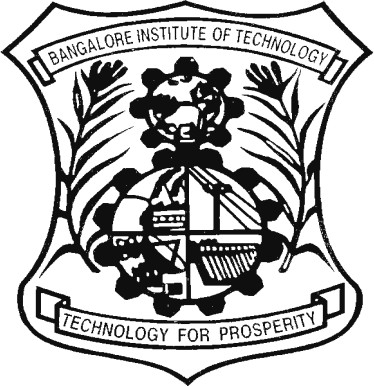
**BANGALORE INSTITUTE OF TECHNOLOGY**

Affiliated to Visvesvaraya Technological University- Belagavi

# K.R.ROAD, V. V. PURA, BENGALURU-560 004



Department of CSE (Data Science)

## Affiliated to

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi, Karnataka –590018

IV-Semester

MICROCONTROLLERS LABORATORY (BCS402)

**Prepared By Mamatha V**

**Assistant Professor**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Bangalore Institute of Technology**  **K R Road, V. V. Pura, Bengaluru- 560004**  **Department of CSE (Data Science)**  **VISION:**  To be a center of excellence in computer engineering education, empowering graduates as highly skilled professionals.  **MISSION:**  **M1** To provide a platform for effective learning with emphasis on technical excellence.  **M2** To train the students to meet current industrial standards and adapt to emerging technologies.  **M3** To instill the drive for higher learning and research initiatives.  **M4** To inculcate the qualities of leadership and Entrepreneurship.  **PROGRAM EDUCATIONAL OBJECTIVES (PEO)**  Provide graduates with both fundamental and advanced concepts of  **PEO-1** computer science and engineering for acquiring knowledge.  To prepare the graduates to apply computer engineering knowledge to build  **PEO-2** professional career.  **PEO-3** To enable students to engage in lifelong learning with the ability to involve in higher studies and research.  **PEO-4** To encourage graduates to imbibe professional ethics and soft skills.  **PROGRAM SPECIFIC OUTCOMES (PSOs)** | | | |
|  | **PSO-1** | The graduates of the program will have the ability to build software products by applying theoretical concepts and programing skills. |  |
| **PSO-2** | The graduates of the program will have the ability to pursue higher studies and Research in the modern computing era. |

**PROGRAM OUTCOMES (POs)**

**Engineering Graduates will be able to:**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of t h e engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**CONTENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MICROCONTROLLERS** | | | | |
| Course Code | | BCS402 | CIE Marks | 50 |
| Teaching Hours/Week (L:T:P:S) | | 3:0:2:0 | SEE Marks | 50 |
| Course Type | | Integrated | Total Marks | 100 |
| Credits | | 04 | Exam Hours | 03 |
| **Course Objectives:**  CLO 1: Understand the fundamentals of ARM-based systems & 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. | | | | |
| **Laboratory Component:** | | | | |
| 1. | Using Keil software, observe the various registers, dump, CPSR, with a simple ALP programme. | | | |
| 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. | | | |
| **Continuous Internal Evaluation (CIE):**   1. **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. 2. On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voice and marks shall be awarded on the same day. 3. 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**. 4. The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.** 5. Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**. 6. The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IP**C**C. | | | | |

**Suggested Learning Resources:**

**Text Books:**

1. Andrew N Sloss, Dominic Symes and Chris Wright, ARM system developers guide, Elsevier, Morgan Kaufman publishers, 2008.

**Reference Books:**

1. Raghunandan. G.H, Microcontroller (ARM) and Embedded System, Cengage learning Publica- tion, 2019.
2. Insider’s Guide to the ARM7 based microcontrollers, Hitex Ltd.,1st edition, 2005

# Course outcomes (Course Skill Set):

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

* Explain the ARM Architectural features and Instructions.
* Develop programs using ARM instruction set for an ARM Microcontroller.
* Explain C-Compiler Optimizations and portability issues in ARM Microcontroller.
* Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.
* Demonstrate the role of Cache management and Firmware in Microcontrollers.

**Note:** Questions mentioned in the SEE paper may include questions from the practical component.

**BANGALORE INSTITUTE OF TECHNOLOGY**

**K R ROAD, V V PURAM, BANGALORE-04**

**DEPARTMENT OF CSE (Data Science)**

**COURSE OBJECTIVES AND OUTCOMES**

|  |  |
| --- | --- |
| **Course Title : Microcontrollers** | **Course Code :BCS402** |
|  |  |
| **Internal Marks:25** |  |
|  |  |

**Prerequisites:**

1. Students should have knowledge of DDCO.
2. Students should have some knowledge of PCD.

**Course Learning Objectives**

This course will help students to achieve the following 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.

Course Outcomes

At the end of the course students should be able to:

1. Explain the ARM Architectural features and Instructions.
2. Develop programs using ARM instruction set for an ARM Microcontroller.
3. Explain C-Compiler Optimizations and portability issues in ARM Microcontroller.
4. Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.
5. Demonstrate the role of Cache management and Firmware in Microcontrollers.

**CO**-**PO MAPPING:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BCS402** | |  | **PO1** | **PO2** | **PO3** | **PO4** | | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | | **PO10** | **PO11** | **PO12** |
| **CO1** | **3** |  |  |  | |  |  |  |  |  | |  |  |  |
| **CO2** | **3** | **2** | **2** |  | | **2** |  |  |  |  | |  |  |  |
| **CO3** | **3** | **2** | **2** |  | |  |  |  |  |  | |  |  |  |
| **CO4** | **3** | **2** | **2** |  | |  |  |  |  |  | |  |  |  |
| **CO5** | **3** | **2** |  |  | |  |  |  |  |  | |  |  |  |
| **BCS402** |  | | | | | | **PSO1** | | | | | | **PSO2** | | | |
| **CO1** | | | | | | **2** | | | | | |  | | | |
| **CO2** | | | | | | **2** | | | | | |  | | | |
| **CO3** | | | | | | **2** | | | | | |  | | | |
| **CO4** | | | | | | **2** | | | | | |  | | | |
| **CO5** | | | | | | **2** | | | | | |  | | | |

**CO**-**PO MAPPING JUSTIFICATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BCS402** | **CO1** | **PO1** | **3** | Students apply the knowledge of engineering fundamentals to understand the ARM architecture. |
| **CO2** | **PO1** | **3** | Apply the knowledge of ARM instruction set to solve given engineering problems. |
| **PO2** | **2** | With the knowledge of ARM instruction set the students will be able to analyse the problem and formulate the solution in Assembly language. |
| **PO3** | **3** | Develop Programs to solve a complex operations using instruction set of ARM microcontroller. |
| **PO5** | **3** | Students explore and use Keil Micro Vision v4 integrated development environment tool for executing ARM assembly programs. |
| **CO3** | **PO1** | **3** | Students use fundamental engineering knowledge to develop programs for C compiler in ARM environment. |
| **PO2** | **2** | With the knowledge of ARM instruction set and basic C Data types analyse the efficient management of system resources. |
| **PO3** | **2** | Students will be able to develop efficient programs in high level language to run over C Compilers. |
| **CO4** | **PO1** | **3** | Students with the basic knowledge on interrupts and exception handling will enhance their knowledge and understand its implementation for ARM models. |
| **PO2** | **2** | With the basic knowledge of interrupts of ARM processor architecture the students will be able analyse its importance of in real time scenarios. |
| **PO3** | **2** | Students will be able to develop a program in simulation model to demonstrate the handling of Interrupt and Exceptions for ARM microcontroller. |
| **CO5** | **PO1** | **3** | Students will be able to understand the Concept of different Memory management Techniques. |
| **PO2** | **2** | Students will be able to gain the Knowledge on the importance of firmware in the design of computing models. |

**CO**-**PSO MAPPING JUSTIFICATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BCS402** | **CO1** | **PSO1** | **2** | Graduates can develop programs in Keil IDE environment by applying algorithmic designs and programming skills. |
| **PSO2** | **2** | Graduates can expand their knowledge and research activities in refining the controller architecture models for the modern computing environment. |
| **CO2** | **PSO1** | **2** | Programming concepts of ARM is applied to develop an embedded product. |
| **CO3** | **PSO1** | **2** | Graduates can analyze efficient code optimization techniques to achieve Portability across the embedded modules in real time environment. |
| **CO4** | **PSO1** | **2** | Graduates can design a cost-effective system with the appropriate knowledge of memory components. |
| **CO5** | **PSO1** | **2** | Graduates can design an efficient computational system with the acquired knowledge on embedded model. |

**Faculty In Charges**

**Course Coordinator Module Coordinator IQAC Programme Co-ordinator**

# About KEIL

* + The µVision IDE is the easiest way for most developers to create embedded applications using the Keil development tools.
  + The µVision help the programmer to develop embedded applications quickly and Keil tools are easy use & is a free software.
  + Keil provides a broad range of development tools like C/C++ compiler, assembler, linker, and a HEX file generator.
  + Keil uVision which can be used for

Writing programs in C/C++ or Assembly language Compiling Programs

Debugging program Creating Hex file

Testing your program without Available real Hardware (Simulator Mode)

# Assembly Language:

# An assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware.

# ASSEMBLER Directives: It indicates how the operand of a program is to be processed by the assembler.

# AREA Directive:

* + - AREA directive allows the programmer to specify the memory location to store code and data.
    - Code and data can reside in different areas of memory.
    - A name must be specified for an area directive.

