RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Estd : 2001

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MICROCONTROLLER LABORATORY MANUAL

For Forth Semester B.E - 2022 Batch [VTU/CBCS, 2022 syllabus]

Subject Code – BCS402

NAME : _____

USN : _____

SECTION : _____ **BATCH :** _____

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

VISION AND MISSION OF CSE DEPARTMENT

Vision

Preparing better computer professionals for a real world

Mission

The Department of Computer Science and Engineering will make every effort to promote an intellectual and an ethical environment in which the strengths and skills of Computer Professionals will flourish by

1. Imparting Solid foundations and Applied aspects in both Computer Science Theory and Programming practices
2. Providing Training and encouraging R&D and Consultancy Services in frontier areas of Computer Science with a Global outlook
3. Fostering the highest ideals of Ethics, Values and creating Awareness on the role of Computingin Global Environment
4. Educating and preparing the graduates, highly Sought-after, Productive, and Well-respected fortheir work culture
5. Supporting and inducing Lifelong Learning practice

PREFACE

We have developed this comprehensive laboratory manual on Digital design & computer organization with the primary objectives: To make the students comfortable with the Verilog hardware description language. The manual will help them to learn various digital circuit modeling issues using Verilog, and some case studies. Through this course students will get exposure to design of digital circuits using Verilog HDL.

Our profound and sincere efforts will be fruitful only when students acquire the extensive knowledge by reading this manual and apply the concepts learned apart from the requirements specified in Digital design & computer organization Laboratory as prescribed by VTU, Belagavi.

Department of CSE

ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Mr. Satish R Shetty, Chairman and Mr. Karan R Shetty, CEO, RNS Group of Companies for providing magnanimous support in all our endeavors.

We are grateful to Dr. M K Venkatesha, Director and Dr. Ramesh Babu H S, Principal, Dr. Kiran P, HOD, CSE, RNSIT for extending their constant encouragement and support.

Our heartfelt thanks to Dr. Girijamma H A, Dr. Devaraju B M, Prof. Lakshmi R, Prof. Chetan Ghatage for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

Department of CSE

MICROCNTROLLERS LABORATORY
INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 25)			
TEST **A**	**REGULAR EVALUATION** **B**	**RECORD** **C**	**TOTAL MARKS** **A+B+C**
10	**10**	**5**	**25**

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No	**Parameters**	**Good**	**Average**	**Needs improvement**
a.	**Understanding of problem (3 marks)**	Clear understanding of problem statement while designing and implementing the program (3)	Problem statement is understood clearly but few mistakes while designing and implementing program (2)	Problem statement is not clearly understood while designing the program (1)
b.	**Writing program(4 marks)**	Program handles all possible conditions (4)	Average condition is defined andverified. (3)	Program does not handlepossible conditions (1)
c.	**Result and documentation(3 marks)**	Meticulous documentation andall conditions are taken care (3)	Acceptable documentation shown(2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**
a.	**Conceptual understanding(10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Excellent**	**Good**	**Needs Improvement**
a.	**Design, implementation and demonstration (5 marks)**	Program follows syntax and semantics of ARM7 assembly instructions and Embedded C programming. Demonstrates the complete knowledge of the program written (5)	Program has few logical errors, moderately demonstrates all possible concepts implemented in programs (3)	Syntax and semantics of programming is not clear (1)
b.	**Result and documentation (5 marks)**	All expected results are demonstrated successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the program and Partial documentation (3)	Expected results are not demonstrated properly, unable to debug the errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 20)				
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**
a.	**Documentation(10 marks)**	Meticulous record writing including program, comments and expected output as per the guidelines mentioned (20)	Write up contains program and expected output, but comments are not included (18)	Write up containsonly program (15)

A. TEST /LAB INTERNALS MARKS (MAX MARKS 10)

TEST #	Write up 10	Execution 30	Viva 10	Sign	Total 50	Avg. 50	Final 10
TEST-1							
TEST-2						40	10

B. REGULAR LAB EVALUATION (MAX MARKS 10)

Program #	Date of Execution	Lab programs	Write up (10)	Exen. (10)	Viva (10)	Total 30	Teacher Signature
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
	Total Marks	330	30			10	

FINAL MARKS OBTAINED			
A : TEST (10)		TOTAL (A+B+C)	Signature Of Lab In Charge:
B : REGULAR EVALUATION (10)			
C: RECORD (5)		25	

DELIVERY PLAN WITH DETAILS

Week No.	Topic/ Programs	Planned Week of Execution	COs covered	Remarks
1	Introduction to keil version 4 software	29th to 05thMay		
2	Basic programs	06th to 12th May		
3	Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).	13th to 18th May		
4	Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program).	20th to 25th May		
5	Develop an ALP to multiply two 16-bit binary numbers.Develop an ALP to find the sum of first 10 integer numbers.	27th to 1st June		
6	Develop an ALP to find the largest/smallest number in an array of 32 numbers. Develop an ALP to count the number of ones and zeros in two consecutive memory locations.	03rd to 8th June		
7	**Internal-1**	10th to 15th June		
8	Simulate a program in C for ARM microcontroller using KEIL to sort the numbers in ascending/descending order using bubble sort.	17th to 22nd June		
9	Simulate a program in C for ARM microcontroller to find factorial of a number.	24th to 29thjune		
10	Simulate a program in C for ARM microcontroller to demonstrate case conversion of characters from upper to lowercase and lower to uppercase.	01st to 6th July		
11	Demonstrate enabling and disabling of Interrupts in ARM.	8th to 13th July		
12	Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM.	15th to 20th July		
13	**Internal-2**	22nd to 27th July		

SYALLABUS

SEMESTER – IV

MICROCONTROLLERS LABORATORY

(Effective from the academic year 2023-2024)

Course Code – BCS402	**CIE Marks - 50**
Number of Contact Hours/Week -3:0:2:0	**SEE Marks - 50**
Total Number of Lab Contact Hours - 20	**Exam Hours - 03**

***COURSE LEARNING OBJECTIVES*:** This course will enable students to
1. Understand the fundamentals of ARM-based systems and basic architecture of CISC and RISC.
2. Familiarize with ARM programming modules along with registers, CPSR and Flags.
3. Develop ALP using various instructions to program the ARM controller.
4. Understand the Exceptions and Interrupt handling mechanism in Microcontrollers.
5. Discuss the ARM Firmware packages and Cache memory polices.

Programs List:

Sl. No	Experiments IDE Used: Keil uVision 4 or 5
1.	Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).
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.
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.
10.	Demonstrate enabling and disabling of Interrupts in ARM.
11.	Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM.

COs	COURSE OUTCOMES
BCS402.1	Explain the ARM Architectural features, Instructions set, and the role of Cache management in Microcontrollers
BCS402.2	Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.
BCS402.3	Develop assembly programs using ARM instruction set.
BCS402.4	Implement embedded C programs for ARM7.
BCS402.5	Analyze C compilers and optimization tasks to generate faster ARM code.

Laboratory Outcomes: The student should be able to:

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

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

CO-PO MATRIX

COURSE OUTCOMES	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	1	1	1										1			
CO2	3	2		2	3									2		
CO3	3	3	3		3				2		1	2		2	2	
CO4	2	2	3		3				2		2	2		2	3	
CO5	2	2	3								2	2			2	

TABLE OF CONTENTS

MICROCONTOLLERS LABORATORY

INTRODUCTION

MICROCONTROLLER

- A small, low-cost computer
- It is designed to do a specific task
- A highly Integrated chip
- Part of Embedded System
- In addition to CPU, it includes RAM, ROM, I/O Ports and Timers also.
- I/O Ports are GPIO Pins

Difference between Microprocessor and Microcontroller

Microprocessor	Microcontroller
Executes big application	Executes a single task within an application
High Cost	Low cost
Not easy to replace	Easy to replace
More power consumption	Less power consumption
It doesn't consist of RAM, ROM, I/O Ports. It uses its pins to interface to peripheral devices.	It consists of CPU, RAM, ROM, I/O Ports.

Application of Microcontrollers:

- Light sensing & controlling devices
- Temperature sensing and controlling devices
- Fire detection & safety devices
- Industrial instrumentation devices
- Process control devices
- Power Tools
- Implantable medical devices
- Automobile Engine control systems
- Remote control appliances, Toys.
- Office Machines.

ARM

- Advanced RISC Machine
- The Project started in the year 1983

- Developed by ARM Holdings, a British company. They licensed the ARM architecture to other companies, who design their own products using this architecture.
- It is a 32-Bit Microcontroller.
- Uses RISC (Reduced Instruction Set Computing) Architecture.
- It has 32 Bit ALU, 32 Bit data bus.
- 32 Bit Address bus. Hence 2 pow 32 = 4GB Memory
- It uses Von Newman Model in which Program and Data stored in common memory
- There are 37 Registers – 32 Bits each (16 Available at a time – R0 to R15)
- Based on Load – Store Model. (All RISC processor work on Registers and not not memory)
- It has 7 operating modes
- It has 7 Interrupts and 7 Addressing modes
- Data formats available in ARM are 8 Bit – Byte format, 16 Bit- Half word format and 32 Bit – Word format
- In ARM everything is aligned.
- ARM instructions are 3 types
  - ARM – When ARM state – 32 Bit instruction
  - Thumb When in Thumb state – Purely 16 bit instruction
  - Jazelle – For Java Byte code – 8 Bit instructions

Difference between Intel Processor and ARM Processor.

Intel	ARM
It uses CISC Architecture	Uses RISC Architecture
Needs more power	Needs less power
High speed	Low speed
More heat generated	Less heat generated High code density It is heterogeneous computing

Applications:

ARM processor is widely found in

- LED TV
- Mobile phone, Tablets
- Multimedia devices, Gaming devices, ipods, Apple's Mobile, Raspberry pi 3

Career Opportunities in Microcontroller, Embedded Systems and Arm Programming

If you know how to make an embedded system then you can automate any task.Embedded system is an ocean.

There are huge products in Medical electronics, aerospace, Telecommunication, Automobiles etc. Every industry needs an Artificial Intelligence System into it. Artificial Intelligence can be given by Embedded Systems only. Embedded system is the future. Lots of career opportunities available in Embedded System. Some of them are as follows.

- Embedded Software Engineer(Firmware)
- System Software Engineer (Kernal and RTOs)
- Application Software Engineer (Device drivers)
- Embedded Hardware Engineer
- Software Test Engineer
- Embedded System Trainer

What skills you should have?

- Embedded C programming on Microcontroller
- Interfacing Microcontrollers with different sensors and peripherals
- Kernel Programming
- Device drivers
- Real Time operating Systems (RTOs)

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:
  - 512 KB on-chip Flash Memory. 32 KB on-chip SRAM.
  - Supports up to 2KB USB RAM.
  - Speed - 60 MHz operation.

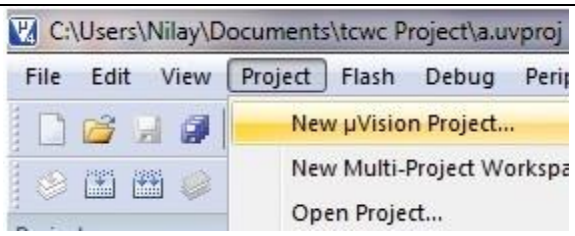
KEIL

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 forWriting programs in C/C++ or Assembly language

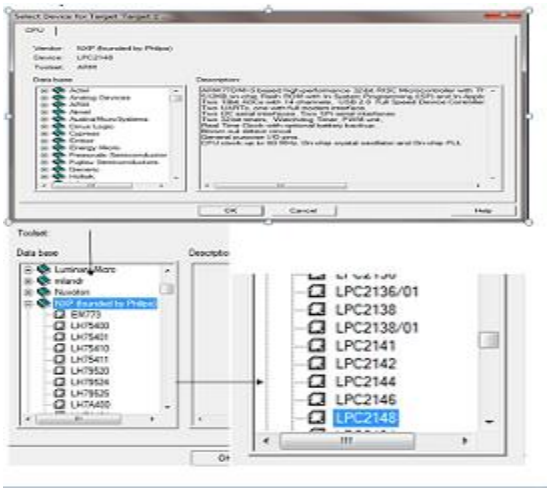
- Compiling and Assembling Programs
- Debugging program
- Creating Hex and Axf file
- Testing your program without Available real Hardware (Simulator Mode)

A SIMPLE GUIDE ON KEILUVISION 4

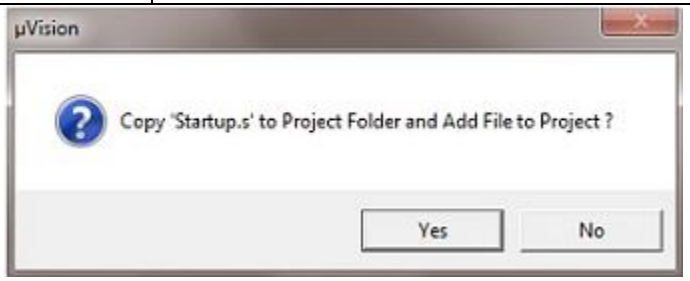
Step 1: After opening Keil uV4, Go to **Project** tab and
Create new uVision project
Now Select new folder and give name to Project.



