MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY

(18CSL48)

LAB MANUAL

For

IV SEMESTER

MAHARAJA INSTITUTE OF TECHNOLOGY, MYSORE

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

MIT MYSORE

VISION AND MISSION OF INSTITUTE

VISION OF THE INSTITUTE

"To be recognized as a premier technical and management institution promoting extensive education fostering research, innovation and entrepreneurial attitude"

MISSION OF THE INSTITUTE

1. To empower students with indispensable knowledge through dedicated teaching and collaborative learning.
2. To advance extensive research in science, engineering and management disciplines.
3. To facilitate entrepreneurial skills through effective institute-industry collaboration and interaction with alumni.
4. To instill the need to uphold ethics in every aspect.
5. To mould holistic individuals capable of contributing to the advancement of the society.

VISION, MISSION, PROGRAM EDUCATIONAL OBJECTIVES AND PROGRAM SPECIFIC OUTCOME OF DEPARTMENT

VISION OF THE DEPARTMENT

To be recognized as the best centre for technical education and research in the field of information science and engineering.

MISSION OF THE DEPARTMENT

- To facilitate adequate transformation in students through a proficient teaching learning process with the guidance of mentors and all-inclusive professional activities.
- To infuse students with professional, ethical and leadership attributes through industry collaboration and alumni affiliation.
- To enhance research and entrepreneurship in associated domains and to facilitate real time problem solving.

PROGRAM EDUCATIONAL OBJECTIVES:

- Proficiency in being an IT professional, capable of providing genuine solutions to information science problems.
- Capable of using basic concepts and skills of science and IT disciplines to pursue greater competencies through higher education.

- Exhibit relevant professional skills and learned involvement to match the requirements of technological trends.

PROGRAM SPECIFIC OUTCOME:

Student will be able

- **PSO1:** Apply the principles of theoretical foundations, data Organizations, networking concepts and data analytical methods in the evolving technologies.
- **PSO2:**Analyse proficient algorithms to develop software and hardware competence in both professional and industrial areas

General Lab Guidelines:

- Conduct yourself in a responsible manner at all times in the laboratory. Intentional misconduct will lead to the exclusion from the lab.
- Do not wander around, or distract other students, or interfere with the laboratory experiments of other students.
- Read the handout and procedures before starting the experiments. Follow all written and verbal instructions carefully. If you do not understand the procedures, ask the instructor or teaching assistant.
- Attendance in all the labs is mandatory, absence permitted only with prior permission from Class teacher.
- The workplace has to be tidy before, during and after the experiment.
- Do not eat food, drink beverages or chew gum in the laboratory.
- Every student should know the location and operating procedures of all Safety equipment including First Aid Kit and Fire extinguisher.

DO'S:-

- Uniform and ID card are must.
- Strictly follow the procedures for conduction of experiments.
- Records have to be submitted every week for evaluation.
- Chairs and stools should be kept under the workbenches when not in use.
- After the lab session, switch off every supply, disconnect and disintegrate the experiments and return the components.
- Keep your belongings in designated area.
- Never use damaged instruments, wires or connectors. Hand these parts to the instructor/ teaching assistant.
- Sign the log book when you enter/leave the laboratory.

DONT'S:-

- Don't touch open wires unless you are sure that there is no voltage. Always disconnect the plug by pulling on the connector body not by the cable. Switch off the supply while you make changes to the experiment.
- Don't leave the experiment table unattended when the experimental setup supply is on.
- Students are not allowed to work in laboratory alone or without presence of the teaching staff/ instructor.
- No additional material should be carried by the students during regular labs.
- Avoid stepping on electrical wires or any other computer cables.

MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY

(Effective from the academic year 2018 -2019)

SEMESTER – IV

Course Code	18CSL48	CIE Marks	40
Number of Contact Hours/Week	0:2:2	SEE Marks	60
Total Number of Lab Contact Hours	36	Exam Hours	03
Credits – 2			

Course Learning Objectives: This course (18CSL48) will enable students to:

- Develop and test Assembly Language Program (ALP) using ARM7TDMI/LPC2148.
- Conduct the experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

Programs List:

PART A

The following experiments by writing Assembly Language Program (ALP) using ARM7TDMI/LPC2148 using an evaluation board/simulator and the required software tool.

1. Write an ALP to multiply two 16 bit binary numbers.
2. Write an ALP to find the sum of first 10 integer numbers.
3. Write an ALP to find factorial of a number.
4. Write an ALP to add an array of 16 bit numbers and store the 32 bit result in internal RAM.
5. Write an ALP to find the square of a number (1 to 10) using look-up table.
6. Write an ALP to find the largest/smallest number in an array of 32 numbers.
7. Write an ALP to arrange a series of 32 bit numbers in ascending/descending order.
8. Write an ALP to count the number of ones and zeros in two consecutive memory locations.

PART B

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

9. Display “Hello World” message using Internal UART.

10. Interface and Control a DC Motor.
11. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.
12. Determine Digital output for a given Analog input using Internal ADC of ARM controller.
13. Interface a DAC and generate Triangular and Square waveforms.
14. Interface a 4x4 keyboard and display the key code on an LCD.
15. Demonstrate the use of an external interrupt to toggle an LED On/Off.
16. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay.