# ENTRY and END Directives:

* Entry point must be specified for every assembly language program.
* The ENTRY directive declares an entry point to a program.
* END directive causes the assembler to stop processing the current source file.

CODE: Contains instructions and READONLY by default.

DATE: Contains data and READWRITE by default.

# Data Reservation Directives (DCB, DCD, DCW):

* ARM assembler supports different data definition directives to insert constants in assembly code.
* This directive allows the programmer to enter fixed data into the program memory and treats that data as a permanent part of the program.
* Different variants of these directives are:
* DCB (Define Constant Byte) to define constants of byte size handles 8bit numbers
* DCW (Define Constant Word) to define constants of word size handles 32-bit numbers or addresses
* DCD (Define Constant Data) which may beused in place of DCW

**Download the Keil Software from the Link:**

<https://drive.google.com/drive/folders/1pYOTX0O-yyrDFuwowjWt1mytyJ9cS-Fv>

CREATION OF PROJECT:

Step 1: Go to “Project” and close the current project “Close Project”

Step 2: Got to “Project” and click on “New Micro vision Project”

Step 3: The new window named “Create New Project” popup and here select destination to save the project.

Step 4: On new “Select Device for Target ‘Target 1’”, select NXP founded by Philips and then. Select LPC2148 and click “Yes”.

Step 5: Go to Next, File>>New

Step 6: Project workspace, editor window and Output window will appear. Write program on editor window.

Step 7: Save the program file, if the program is in “C” save as “filename.C” or else it is in ASM save as “filename.s” or “filename.src”, ”filename.a”.

Step 8: Go to “Project workspace” >> “Target 1” >> “Add Existing Files to Group ‘source group1’” on Add files to the ‘Source Group’ window >> Add.

Step 9: Go to “Project” >> “Build Target” or “F7” >> Debug >> “Start/stop Debug Session”

**Experiments**

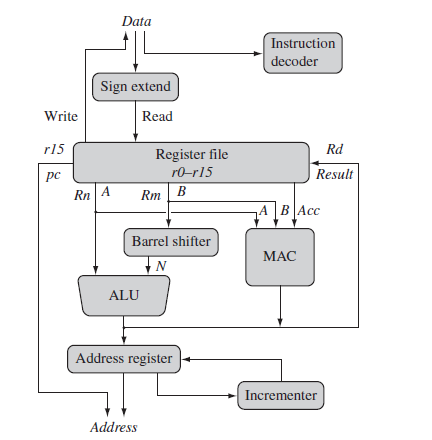
**Module – 1**

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

## ARM Architecture:

ARM processors are mainly used for low-power and low cost applications such as mobile phones, communication modems, automotive engine management systems, and hand-held digital systems.

Here is a diagram of the ARM architecture for your reference.



ARM Architecture is Enhanced RISC Architecture.

It has large uniform Register file and uses Load Store Architecture.

The operations operate on registers and not in memory locations.

ARM Architecture instructions are of uniform and fixed length.

It is a 32 bit processor. It also has 16 bit variant called THUMB.

i.e. it can be used as 32 bit and as 16 bit processor.

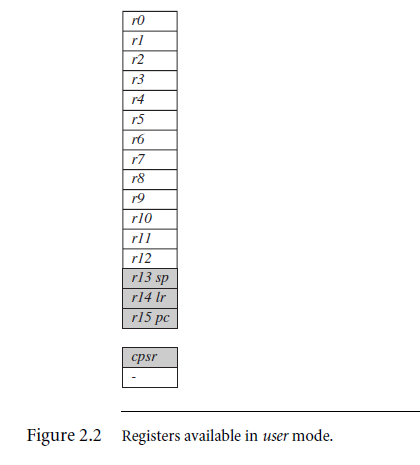
## ARM Core Registers:

* All the registers shown **are 32 bits in size.**
* There are up to **18 active registers**: **16 data registers and 2 processor status registers.**
* The data registers are visible to the **programmer as *r0 to r15****.*
* The ARM processor has **three registers assigned** to a particular task or special function:

**Register r13** is traditionally used as the stack pointer (sp) and stores the head of the stack in the current processor mode.

**Register r14** is called the link register (lr) and is where the core puts the return address whenever it calls a subroutine.

**Register r15** is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.



## Current Program Status Register (CPSR):

* The ARM core uses the CPSR to monitor and control internal operations. The CPSR is a dedicated 32-bit register and resides in the register file.
* The shaded parts are reserved for future expansion.
* The CPSR is divided into four fields, each 8 bits wide:

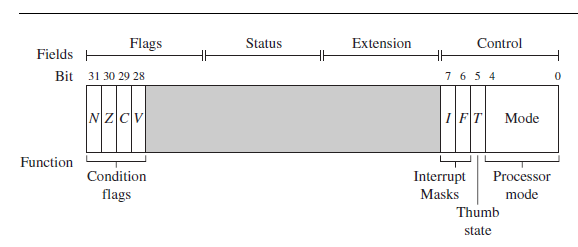
-flags,

- status,

- extension, and

- control.

* In current designs the extension and status fields are reserved for future use.



 Conditonion code flags in CPSR:

N - Negative or less than flag

Z - Zero flag

C - Carry or bowrrow or extended flag

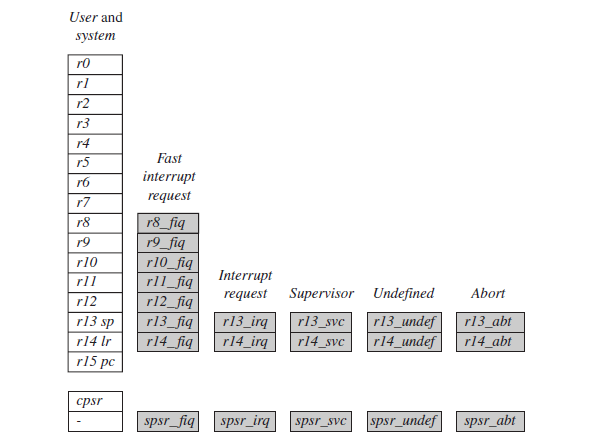
V - Overflow flag

**Processor mode:** There are seven processor modes in ARM core model.

**Six privileged modes:** abort, fast interrupt request, interrupt request, supervisor, system, and undefined

**One non privileged mode:** User Mode

## Banked Register of ARM:



**State Bit:** **There are three instruction states:**

* + ARM state: The ARM instruction set (32 bit) is only active when the processor is in ARM state (T=0 and J=0)
  + Thumb state: Similarly the Thumb instruction set (16 bits) is only active when the processor is in Thumb state (T=1 and J=0).
  + Jazelle state: Jazelle executes 8-bit instructions and is a hybrid mix of software and hardware designed to speed up the execution of Java byte-codes(T=0 and J=1).

**Interrupt mask bits:** The cpsr has two interrupt mask bits, 7 and 6 (or I and F), which control the masking of IRQ and FIQ, respectively.

* + The I bit masks IRQ when set to binary 1, and similarly the F bit masks FIQ when set to binary 1. (Interrupts are masked means they are not recognized by processor or disabled.)