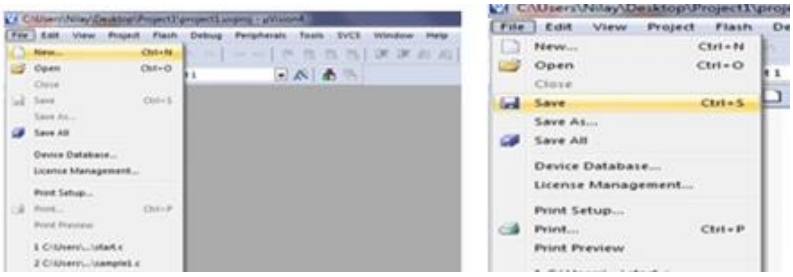
Step 2: After Creating project now **Select your device model**. Example.NXP-LPC2148



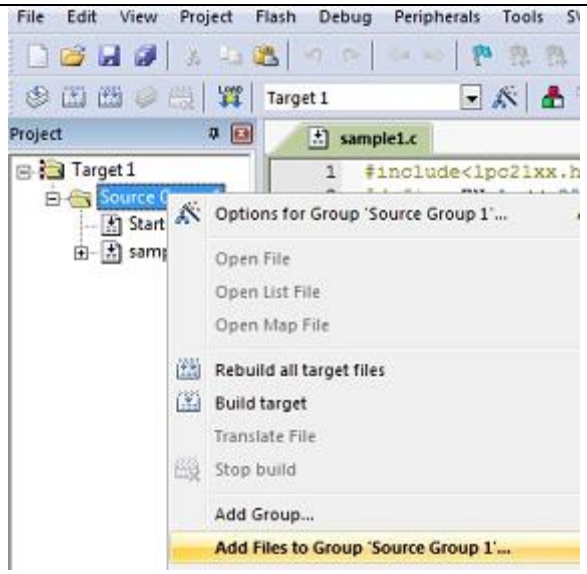
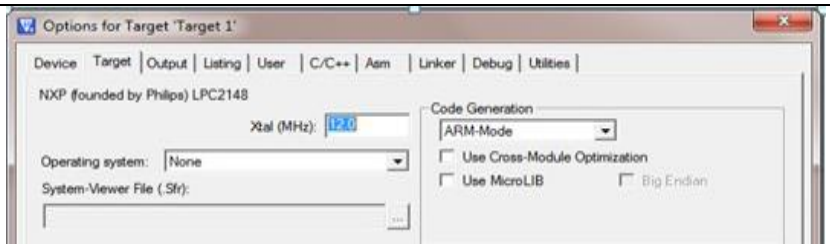
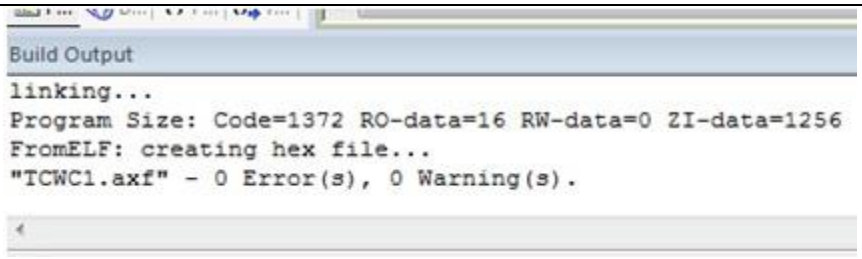
Step 3: so now your project is created and **Message** window will appear.
For C programs – Add startup.s file
For assembly programs – Not necessary



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.	
Step 7 and 8 are for Hardware Demonstration Programs only. Omit these steps for Software Assembly and Embedded C programs	
Step 7: **Right click on target** and click on **options for target**	
Step 8: **The various setting are as follows:**	**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 "LPC2000 IAP2 512 KB Flash" and then Click OK button.**
Step 9: Now Click on **Build target**. You can find it under Project tab or in toolbar. It can also be done by pressing **F7** key.	
Step 10: 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]	

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.

The Reasons to write computer instructions in assembly language:

01. To speed computer operation
02. To reduce the size of the program.
03. To write programs for special situations.
04. To save money
05. To better understand how computer operate.

Arm assembly programming consists of Symbols, Labels, Instructions Mnemonic, Directives

Symbols: The name given to identifier by programmer. It is case sensitive.

Labels: The label field is the first field in an assembly language instruction. It is the symbol that represents memory address of an instruction or data. The ARM assembler requires labels to start at the first character of a line. Labels are most frequently used in branch instruction. Examples for labels are START, LOOP, and MAIN

Instruction mnemonic: most of the arm instruction consists of three letters. For Example sub, mov, str. usually instruction followed by one or more operands. Some instructions have no operands.

Directives: These are instructions to assembler. It used at assembly time. Example: AREA

Difference between instruction and directive.

Instruction carried out by processor at run time. Instructions are assembled into machine code. They are linked to final executable. Instructions are part of program that will be executed when the program is run. Instruction generates machine code, thus contributes towards the size of program.

Directive is instruction to assembler. It is not part of program. It is used at assembly time. It does not create any machine code.

ARM INSTRUCTIONS AND DIRECTIVES

Instruction	Syntax	Remark
ADC	ADC Rd Rn, op1	Adds the value of op1 and Carry flag to Rn and stores the result in Rd. Add the most significant words (In ARM word=32 bits)
ADD	ADD Rd Rn, op1	Add the value of op1 to Rn and stores the result in Rd
ADDS	ADDS r4, r0, r2	Add a 64-bit integer contained in r2 to another integer contained in r0 and stores the result in r4. Least significant words are added.
AND	AND Rd, Rn, op1	Performa a bitwise AND of the value of register Rn with the value of op1 and stores the result in Rd
B,BL	B label BL func	B-Branch unconditionally to label, BL-Branch and link. Subroutine call to function "func"
CMP	CMP Rn, op1	Compares a register value with another arithmetic value. The condition flags are updated, based on the result of subtracting op1 from Rn, so that subsequent instructions can be conditionally executed.
EOR	EOR Rd,Rn, op1	Performs bitwise Exclusive OR of the value of register Rn with the value of op1, and stores the result in the Rd. condition code flags are optionally updated based on the result.
LDR	LDR Rd, op2	Loads a word from the memory address calculated by op2 and writes it to register Rd.

Instruction	Syntax	Remark
LDRB	LDRB Rd,op2	Loads a byte from memory address calculated by op2, zero extends the byte to a 32-bit word and

		writes the word to register Rd
MOV	MOV Rd, op1	Moves the value of op1 to destination register Rd
MVN	MVN Rd,op1	Moves the logical one's complement of the value of op1 to the destination Rd.
ORR	ORR Rd, Rn, op1	Performs a bitwise OR of the value of Register Rn with the value of op1 and stores the result in Rd.
SBC	SBC Rd, Rn, op1	Subtracts the value of op1 and the value of NOT(Carry flag) from the value of Register Rn and stores the result in Rd. Rd=Rn-op1-CPSR(C)
STR	STR Rd, op2	Stores a word from register Rd to the memory address calculated by op2
STRB	STRB Rd, op2	Stores a byte from the least significant byte of register Rd to the memory address calculated by op2
SUB	SUB Rd, Rn, op1	Subtracts the value of op1 from the value of Register Rn and stores the result in the destination register Rd

SWP	SWP Rd, Rm,[Rn]	Swaps a word between registers and memory. SWP loads a word from the memory address given by the value of register Rn. The value of register Rm is then stored to the memory address given by the value of Rn, and the original loaded value is written to register Rd. If the same register is specified for Rd and Rm, this instruction swaps the value of the register and the value at the memory address.
SWPB	SWPB Rd, Rm, [Rn]	Swaps a byte between registers and memory. SWPB loads a byte from the memory address given by the value of register Rn. The value of least significant byte of Register Rm is stored to the memory address given by the value of Rn, and the original loaded value is zero-extended to

		a 32-bit word and the word is written to register Rd. If the same register is specified for Rd and Rm, this instruction swaps the value of the least significant byte of register and the byte value at the memory address

Directives

Directive	Remark
AREA	The AREA directive allows the programmer to specify the memory locations where programs, subroutines, or data will reside.
EQU	The EQUATE directive allows the programmer to equate names with addresses or data.
ALIGN	
DATA	Indicates that the label points to the data rather than code.
ENTRY	This directive specifies the program entry point for the linker.

HOUSE KEEPING DIRECTIVES

Directive	Remark
TTL	Title of the Program
END	Marks the end of the assembly language source program.
INCLUDE	Will include the contents of a named file into the current file.

DEFINE CONSTANT (DATA) DIRECTIVE

Directive	Remark
DCB	Allocates one or more bytes of memory, aligned on 2-byte boundaries, and defines the initial runtime contents of the memory
DCD	Allocates one or more words of memory, aligned on 2-byte boundaries, and defines the initial runtime contents of the memory
DCW	Allocates one or more halfwords of memory, aligned on 2-byte boundaries, and defines the initial runtime contents of the memory

Embedded C Programming Language

Embedded C Programming is the soul of the processor functioning inside each and every embedded system we come across in our daily life, such as mobile phone, washing machine, and digital camera. Embedded C language

is most frequently used to program the microcontroller.

Earlier, many embedded applications were developed using assembly level programming. However, they did not provide portability. This disadvantage was overcome by the advent of various high level languages like C, Pascal, and COBOL. However, it was the C language that got extensive acceptance for embedded systems, and it continues to do so. The C code written is more reliable, scalable, and portable; and in fact, much easier to understand.

Differences between C and Embedded C

C Programming	Embedded C Programming
Possesses native development in nature	Possesses cross development in nature
Independent of hardware architecture.	Dependent on hardware architecture (Microcontroller or other devices)
Used for desktop applications, OS and PC memories.	Used for limited resources like RAM, ROM and I/O peripherals on embedded controller.

Advantages of embedded C program:

- Its takes less time to develop application program.
- It reduces complexity of the program.
- It is easy to verify and understand.
- It is portable in nature from one controller to another.

SAMPLE PROGRAMS

Data Processing Instructions

1. Mov Instruction

```
        AREA ALP1, CODE, READONLY

      ENTRY
      MOV R5, #5
      MOV R7,#8
      MOV R7,R5
STOP B STOP

      END
```

2. Logical Shift Left(LSL)

```
        AREA ALP2, CODE, READONLY

        ENTRY
        MOV R5,#5
        MOV R7, #8
        MOV R7, R5, LSL #2
STOP B STOP
        END
```

3. ArithmeticOperations

```
        AREA ALP3,CODE,READONLY

        ENTRY
        LDR R0,=0x00000000
        LDR R1,=0x00000002
        LDR R2,=0x00000001
        ADDr0,r1,r2
        SUB r0,r1,r2
 STOPBSTOP
        END
```

4. Reverse subtraction

```
        AREA ALP4, CODE, READONLY

        ENTRY
        LDR R0,=0x00000000
        LDR R1,=0x00000077
        RSB   R0,R1,#0   ;RSB subtracts r1 from constant value #0  R0 = -R1
STOP B STOP

        END
```

5. Addition with barrel shifter

```
        AREA ALP5, CODE, READONLY
        ENTRY
        LDR R0,=0x00000000
        LDR R1,=0x00000005
        ADD  R0,  R1,  R1, LSL#1
STOP  B  STOP
        END
```

6. Logical Instructions

```
        AREA ALP6 ,CODE, READONLY
        ENTRY
        LDR R0,=0x00000000
        LDR R1,=0x02040608
        LDR R2,=0x10305070
        ORR R0, R1, R2
        AND R0,R1,R2
        EOR R0,R1,R2
STOP   B  STOP
        END
```

7. MVN Instruction

```
        AREA ALP7 ,CODE, READONLY
        ENTRY
        LDR R0,=0x0000000A
        MVN R1,R0
STOP   B  STOP
   END
```

8. Load and store instructions

```
        AREA ALP8,CODE, READONLY
        ENTRY
        LDR R0, =0x40000000
        LDR R1,[R0]
        LDR R2,=0x40000050
        STR R1,[R2]
STOP    B   STOP
        END
```

9. To check for carry flag C in CPSR

```
        AREA ALP9,CODE,READONLY
```

```
        ENTRY
        LDR R0, =0x00000000
        LDR R1, =0x80000004
        MOVS  R0,R1,LSL#1
STOP    B  STOP

        END
```

10.Comparison instruction to check for C, Z flags in CPSR

```
        AREA ALP10, CODE, READONLY

        ENTRY
        LDR R0, =0x00000002
        LDR R1, =0x00000002
        CMP R0,R1
STOP   B   STOP

        END
```

11. Instruction to check for C, Z in CPSR

```
        AREA ALP11, CODE, READONLY

        ENTRY
        LDR R1,=0x00000001
        SUBS R1,R1,#1
STOP  BSTOP
        END
```

12. Logical bit clear

```
        AREA ALP12, CODE, READONLY

        ENTRY
        LDR  R1, = 0Xf
        LDR  R2, =0x5
        BIC R0,R1,R2  ; R0 = R1 AND (!R2)
STOP   B  STOP
        END
```

13. MOVS and MOVCS instructions

```
        AREA ALP13,  CODE,  READONLY

        ENTRY
        LDR R0, =0x00000000
        LDR R1, =0x80000004
        MOVS  R0,R1,LSL#1
        MOVCS R0, R1   ;if carry is set then R0:=R1
STOP   B   STOP
        END
```

14. ADDCS instruction

```
        AREA ALP14, CODE,  READONLY

        ENTRY
        LDR R0, =0x00000000
        LDR R1, =0x80000004
        MOVS R0,R1,LSL #1
        ADDCS R2, R0, R1
STOP    B   STOP
        END
```

15. ADDEQ Instruction

```
        AREA  ALP15, CODE, READONLY

ENTRY

        LDR R0, =0x00000004

        LDR R1, =0x00000004

        CMP R0,R1
        ADDEQ R2, R0, R1
STOP  B  STOP
        END
```

