features like simple and complex breakpoints, watch windows, and execution control and provides full visibility to device peripherals.

LPC2148

- LPC means Linear Programming Control. 2148 is the version of LPC controller.
- It is one of the most commonly used ARM based Microcontroller.
- It is a product of NXP (A subsidiary of Philips Company).
- It is a 32-Bit microcontroller. It is based on ARM7 and uses RISC Technology.
- It has Two Input/output port namely P0, P1. Each port has 32 pins. These pins are called GPIO Pins. Most of the pins are multifunctional.
- Its Features: o 512 KB on-chip Flash Memory. 32 KB on-chip SRAM. o Supports up to 2KB USB RAM. o Speed - 60 MHz operation

Pin diagram of LPC2148

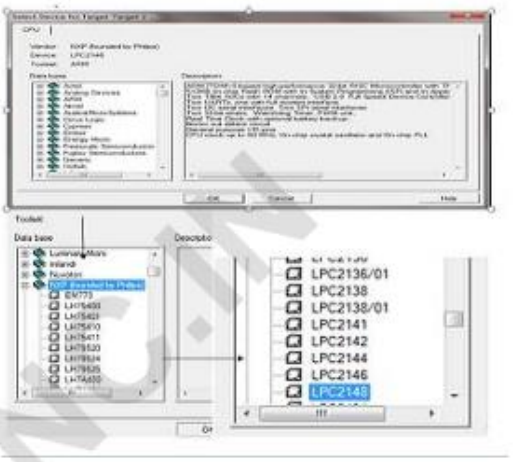
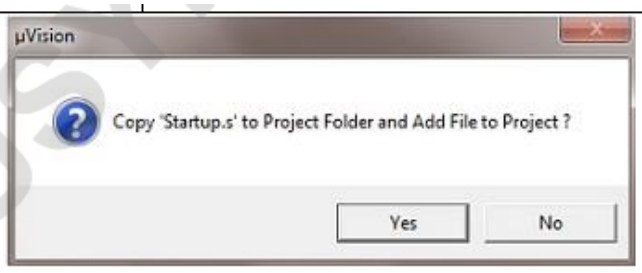


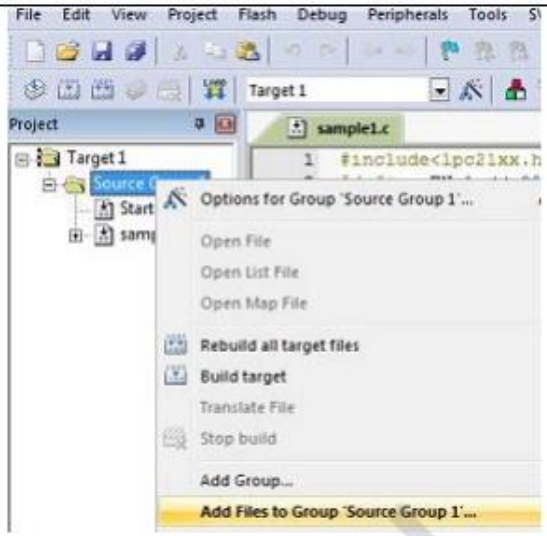
KEIL

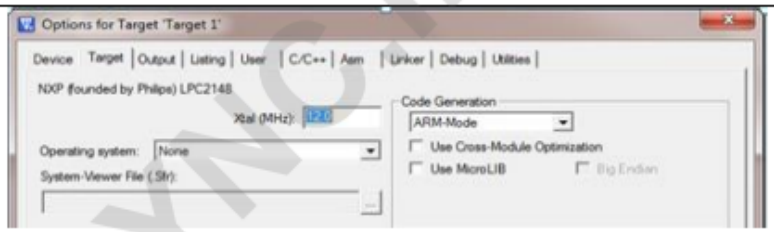
KeilMicroVision is free software which solves many of the pain points for an embedded program developer. This software is an integrated development environment (IDE), which integrated a text editor to write programs, a compiler and it will convert your source code to hex files too. Keil can be used for Writing programs in C/C++ or Assembly language

- Compiling and Assembling Programs
- Debugging program
- Creating Hex and Axf file
- Testing your program without Available real Hardware (Simulator Mode)

A SIMPLE GUIDE ON KEILUVISION 4

Step 1: After opening Keil uV4, Go to **Project** tab and **Create new uVision project** Now Select new folder and give name to Project.	
Step 2: After Creating project now **Select your device model**. Example.NXP-LPC2148	
Step 3: so now your project is created and **Message** window will appear. For C programs – Add startup.s file For assembly programs – Not necessary	

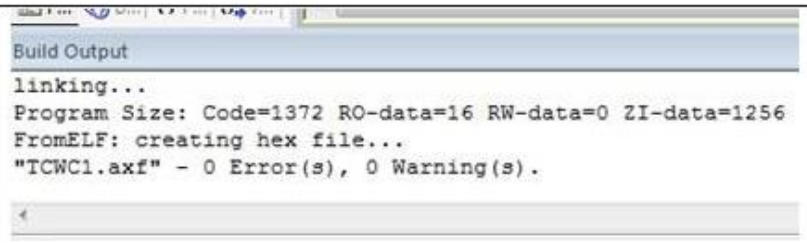
Step 4: Now go to File and create new file and save it with **.C** extension if you will write program in C language or save with **.asm** for **assembly** language.	
Step 5: Now write your program and save it again. Now come on Project window.	
Step 6: Now Expand target and you will see source group Right click on group and click on **Add files to source group.** Now add your program file which you have written in C/assembly. You can see program file added under source group.	

<table>
<tr><td colspan="2">Step 7 and 8 are for Hardware Demonstration Programs only. Omit these steps for Software Assembly and Embedded C programs</td></tr>
<tr><td><u>Step 7:</u>
Right click on target and click on options for target</td><td> 
</td></tr>
<tr><td><u>Step 8:</u>

The various setting are as follows:</td><td>Output Tab: Check Create hex file

Linker Tab: Check Use Memory layout from Target Dialog

Debug Tab: Click Use Simulator Radio button.

Utilities Tab: Click Setting button. To the Flash download setup , Add “<u>LPC2000 IAP2 512 KB Flash</u>” and then Click OK button.</td></tr>
<tr><td colspan="2"><u>Step 9:</u> Now Click on Build target. You can find it under Project tab or in toolbar. It can also be done by pressing F7 key.</td></tr>
<tr><td><u>Step 10:</u>
you can see Status of your program in Build output window
[If it’s not there go to view and click on Build output window]</td><td> 
</td></tr>
</table>

ARM PROGRAMMING

We do programming using

- Assembly Language
- Embedded C Programming Language Assembly Language: After machine level language, the next level of development in the evolution of computer languages was the Assembly Language. Assembly language is a low-level programming language for a computer or other programmable device. Programs written in assembly languages are compiled by an assembler. Every assembler has its own assembly language, which is designed for one specific computer architecture.