**Module – 2**

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

# AREA calculator, CODE, READONLY ; name the code block

# ENTRY ; marker of first executable instruction

# START ; label

# MOV R0, #08 ; put the value 08 in register0

# MOV R1, #06 ; put the value 06 in register1

# ADD R3, R0, R1 ; R3 = R0 + R1

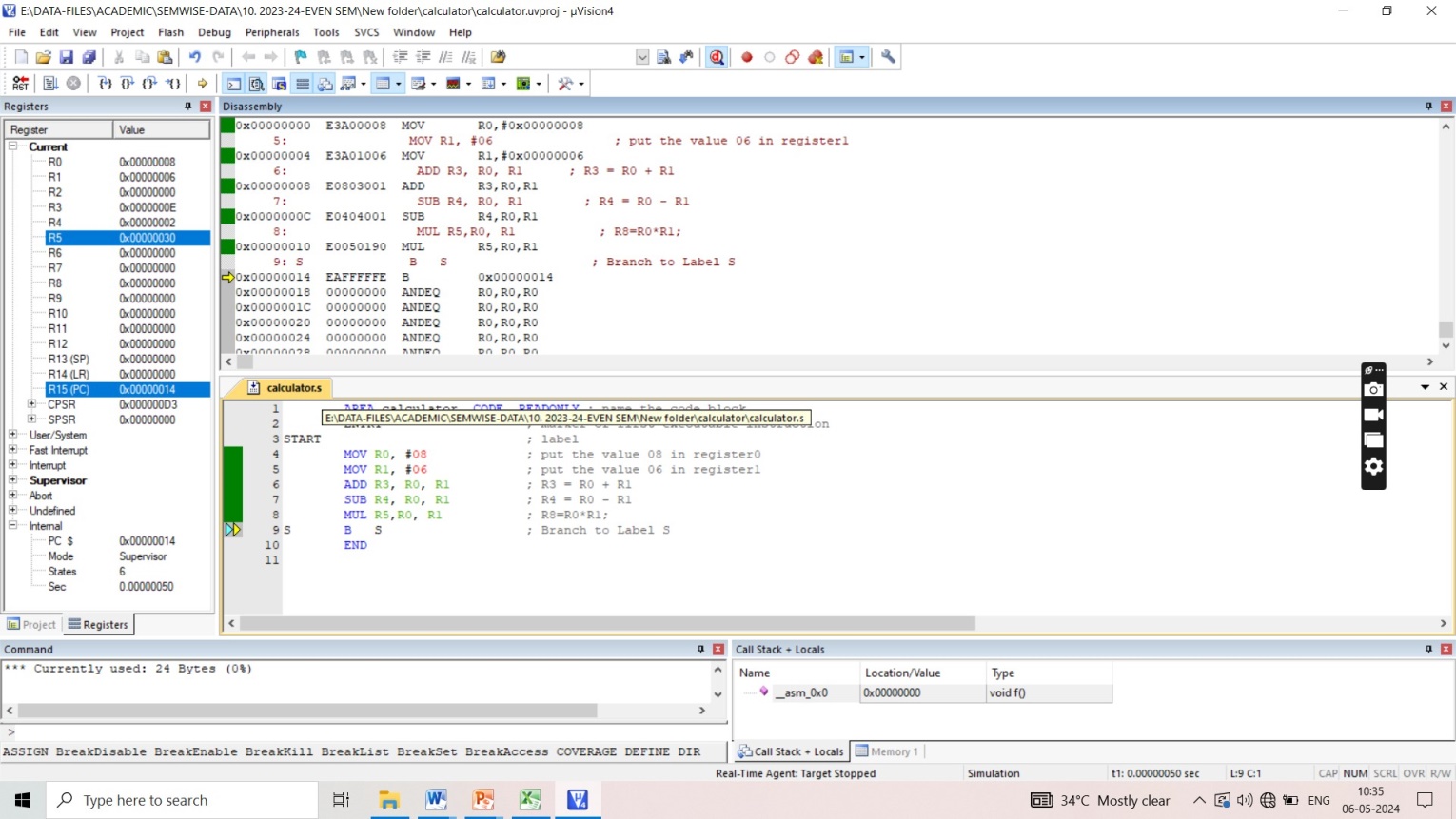
# SUB R4, R0, R1 ; R4 = R0 - R1

# MUL R5, R0, R1 ; R5=R0\*R1;

# S B S ; Branch to Label S

# END

# OUTPUT:



# Logical Instructions: Logical instructions perform bitwise logical operations on the two source registers.

# AREA logical, CODE, READONLY ; name the code block

# ENTRY ; marker of first executable instruction

# START ; label

# MOV R0, #0x08 ; put the value 08 in register0

# MOV R1, #0x0A ; put the value 10 in register1

# AND R3, R0, R1 ;Logical Bitwise AND: R3 = R0 & R1

# ORR R4, R0, R1 ; Logical Bitwise OR: R3 = R0 | R1

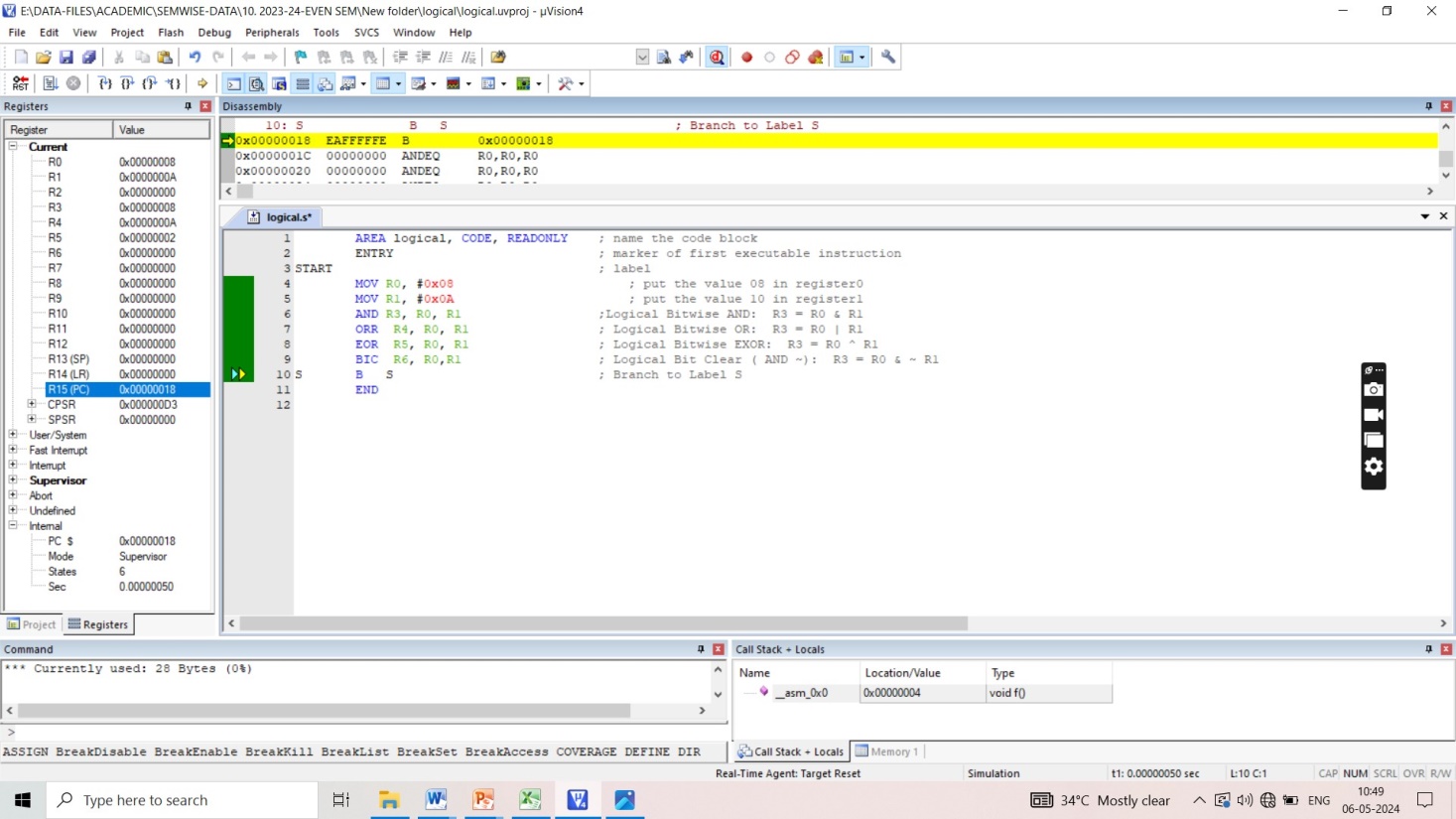
# EOR R5, R0, R1 ; Logical Bitwise EXOR: R3 = R0 ^ R1