LABORATORYPROGRAMS

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

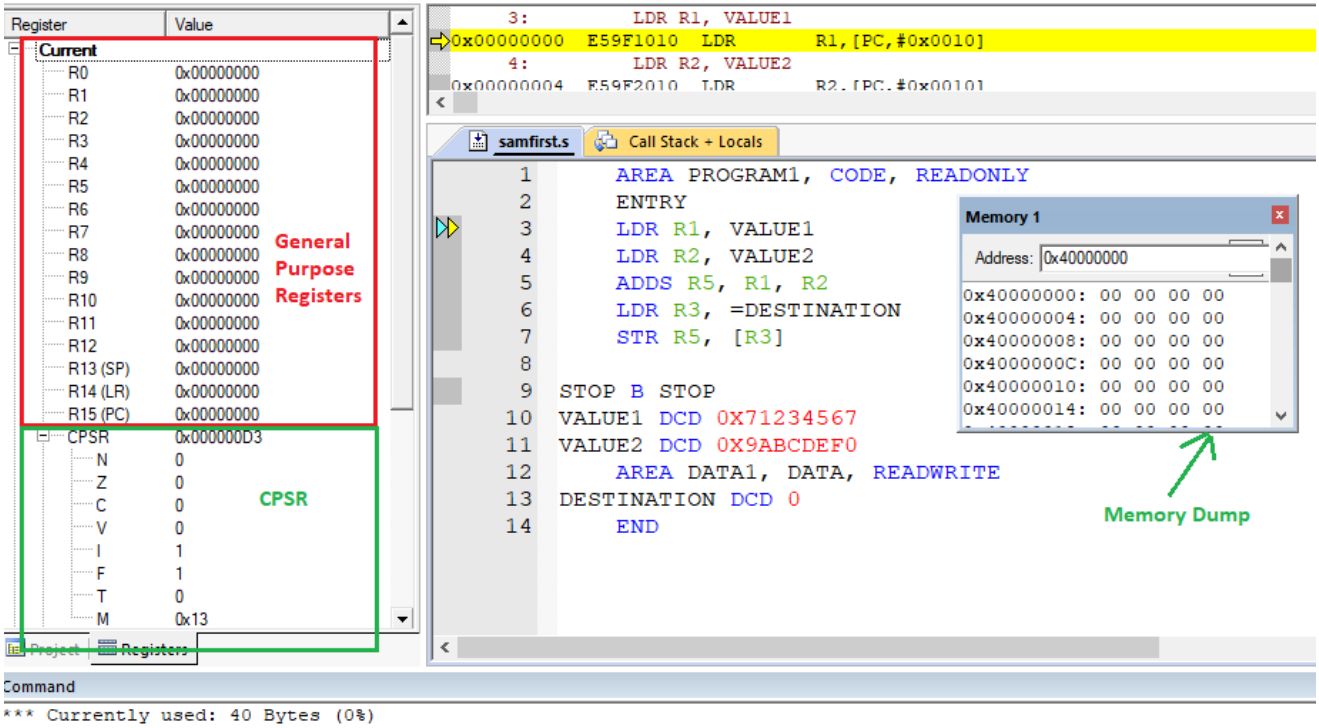
OBJECTIVE:
Get familiarize with KEIL SOFTWARE. In this program we are going to learn about the basic structure of ARM assembly program and symbols, instructions, directives, variable.In the process we will learn how to observe the register contents, memory dump, and CPSR.

NOTE:
- Users should create a working directory and in that directory project, programs should be stored.
- All user defined variables names, label names should be typed starting from Column 1.
- Example: Labels like MAIN, STOP, L1, L2, etc., variables like Value1, Value2 should start from Column 1 (should not have a space before it)
- All directives like TTY,AREA,ENTRY,END and instructions like LDR, MOV, STR, ADDS, SUB etc. should start after one tab space.
- Assembly program file should be stored with extension **.asm** or **.s**
- Always either type in uppercase or lower case but not in mixed case.

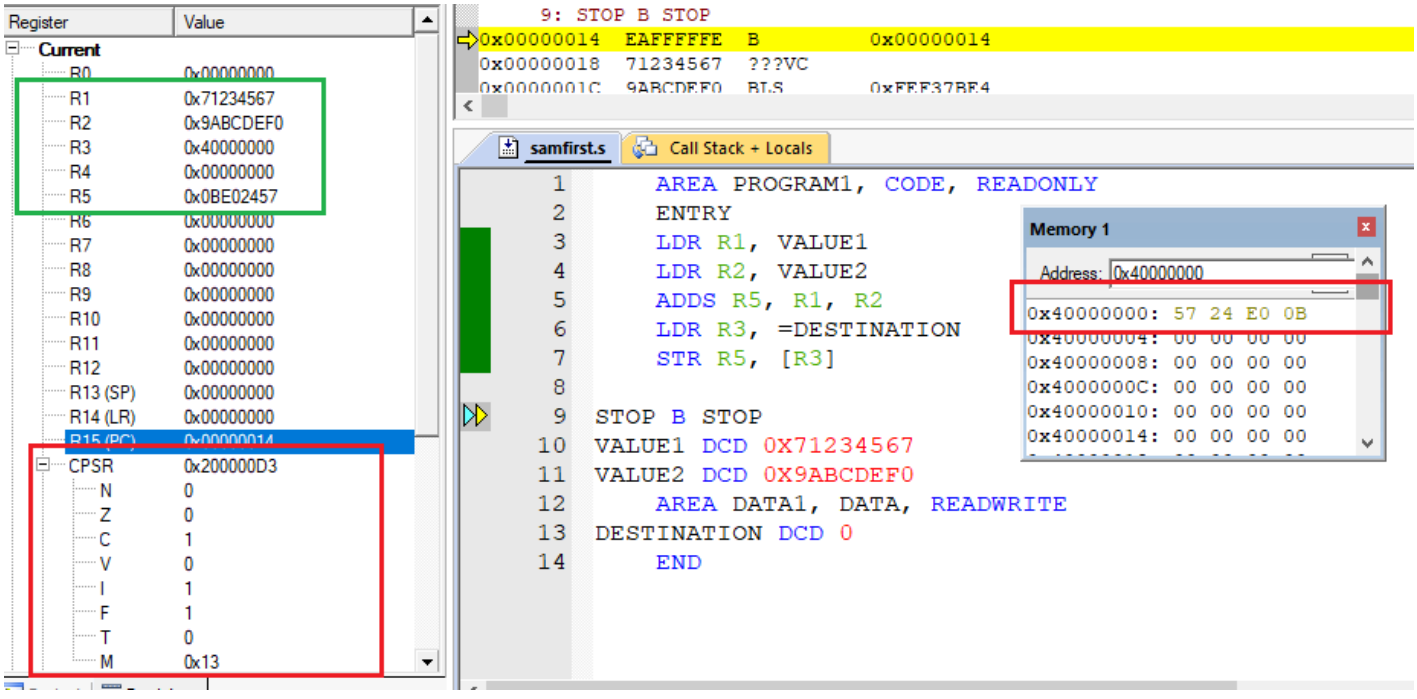
PROGRAM:

```
AREA PGM1, CODE, READONLY          ; Instructs the assembler to assemble a new code section
        ENTRY                      ; Marks the beginning of the code section
MAIN                               ; Label indicating the starting point (optional)
        LDR R1, VALUE1             ; Load the content of memory location VALUE1 to register R1.
        LDR R2, VALUE2             ; Load the content of memory location VALUE2 to register R2
        ADDS R1, R2                ; R1 = R1 + R2 and update the carry flag. Check the flag in CPSR
        LDR R3, =DESTINATION       ; Load the address of DESTINATION memory
        STR R1, [R3]               ; Store the sum R1 to memory location pointed to by R3.
STOP B STOP
VALUE1 DCD 0X71234567              ; Assign the value 0X71234567, and 0X9ABCDEF to memory location VALUE1 and VALUE2
VALUE2 DCD 0X9ABCDEF0
AREA DATA1, DATA, READWRITE        ; Create a data section to store the result with read-write permission.
DESTINATION DCD 0
        END
```

OUTPUT:



After the execution, observe the changes in contents of registers involved in the program, CPSR, and memory contents.



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

OBJECTIVE:

Get familiarize with KEIL SOFTWARE. In this program we are going to learn about the basic structure of ARM assembly program and symbols, instructions, directives, variable .In the process we will learn how to declare variables, data transfer, arithmetic, logic operations.

NOTE:

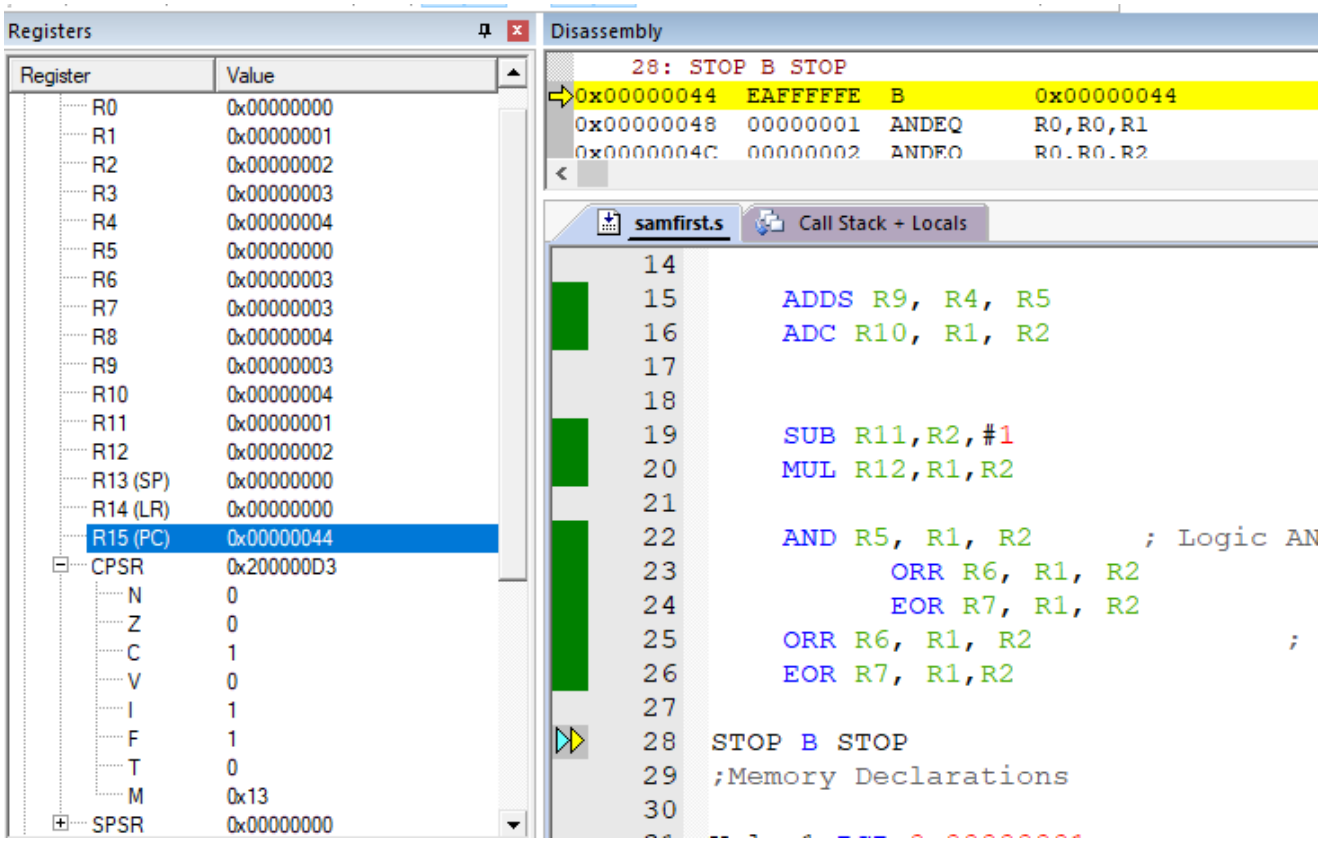
- Users should create a working directory and in that directory project, programs should be stored.
- All user defined variables names, label names should be typed starting from Column 1.
- Example: Labels like MAIN, STOP, L1, L2, etc., variables like Value1, Value2 should start from Column 1 (should not have a space before it)
- All directives like TTY,AREA,ENTRY,END and instructions like LDR, MOV, STR, ADDS, SUB etc. should start after one tab space.
- Assembly program file should be stored with extension **.asm** or **.s**
- Always either type in uppercase or lower case but not in mixed case.

PROGRAM:

```
    AREA PGM2, CODE, READONLY    ;Assembler directive to indicate code segment
  ENTRY;Marks beginning of the program

    LDR R1, Value1              ; Loads the contents of memory location Value1 to register R1
    LDR R2, Value2
    LDR R3, Value3
    LDR R4, Value4
    LDR R5, Value5

    MOV R6,R1                   ; Contents of R1 is moved to R6

    ADD R7,R1,R2                ;R7=R1+R2
    ADD R8,R3,#1                ;Add value #1 and R1 and store the result in R1

    ADDS R9,R4,R5
    ADC R10,R1,R2               ;add R1 and R2 with carry Bit from previous ADDS,
                                  ;store result to R10

    SUB R11,R2,#1
    MUL R12,R1,R2

    AND R5,R1,R2            ; Logic AND operation
ORR R6,R1,R2                ; Logic OR operation
EOR R7,R1,R2                ; Logic Ex-OR operation STR  R1, Result
    ORR R6,R1,R2            ; Logic OR operation
    EOR R7,R1,R2 ; Logic Ex-OR operation
STOP B STOP
;Memory Declarations
Value1 DCD 0x00000001
Value2 DCD 0x00000002
Value3DCD 0x00000003
Value4DCD 0x00000004
Value5 DCD 0xFFFFFFFF
    END
```

OUTPUT:

- Build the program and fix all the errors.
- Go to Debugging session to run the program and view the results.
- Perform single step execution and observe the changes in register contents.



Notes:

1. The semicolon indicates a user-supplied comment. Anything following a semicolon on the same line is ignored by the assembler.
2. The first line is AREA PROGRAM, CODE, READONLY is an assembler directive and is required to set up the program. It is a feature of the development system and not the ARM assembly language. An assembler from a different company may have a different way of defining the start of a program. In this case, AREA refers to the segment of code, PROGRAM is user defined name, and CODE indicates executable code rather than data and READONLY state that it cannot be modified at run time.
3. Anything starting in column 1 (in this case START) is a label that can be used to refer to that line.
4. The instruction STOP B STOP means 'Branch to the line labeled STOP' and is used to create an infinite loop. This is a convenient way of ending programs in simple examples like these.
5. The last line END is an assemble directive that tells the assembler there is not more code to follow. It ends the program.