Laboratory Outcomes: The student should be able to:

- Develop and test Assembly Language Program (ALP) using ARM7TDMI/LPC2148
- Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

Conduct of Practical Examination:

- All laboratory experiments, excluding the first, are to be included for practical examination.
- Experiment distribution
  - For questions having only one part: Students are allowed to pick one experiment from the lot and are given equal opportunity.
  - For questions having part A and B: Students are allowed to pick one experiment from part A and one experiment from part B and are given equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure part to be made zero.
- Marks Distribution *(Subjected to change in accordance with university regulations)*
  - For questions having only one part – Procedure + Execution + Viva-Voce: 15+70+15 =100 Marks
  - For questions having part A and B
    i. Part A – Procedure + Execution + Viva = 4 + 21 + 5 = 30 Marks
    ii. Part B – Procedure + Execution + Viva = 10 + 49+ 11 = 70 Marks

INDEX

INTRODUCTION

Evolution of Microprocessor:

The microprocessor forms the brain of the Central Processing Unit (CPU). Microprocessor is an engine which can compute various operations fabricated on a single chip. The internal architecture of microprocessor determines what operations can be performed on a microprocessor and how it can be performed.

The first microprocessor was introduced in the year 1971. It was introduced by Intel and was named Intel 4004.Intel 4004 is a 4 bit microprocessor and it was not a powerful microprocessor. It can perform addition and subtraction operation on 4 bits at a time.

However it was Intel's 8080 was the first microprocessor to make it to Home computers. It was introduced during the year 1974 and it can perform 8 bit operations. Then during the year 1976, Intel introduced 8085 processors which is nothing but an update of 8080 processors.8080 processors are updated by adding two Enable/Disable Instructions, Three added interrupt pins and serial I/O pins.

Intel introduced 8086 pins during the year 1976. The major difference between 8085 and 8086 processor is that 8085 is an 8 bit processor, but 8086 processor is a 16 bit processor. The greatest advantage of the above processors are that it do not contain Floating point instructions. Here floating point refers to the radix point or decimal point. For example: 123.456 is a floating point representation. Processors such as 8085 and 8086 do not support such representations and instructions.

Intel later introduced 8087 processor which was the first math co-processor and later the 8088 processor which was incorporated into IBM personal computers.

As the years progressed lots of processors from 8088,80286,80386,80486,Pentium II, Pentium III, Pentium IV and now Core2Duo,Dual Core and Quad core processors are the latest in the market.

Types of microprocessors:

- **Complex instruction set microprocessor**
  The processors are designed to minimise the number of instructions per program and ignore the number of cycles per instructions. The compiler is used to translate a high level language to assembly level language because the length of code is relatively short and an extra RAM is used to store the instructions. These processors can do tasks like downloading, uploading and recalling data from memory. Apart from these tasks these microprocessor can perform complex mathematical calculation in a single command. Example: IBM 370/168, VAX 11/780.
- **Reduced instruction set microprocessor**
  These processor are made according to function. They are designed to reduce the execution time by using the simplified instruction set. They can carry out small things in specific commands. These processors complete commands at faster rate. They require only one clock cycle to implement a result at uniform execution time. There are number of registers and less number of transistors. To access the memory location LOAD and STORE instructions are used. Example: Power PC 601, 604, 615, 620
- **Super scalar microprocessor**
  These processors can perform many tasks at a time. They can be used for ALUs and multiplier like array. They have multiple operation unit and perform tasts my executing multiple commands.
- **Application specific integrated circuit**
  These processors are application specific like for personal digital assistant computers. They are designed according to proper specification.
- **Digital signal multiprocessor**
  These processors are used to convert signals like analog to digital or digital to analog. The chips of these processors are used in many devices such as RADAR SONAR home theatres etc.

Advantages of microprocessor –

1. High processing speed
2. Compact size
3. Easy maintenance
4. Can perform complex mathematics
5. Flexible
6. Can be improved according to requirement

Disadvantages of microprocessors –

1. Overheating occurs due to overuse
2. Performance depends on size of data
3. Large board size than microcontrollers
4. Most microprocessors do not support floating point operations

Evolution of Micro Controller

The Micro controller was developed by Intel in 1980. The 8051 is one of the most popular 8 bit micro controllers and combines an instruction set that allows tight coding of small particularly I/O application with enough power and a large enough program space that it can be used with C Language.
The development of 16 and 32 bit microprocessors contributed to the growth of powerful personal computers that are used in all walks of life. Because of their processing power and speed, these 16 and 32 bit microprocessors have also found their way into the design of standalone products such as electronic instruments which require sophisticated control capability. In the evolution of microprocessor capability, instead of focusing upon larger word widths and address spaces. The present emphasis 'is upon exceedingly fast real time control. The development of microcontrollers has focused upon the integration of the facilities needed to support fast control into a single chip.

Intel has introduced standard 8-bit microcontroller 8048 in 1976. The same company has continued to drive the evolution of single chip microcontrollers. In the year 1980, Intel has introduced the 8051 microcontroller, with higher performance than 8048. With the advantages of 8051, the microcontroller's applications took a peak level. The 8-bit microcontroller, 8051 family. Quickly gained the position of the second generation world standard microcontrollers.

Because of the advanced semiconductor technology, it has become possible to integrate more than 1.00.000 transistors onto a single silicon chip. Intel has made use of this advanced process technology and developed a new generation of single chip 16 bit microcontrollers called the MCS-96 (8096 family). The 8096 family offers the highest level of system integration ever achieved on a single chip microcontroller with l,20,000 transistors. This 8096 microcontroller has 16 bit CPU, 8K bytes of program memory. 232 bytes of data memory and both analog and digital type of 1/ O features.

The Motorola Microcontroller family was first introduced to the market in 1978 and is built in the same pattern of the microprocessor 6800. Even though the Microcontroller 6801 family was designed similar to the microprocessor 6800, its design and instruction set have been modified to suit the control applications.

The microcontroller 6801 family includes on-chip Input/output ports, an Asynchronous serial communication device and 16 bit timer modules. The Microcontrollers 6801. 6803, 6805. 6811 are available from Motorola Company. The 6811 microcontroller family have different version with ROM, RAM, EPROM, and EEPROM. These versions are denoted by suffix characters and numbers.