# BIC R6, R0, R1 ; Logical Bit Clear ( AND ~): R3 = R0 & ~ R1

# S B S ; Branch to Label S

# END

# OUTPUT:



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

AREA MULTIPLY, CODE, READONLY

ENTRY ; Mark first instruction to execute

START

MOV R1, #6400 ; STORE FIRST NUMBER IN R0

MOV R2, #3200 ; STORE SECOND NUMBER IN R1

MUL r3, r1, r2 ; MULTIPLICATION

back B back

END ; Mark end of file

; **Program to multiply two 16 bit numbers defined in memory and display the result in register**

AREA MULTIPLY, CODE, READONLY ; NAME THE CODE BLOCK

ENTRY

START

LDR R0, =VALUE1

LDRH R1, [ R0]

LDR R0, =VALUE2

LDRH R2, [R0]

MUL R3, R2, R1

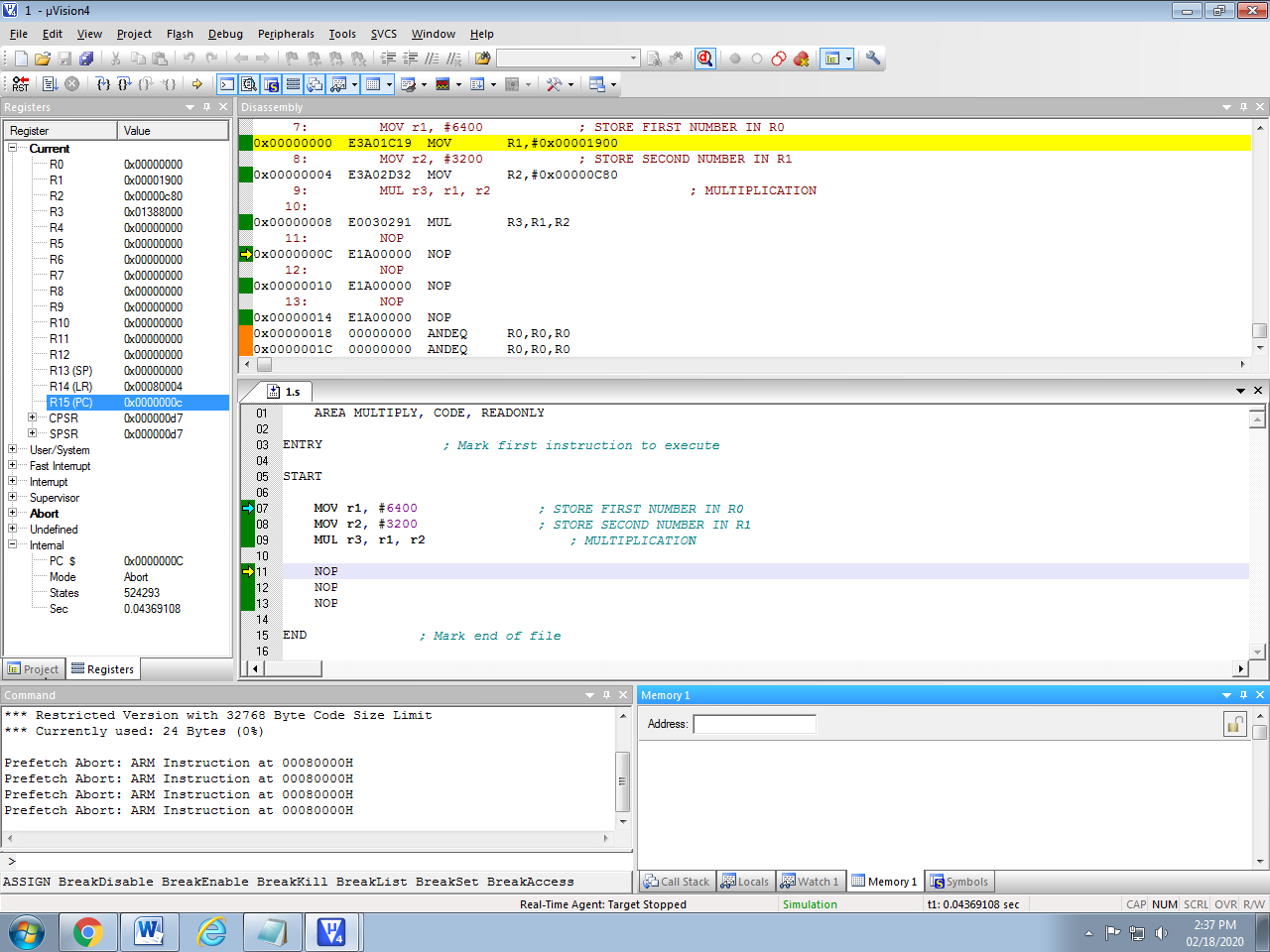
BACK B BACK

VALUE1 DCW & BBBB; OR 0XBBBB

VALUE2 DCW & CCCC; OR 0XCCCC

END

**Output:**



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

AREA SUM, CODE, READONLY

ENTRY

MOV R1, #10 ; load 10 to register

MOV R2, #0 ; empty the register to store result

Loop

ADD R2, R2, R1 ; add the content of R1 with result at R2

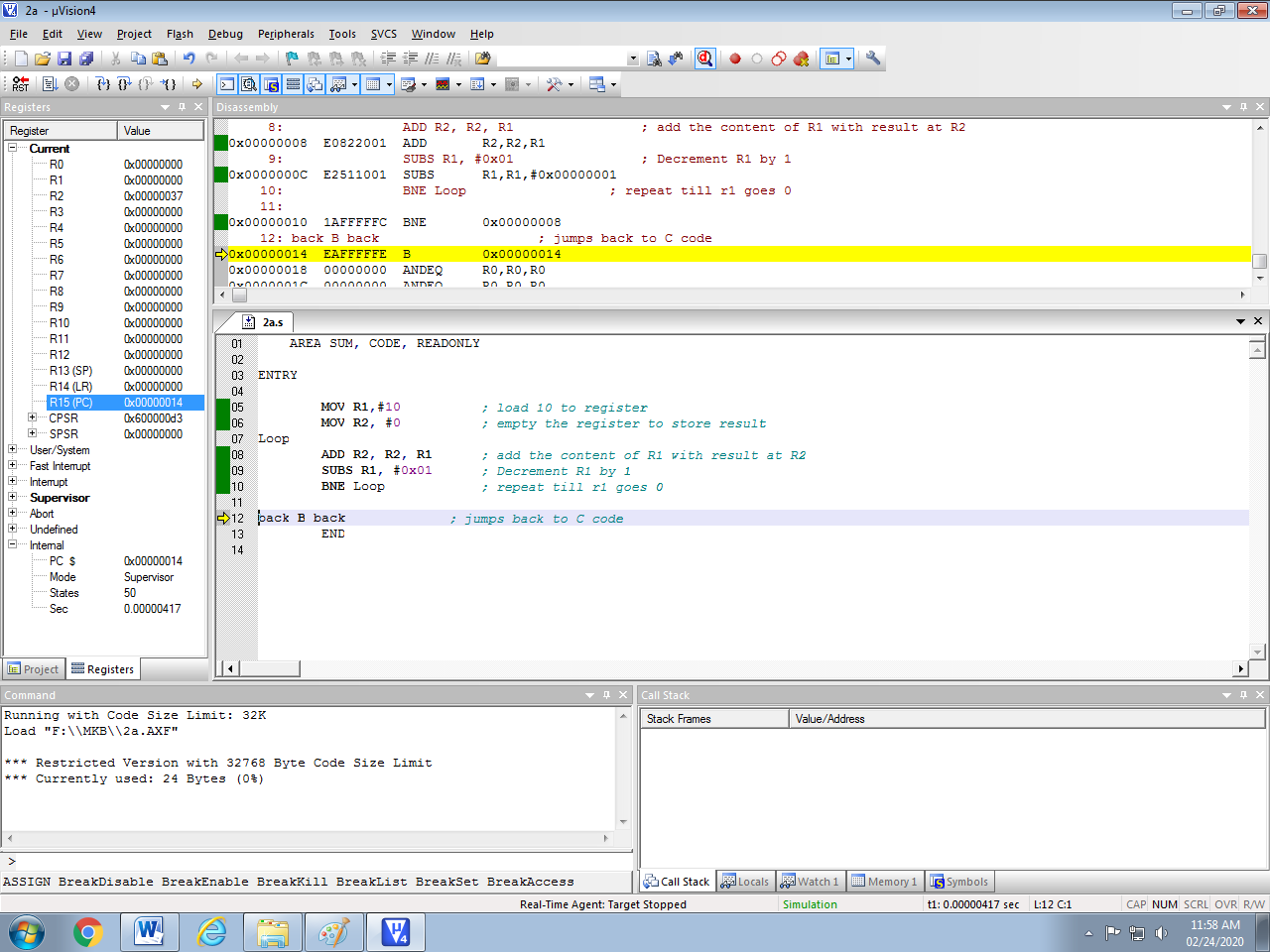
SUBS R1, #0x01 ; Decrement R1 by 1

BNE loop ; repeat till r1 goes 0

back B back ; jumps back to C code

END

**Output:**



1. **Write a program to find the largest/smallest number in an array of 32 numbers**

