DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige malleswara hills , kumaraswamy layout bengaluru-560078

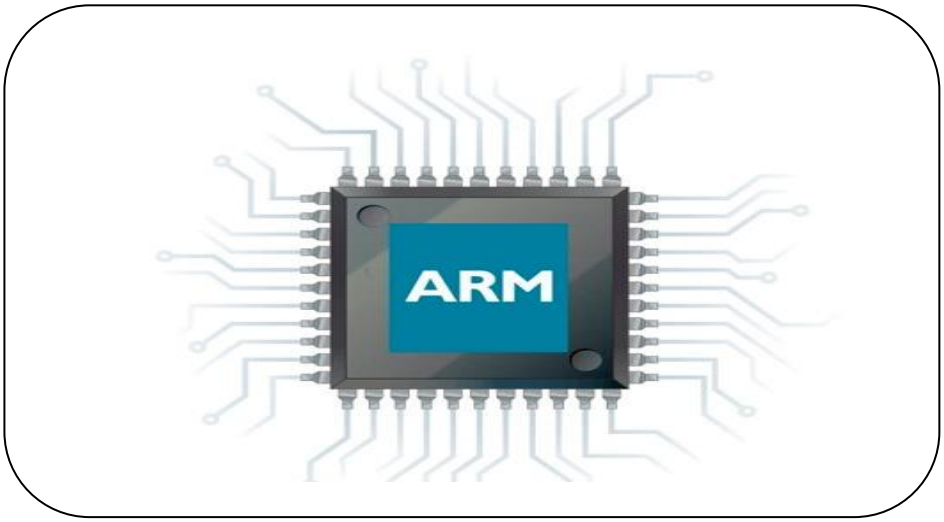
Accredited by National Assessment & Accreditation Council (NAAC) with 'A' Grade & ISO 9001:2015 Certified

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)

ARM PROCEESOR LABORATORY

IV Semester

CODE: 22EI44



DEPARTMENT OF ELECTRONICS & INSTRUMENTATION ENGINEERING

Accredited by National Board of Accreditation (NBA)

Prepared by

Dr. Meharunnisa S.P
Associate Professor
Dept. of EIE, DSCE

VISION OF THE INSTITUTE

To Impart Quality Technical Education with a focus on Research and Innovation emphasizing on Development of Sustainable and Inclusive Technology for the benefit of Society.

MISSION OF THE INSTITUTE

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence
- To nurture team work in order to transform individuals as responsible leaders and entrepreneurs
- To train the students to the changing technical scenario and make them to understand the importance of Sustainable and Inclusive technologies.

DAYANANDA SAGAR COLLEGE OF ENGINEERING

QUALITY POLICY

We, at

Dayananda Sagar College of Engineering

Are committed to continually improve and deliver competitive quality technical education to the utmost satisfaction of students, their parents and potential Employers.

Dr. D. Hemachandra Sagar
Chairman

VISION OF THE DEPARTMENT

To meet the challenges of industry and research in the field of Electronics, Instrumentation and Control Engineering by providing quality technical education.

MISSION OF THE DEPARTMENT

- To impart quality education with an understanding of basic concepts.
- To enhance knowledge in the core domains of Electronics and Instrumentation by providing a conducive learning environment.
- To nurture the students with inter - disciplinary technology by contributing solutions for techno-social issues.

PROGRAM EDUCATIONAL OBJECTIVES(PEOs)

PEO1: Have successful professional career in the core and allied domains.

PEO2: Engage in continuous learning and adapt to modern technology.

PEO3: Work in diverse environments exhibiting team work and leadership quality.

PEO4: Analyze and provide the solutions to real-life engineering problems for the betterment of society.

PROGRAM SPECIFIC OUTCOMES(PSOs)

PSO1: Apply the knowledge of control and automation to develop the industrial control system for process Industries.

PSO2: Acquire the concepts of embedded systems and apply them to solve the engineering problems.

PSO3: Ability to design instrumentation systems to solve real time applications.

LABORATORY RULES

DO's

- ✓ Uniform should be worn during laboratory. Dress properly during alaboratory activity
- ✓ Perform only those experiments authorized by your teacher. Carefully follow all instructions, both written and oral**.**
- ✓ Before use of equipment, Labels and instructions must be read carefully. Set up and use the equipment as directed by your teacher.
- ✓ Conduct yourself in a responsible manner at all times in the Laboratory. Don't talk aloud or crack jokes in lab.
- ✓ Any failure / break-down of equipment must be reported to the teacher**.**
- ✓ Observe good housekeeping practices. Replace the materials in proper place after work to keep the lab area tidy.
- ✓ Coats, bags, and other personal items must be stored in designated areas.
- ✓ Know the location of and know how to operate the following Fireextinguishers & First-aid kits

DON'Ts

- ✘ Students are not allowed to touch any equipment in the laboratory area until you are instructed by Teacher or Technician.
- ✘ Students are not allowed to work in laboratory alone or without presence of the Teacher or Technician
- ✘ Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.
- ✘ Do not eat food, drink beverages or chew gum in the laboratory.
- ✘ The use of personal audio or video equipment is prohibited in the laboratory.
- ✘ Do not sit on laboratory tables.