Classification According to Memory Architecture:

Memory architecture of microcontroller are two types, they are namely:

- **Harvard Memory Architecture Microcontroller**: The point when a microcontroller unit has a dissimilar memory address space for the program and data memory, the microcontroller has Harvard memory architecture in the processor.

- **Princeton Memory Architecture Microcontroller**: The point when a microcontroller has a common memory address for the program memory and data memory, the microcontroller has Princeton memory architecture in the processor.

Classification According to Memory Devices

- **Embedded memory microcontroller**: When an embedded system has a microcontroller unit that has all the functional blocks available on a chip is called an embedded microcontroller. For example, 8051 having program & data memory, I/O ports, serial communication, counters and timers and interrupts on the chip is an embedded microcontroller.

- **External Memory Microcontroller**: When an embedded system has a microcontroller unit that has not all the functional blocks available on a chip is called an external memory microcontroller. For example, 8031 has no program memory on the chip is an external memory microcontroller.

Classification According to Instruction Set

- **CISC**: CISC is a Complex Instruction Set Computer. It allows the programmer to use one instruction in place of many simpler instructions.

- **RISC**: The RISC is stands for Reduced Instruction set Computer, this type of instruction sets reduces the design of microprocessor for industry standards. It allows each instruction to operate on any register or use any addressing mode and simultaneous access of program and data.

Classification According to Number of Bits

- The **16-bit** microcontroller performs greater precision and performance as compared to 8-bit. For example 8 bit microcontrollers can only use 8 bits, resulting in a final range of 0×00 – 0xFF (0-255) for every cycle. In contrast, 16 bit microcontrollers with its 16 bit data width has a range of 0×0000 – 0xFFFF (0-65535) for every cycle. A longer timer most extreme worth can likely prove to be useful in certain applications and circuits. It can automatically operate on two 16 bit numbers. Some examples of 16-bit microcontroller are 16-bit MCUs are extended 8051XA, PIC2x, Intel 8096 and Motorola MC68HC12 families.

The **32-bit** microcontroller uses the 32-bit instructions to perform the arithmetic and logic operations. These are used in automatically controlled devices including implantable medical devices, engine control systems, office machines, appliances and other types of embedded systems. Some examples are Intel/Atmel 251 family, PIC3x.

ARM (ADVANCED RISC MACHINE)

The **ARM processor** is a 32-bit RISC processor, meaning it is built using the RISC (reduced instruction set computer) ISA (instruction set architecture). ARM processors are microprocessors and are widely used in many of the mobile phones sold each year, as many as 98% of mobile phones. They are also used in PDAs (personal digital assistants), digital media and music layers, hand-held gaming systems, calculators, and even computer drives. The first ARM processor-based computer was the Acorn Archimedes, released in 1987. Apple Computer became involved with helping to improve the ARM technology in the late 1980s, with their work resulting in the ARM6 technology in 1992. Later, Acorn used the ARM6-based ARM 610 processor in their RISC PC's in 1994. Today, the ARM architecture is licensed to many companies, including Apple, Cirrus Logic, Intel, LG, Microsoft, NEC, Nintendo, Nvidia, Sony, Samsung, Texas Instruments, and many more. The latest developed ARM processor families include ARM11 and Cortex. ARM processors capable of 64-bit processing are currently in development.

ARM processor features include:

- Load/store architecture.
- An orthogonal instruction set.
- Mostly single-cycle execution.
- Enhanced power-saving design.
- 64 and 32-bit execution states for scalable high performance.
- Hardware virtualization support.

ARM Holdings offers users the following types of processors:

- **Cortex-A:** built for advanced operating systems and exhibits the highest possible performance;
- **Cortex-R:** caters perfectly to the needs of real-time applications and provides its users with the fastest response times;
- **Cortex-M:** mainly built for microcontrollers;
- **SecurCore:** takes care of security applications;
- **Machine Learning:** for ML application purposes.

- **PROJECT CREATION IN KEILUV4 IDE:**

Create a project folder before creating NEW project.

- Open Keil uVision4 IDE software by double clicking on “Keil Uvision4” icon.
- Go to “Project” then to “New uVision Project” and save it with a name in the respective project folder, already you created.
- Select the device as “NXP” In that “LPC2148” then press OK and then press “YES” Button to add “startup.s” file.
- In startup file go to Configuration Wizard. In Configuration Wizard window uncheck PLL Setup and check VPBDIV Setup.
- Go to “File” In that “New” to open an editor window. Create your source file and use the header file “lpc21xx.h” in the source file and save the file. Colour syntax highlighting will be enabled once the file is saved with a extension such as “.C “.
- Right click on “Source Group 1” and select the option “Add Existing Files to Group
- Source Group 1“add the *.C source file(s) to the group. After adding the source file you can see the file in Project Window. Then go to “Project” in that “Translate” to compile the File (s). Check out the Build output window.

Right click on Target1 and select options for Target Target1.

Then go to option “Target” in that

- Xtal 12.0MHz
- Select “Use MicroLIB”.
- Select IROM1 (starting 0x0 size 0x80000).
- Select IRAM1 (starting 0x40000000 size 0x8000).

Then go to option “Output”

- Select “Create Hex file”

Then go to option “Linker”

- Select “Use Memory Layout for Target Dialog”.

To come out of this window press OK.

Go to “Project” in that “Build Target” for building all source files such as “.C”,”.ASM”, “.h”, files, etc…This will create the *.HEX file if no warnings & no Errors. Check out the Build output window.