**LARGEST NUMBER:**

AREA LARGEST, CODE, READONLY

ENTRY ; Mark first instruction to execute

START

MOV R5,#6 ; INTIALISE COUNTER TO 6(i.e. N=7)

LDR R1,=VALUE1 ; LOADS THE ADDRESS OF FIRST VALUE

LDR R2, [R1],#4 ; WORD ALIGN T0 ARRAY ELEMENT

LOOP

LDR R4, [R1], #4 ; WORD ALIGN T0 ARRAY ELEMENT

CMP R2, R4 ; COMPARE NUMBERS

BHI LOOP1 ; IF THE FIRST NUMBER IS > THEN GOTO LOOP1

MOV R2, R4 ; IF THE FIRST NUMBER IS < THEN MOV CONTENT R4 TO R2

LOOP1

SUBS R5, R5, #1 ; DECREMENT COUNTER

CMP R5, #0 ; COMPARE COUNTER TO 0

BNE LOOP ; LOOP BACK TILL ARRAY ENDS

LDR R4, =RESULT ; LOADS THE ADDRESS OF RESULT

STR R2, [R4] ; STORES THE RESULT IN R2

BACK B BACK

NOP

NOP

; ARRAY OF 32 BIT NUMBERS (N=7)

VALUE1

DCD 0X44444444 ;

DCD 0X22222222 ;

DCD 0X11111111 ;

DCD 0X33333333 ;

DCD 0XAAAAAAAA ;

DCD 0X88888888 ;

DCD 0X99999999 ;

AREA DATA2, DATA, READWRITE; TO STORE RESULT IN GIVEN ADDRESS

RESULT DCD 0X0

END ; Mark end of file

**SMALLEST NUMBER:**

AREA SMALLEST, CODE, READONLY

ENTRY ; Mark first instruction to execute

START

MOV R5, #6 ; INTIALISE COUNTER TO 6(i.e. N=7)

LDR R1, =VALUE1 ; LOADS THE ADDRESS OF FIRST VALUE

LDR R2, [R1], #4 ; WORD ALIGN T0 ARRAY ELEMENT

LOOP

LDR R4, [R1], #4 ; WORD ALIGN T0 ARRAY ELEMENT

CMP R2, R4 ; COMPARE NUMBERS

BLS LOOP1 ; IF THE FIRST NUMBER IS < THEN GOTO LOOP1

MOV R2, R4 ; IF THE FIRST NUMBER IS > THEN MOV CONTENT R4 TO R2

LOOP1

SUBS R5, R5, #1 ; DECREMENT COUNTER

CMP R5, #0 ; COMPARE COUNTER TO 0

BNE LOOP ; LOOP BACK TILL ARRAY ENDS

LDR R4,=RESULT ; LOADS THE ADDRESS OF RESULT

STR R2, [R4] ; STORES THE RESULT IN R2

BACK B BACK

NOP

NOP

; ARRAY OF 32 BIT NUMBERS (N=7)

VALUE1

DCD 0X44444444 ;

DCD 0X22222222 ;

DCD 0X11111111 ;

DCD 0X22222222 ;

DCD 0XAAAAAAAA ;

DCD 0X88888888 ;

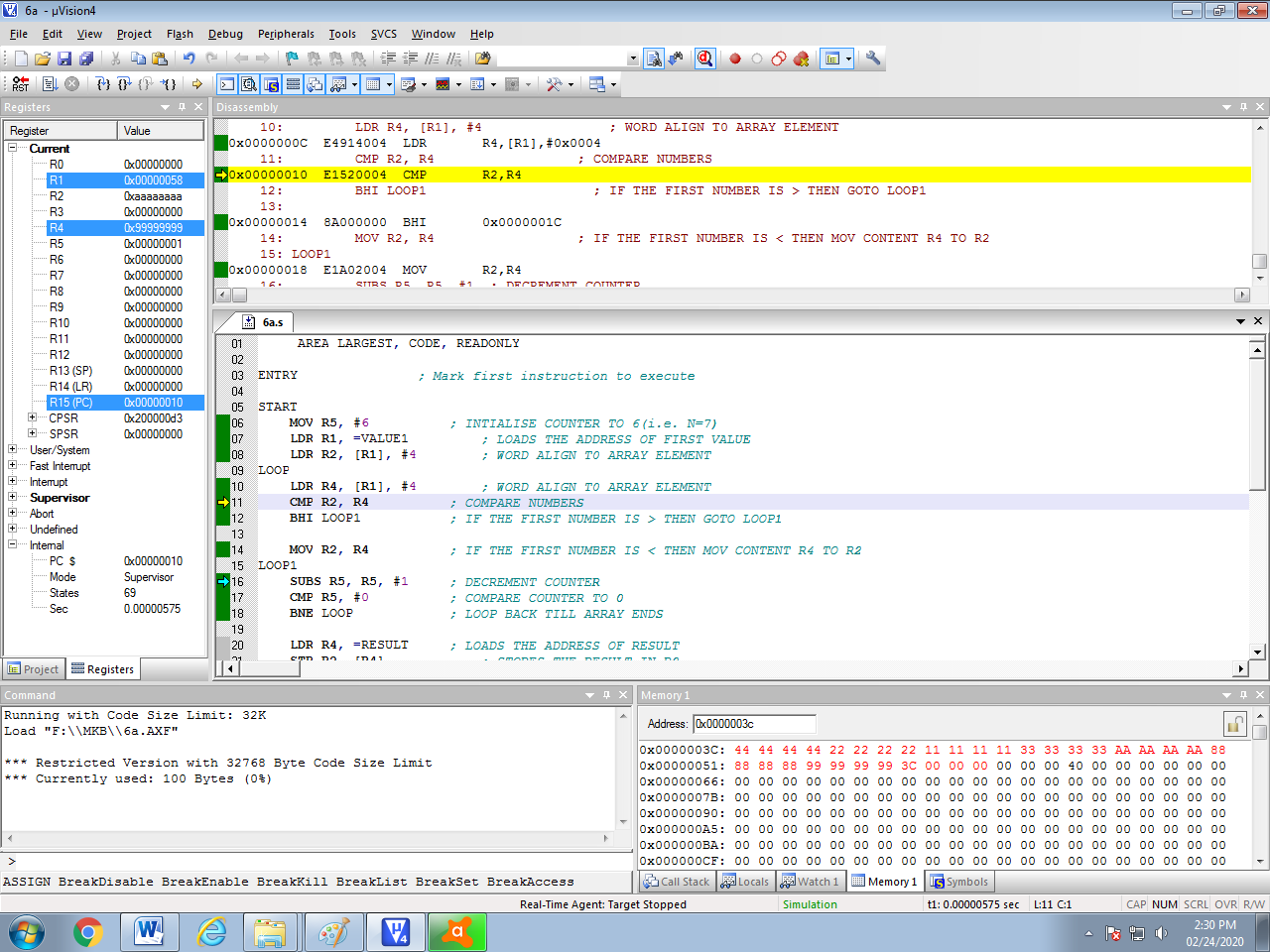
DCD 0X99999999 ;