HOD – E&IE

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF ELECTRONICS & INSTRUMENTATION ENGINEERING(NBA ACCREDITED)

IV SEMESTER B. E (EIE)

Course code: 22EI44 **Credits: 04**
L: T: P: S: 3:0:2:0 **CIE Marks: 50**
Exam Hours: 02 **SEE Marks: 50**

Course Objectives

1	Understand Microprocessor & Controller design philosophy and ARM processor architecture and fundamentals
2	Analyze the Thumb instructions, Various Interrupts and exception handling of ARM controller
3	Evaluate various interfacing concepts to develop Embedded C program for LED, Keyboard, LCD, DC motor, Stepper motor
4	Design solutions for engineering problems using ARM controller

Course Outcomes: After completing the course, the students will be able to	
CO1	Comprehend the architecture, design philosophy of Microprocessors & Controllers
CO2	Analyse features of ARM & THUMB instruction set & exercise instructions with simple program.
CO3	Analyze features of Exception & Interrupts handling schemes of ARM & develop simple assembly language program to solve engineering problems
CO4	Design a system to interface with different Peripheral devices & develop embedded C Program for different applications & case study on various applications.

MAPPING OF COs WITH POs AND PSOs

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	-	-	-	-	-	-	-	3	-	-	-	-	-
CO2	-	3	3	-	-	-	-	-	3	3	-	-	-	-	-
CO3	-	-	3	3	-	-	-	-	3	3	-	-	-	-	-
CO4	-	-	-	3	-	-	-	-	3	3	-	3	-	3	-

LIST OF EXPERIMENTS

Sl. No	Experiment Name	Hours	Cos
1.	Develop an assembly language program to multiply two 16-bit binary numbers.	02	**CO1, 2,3,4**
2.	Develop an assembly language program to find the sum of first 10 integer numbers.	02	**CO1, 2,3,4**
3.	Develop an assembly language program find factorial of a number.	02	**CO1, 2,3,4**
4.	Develop an assembly language program to find the largest/smallest number in an array of 32 numbers	02	**CO1, 2,3,4**
5.	Develop an embedded C program to Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.	02	**CO1, 2,3,4**
6.	Develop an embedded C program to demonstrate the use of an external interrupt to toggle an LED On/Off.	02	**CO1, 2,3,4**
Open-ended Experiments			
1.	Develop an embedded C program to Interface a 4x4 keyboard and display the key code on an LCD.	02	**CO1, 2,3,4**
2.	Develop an embedded C program to display “Hello World” message using Internal UART.	02	**CO1, 2,3,4**

Introduction

An embedded system is an electronic/electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (software). An embedded system combines mechanical, electrical, and chemical components along with a computer, hidden inside, to perform a single dedicated purpose.

There are more computers on this planet than there are people, and most of these computers are single-chip microcontrollers that are the brains of an embedded system. Embedded systems are a ubiquitous component of our everyday lives. We interact with hundreds of tiny computers every day that are embedded into our houses, our cars, our bridges, our toys, and our work. As our world has become more complex, so have the capabilities of the microcontrollers embedded into our devices. Therefore, the world needs a trained workforce to develop and manage products based on embedded microcontrollers.

A general-purpose computing system is a combination of generic hardware and general-purpose operating system for executing a variety of application, whereas an embedded system is a combination system is a combination of special purpose hardware and embedded OS/firmware for executing a specific set of applications.

The ARM microcontroller stands for Advance Risk Machine; it is one of the extensive and most licensed processor cores in the world. The first ARM processor was developed in the year 1978 by Cambridge University, and the first ARM RISC processor was produced by the Acorn Group of Computers in the year 1985.

These processors are specifically used in portable devices like digital cameras, mobile phones, home networking modules and wireless communication technologies and other embedded systems due to the benefits, such as low power consumption, reasonable performance, etc. This article gives an overview of ARM architecture with each module's principle of working.

Keil MDK is the complete software development environment for a wide range of ARM Cortex-M based microcontroller devices. MDK includes the µVision IDE and debugger, Arm C/C++ compiler, and essential middleware components. It supports all silicon vendors with more than 6,000 devices.

Features of ARM Processor:

The **ALS/EVBRD/ARM7T9** board is a study Board which includes LPC2148 ARM7TDMI-S micro-controller with USB 2.0 Full speed device, multiple UARTs, SPI, I2C & on- chip 512K Flash and SRAM up to 40kB, produced by NXP semiconductors. The **ARM7TDMI-S** is a general-purpose **32-bit** microprocessor, which offers high performance and very low power consumption. The ARM processor is based on Reduced Instruction Set (**RISC**) architecture, And the instruction set and related decode mechanism are much simpler than those of micro programmed Complex Instruction Set Computers. This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective processor Core. Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

The ARM7TDMI-S processor also employs a unique architectural strategy known as THUMB, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue. The key idea behind THUMB is that of a super reduced instruction set. Essentially, the ARM7TDMI-S processor has two Instruction sets:

• The standard **32-bit** ARM instruction set.

• A **16-bit** THUMB instruction set.

The THUMB set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the Arm's performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because THUMB code operates on the same 32-bit register set as ARM code. THUMB code is able to provide up to 65% of the code size of ARM, and 160% of the performance of an equivalent ARM Processor connected to a 16-bit memory system.