4.3 FLASH MAGIC VERSION 6.1: Installation of Flash Magic as follows.

Go to EXE folder and then Flash Magic 6.1 folder in the CD & run FlashMagic.exe

- Next
- Click on the option "I Accept the Agreement" and then give Next
- Then it asks the Destination location, Click Next.
- Further Select start menu folder, Click Next.
- Select "Create a desktop icon" then Next
- It prompts "Ready to Install" Click INSTALL.
- Click Finish to complete the installation.

ISP PROGRAMMING:

FLASH MAGIC software can be used to download the HEX files to the Flash memory of controller.

SETTINGS IN FLASH MAGIC:

Step1. Communications:

- Device : LPC2148
- Com Port : COM1(Check and connect)
- Baud Rate : 9600
- Interface : None(ISP)
- Oscillator : 12MHz

Step2. ERASE:

- Select "Erase Blocks Used By Hex File".

Step3. Hex file:

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

Step4. Options

- Select "Verify after programming".

Step5. Start:

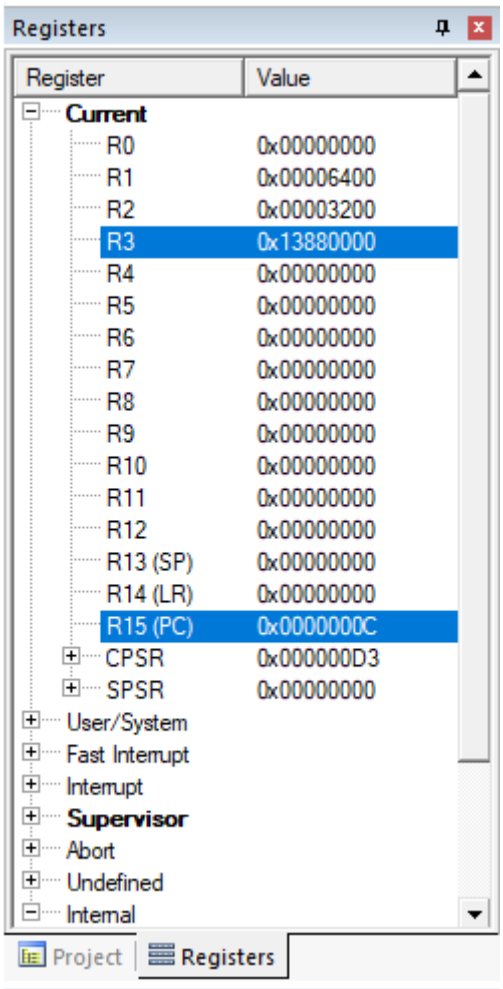
- Click Start to download the hex file to the controller.

After downloading the code the program starts executing in the hardware, then remove the ISP jumper JP7.

PART A

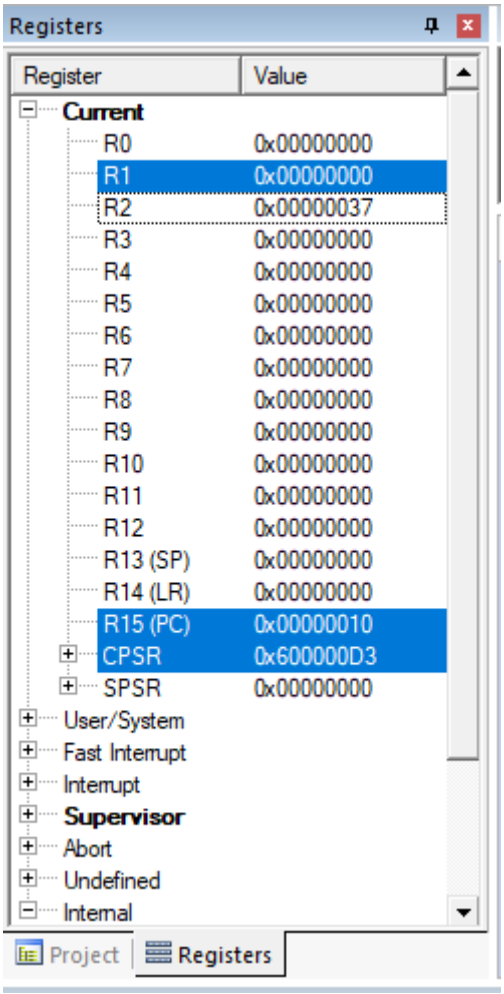
1. ALP TO MULTIPLY TWO 16BIT NUMBERS

LABEL FIELD	MNEMONIC FIELD	COMMENTS FILED
	AREA MULTIPLY, CODE, READONLY	
	ENTRY	; Mark first instruction to execute
	MOV R1,#0X6400	; STORE FIRST NUMBER IN R0
	MOV R2,#0X3200	; STORE SECOND NUMBER IN R1
	MUL R3,R1,R2	; MULTIPLICATION
HERE	B HERE	
	END	; MARK END OF FILE