AREA DATA2, DATA, READWRITE; TO STORE RESULT IN GIVEN ADDRESS

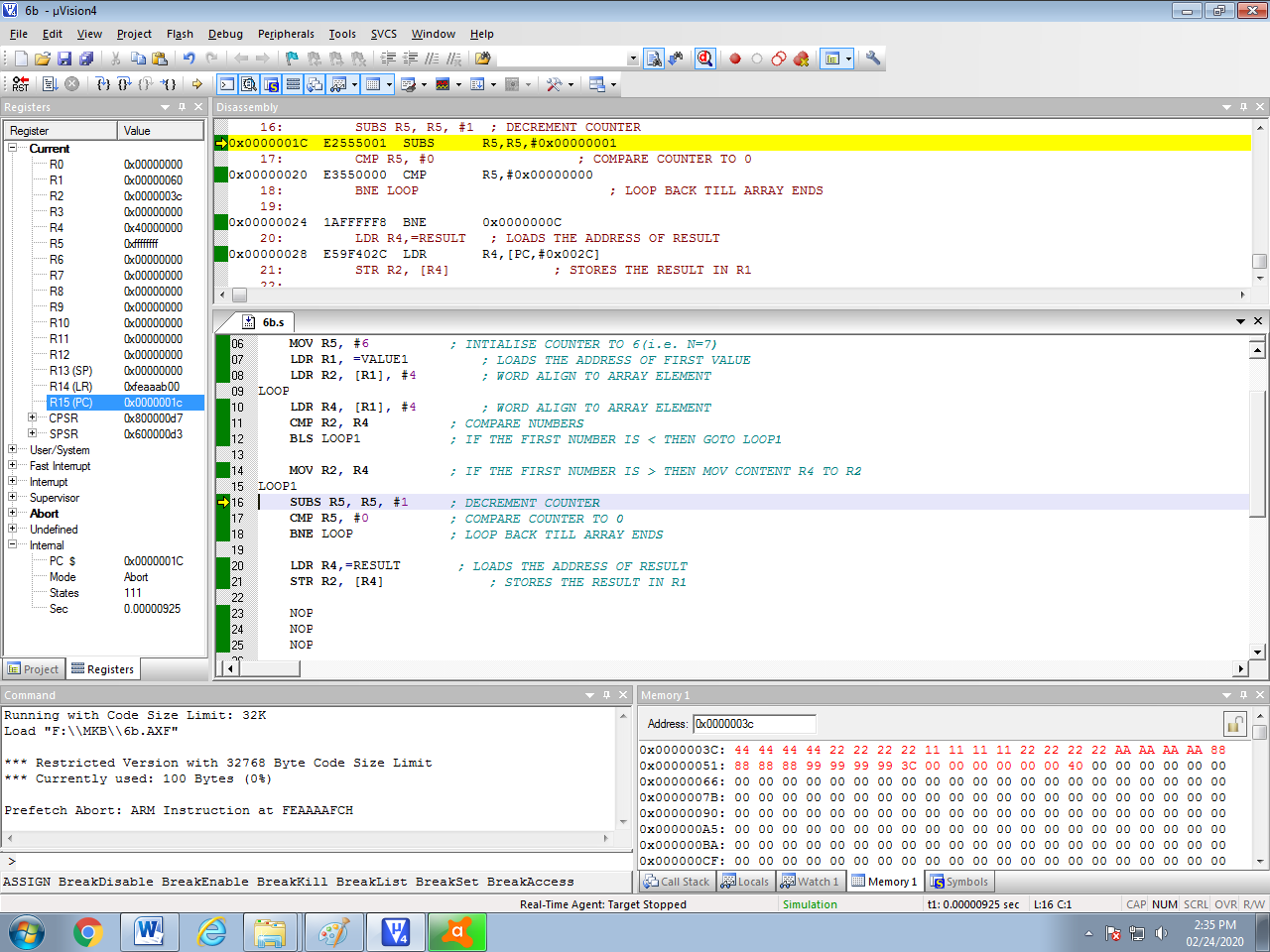
RESULT DCD 0X0

END ; Mark end of file

**Output: LARGEST NUMBER:**

****

**Output: SMALLEST NUMBER:**



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

AREA ONEZERO, CODE, READONLY

ENTRY ; Mark first instruction to execute

START

MOV R2, #0 ; COUNTER FOR ONES

MOV R3, #0 ; COUNTER FOR ZEROS

MOV R7, #2 ; COUNTER TO GET TWO WORDS

LDR R6, =VALUE ; LOADS THE ADDRESS OF VALUE

LOOP MOV R1, #32 ; 32 BITS COUNTER

LDR R0, [R6], #4 ; GET THE 32 BIT VALUE

LOOP0 MOVS R0, R0, ROR #1 ; RIGHT SHIFT TO CHECK CARRY BIT (1's/0's)

BHI ONES ; IF CARRY BIT IS 1 GOTO ONES BRANCH OTHERWISE NEXT

ZEROS ADD R3, R3, #1 ; IF CARRY BIT IS 0 THEN INCREMENT THE COUNTER BY 1(R3)

B LOOP1 ; BRANCH TO LOOP1

ONES ADD R2, R2, #1 ; IF CARRY BIT IS 1 THEN INCREMENT THE COUNTER BY 1(R2)