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

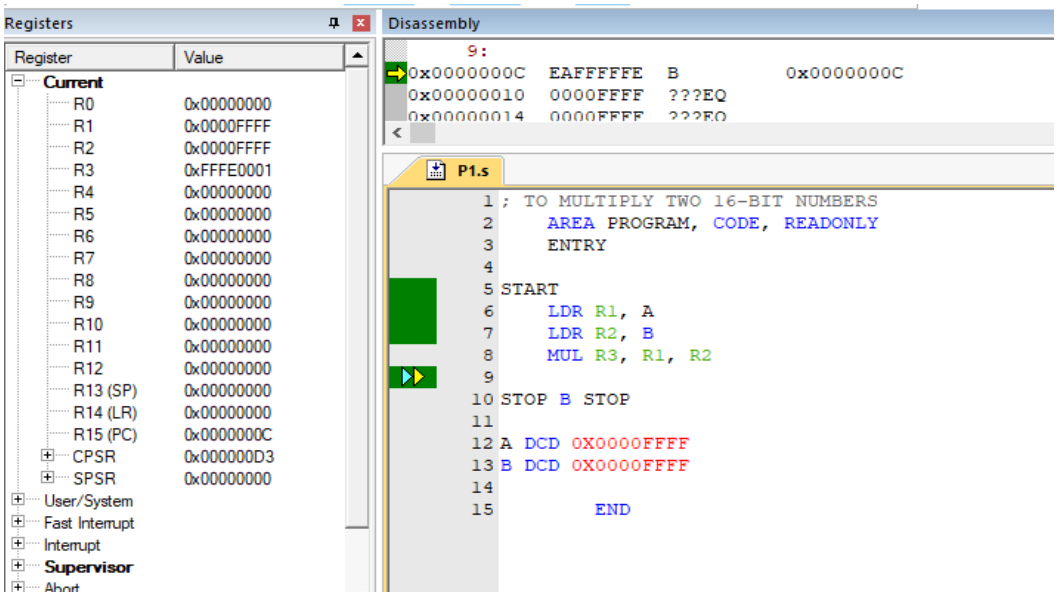
OBJECTIVE:
To to learn about Bit Number format and MUL instruction. The bit number format is as follows.

Number of Bits	Max. Hexadecimal value	Binary
1 bit	1	1
2 bit	3	11
3 bit	7	111
4 bit	F	1111
8 bit	FF	1111 1111
16 bit	FFFF	1111 1111 1111 1111
32 bit	FFFF FFFF	1111 1111 1111 1111 1111 1111 1111 1111

PROGRAM:

```
; TO MULTIPLY TWO 16-BIT NUMBERS

        AREA PGM3, CODE, READONLY          ;  Instructs the assembler to assemble a new code section

        ENTRY                              ;  Marks the beginning of the code section
START                                      ;  Label indicating the starting point (optional)
        LDR R1, A                          ;  Load the content of memory location A to register R1.

        LDR R2, B
        MUL R3, R1, R2                     ;  R3 = R1 * R2

STOP B STOP

A DCD 0X0000FFFF                           ;  Assign the value 0X0000FFFF to memory location A

B DCD 0X0000FFFF                           ;  Assign the value 0X0000FFFF to memory location

                END
```

OUTPUT:



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

OBJECTIVE:
To learn about the loop and branching instructions.

Branching:

Branching instruction is used to change the control flow. The Branch instruction can be used in two ways, conditionally and unconditionally. An unconditional branch will always branch no matter what the state of the flags, For a conditional branch, one of the condition codes is given immediately after B.

BEQ instruction is used to branch on equal condition (the Z flag is set).
BNE Branch if not equal
BHI Branch if higher than
BHS Branch if higher or same
BLO Branch if lower
BLS Branch if lower or same
BGT Branch if greater than
BGE Branch if greater than or equal
BLT Branch if less than
BLE Branch if less than or equal

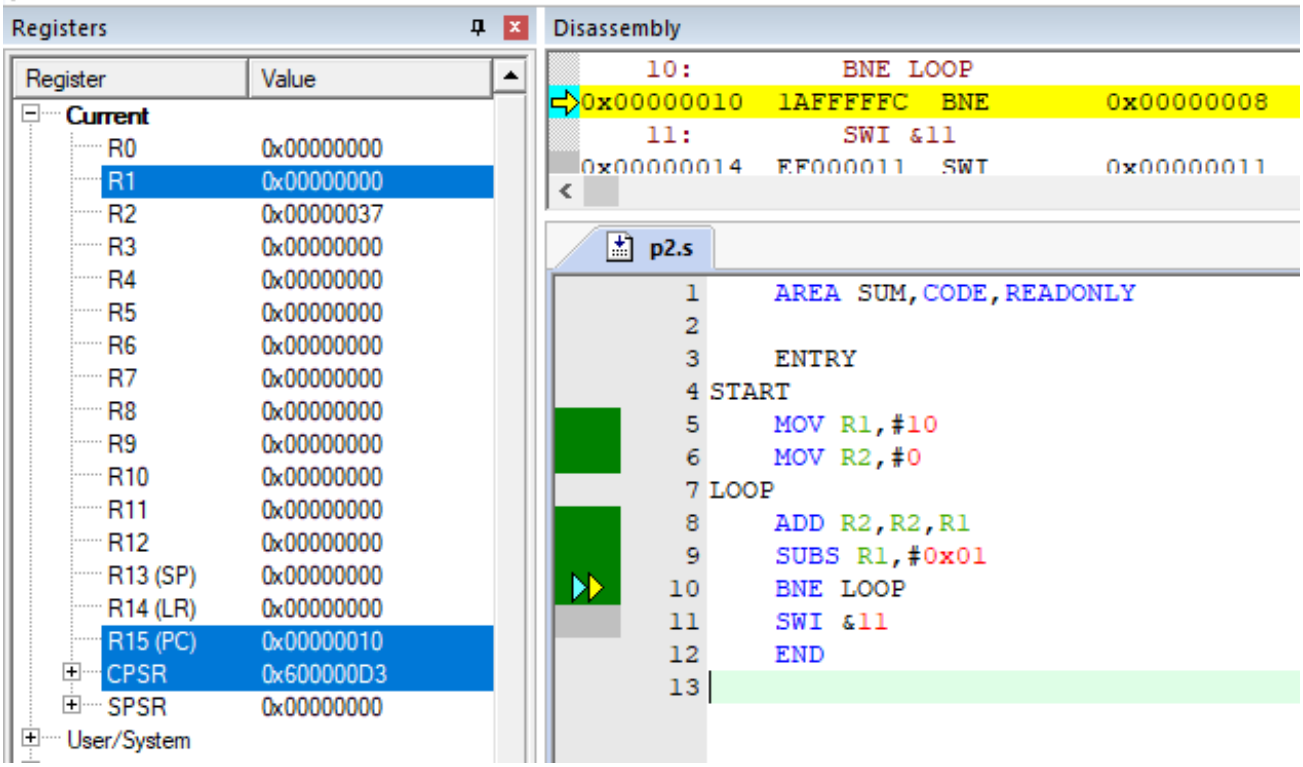
Example:

In this program we are going to add numbers from 10 to 1. The total we get will be 55. The total we get in Hexadecimal format will be in R2 register.

PROGRAM:

```
        AREA SUM,CODE,READONLY

        ENTRY
START
        MOV R1,#10          ;  Load 10 to register R1. R1 is a counter
        MOV R2,#0           ;  Empty the register to store result
LOOP
        ADD R2,R2,R1        ;  Add the content of R1 with result at R2
        SUBS R1,#0x01       ;  Decrement R1 by 1 and update the flags
        BNE LOOP            ;  BNE results in true value as long as R1 contains non-
                               zero value in it and takes the control to a line labeled
                               LOOP.When R1 becomeszero, BNE results in false
                               value and  theloop terminates
STOP B  STOP
       END
```

OUTPUT:


Registers
Register
Value
Current
R0 0x00000000
R1 0x00000000
R2 0x00000037
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000010
CPSR 0x600000D3
SPSR 0x00000000
User/System
Disassembly
10: BNE LOOP
0x00000010 1AFFFFFC BNE 0x00000008
11: SWI &11
p2.s
1 AREA SUM,CODE,READONLY
2
3 ENTRY
4 START
5 MOV R1,#10
6 MOV R2,#0
7 LOOP
8 ADD R2,R2,R1
9 SUBS R1,#0x01
10 BNE LOOP
11 SWI &11
12 END
13

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

OBJECTIVE: To learn about comparisons.
For comparisons in case of unsigned numbers we use HI, HS, LO, LS
For comparisons in case of signed numbers we use GT, GE, LT, LE
For generic we use EQ, NE

Example:

Input : Array containing 32 bit numbers

Output : Largest number is stored in location 0x40000000 and also in R2.

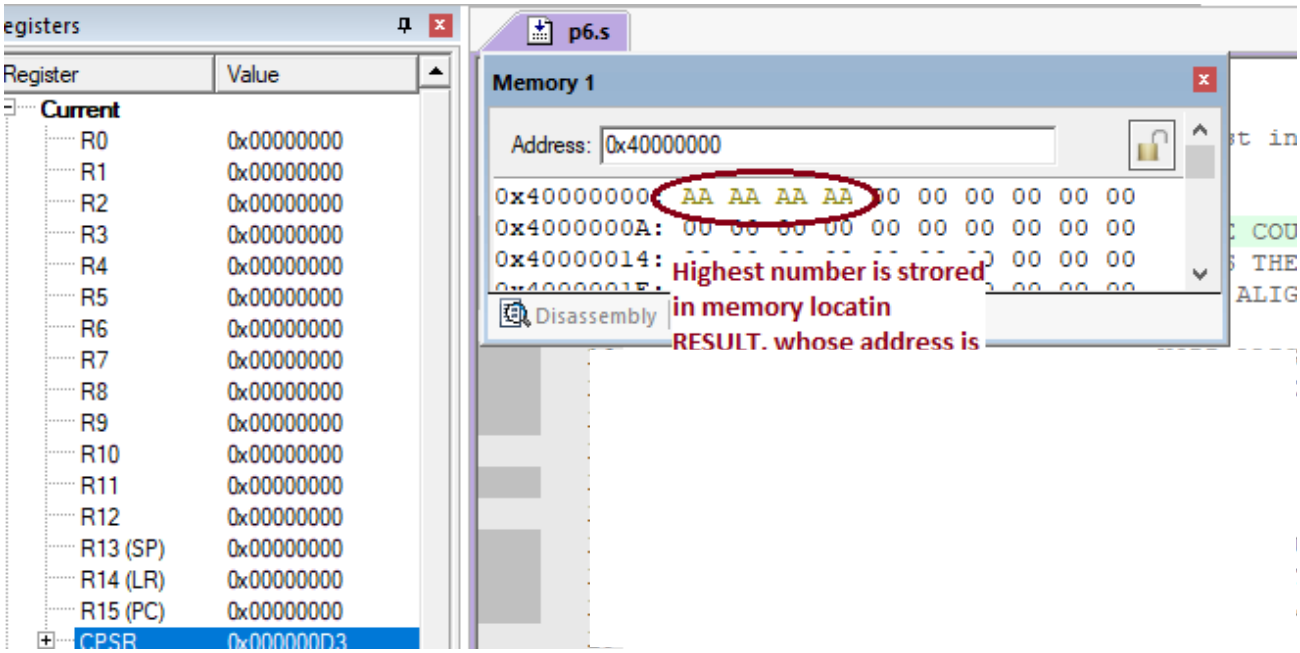
Check the result in R2 or memory location 0x40000000.

First Part – To find the Largest Number

PROGRAM:

```
AREA LARGEST, CODE, READONLY
ENTRY
START                                   ; mark first instruction to execute
        MOV R5,#6
        LDR R1,=MYARRAY                 ; initialize counter to 6(i.e. n=7)
        LDR R4,=RESULT                  ; loads the base address of the array MYARRAY to R1
LDR R2,[R1],#4                          ; loads the address of result to R4
                                        ; word align to array element. Load the content of memory
LOOP                                      location pointed to by R1 to R2, then increment R1 by 4.
        LDR R3,[R1],#4
        CMP R2,R3                       ; Load the next element to R3, and word align to array element
        MOVLO R2, R3                    ; compare numbers
                                        ; If R2 < R3, R3 is considered as highest and move to R2 –
        SUBS R5,R5,#1                     unsigned comparisons
        BNE LOOP                        ;       decrement counter and update the status flag
        STR R2,[R4]                     ; loop back till array ends
STOP B STOP                             ; stores the result where R4 is pointed to
MYARRAY
                DCD     0X44444444     ; ARRAY OF 32 BIT NUMBERS(N=7)
                DCD     0X22222222
                DCD     0X11111111
                DCD     0X33333333
                DCD     0XAAAAAAAA
                DCD     0X88888888
                DCD     0X99999999

        AREA DATA2,DATA,READWRITE
RESULT DCD 0
END                                     ; to store result in given address
                                        ; Mark end of file
```

OUTPUT:



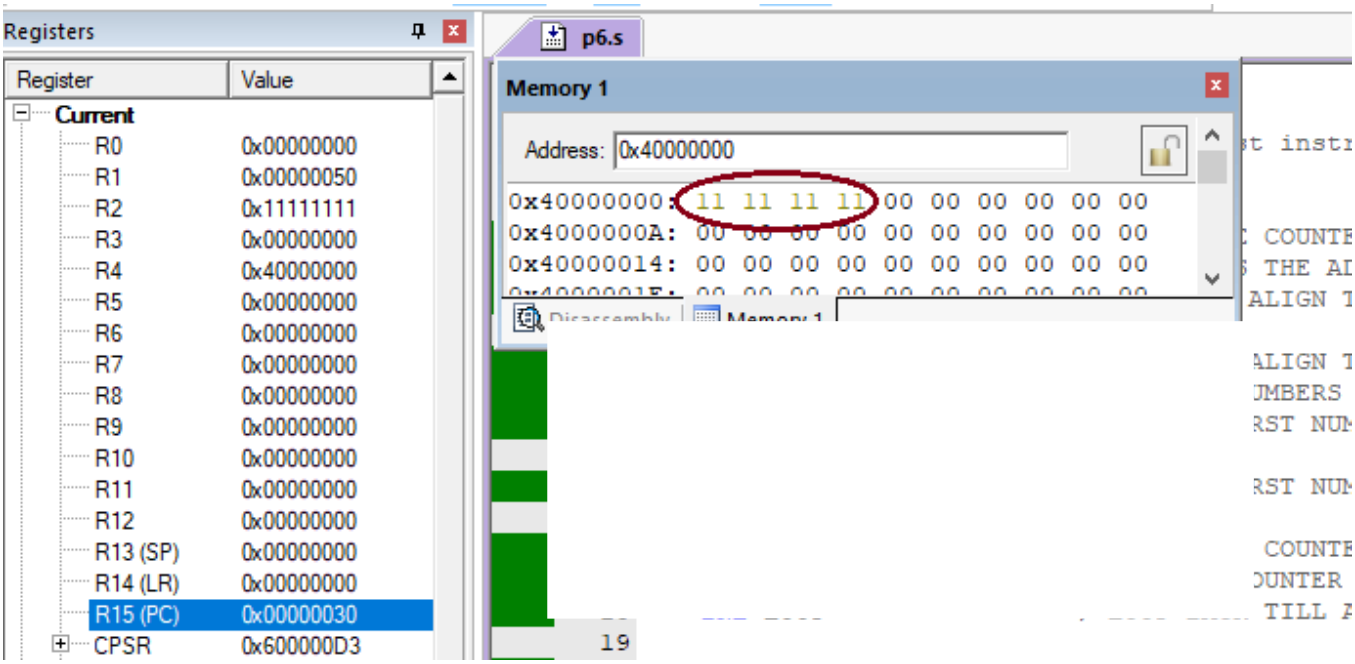
Second Part – To find the Smallest Number.

PROGRAM:

```
        AREA SMALLEST, CODE, READONLY
        ENTRY
START                                   ; mark first instruction to execute
        MOV R5, #6
        LDR R1,=MYARRAY                 ; initialize counter to 6(i.e. n=7)
        LDR R4,=RESULT                  ; loads the base address of the array MYARRAY to R1
        LDR R2, [R1], #4                ; loads the address of result to R4
                                        ; word align to array element. Load the content of memory
                                          location pointed to by R1 to R2, then increment R1 by 4.
LOOP
        LDR R3, [R1], #4
                                        ; Load the next element to R3, and word align to array element
        CMP R2, R3                      ; compare numbers
        MOVHS R2, R3                    ; If R2 >= R3, R3 is considered as highest and move to R2 -
                                          unsigned comparisons
        SUBS R5, R5, #1                 ;       decrement counter and update the status flag
        BNE LOOP                        ; loop back till array ends
        STR R2, [R4]                    ; stores the result  where R4 is pointed to
STOP B STOP
MYARRAY                                 ; ARRAY OF 32 BIT NUMBERS(N=7)
        DCD     0X44444444
        DCD     0X22222222
        DCD     0X11111111
        DCD     0X33333333
        DCD     0XAAAAAAAA
        DCD     0X88888888
```

```
                DCD     0X99999999

                                        ; to store result in given address
        AREA DATA2,  DATA,   READWRITE
RESULT  DCD  0                          ; Mark end of file
        END
```

OUTPUT:



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

OBJECTIVE:

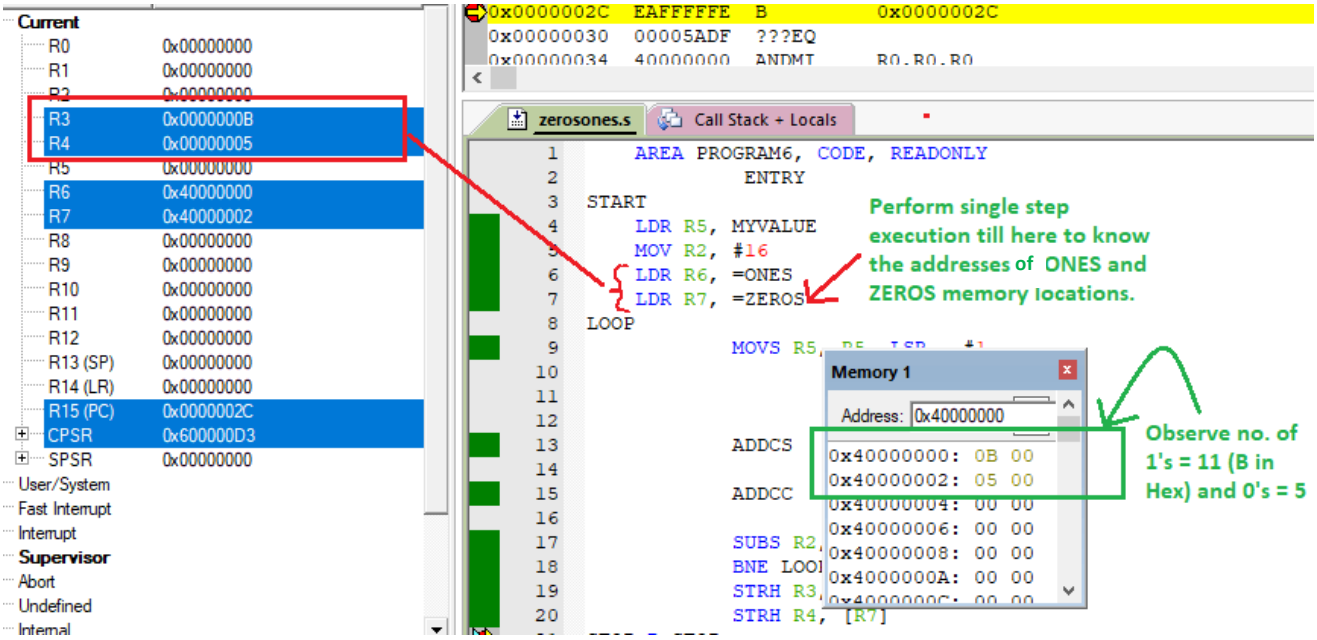
- To learn about barrel shift instructions – LSR
- To understand how conditional execution simplifies the program development.
- To learnt new instructions ADDCS and ADDCC

LSR – LOGICAL SHIFT RIGHT: Right Shifts behave like dividing the contents of a register by 2^s where s is the shift amount, if you assume the contents of the register are unsigned.

PROGRAM:

```
        AREA COUNT, CODE, READONLY
        ENTRY
START                                   ; mark first instruction to execute
        LDR R5, MYVALUE                 ; Load the contents of memory location MYVALUE to R5
MOV R2, #16                             ; Since we need to count 16 bits, load this counter to R2
LDR R6, =ONES                           ; Load the address of memory location 'ONES' to R6
LDR R7, =ZEROS                          ; Load the address of memory location 'ZEROS' to R7
LOOP
        MOVS R5, R5, LSR  #1            ; Logical shift Right the contents of R5 and move back the
                                          shifted value to R5 also update the status. If the shifted bit
                                          will be moved to carry flag. If the shifted bit is 1, CF
                                          becomes 1, otherwise it becomes 0.
ADDCS   R3, #1                          ; R3 will be incremented by 1 if the CF = 1. ADDCS – add if
                                          and only is carry flag is set
 ADDCC   R4, #1                         ; R4 will be incremented by 1 if the CF = 0 - ADDCC – add
                                          if and only is carry flag is cleared
SUBS R2, #1                             ; Decrement the counter R2 and update the status flag.
        BNE LOOP                        ; If R2 is not 0, process the next bit.
STRH R3, [R6]                           ; Store half-word: ONES and ZEROS are declared as DCW
STRH R4, [R7]                             that allocates 16 bits (or half words). The contents of R3
                                          and R4 are respectively stored where R6 and R7 are
                                          pointing to.
STOP B STOP
MYVALUE    DCW        0X5ADF
      AREA DATA1, DATA, READWRITE
ONES DCW 0                              ; Data section is created with read and write permission to
ZEROS DCW                                 store number of ones and zeros in memory locations ONES
                                          and ZEROS respectively.
        END
```

OUTPUT:


Current
R0 0x00000000
R1 0x00000000
R3 0x0000000B
R4 0x00000005
R6 0x40000000
R7 0x40000002
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x0000002C
CPSR 0x600000D3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
0x0000002C EAFFFFFE B 0x0000002C
0x00000030 00005ADF ???EQ
zerosones.s
Call Stack + Locals
AREA PROGRAM6, CODE, READONLY
ENTRY
START
LDR R5, MYVALUE
MOV R2, #16
LDR R6, =ONES
LDR R7, =ZEROS
LOOP
ADDCS
ADDCC
SUBS R2,
BNE LOO
STRH R3,
STRH R4, [R7]
Perform single step execution till here to know the addresses of ONES and ZEROS memory locations.
Memory 1
Address: 0x40000000
0x40000000: 0B 00
0x40000002: 05 00
0x40000004: 00 00
0x40000006: 00 00
0x40000008: 00 00
0x4000000A: 00 00
Observe no. of 1's = 11 (B in Hex) and 0's = 5

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

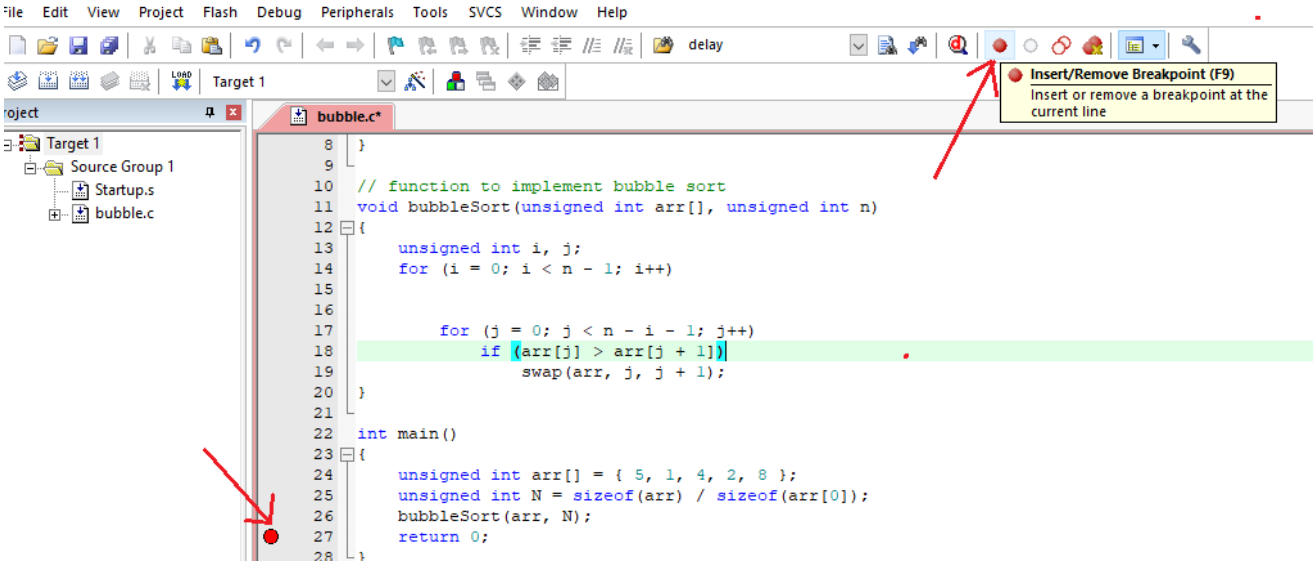
OBJECTIVE:
To gain hands-on experience in analyzing memory usage and register allocation, by observing how values are assigned to registers and tracking changes in register contents during the sorting process,

PROGRAM:

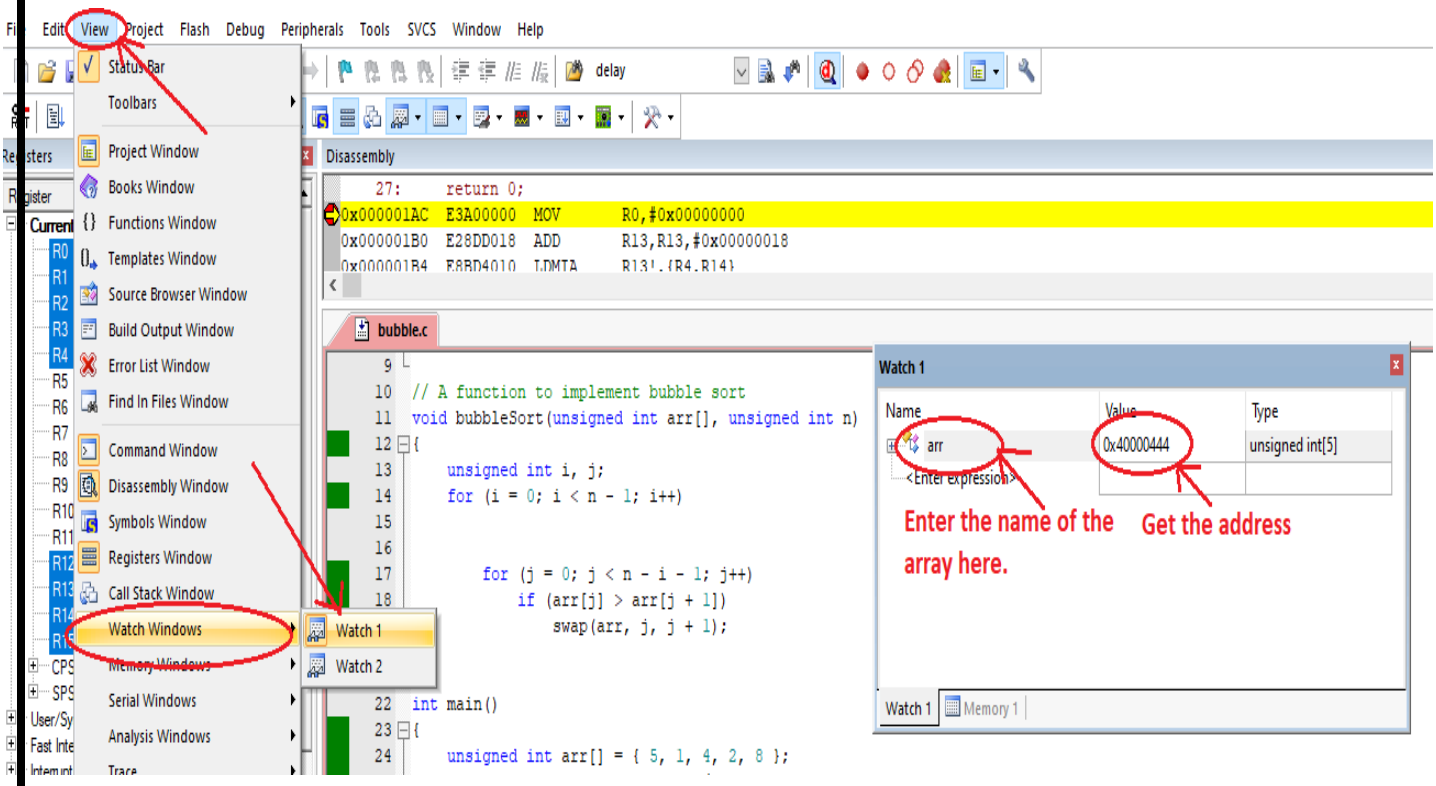
```
#include <lpc21xx.h>

void swap(unsigned int* arr, unsigned int i, unsigned int j)
{
   unsigned int temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
}

// function to implement bubble sort
void bubbleSort(unsigned int arr[], unsigned int n)
{
   unsigned int i, j;
   for (i = 0; i < n - 1; i++)


      for (j = 0; j < n - i - 1; j++)
         if (arr[j] > arr[j + 1])
            swap(arr, j, j + 1);
}

int main()
{
   unsigned int arr[] = { 5, 1, 4, 2, 8 };
   unsigned int N = sizeof(arr) / sizeof(arr[0]);
   bubbleSort(arr, N);
   return 0;
}
```

OUTPUT:
Step 1: Place a breakpoint for return statement. Place the cursor where breakpoint is needed and click on Insert/Remove breakpoint as shown in the below snapshot.
Step 2: Build the program
Step 3: Start debugging session.
Step 4: View → Watch Windows → Watch 1
Step 5: In the watch window, enter the name of the variable and name of the array.
Step 6: Get the address of the required variable. Go to view → Memory Windows → Memory. Type the address in memory window address bar.
Step 7: Run the program and observe the contents in the given location.

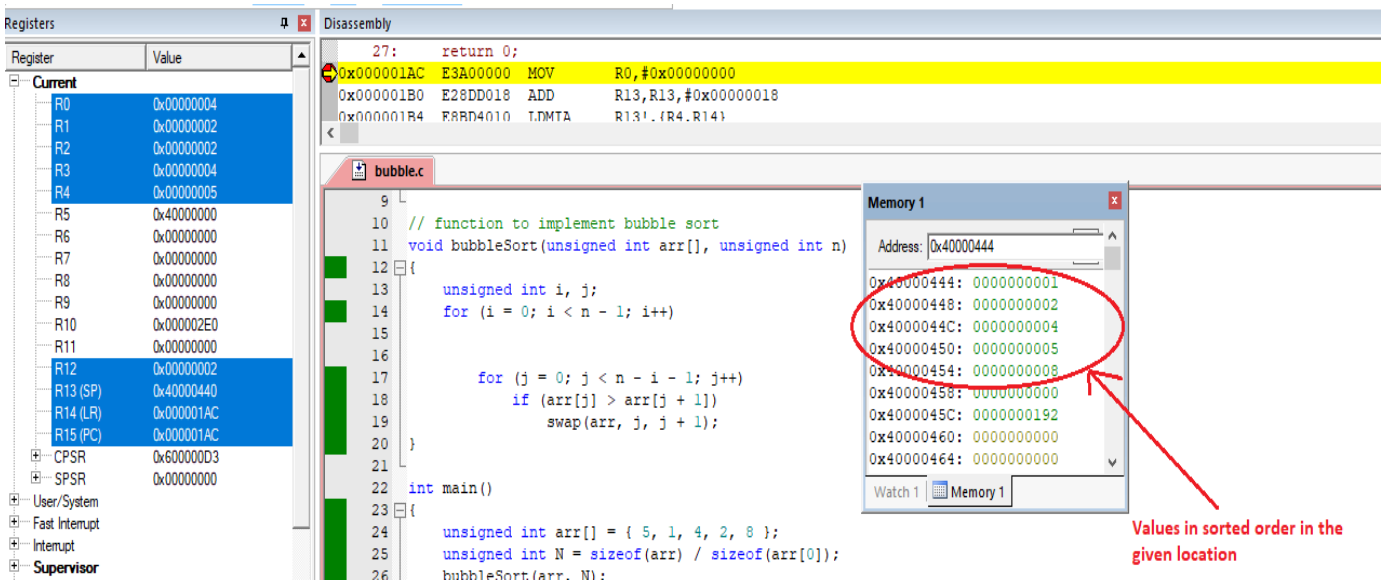
Breakpoint:



Watch window



Go to view → Memory Windows → Memory. Type the address in memory window address bar.


Registers
Register
Value
Current
R0 0x00000004
R1 0x00000002
R2 0x00000002
R3 0x00000004
R4 0x00000005
R5 0x40000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x000002E0
R11 0x00000000
R12 0x00000002
R13 (SP) 0x40000440
R14 (LR) 0x000001AC
R15 (PC) 0x000001AC
CPSR 0x600000D3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Disassembly
27: return 0;
0x000001AC E3A00000 MOV R0,#0x00000000
0x000001B0 E28DD018 ADD R13,R13,#0x00000018
bubble.c
// function to implement bubble sort
void bubbleSort(unsigned int arr[], unsigned int n)
{
unsigned int i, j;
for (i = 0; i < n - 1; i++)
for (j = 0; j < n - i - 1; j++)
if (arr[j] > arr[j + 1])
swap(arr, j, j + 1);
}
int main()
{
unsigned int arr[] = { 5, 1, 4, 2, 8 };
unsigned int N = sizeof(arr) / sizeof(arr[0]);
Memory 1
Address: 0x40000444
0x40000444: 0000000001
0x40000448: 0000000002
0x4000044C: 0000000004
0x40000450: 0000000005
0x40000454: 0000000008
0x40000458: 0000000000
0x4000045C: 0000000192
0x40000460: 0000000000
0x40000464: 0000000000
Watch 1
Memory 1
Values in sorted order in the given location

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

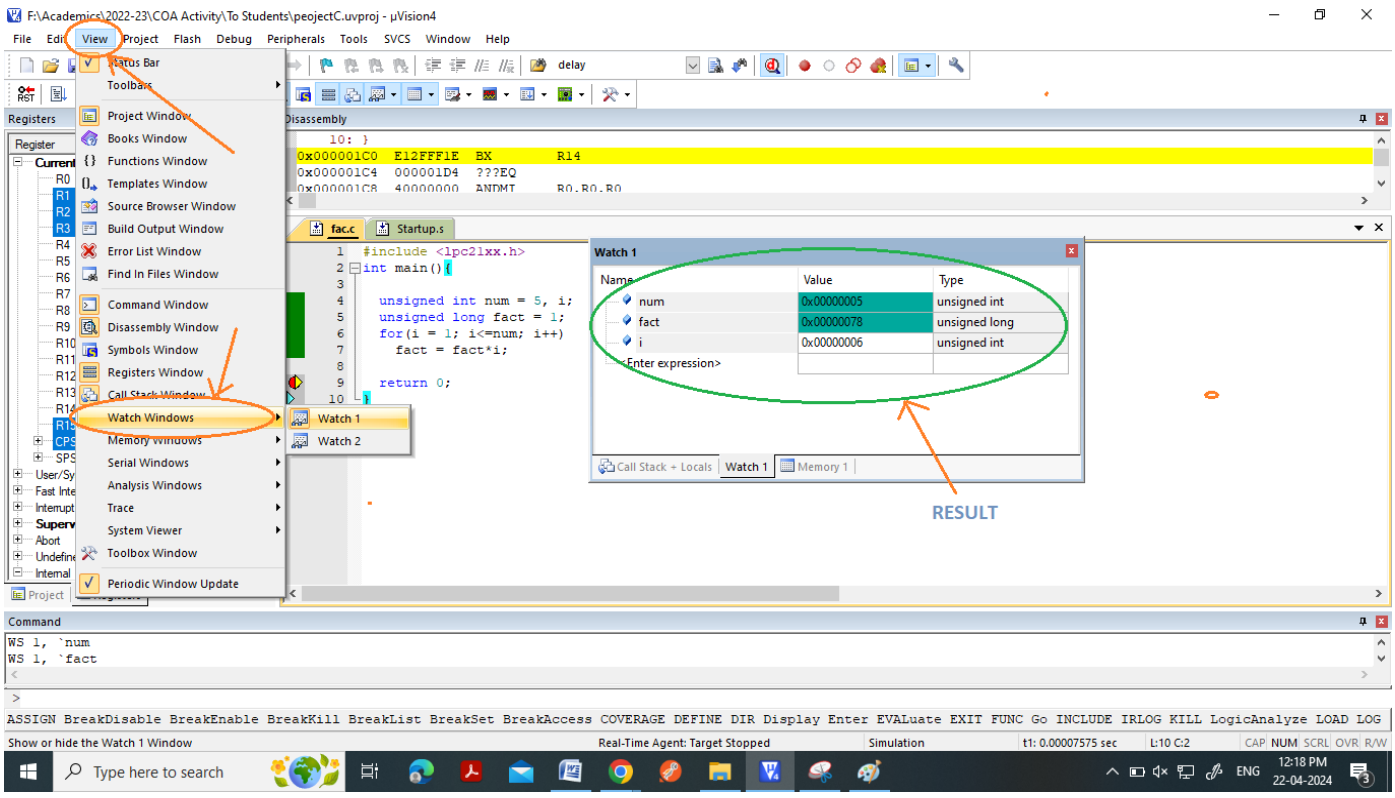
OBJECTIVE:

To gain insight into the intricacies of register-level operations and memory management, by observing how values are assigned to registers and tracking changes in register contents during program execution.

PROGRAM:

```
#include <lpc21xx.h>
int main(){

        unsigned int num = 5, i;
        unsigned long fact = 1;
        for(i = 1; i<=num; i++)
                fact = fact*i;

        return 0;
}
```

OUTPUT:



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

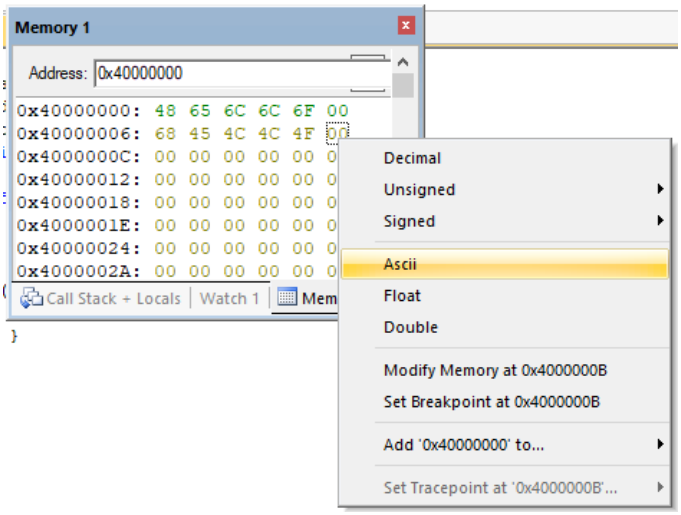
OBJECTIVE:

To understand the process of memory dump analysis and ASCII representation.

In Keil uVision, when we perform a memory dump and the values are displayed in hexadecimal format, we can easily see their equivalent ASCII characters by looking at the ASCII representation of each hexadecimal value.