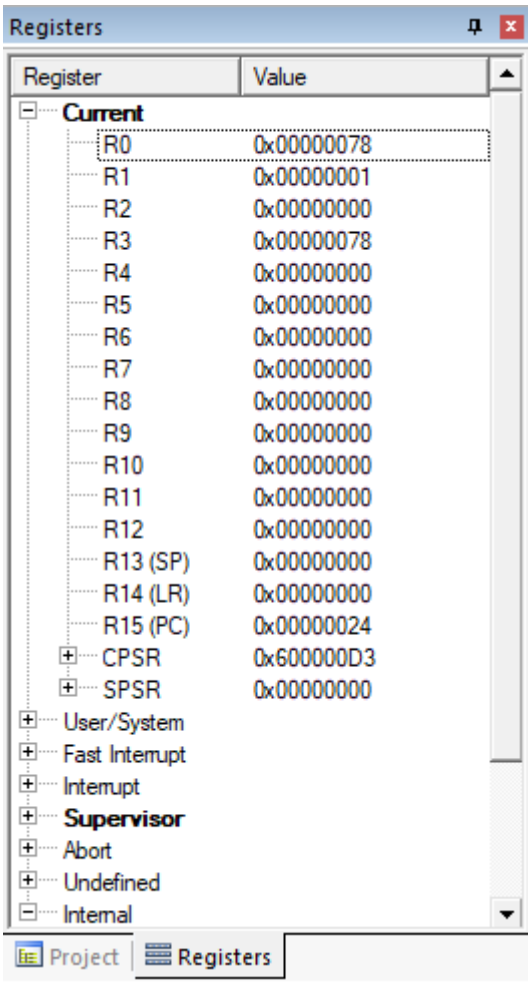
2. ALP TO FIND THE SUM OF FIRST 10 INTEGER NUMBERS

LABEL FIELD	MNEMONIC FIELD	COMMENTS FILED
	AREA INTSUM, CODE, READONLY	
	ENTRY	; Mark first instruction to execute
	MOV R1,#10	; LOAD 10 TO REGISTER
	MOV R2,#0	; EMPTY R2 REGISTER TO STORE RESULT
LOOP	ADD R2,R2,R1	; ADD THE CONTERNT OF R1 WITH RESULT AT R2
	SUBS R1,#0X01	; DECREMENT R1 BY 1
	BNE LOOP	; REPEAT TILL R1 GOES TO ZERO
HERE	B HERE	
	END	



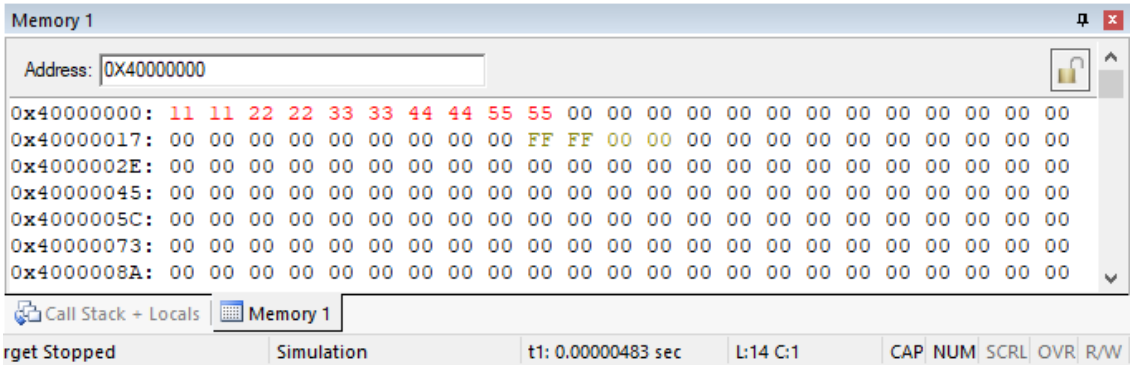
3. ALP TO FIND FACTORIAL OF A NUMBER.

LABEL FIELD	MNEMONIC FIELD	COMMENTS FILED
	AREA FACTORIAL,CODE, READONLY	
	ENTRY	; MARK FIRST INSTRUCTION TO EXECUTE
	MOV R0, #5	; 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	NOP	
HERE	B HERE	
	END	; MARK END OF FILE



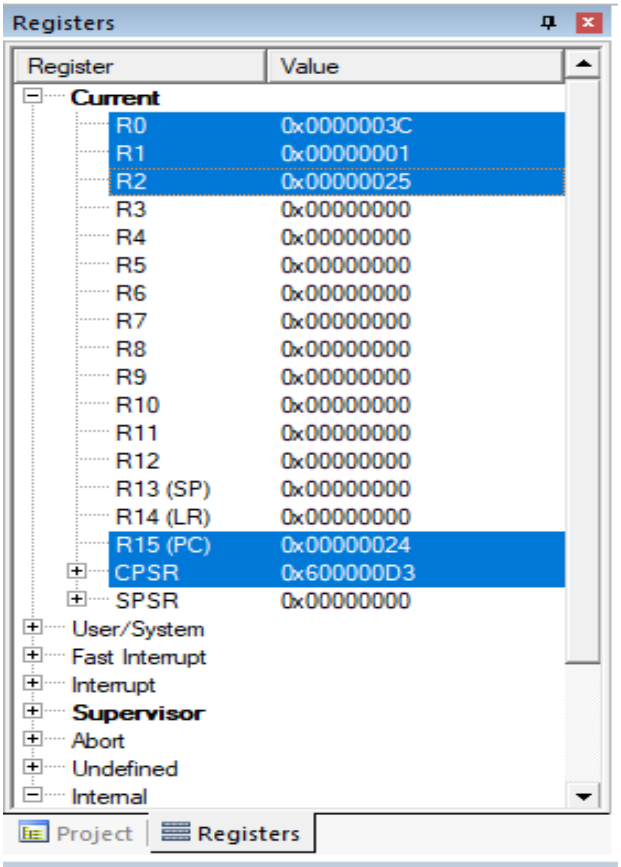
4. ALP TO ADD AN ARRAY OF 16BIT NUMBERS AND STORE THE 32BIT RESULT IN INTERNAL RAM.