LOOP1 SUBS R1, R1, #1 ; COUNTER VALUE DECREMENTED BY 1

BNE LOOP0 ; IF NOT EQUAL GOTO TO LOOP0 CHECKS 32BIT

SUBS R7, R7, #1 ; COUNTER VALUE DECREMENTED BY 1

CMP R7, #0 ; COMPARE COUNTER R7 TO 0

BNE LOOP ; IF NOT EQUAL GOTO TO LOOP

BACK B BACK

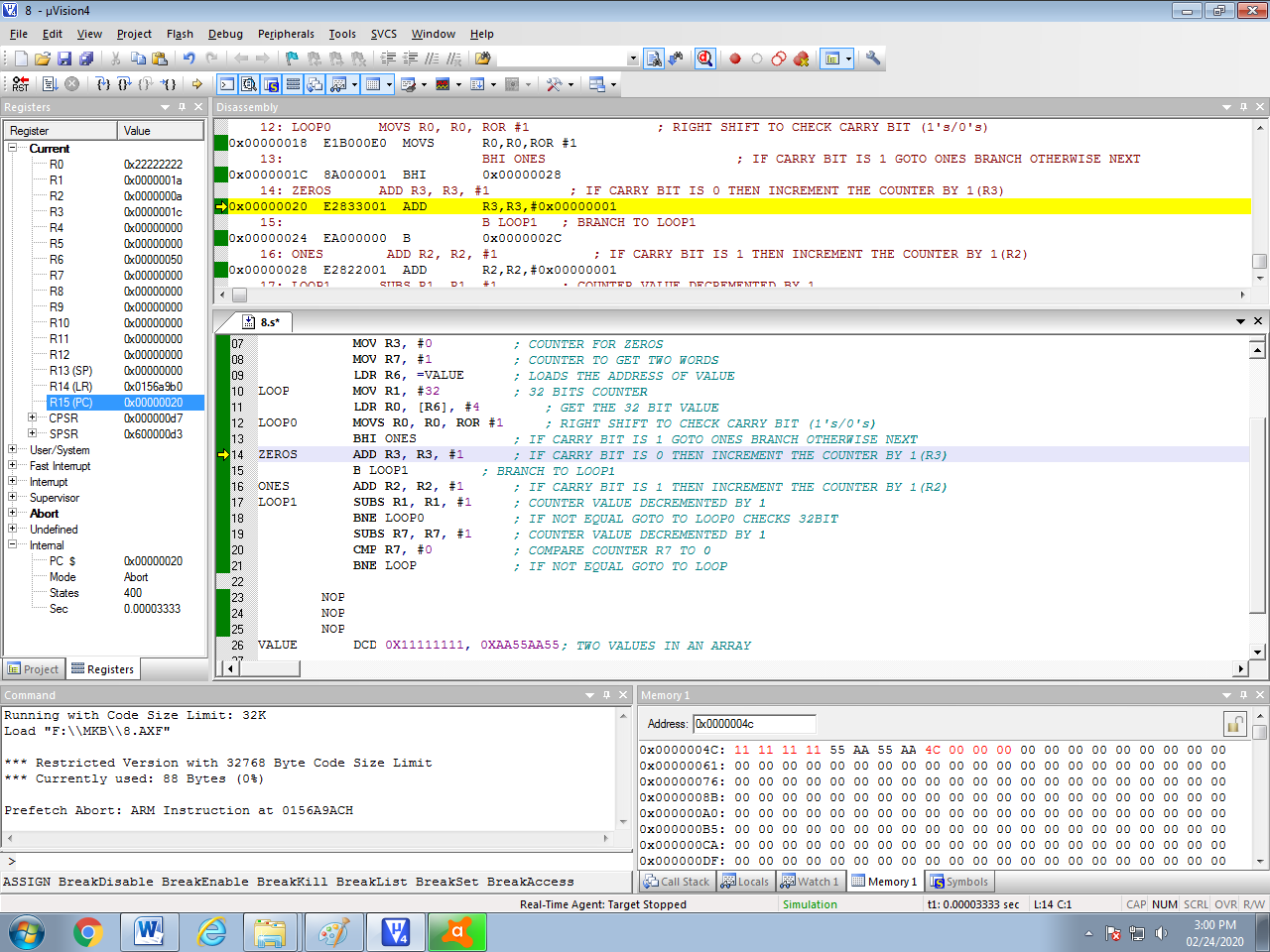
NOP

NOP

VALUE DCD 0X11111111, 0XAA55AA55; TWO VALUES IN AN ARRAY

END ; Mark end of file

**Output:**



**Module – 3**

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

#include <stdint.h>

#define NUM\_ELEMENTS 10

// Function to swap two integers

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

//Function to perform bubble sort

void bubbleSort(int arr[], int n)

{

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], &arr[j + 1]);

}

}

}

}

int main(void)

{

int numbers[NUM\_ELEMENTS] = {5, 2, 9, 1, 7, 3, 8, 6, 4, 0};

int i, j;

//Sort the array

bubbleSort(numbers, NUM\_ELEMENTS);

//code to handle the sorted array goes here

//For example, printing the sorted array

for (i=0; i < NUM\_ELEMENTS; i++) {

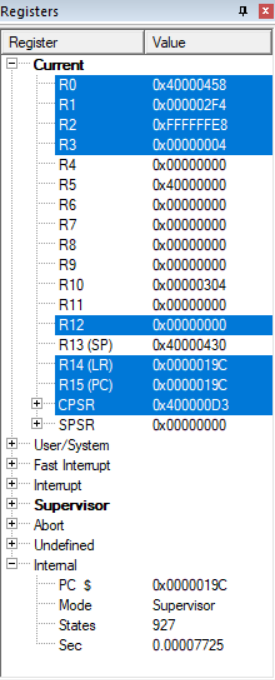
// code to output the sorted numbers

}

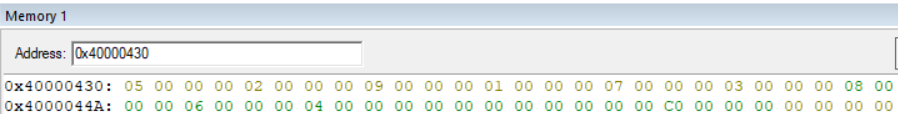
}

**Output:**

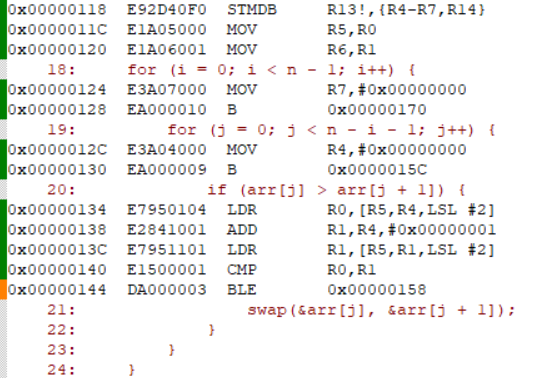
**Register Window:**



**Input Array: Before Sorting**



**Disassembly Window:**



**In Bubble Sort:**

* **Total no. of passes:**n-1
* **Total no. of comparisons:**n\*(n-1)/2

R5= 0x40000430; Starting address memory the input array elements are stored .

R7=00 (Refers to index variable  **i** to keep track of number of passes from 0 to n-1)

R4=00(Refers to variable **j** to keep track of number of comparisons in each pass from 0 to n-i-1)

R6=0x0A, R0=09

**LDR R0, [R5,R4,LSL #2]:** Scaled immediate offset addressing mode. This instruction is used to read the next word from the 32 bit data array. Initially R4=0

Ex: Load First array Element R0 , [0x40000430+0 < 2]

R0, [0x40000430+0]

R0, [0x40000430]

Load First Element R0=0X00000005

After reading the first element the R4 value is incremented by 1 by using ADD instruction.

**LDR R1,[R5,R4,LSL #2]:** Scaled immediate offset addressing mode. This instruction is used to read the next word from the 32 bit data array.

Load Second array Element by computing address R1, [0x40000430+1 < 2]

R1, [0x40000430+4]

R1, [0x40000434]

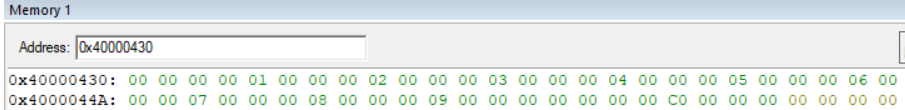
Load R1=0X00000002

**CMP R0, R1**; It performs R0-R1 if R0<R1,CF=0 hence no Swapping…

Here First Number 0x05 is compared with Second Number 0x02, 05-02, **CF=1** thus it swaps the array elements in the memory using swap function , hence it continues with next comparisons, and so on.

The above Process is repeated until all the elements are sorted in an ascending order.

**Final Sorted Elements are:**



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

#include <LPC214x.H> // Device header for LPC214x series

unsigned int factorial(unsigned int n) {

unsigned int result = 1;

unsigned int i;

for ( i = 1; i <= n; ++i) {

result \*= i;

}

return result;

}

int main(void) {

unsigned int num = 5; // Number to calculate the factorial of

unsigned int result;

// Calculate the factorial

result = factorial(num);

// Endless loop to keep the program running

while (1) {

// Use a debugger to inspect the value of 'result'

}

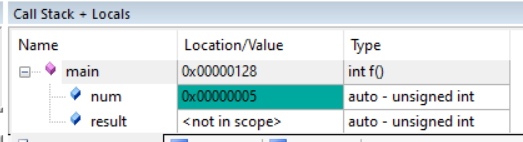
// Return 0 (not necessary as this point will never be reached)

return 0;

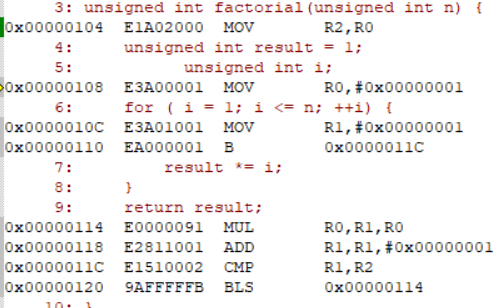
}

**Output:**

**Input window with num=5**

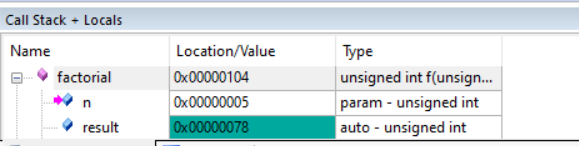


**Disassembly Window:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R4=5;  Num variable in the Main Program is mapped to register R4. | R2=05;  variable n in the factorial function  is mapped to Register R2 (called function) | R0=1; **result** variable | R1=1;index value **i** incremented after every iteration | MUL R0,R1,R0 |
| **1** | **1** | **1** |
| **1** | **1+1=2** | **2\*1=2** |
| **2** | **2+1=3** | **3\*2=6** |
| **6** | **3+1=4** | **4\*6=24**  **(18 in hexa)** |
| **24** | **4+1=5** | **5\*24=120**  **(78 in hexa)** |
| **78** | **5+1=6-STOP** |  |

**Output window with factorial of a number stored at memory location result:**



**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>

// Convert uppercase to lowercase (and vice versa)

char convertCase(char c) {

if (c >= 'A' && c <= 'Z') {

// Convert uppercase to lowercase

return c =c+32;

} else if (c >= 'a' && c <= 'z') {

// Convert lowercase to uppercase

return c =c-32;

}

// Return unchanged for non-alphabetic characters

return c;

}

int main() {

int i;

char input[] = "AaBb"; // Your input string

int len = sizeof(input) - 1; // Exclude the null terminator

for (i = 0; i < len; ++i) {

input[i] = convertCase(input[i]);