Typically, in the memory dump window, there will be two columns - one displaying the memory address and the other displaying the memory contents in hexadecimal format. To see the ASCII representation:

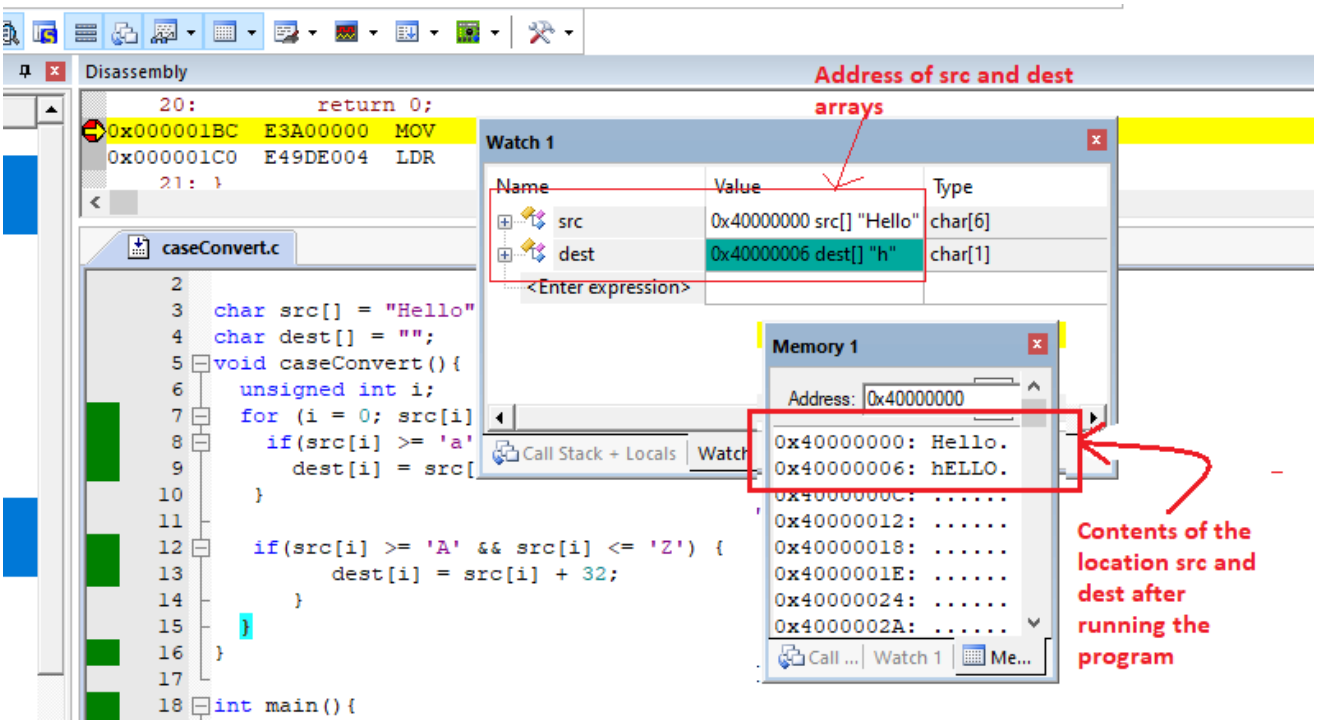
1. Locate the column displaying the hexadecimal values.
2. For each byte of memory content displayed in hexadecimal, you can refer to the ASCII representation of that byte by **right clicking on it and changing to "ASCII"**



PROGRAM:

```
# include <lpc21xx.h>
char src[] = "Hello";
char dest[] = "";
void caseConvert(){
        unsigned int i;
        for (i = 0; src[i]!='\0'; i++){
                if(src[i] >= 'a' && src[i] <= 'z') {
dest[i] = src[i] - 32;
}

                if(src[i] >= 'A' && src[i] <= 'Z') {
                        dest[i] = src[i] + 32;
                }
        }
}

int main(){
        caseConvert();
        return 0;
}
```

OUTPUT:


Disassembly
20: return 0;
0x000001BC E3A00000 MOV
0x000001C0 E49DE004 LDR
21: }
caseConvert.c
2
3 char src[] = "Hello"
4 char dest[] = "";
5 void caseConvert(){
6 unsigned int i;
7 for (i = 0; src[i]
8 if(src[i] >= 'a'
9 dest[i] = src[
10 }
11
12 if(src[i] >= 'A' && src[i] <= 'Z') {
13 dest[i] = src[i] + 32;
14 }
15 }
16 }
17
18 int main(){
Address of src and dest arrays
Watch 1
Name
Value
Type
src
0x40000000 src[] "Hello"
char[6]
dest
0x40000006 dest[] "h"
char[1]
<Enter expression>
Call Stack + Locals
Watch
Memory 1
Address: 0x40000000
0x40000000: Hello.
0x40000006: hELLO.
0x40000012: ......
0x40000018: ......
0x4000001E: ......
0x40000024: ......
0x4000002A: ......
Call ...
Watch 1
Me...
Contents of the location src and dest after running the program

10. Demonstrate enabling and disabling of Interrupts in ARM.

OBJECTIVE:

To learn about inline assembly in Embedded C, specifically the use of MSR and MRS instructions for managing the CPSR register on an ARM7 processor. Additionally, the program aims to understand how these instructions function within the Supervisor (svc) mode to effectively disable/enable interrupts.

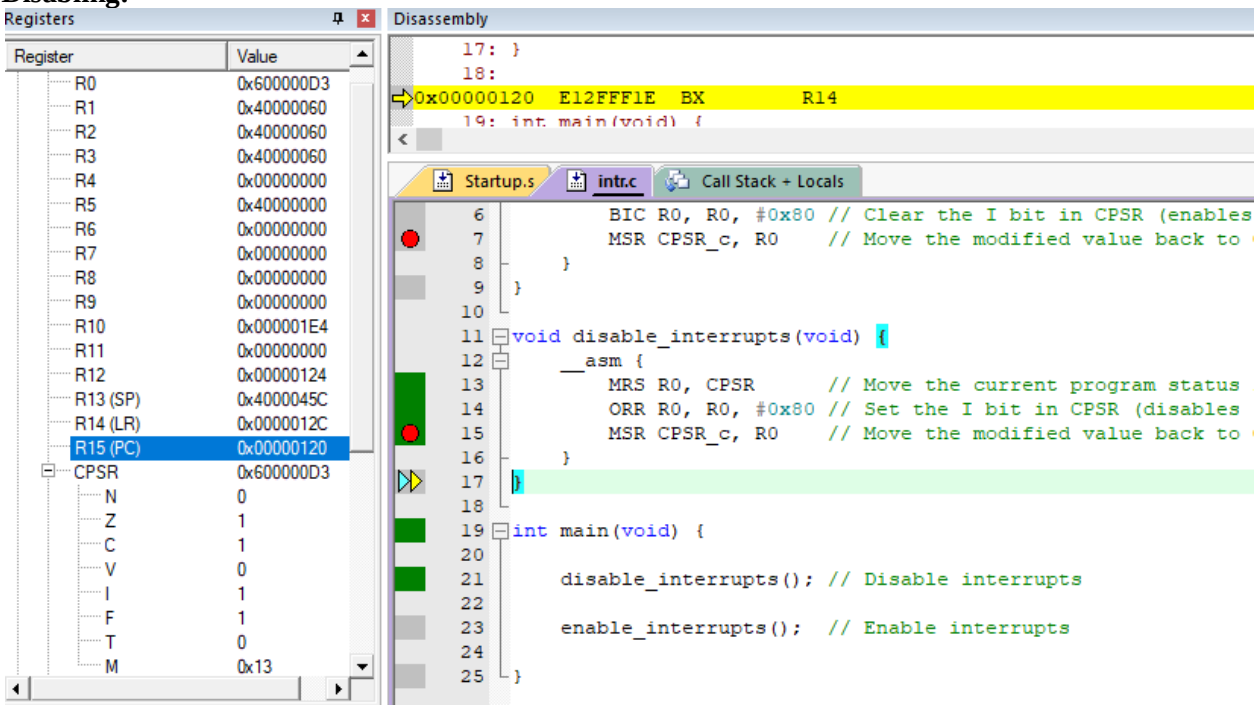
PROGRAM:

```
#include <LPC21xx.h> // Include the header file for the specific ARM7 microcontroller

void enable_interrupts(void) {
    __asm {
        MRS R0, CPSR      // Move the current program status register to R0
        BIC R0, R0, #0x80 // Clear the I bit in CPSR (enables IRQ)
        MSR CPSR_c, R0    // Move the modified value back to CPSR
    }
}
void disable_interrupts(void) {
    __asm {
        MRS R0, CPSR      // Move the current program status register to R0
        ORR R0, R0, #0x80 // Set the I bit in CPSR (disables IRQ)
        MSR CPSR_c, R0    // Move the modified value back to CPSR
    }
}

int main(void) {
    disable_interrupts(); // Disable interrupts
    enable_interrupts();  // Enable interrupts
}
```

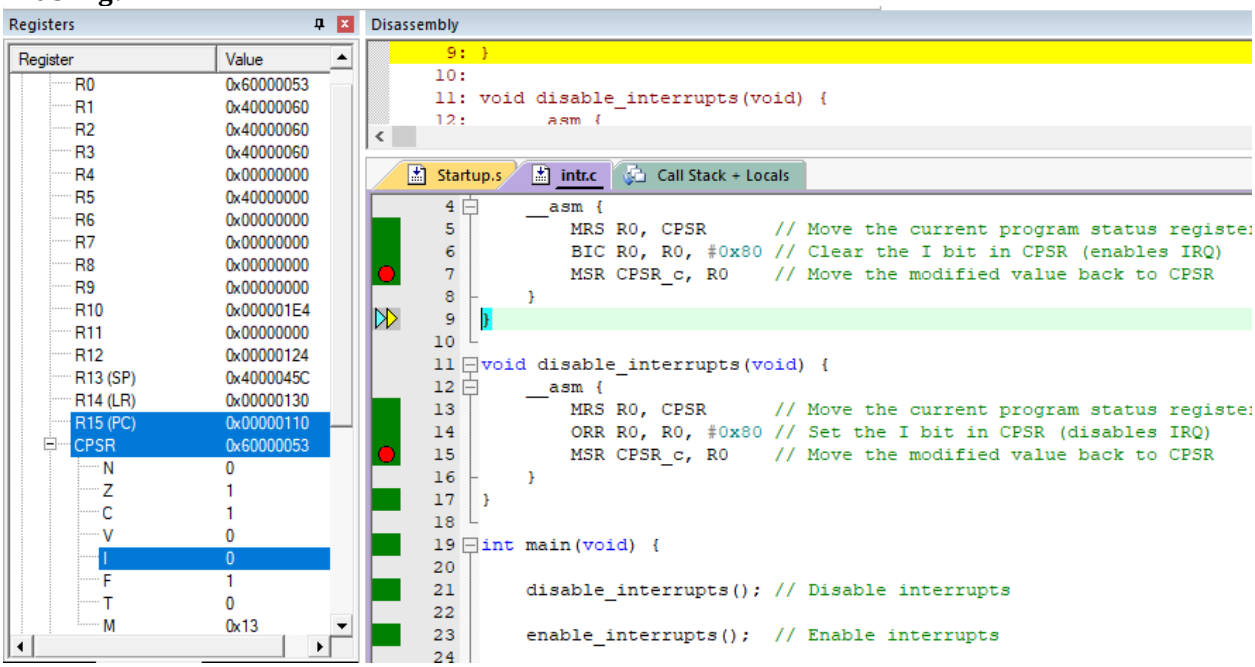
OUTPUT:

Accept startup.s file as it is needed for our application program to start in supervisor (SVC) mode. Only in privileged mode, changes can be done to CPSR and hence enable and disable IRQ effectively.

Disabling:



Enabling:


Registers
Register
Value
R0 0x60000053
R1 0x40000060
R2 0x40000060
R3 0x40000060
R4 0x00000000
R5 0x40000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x000001E4
R11 0x00000000
R12 0x00000124
R13 (SP) 0x4000045C
R14 (LR) 0x00000130
R15 (PC) 0x00000110
CPSR 0x60000053
N 0
Z 1
C 1
V 0
I 0
F 1
T 0
M 0x13
Disassembly
9: }
10:
11: void disable_interrupts(void) {
12: asm {
Startup.s
intr.c
Call Stack + Locals
4 __asm {
5 MRS R0, CPSR // Move the current program status registe
6 BIC R0, R0, #0x80 // Clear the I bit in CPSR (enables IRQ)
7 MSR CPSR_c, R0 // Move the modified value back to CPSR
8 }
9 }
10
11 void disable_interrupts(void) {
12 __asm {
13 MRS R0, CPSR // Move the current program status registe
14 ORR R0, R0, #0x80 // Set the I bit in CPSR (disables IRQ)
15 MSR CPSR_c, R0 // Move the modified value back to CPSR
16 }
17 }
18
19 int main(void) {
20
21 disable_interrupts(); // Disable interrupts
22
23 enable_interrupts(); // Enable interrupts
24

11. Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM.

OBJECTIVE: To learn about exception vectors, SWI handling and running the code in SVC mode

EXPLANATION:

1.Exception Vectors:

- The vector table points to the corresponding handlers for reset, undefined instruction, software interrupt (SWI), prefetch abort, data abort, IRQ, and FIQ.

2.SWI Handler:

- The SWI_Handler reads the SWI number by loading the instruction that caused the SWI and masking out the upper 24 bits to isolate the SWI number.
- Depending on the value of the SWI number (R0), it branches to the appropriate handler (Handle_Divide_By_Zero, Handle_Invalid_Operation, Handle_Overflow).

3.Exception Handlers:

- Handle_Divide_By_Zero, Handle_Invalid_Operation, and Handle_Overflow each load a specific value into R0 to simulate handling the exception, then branch to SWI_Return.

3. SWI Return:

- The SWI_Return handler copies LR to PC and returns to the instruction following the SWI.