LABEL FIELD	MNEMONIC FIELD	COMMENTS FILED
	AREA FACTORIAL,CODE, READONLY	
	ENTRY	; MARK FIRST INSTRUCTION TO EXECUTE
	MOV R1,#05	; COUNTER BIT FOR 5 16BIT ADDITION
	SUB R1,#01	; DECREMENTED BY 1 BECAUSE WE ADD ONLY 4 TIME
	MOV R0,#0X40000000	R0 POINTING TO 0X40000000 MEMORY LOCATION
	LDRH R2,[R0]	; LODING HALF WORD POINTED BT RO TO R2
UP	ADD R0,R0,#2	; MEMORY POINTER INCREMENTED BY 2
	LDRH R3,[R0]	; SECOND 16BIT NUMBER IS LOADED TO R3
	ADD R2,R2,R3	; ADDITION IS DONE
	SUBS R1,#01	; DECREMENTS COUNTER BIT FOR NUMBER OF ADDITION
	BNE UP	; IF COUNTER≠0 THE EXECUTION JUMPS TO THE LABEL 'UP'
	MOV R0,#0X40000020	; MEMORY LOCATION WHERE RESULT SHOULD BE SAVED
	STR R2,[R0]	; STORING OF RESULT
HERE	B HERE	
	END	; MARK END OF FILE



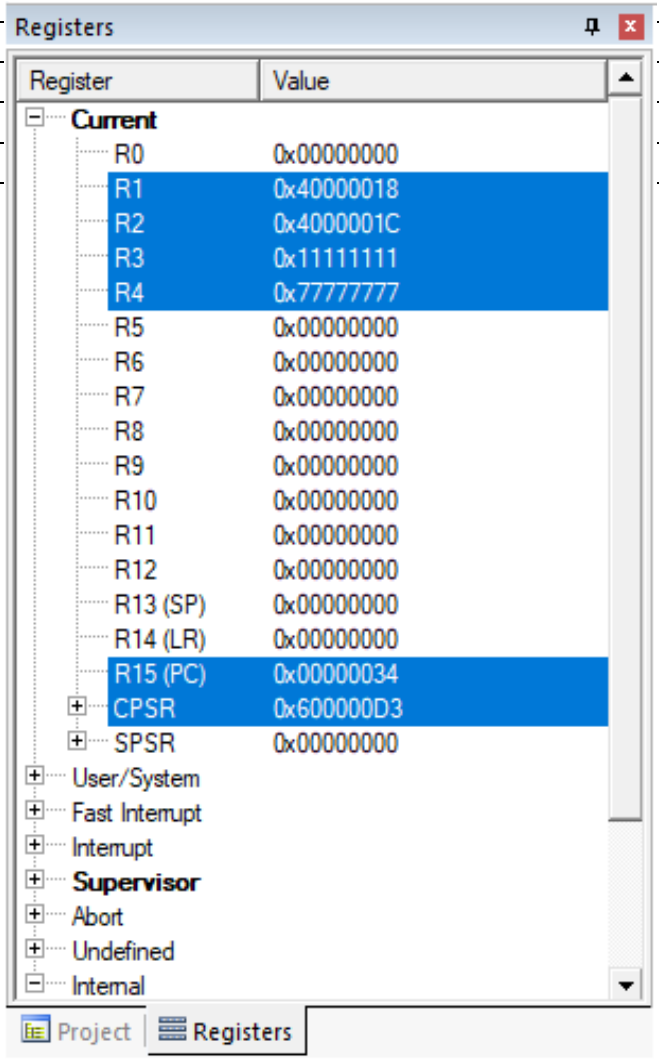
5. ALP TO FIND THE SQUARE OF A NUMBER (1 TO 10) USING LOOK-UP TABLE.

LABEL FIELD	MNEMONIC FIELD	COMMENTS FILED
	AREA SQUARE,CODE, READONLY	
	ENTRY	; MARK FIRST INSTRUCTION TO EXECUTE
	LDR R0, =DATATABLE	; Load start address of Lookup table
	LDR R1,VALUE	; Load no whose square is to be find
	CMP R1,#1	; check whether it is one
	BEQ LOOP1	; if one then return the first address value as square(1)=1
LOOP2	ADD R0,R0,#4	; if not increment the address location
	SUBS R1,R1,#1	; decrement the value of the number
	CMP R1,#1	; check whether it is 1, idea here is to check for the location
	BNE LOOP2	; if not one then go to loop2 else for loop1
LOOP1	LDR R2,[R0]	; store the result in r2
HERE	B HERE	; infinite loop
	END	; MARK END OF FILE
	;Lookup table contains Squares of no's from 0 to 10 (in hex)	
	AREA DATATABLE, DATA, READONLY	
	DCD 0X00000001;	
	DCD 0X00000004;	
	DCD 0X00000009;	
	DCD 0X00000016;	
	DCD 0X00000025;	
	DCD 0X00000036;	
	DCD 0X00000049;	
	DCD 0X00000064;	
	DCD 0X00000081;	
	DCD 0X00000100;	
	VALUE DCB 5	
	ALIGN	
	RESULT DCW 0	
	END	