MICROCONTROLLER FEATURES:

Following are the features of 16 bit /32 bit LPC2148 Arm Microcontroller

- PHILIPS LPC2148 is a 16-bit or 32-bit Microcontroller in a LQFP64-pin Package.
- 40 kB of on-chip static RAM and 512 kB of on-chip flash memory. 128-bit wide Interface/accelerator enables high-speed 60 MHz operation.
- The LPC2148 provides 100000 erase/write cycles and 20 years of Data-retention.
- In-System Programming/In-Application Programming (ISP/IAP) via on-chip boot Loader software. Single flash sector or full chip erases takes 400ms and Flash Programming takes 1ms per 256-byte line.
- USB 2.0 Full speed compliant device controller with 2 kB of endpoint RAM. In Addition, the LPC2148 provides 8 kB of on-chip RAM accessible to USB by DMA.
- Embedded ICE-RT and Embedded Trace Macro cell (**ETM**) interfaces offer real time Debugging with on-chip Real Monitor software and high-speed real-time tracing of Instruction execution.
- Two 10-bit ADCs provide a total of 14 analog inputs, with conversion times as low as 2.44μs per channel.
- Single 10-bit DAC provides variable analog output.
- Two 32-bit Timers/External event Counters (with four Capture and four Compare Channels each), PWM unit (six outputs) and watchdog.
- Low power Real-Time Clock (RTC) with independent power and 32 kHz clock input.
- Multiple serial interfaces including two UARTs (16C550 equivalent), two Fast I2Cbus(400 kbit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored interrupt controller (VIC) with configurable priorities and vector addresses. Up to 45 numbers of 5 V tolerant fast general purpose I/O pins in a tiny LQFP64 Package.
- Up to nine edge or level sensitive external interrupt pins available.
- 60 MHz maximum CPU clock available from programmable on-chip PLL with settling Time of 100 μs.
- On-chip integrated oscillator operates with an external crystal in range from 1 MHz to 30 MHz and with an external oscillator up to 50 MHz
- Power saving modes include Idle and Power-down.
- Individual power enable/disable of peripheral functions as well as peripheral clock Scaling for additional power optimization.
- Processor wake-up from Power-down mode via external interrupt, USB, Brown-Out Detect (BOD) or Real-Time Clock (RTC).
- Single power supply chip with Power-On Reset (POR) and BOD circuits: CPU Operating voltage range of 3.0 V to 3.6 V (3.3 V } 10 %) with 5 V tolerant I/O pads.

CONNECTORS DETAILS:

CN1 CONNECTOR: 28 pin 14 X 2 HEADER is connected to the controller.

PIN #	DESCRIPTION	PIN #	DESCRIPTION
1	TDO-JTAG	15	GND
2	TD1-JTAG	16	RTCX1
3	TMS-JTAG	17	RESET
4	TRST-JTAG	18	RTCX2
5	TCK-JTAG	19	VBAT
6	P0.11	20	P0.25
7	P0.11	21	NC
8	P0.11	22	NC
9	P0.11	23	NC
10	3.3V	24	NC
11	GND	25	NC
12	3.3V	26	P1.24(2148)
13	NC	27	NC
14	NC	28	NC

CN2 CONNECTOR: 28 pin 14 X 2 HEADER is connected to the controller.

PIN #	DESCRIPTION	PIN #	DESCRIPTION
1	NC	15	NC
2	NC	16	NC
3	NC	17	NC
4	P0.8	18	GND
5	3.3V	19	3.3V
6	NC	20	NC
7	NC	21	NC
8	GND	22	NC
9	NC	23	NC
10	NC	24	NC
11	P1.20	25	P0.22
12	P1.21	26	P0.22
13	P1.22	27	P0.22
14	P1.23	28	NC

CN3 CONNECTOR: 28 pin 14 X 2 HEADER is connected to the controller.

PIN #	DESCRIPTION	PIN #	DESCRIPTION
1	NC	15	P1.17
2	NC	16	P0.7
3	P1.25	17	P0.6
4	P0.3	18	P0.5
5	P0.14	19	P0.4
6	3.3V	20	P0.3
7	GND	21	P0.2
8	P0.31	22	P0.15
9	P0.30	23	3.3V
10	P0.22	24	GND
11	P0.28	25	P0.14
12	P1.19	26	P0.13
13	P1.18	27	P0.12
14	P1.16	28	NC

CN4 CONNECTOR: 28 pin 14 X 2 HEADER is connected to the controller.

PIN #	DESCRIPTION	PIN #	DESCRIPTION
1	P0.21	15	NC
2	P0.20	16	NC
3	P0.19	17	NC
4	P0.18	18	NC
5	P0.17	19	NC
6	P0.16	20	NC
7	NC	21	3.3V
8	GND	22	GND
9	3.3V	23	TXD0
10	NC	24	RXD0
11	NC	25	RTCK-JTAG
12	NC	26	NC
13	NC	27	NC
14	NC	28	NC

CN7 CONNECTOR: 26 pin FRC connected to the controller, which is compatible Standard External Interfaces. SHORT jumper JP4 while using External Interfaces.