5.Reset Handler:

- Initializes the stack pointer.
- Triggers the three exceptions using SWI #0, SWI #1, and SWI #2, each setting R0 to 0, 1, and 2 respectively.
- Enters an infinite loop to halt the processor.
-

Running the Code in Keil

4. Create a new project in Keil and add the startup.s file.
5. Delete the contents of startup.s automatically created and Copy the below program in startup.s
6. Run the project and use the Keil debugger to observe how each exception is handled, specifically by checking the values loaded into R0.
   i. Value 0xDDDDDDDD in R0 indicates the handling of Divide by Zero exception
   ii. Value 0xBBBBBBBB in R0 indicates the handling of Invalid Operation Exception
   iii. Value 0xFFFFFFFF in R0 indicates the handling of Overflow Exception

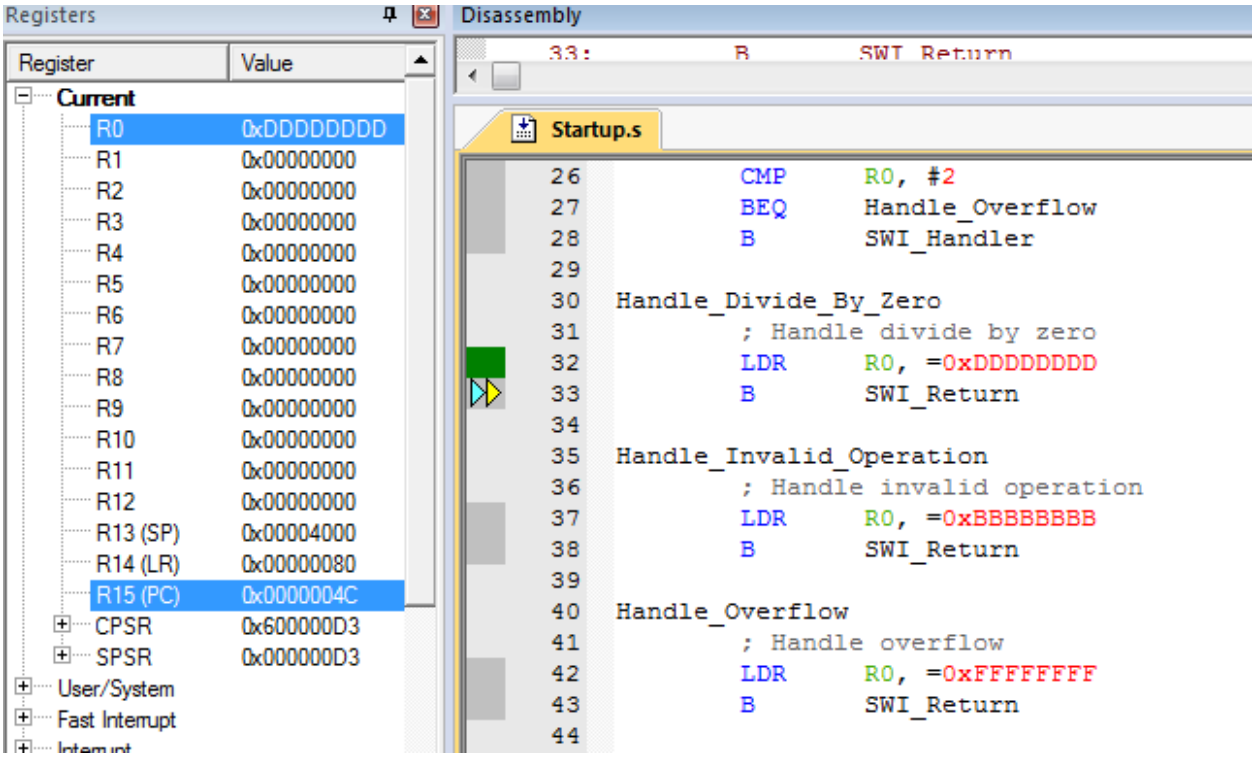
PROGRAM:

```
AREA RESET, CODE, READONLY
    ENTRY
    ; Vector Table
    LDR    PC, =Reset_Handler        ; Reset vector
    LDR    PC, =Undefined_Handler    ; Undefined instruction vector
    LDR    PC, =SWI_Handler          ; Software interrupt vector
    LDR    PC, =Prefetch_Handler     ; Prefetch abort vector
    LDR    PC, =Data_Handler         ; Data abort vector
    NOP                              ; Reserved
    LDR    PC, =IRQ_Handler          ; IRQ vector
    LDR    PC, =FIQ_Handler          ; FIQ vector

    ; Exception Handlers
Undefined_Handler
    B      Undefined_Handler         ; Undefined instruction handler
SWI_Handler
    ; Check which SWI was triggered and handle accordingly
    LDR    R0, [LR, #-4]             ; Load the instruction that caused the SWI
    BIC    R0, R0, #0xFF000000       ; Mask to get SWI number
    CMP    R0, #0
```

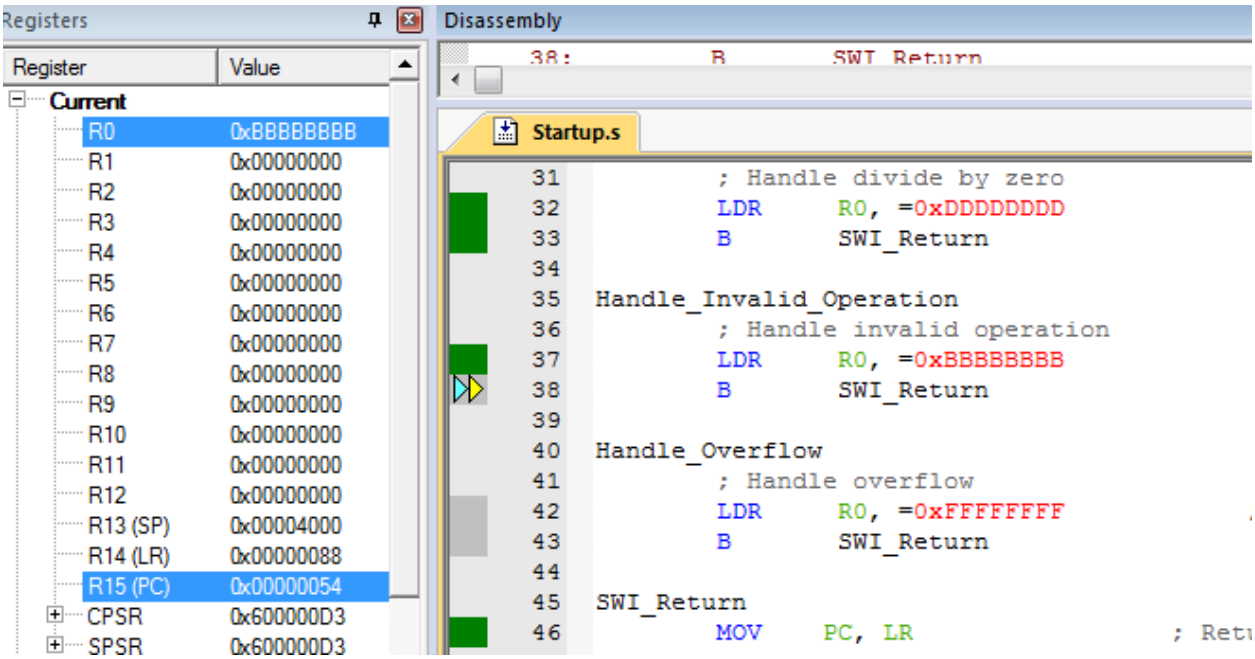
```
        BEQ     Handle_Divide_By_Zero
        CMP     R0, #1
        BEQ     Handle_Invalid_Operation
        CMP     R0, #2
        BEQ     Handle_Overflow
        B       SWI_Handler             ; Unhandled SWI

Handle_Divide_By_Zero
        ; Handle divide by zero
        LDR     R0, =0xDDDDDDDD         ; Example handling action
        B       SWI_Return

Handle_Invalid_Operation
        ; Handle invalid operation
        LDR     R0, =0xBBBBBBBB         ; Example handling action
        B       SWI_Return

Handle_Overflow
        ; Handle overflow
        LDR     R0, =0xFFFFFFFF         ; Example handling action
        B       SWI_Return

SWI_Return
        MOVS    PC, LR                  ; Return from SWI handler

Prefetch_Handler
        B       Prefetch_Handler        ; Prefetch abort handler
Data_Handler
        B       Data_Handler            ; Data abort handler
IRQ_Handler
        B       IRQ_Handler             ; IRQ handler
FIQ_Handler
        B       FIQ_Handler             ; FIQ handler

        ; Reset Handler
Reset_Handler
        LDR     SP, =0x4000             ; Initialize stack pointer

        ; Trigger divide by zero exception
        MOV     R0, #0
        SWI     #0

        ; Trigger invalid operation exception
        MOV     R0, #1
        SWI     #1

        ; Trigger overflow exception
        MOV     R0, #2
        SWI     #2

        ; Infinite loop to halt the processor
        B       .

        END
```

OUTPUT:

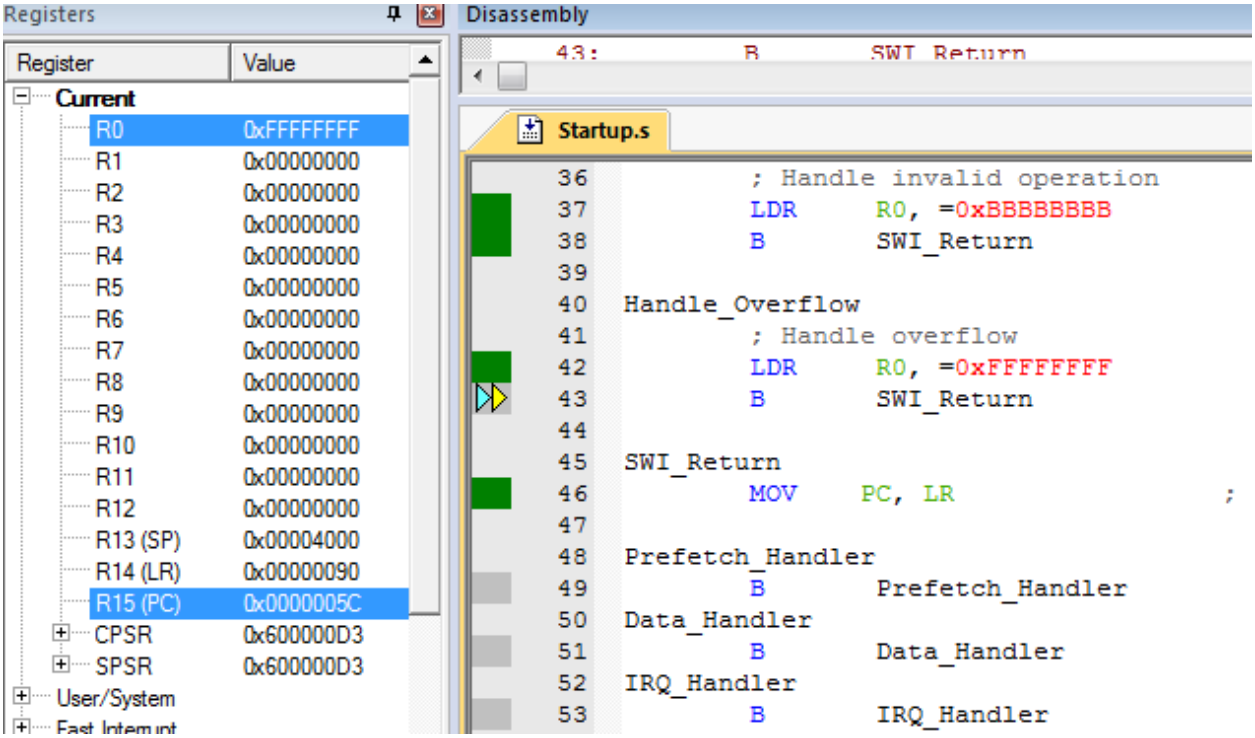
Triggering and handling Divide by Zero Exception (SWI 0 in the program)



Triggering and handling Invalid Operation Exception (SWI 1 in the program)



Triggering and handling Overflow Exception (SWI 2 in the program)



VIVA QUESTIONS

1. What is the purpose of following assembler directives?
   AREA CODE DATA ENTRY START STOP END
2. DEFINE: DCD, DCB,DCW
3. What is ARM (TDMI)?
4. Mention the design features of ARM.
5. Name different branch instructions used in our lab programs
6. What is the difference between SUB and SUBS?
7. How do we compile and run an assembly program?
8. What is the difference between ALL and HLL?
9. Difference between conditional and unconditional branch?
10. Name the status registers? What is difference between them?
11. Define all the bits of CPSR.
12. List the ARM processor modes.
13. How many registers are present in ARM microcontroller?
14. What is the register numbers of LR, SP, PC?
15. Explain the purpose of all the special purpose registers.
16. List the indexing methods in ARM. Explain each one of them.
17. What is size of each register in ARM?
18. Difference between LDR and STR.
19. Explain how masking is done in program 4?
20. What is a look up table?
21. What is a barrel shifter? Give example of a shifting operation.
22. Give examples of rotation instructions.
23. What is difference between shifting and rotating bits of a register?
24. Give an example to show how CMP instruction affects the status register.
25. Explain pipelining concept.
26. What is meant by flushing a pipeline?
27. List all arithmetic instructions
28. List all logical instructions
29. List all data processing instructions.
30. Explain how SP, LR, PC can be used in ARM programs.

LIST OF ADDITIONAL PROGRAMS

1. Write an ALP to generate Fibonacci Series
2. Write an ALP to find the 1st and 2nd largest number in an array.
3. Write an ALP to find the 1st and 2nd smallest number in an array.
4. Write an ALP to search an element in an array (linear search)
5. Write an ALP to search an element in an array (binary search)
6. Write an ALP to find if the given number is prime or not.
7. Write an ALP to find factors of a given number
8. Write an ALP to count the occurrence of a given number in an array.
9. Write an ALP to find the position of a given number in an array
10. Write an ALP to generate multiplication table of 3 or 5
11. Write an ALP to find the trace of a matrix of the order 3x3
12. Write an ALP to find the transpose of 3x3 matrix
13. Write an ALP to add 2 matrices of order 3x3
14. Write an ALP to find the LCM of 2 32 bit numbers
15. Write an ALP to find the GCD of 2 32 bit numbers
16. Write an ALP to covert a BCD number to hex
17. Write an ALP to convert a hex number to BCD
18. Write an ALP to find factorial of a number.
19. Write an ALP to count leading 0's
20. Write an ALP to Insert 0 in a given 32 bit data from bit 5 to bit 10
21. Write an ALP to Insert 1 in a given 32 bit number from bit 22 to bit 27
22. Write an ALP to convert a binary number into a gray number
23. Write an ALP to convert a gray number into a binary number
24. Write an ALP to exchange block of ten 32 bit numbers
25. Write an ALP to generate quotient and reminder
26. Write an ALP to find average of N words
27. Write an ALP to read and store 5 consecutive memory locations without using loop.
28. Write an ALP to generate odd/even series between x1 and x2 range
29. Write an ALP to perform subtraction of two 64 bit numbers
30. Write an ALP to perform multiplication of two 64 bit numbers
31. WAP to check if the given year is leap or not.
32. WAP to evaluate ΣXiYi where i ranges from 0 to 10.