6. ALP TO FIND THE LARGEST/SMALLEST NUMBER IN AN ARRAY OF 32 NUMBERS.

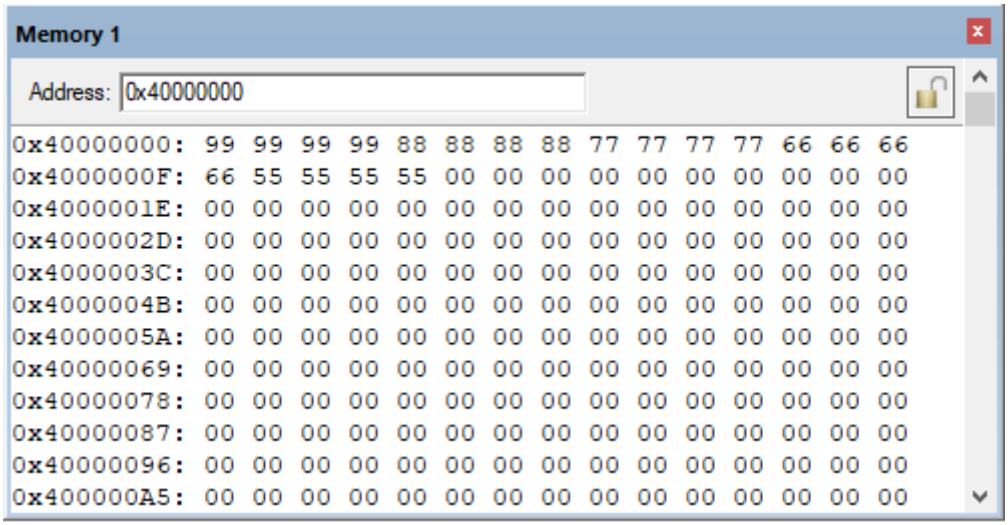
LABEL FIELD	MNEMONIC FIELD	COMMENT FIELD
	AREA LAR_SMAL, READONLY	
	ENTRY	
	MOV R5,#06	; COUNTER VALUE E.G 7 NUMBERS
	MOV R1,#0X40000000	; START OF THE DATA MEMORY
	MOV R2,#0X4000001C	; RESULT LOCATION
	LDR R3,[R1]	; GET THE FIRST DATA
	MOV R2,#0X4000001C	; RESULT LOCATION
	LDR R3,[R1]	; GET THE FIRST DATA
LOOP	ADD R1,R1,#04	; MEMORY POINTER UPDATED TO FETCH 2ND DATA
	LDR R4,[R1]	; GET SECOND NUMBER
	CMP R3,R4	; COMPARE BOTH NUMBERS
	BLS LOOP1	;BHI → for large; IF 1ST> 2ND THAN LOOP1
	MOV R3,R4	
LOOP1	SUBS R5,R5,#01	; DECREMENT THE COUNTER
	CMP R5,#00	
	BNE LOOP	
	STR R3,[R2]	
STOP	B STOP	
	END	



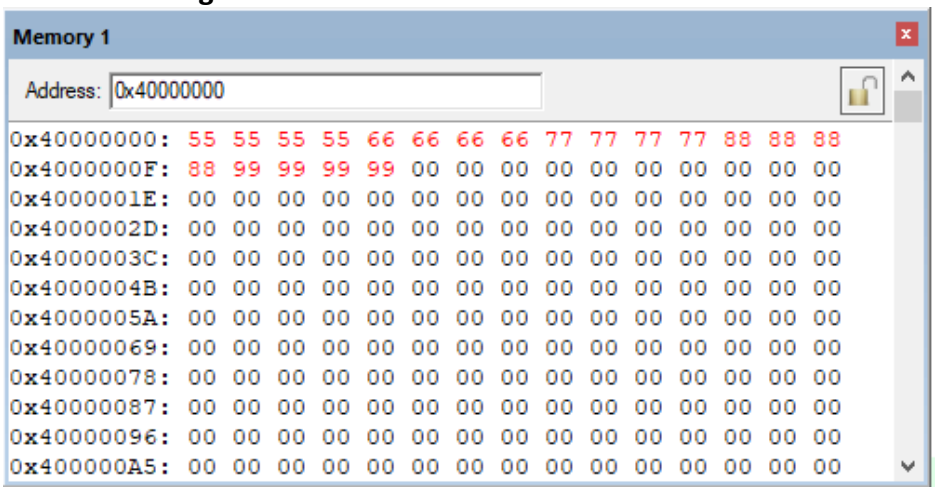
7. ALP TO ARRANGE A SERIES OF 32 BIT NUMBERS IN ASCENDING/DESCENDING ORDER.

LABEL FIELD	MNEMONIC FIELD	COMMENT FIELD
	AREA ASCENDING , CODE, READONLY	
	ENTRY	
	MOV R0,#05	; OUTER LOOP
OUTTERLOOP	MOV R5,#0X40000000	; DATA ADDRESS
	ADD R6,R5,#4	; INC TO CMP WITH NEXT DATA
	MOV R3,#4	; INNER LOOP
INNERLOOP	LDR R1,[R5]	; GET 1ST DATA
	LDR R2,[R6]	; GET 2ND DATA
	CMP R1,R2	; COMPARE 2 NO'S
	BLO LOOP3	; IF 1>2 THEN NO NEED TO EXCHANGE; BHI
	MOV R4,R2	; IF 1<2 THEN EXCHANGE
	MOV R2,R1	
	MOV R1,R4	
LOOP3	STR R1,[R5]	
	STR R2,[R6]	
	ADD R5,R5,#04	; INC 4 TIMES TO GET NEXT DATA FOR CMP
	ADD R6,R6,#04	
	SUBS R3,R3,#01	; DECREMENT INNER LOOP
	BNE INNERLOOP	
	SUBS R0,R0,#1	
	BNE OUTTERLOOP	; DECREMENT OUTTER LOOP
STOP	B STOP	
	END	