PIN #	DESCRIPTION	PIN #	DESCRIPTION
1	P0.12	14	P0.17
2	P0.13	15	P0.30
3	P0.10	16	P0.31 THROUGH JP2(1,2) OR P1.24 THROUGH JP2(2,3)
4	P0.11	17	P0.28
5	P0.8	18	P0.28
6	P0.9	19	P0.28
7	P0.22	20	P0.28
8	P0.23	21	P0.28
9	P0.20	22	P0.28
10	P0.21	23	P0.14(Short JP13 to use this pin)
11	P0.18	24	P0.15
12	P0.19	25	+5V THROUGH JP4 13 P0.16
13	P0.16	26	GND

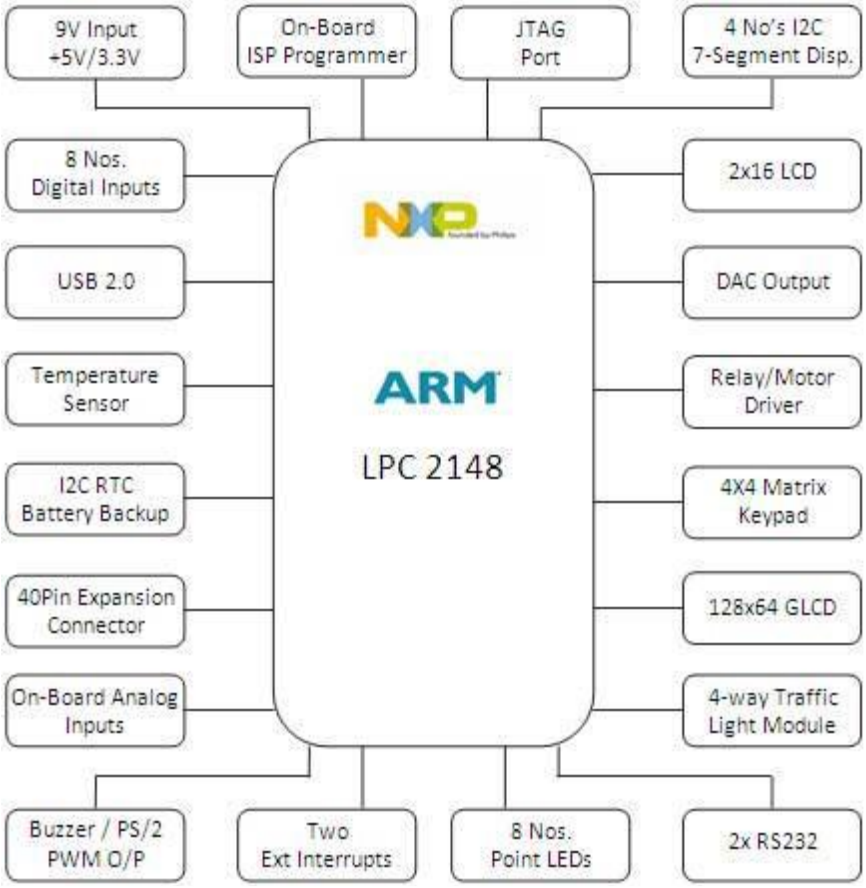
TEST POINT DETAILS:

TP1	+5V
TP2	+3.3V
TP3	GROUND
TP4	PWM output

POT DETAILS:

POT1	50K ANVI POT for **LCD** Contrast.

General Block Diagram:



POWER SUPPLY:

The external power can be AC or DC, with a voltage between (9V/12V, 1A output) at 230V AC input.

The ARM board produces +5V using an LM7805 voltage regulator, which provides supply to the Peripherals.

• LM1117 Fixed +3.3V positive regulator used for processor & processor related peripherals.

FLASH PROGRAMMING UTILITY:

NXP (Philips) NXP Semiconductors produce a range of Microcontrollers that feature both on-chip Flash memory and the ability to be reprogrammed using In-System Programming technology.

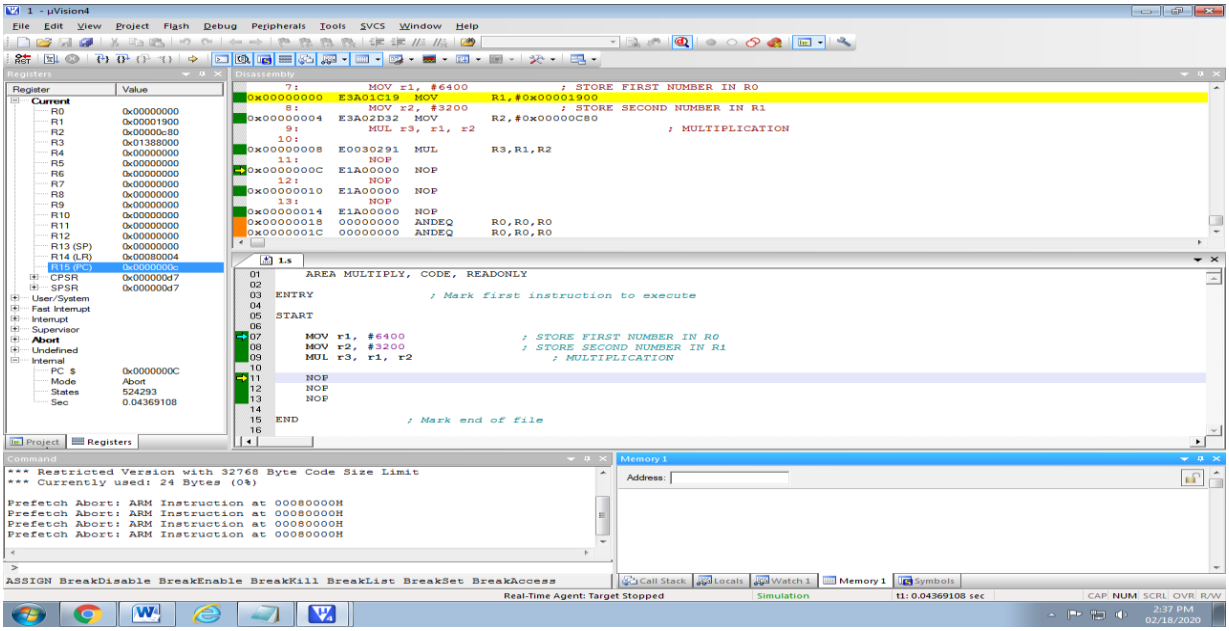
1. Write a program to multiply two 16 bit binary numbers.

```
        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
```