}

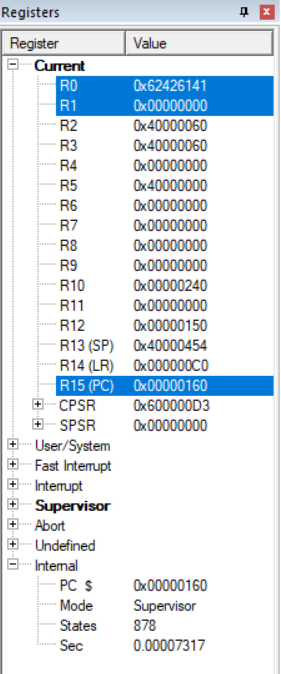
//printf("Converted string: %s\n", input);

return 0;

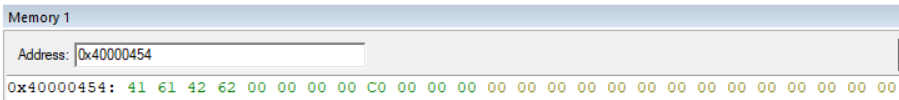
}

**OUTPUT:**

**Register Window:**

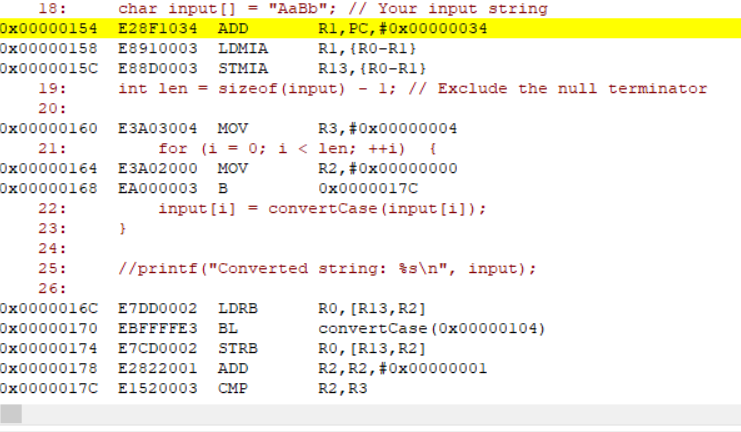


**Input Window:**

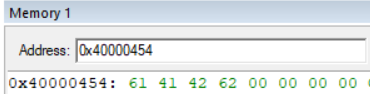


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Character | A | a | B | b |
| ASCII Equivalent in Decimal | 65 | 97 | 66 | 98 |
| ASCII Equivalent in Hexadecimal | 41 | 61 | 42 | 62 |
| **After the conversion the output is** | **61** | **41** | **62** | **42** |

**Disassembly Window:**



**Output Window:**



**Description:**

ASCII value of Character A=65 and Z=90 and **thus in Hexadecimal it ranges from 41 to 5A.**

Similarly ASCII value of Character a=97 and z=122 and **thus in Hexadecimal it ranges from 61 to 7A.**

Thus to Convert **Uppercase of A to lowercase a** in decimal we should add 65+32=97. Hence for all other Upper case character conversion to lowercase.

**// Convert uppercase to lowercase**

**str[i] = str[i] + 32;**

Thus to Convert **lowercase character to uppercase** 97-32=65.

**// Convert uppercase to lowercase**

**str[i] = str[i] - 32;**

Since array is declared as char, C Compiler generates Assembly code as above:

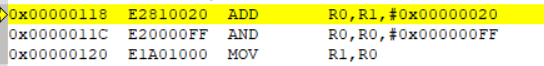
Using LDRB R0, [R13,R2]: We are loading a 8 bit data into register R0.Then call convertCase function to convert from uppercase to lowercase and Viceversa.

CMP R1, 0X00000041; Checks if input is a uppercase character or lowercase.

**Conversion from Uppercase to Lowercase:**

//**str[i] = str[i] + 32;**

Equivalent assembly code is:



**ADD R0,R1,0X20; (Hexadecimal value of 32)**

**AND R0,R0, 0x000000FF;(Masking higher order 24 bits)**

**Conversion from Lowercase to Uppercase:**

**// str[i] = str[i] - 32;**

Equivalent assembly code is:



**SUB R0, R1, 0X20; (Hexadecimal value of 32)**

**AND R0, R0, 0X000000FF ;( Masking higher order 24 bits)**

**Module – 4 and 5**

**9. Demonstrate enabling and disabling of Interrupts in ARM.**

AREA INTERR,CODE, READONLY

ENTRY

MRS R1, CPSR

BIC R1, R1, #0X80 ; BIC R1, R1, #0X40 FOR fiq (Enable Fast Interrupt)

MSR CPSR\_C, R1

MRS R1, CPSR

ORR R1, R1, #0X80 ; BIC R1, R1, #0X40 FOR fiq (Disable Fast Interrupt)

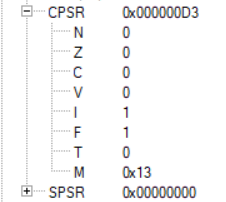
MSR CPSR\_C, R1

NOP

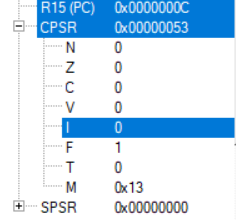
END

**OUTPUT:**

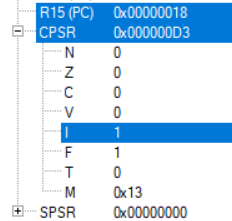
**Initial:**

****

**Enable Interrupt:**

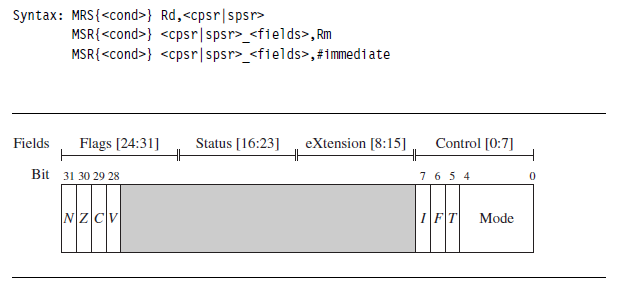
****

**Disable Interrput:**

****

**Explanation:**

MRS Instruction: The MRS instruction transfers the contents of either the CPSR or SPSR into a general Purpose register. **Rd=CPSR**



**Enabling Interrupts:**

**Initially: CPSR=0X000000D3;**

**0XD3= 1101 0011(nzcvIFt\_SVC); SVC=10011**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | b7(I) | b6(F) | b5(t) | Processor Mode(b4-b0) | | | | |
| PRE R1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| BIC R1, R1,#0X80 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| POST R1 | **0** | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

**POST=0X00000053(nzcviFt\_SVC)**

**Steps For Enabling Normal Interrpt(i)**

* The MSR first copies the CPSR = 0XD3 into register R1.
* The BIC instruction clears bit 7 of R1.
* Register r1 is then copied back into the CPSR, which enables IRQ interrupts.
* This code preserves all the other settings in the CPSR and only modifies the I bit in the control field.
* This example is in SVC mode. In user mode you can read all CPSR bits, but you can only update the condition flag field.

The MSR instruction: The MRS transfers the contents of a register into the CPSR or SPSR. CPSR=Rs

**Disabling Interrupt:**

**Initially: CPSR=0X00000053;**

**0X53= 0101 0011(nzcviFt\_SVC)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | b7(I) | b6(F) | b5(t) | Processor Mode(b4-b0) | | | | |
| PRE R1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| ORR R1, R1, 0X80 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| POST R1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

**POST=0X000000D3(nzcvIFt\_SVC)**

**Steps For Disabling Normal Interrpt(I)**

* The MSR first copies the CPSR = 0X53 into register R1.
* The ORR instruction sets bit 7 of R1.
* Register R1 is then copied back into the CPSR, which disables IRQ interrupts.
* This code preserves all the other settings in the CPSR and only modifies the I bit in the control field.
* This example is in SVC mode. In user mode you can read all CPSR bits, but you can only update the condition flag field.

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

AREA PGM, CODE, READONLY

ENTRY

MOV R1, #00

MOV R2, #02

CMP R1, #00

DIV R2, R1 ; Check for invalid operation

LDR R3, =0X70000000 ; Check for overflow

LDR R4, =0X7FFFFFFF

ADDS R3, R4

NOP

END

**OUTPUT:**