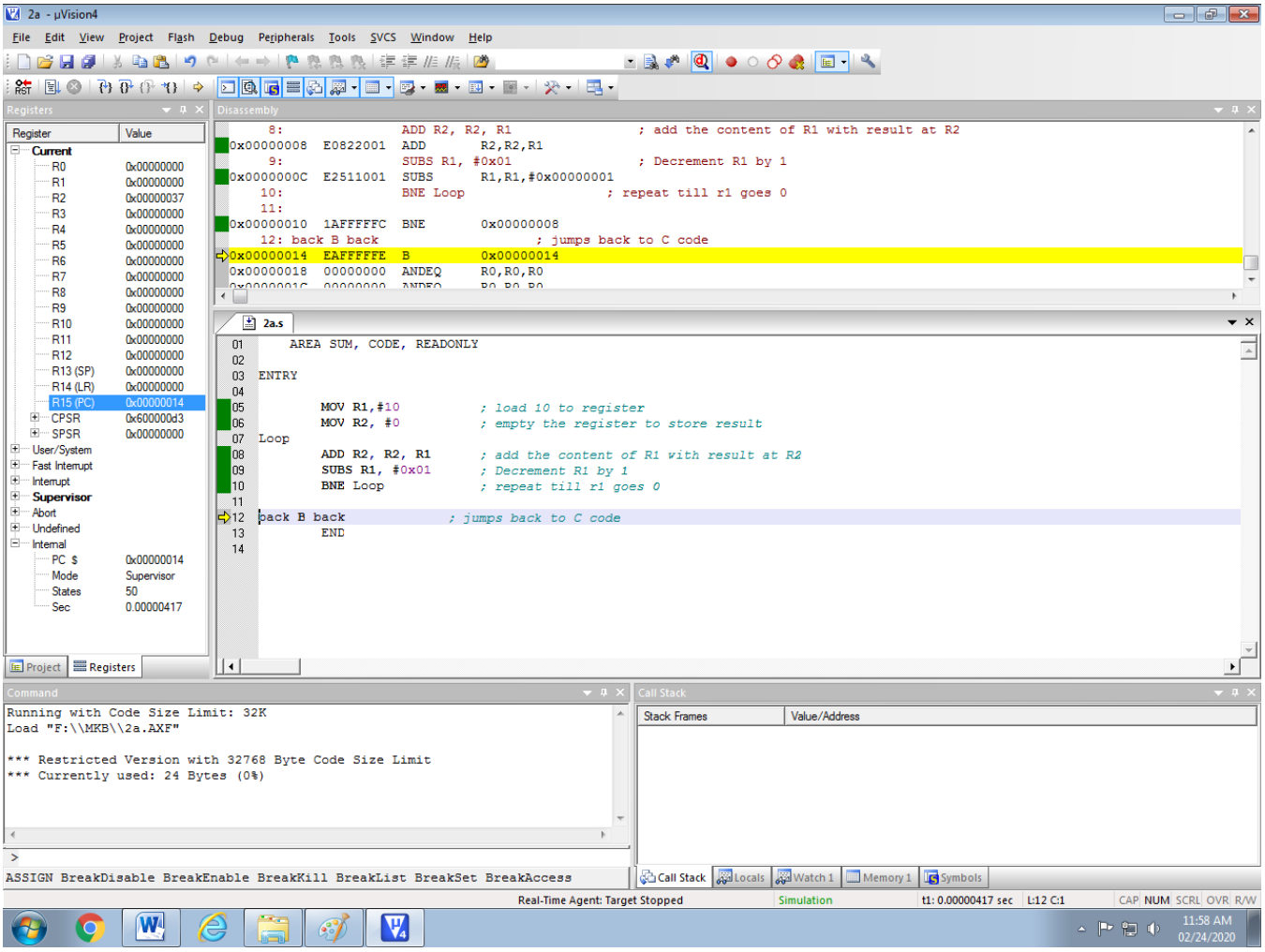
Output:



2. Write a program 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
```

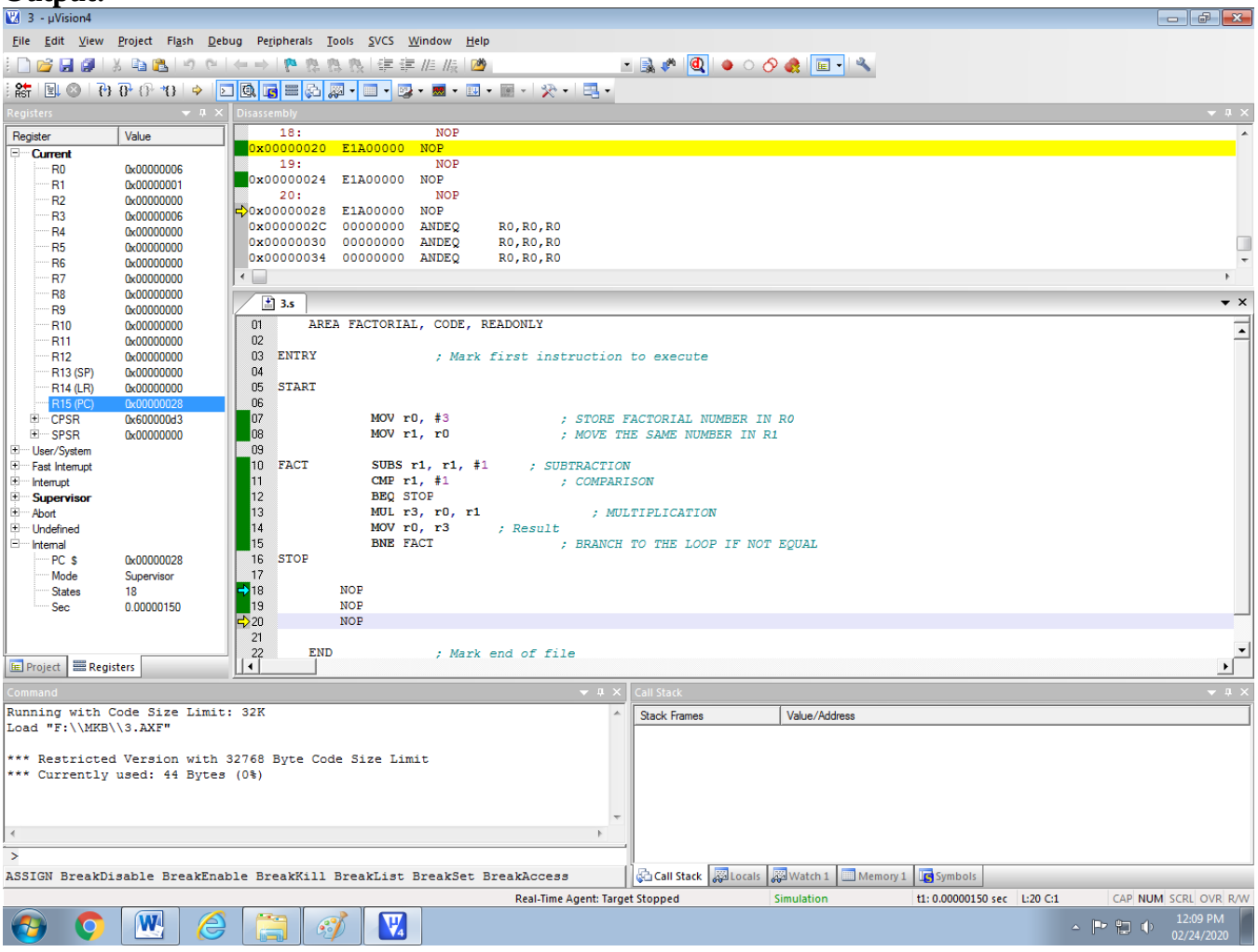
Output:



3. Write a program to find factorial of a number.

```
        AREA FACTORIAL, CODE, READONLY

        ENTRY                           ; Mark first instruction to execute

START

                MOV r0, #3              ; STORE FACTORIAL NUMBER IN R0
                MOV r1, r0      ; MOVE THE SAME NUMBER IN R1

FACT            SUBS r1, r1, #1         ; SUBTRACTION
                CMP r1, #1              ; COMPARISON
                BEQ STOP
                MUL r3, r0, r1  ; MULTIPLICATION
                MOV    r0, r3           ; Result
                BNE FACT                ; BRANCH TO THE LOOP IF NOT EQUAL
STOP

back            B   back
        NOP
        NOP

        END                             ; Mark end of file
```

Output:



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

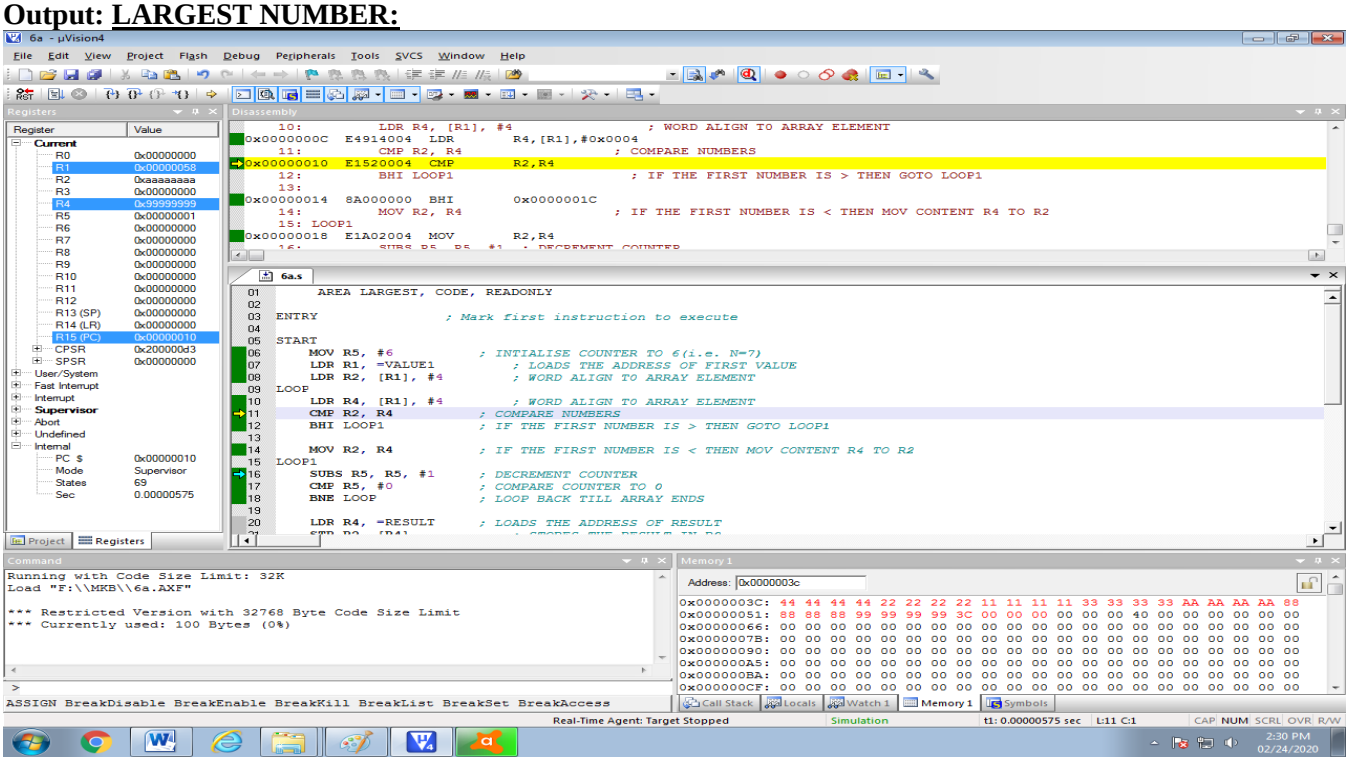
LARGEST NUMBER:

```
        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
```

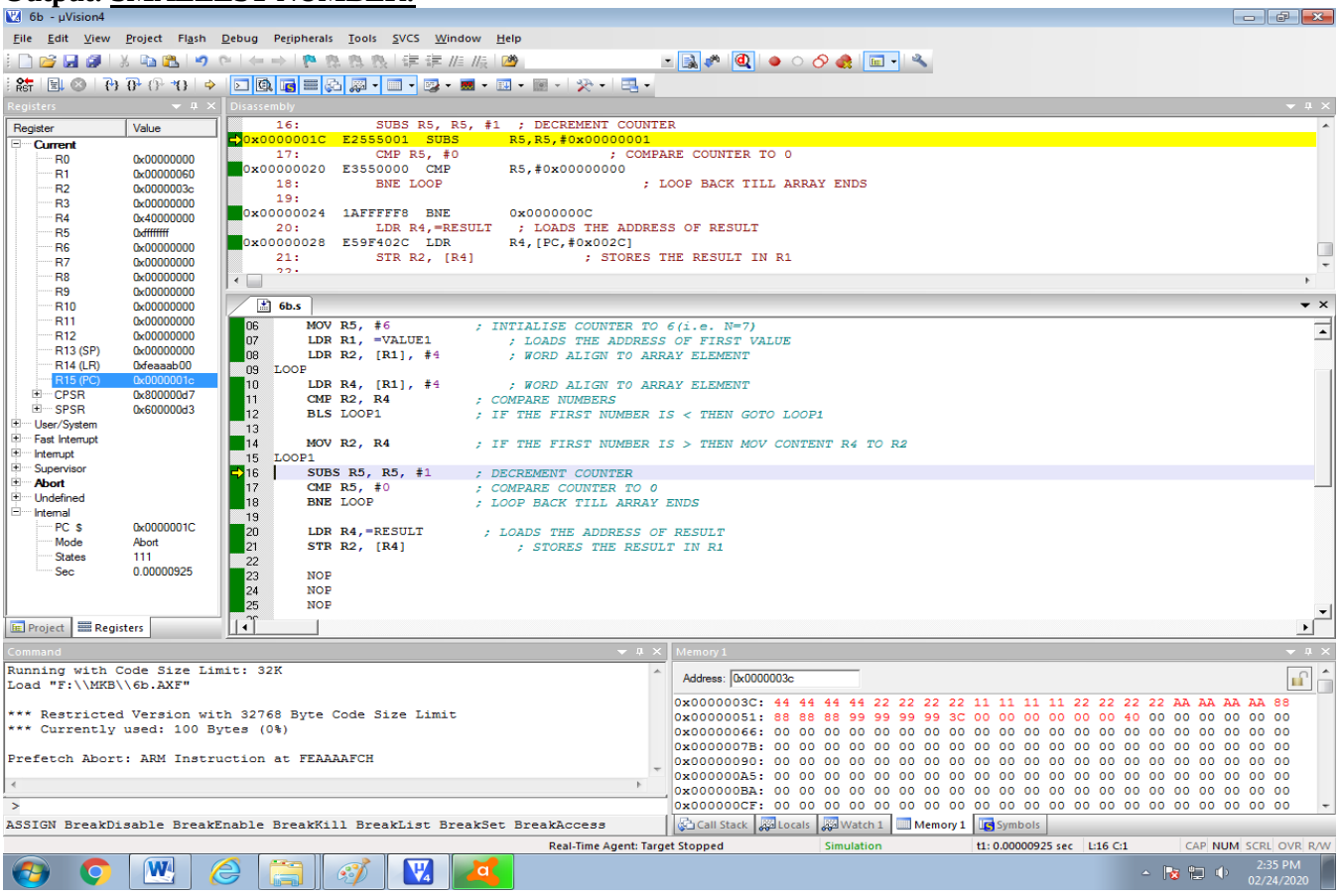
SMALLEST NUMBER:

```
        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 R1

        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:



5. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.

```
#include <LPC21xx.h>

voidclock_wise(void) ;
voidanti_clock_wise(void) ;

unsigned  int var1 ;
unsigned long inti = 0 , j = 0 , k = 0 ;

int main(void)
{
        PINSEL2 = 0x00000000;                   //P1.20 to P1.23 GPIO
        IO1DIR |= 0x00F00000 ;                  //P1.20 to P1.23 made as output

                while(1)
                {

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

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

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

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

        }

}// End of main

voidclock_wise(void)
{
        var1 = 0x00080000;                      //For Clockwise
for(i = 0 ; i<= 3 ; i++ )     // for A B C D Stepping
        {
                var1 <<= 1 ;

                IO1CLR =0x00F00000 ;            //clearing all 4 bits

                IO1SET = var1 ;                         // setting perticular bit

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


        }

}

voidanti_clock_wise(void)
{
        var1 = 0x00800000 ;                             //For Anticlockwise

        IO1CLR =0x00F00000 ;                                            //clearing all 4 bits
```

```
            IO1SET = var1 ;

            for( k = 0 ; k < 3000 ; k++ ) ;

    for(i = 0 ; i< 3 ; i++ )                    // for A B C D Stepping
            {
                        var1 >>=1;                                          //rotating bits
                    IO1CLR =0x00F00000 ;            // clar all bits before setting

                        IO1SET = var1 ;                                     //          setting
    perticular bit

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

            }
    }
```

6. Demonstrate the use of an external interrupt to toggle an LED On/Off.

```
#include <LPC21xx.h>

unsignedintdelay,lg;

int main ()
{
        PINSEL1 = 0x00000000 ;              // Configure P0.16 to P0.31 as GPIO
        IO0DIR  = 0x00FF0000 ;              // Configure P0.16 to P0.23 as Output

        while(1)
        {
                IO0CLR = 0x00FF0000; // CLEAR (0) P0.10 to P0.13 and P0.18 to P0.21, LEDs ON
                for(delay=0; delay<500000; delay++);    // delay
                IO0SET = 0x00FF0000;       // SET (1) P0.10 to P0.13 and P0.18 to P0.21, LEDs OFF
                for(delay=0; delay<500000; delay++);    // delay
        }

}
```

7. Display "DSCE-EIE-ARM LAB" message using Internal UART.

```
#include<lpc214x.h>

voiduart_init(void);
unsignedint delay;
unsigned char *ptr;
unsigned char arr[]="DSCE-EIE-ARM LAB\r";

int main()
{
        while(1)
        {
                uart_init();
                ptr = arr;
                while(*ptr!='\0')
                {
                        U0THR=*ptr++;
                        while(!(U0LSR & 0x40)== 0x40);
                        for(delay=0;delay<=600;delay++);
                }
                for(delay=0;delay<=60000;delay++);
        }
}
voiduart_init(void)
{
  PINSEL0=0X0000005;                    //select TXD0 and RXD0 lines
        U0LCR = 0X00000083;             //enable baud rate divisor loading and
        U0DLM = 0X00;                   //select the data format
        U0DLL = 0x13;                           //select baud rate 9600 bps
        U0LCR = 0X00000003;

}
```

8. Interface a 4x4 keyboard and display the key code on an LCD.

```
#include<lpc214x.h>
#include<stdio.h>

//Function prototypes
voidlcd_init(void);
voidwr_cn(void);
voidclr_disp(void);
void delay(unsigned int);
voidlcd_com(void);
voidwr_dn(void);
voidlcd_data(void);

unsigned char temp1;
unsigned long inttemp,r=0;
unsigned char *ptr,disp[] = "ALS,R&D SECTION,",disp1[] = "BENGALURU-58";

int main()
{
        IO0DIR = 0x000000FC;            //configure o/p lines for lcd
        IO0PIN = 0X00000000;

        delay(3200);                    //delay
        lcd_init();             //lcdintialisation
        delay(3200);                    //delay
clr_disp();                             //clear display
        delay(3200);            //delay

//.....LCD DISPLAY TEST.....//
        temp1 = 0x80;                   //Display starting address      of first line 1st pos
        lcd_com();

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

        temp1 = 0xC0;               // Display starting address of second line 4th pos
        lcd_com();

        ptr = disp1;
        while(*ptr!='\0')
   {
        temp1 = *ptr;
        lcd_data();
                ptr ++;
   }
        while(1);
}  //end of main()

// lcd initialisation routine.
```

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

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

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

        temp = 0x20;    // change to 4 bit mode from default 8 bit mode
        wr_cn();
        delay(3200);

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

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

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

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

        temp1 = 0x80;  // set the cursor to beginning of line 1
        lcd_com();
        delay(800);
}

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

// command nibble o/p routine
voidwr_cn(void)             //write command reg
{
        IO0CLR  = 0x000000FC;                   // clear the port lines.
        IO0SET          = temp;                         // Assign the value to the PORT lines
        IO0CLR  = 0x00000004;                   // clear bit  RS = 0
```

```
        IO0SET          = 0x00000008;           // E=1
        delay(10);
        IO0CLR  = 0x00000008;
}

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

// data o/p routine which also outputs high nibble first
// and lower nibble next
voidlcd_data(void)
{
        temp = temp1 & 0xf0;
temp = temp ;//<< 6;
wr_dn();
temp= temp1 & 0x0f;
temp= temp << 4;
wr_dn();
delay(100);
}

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

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