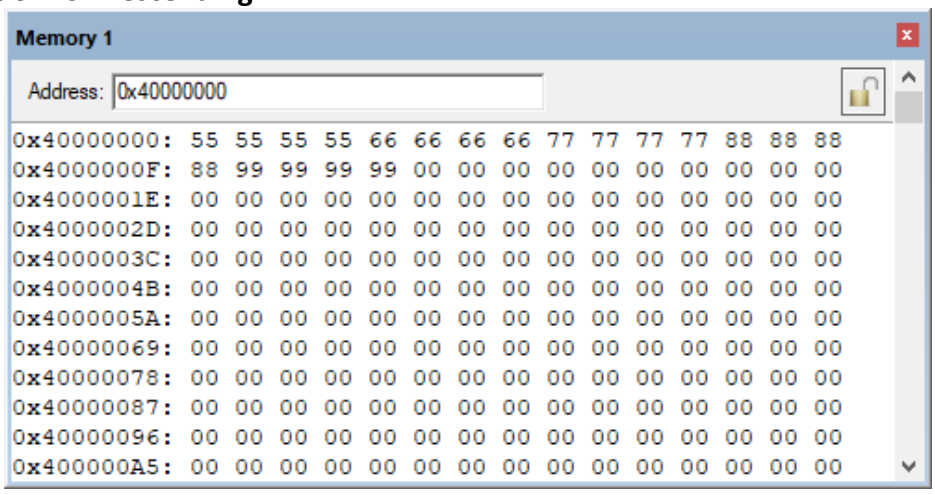
Before Sorting for ascending:



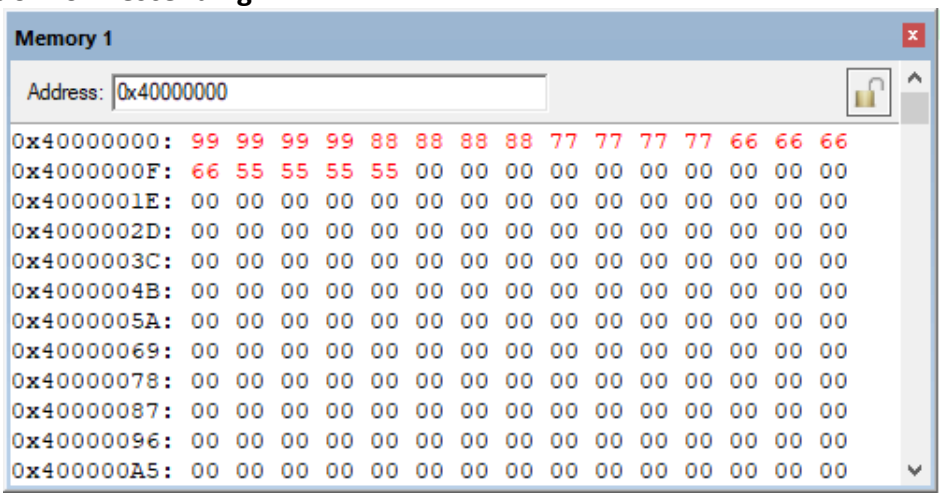
After excution for ascending:



Before Exection for Descending:



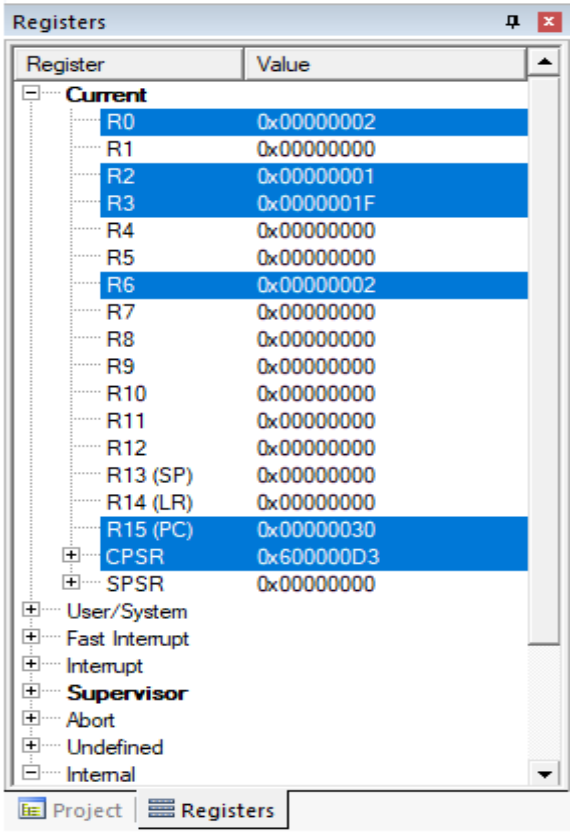
After Execution for Descending:



NOTE: BLO instruction for Descending
BHI instruction for Ascending

8. ALP TO COUNT THE NUMBER OF ONES AND ZEROS IN TWO CONSECUTIVE MEMORY LOCATIONS.

LABEL FIELD	MNEMONIC FIELD	COMMENT FIELD
	AREA ONEZERO , CODE, READONLY	
	ENTRY	;MARK FIRST INSTRUCTION TO EXECUTE
	MOV R2,#0	; COUNTER FOR ONES
	MOV R3,#0	; COUNTER FOR ZEROS
	MOV R6,#0X00000002	; LOADS THE VALUE
	MOV R1,#32	; 32 BITS COUNTER
	MOV R0,R6	; GET THE 32 BIT VALUE
	MOV R0,R6	; GET THE 32 BIT VALUE
LOOP0	MOVS R0,R0,ROR #1	; RIGHT SHIFT TO CHECK CARRY BIT (1'S/0'S)
	BHI ONES	; IF C=1 GOTO ONES BRANCH OTHERWISE NEXT
ZEROS	ADD R3,R3,#1	; IF C= 0 THEN INCREMENT THE COUNTER BY 1(R3)
	B LOOP1	; BRANCH TO LOOP1
ONES	ADD R2,R2,#1	; IF C=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
STOP	B STOP	
	END	



PART B

9. Display “Hello World” message using Internal UART.

```
INCLUDE <LPC21XX.H>          /* LPC21XX DEFINITIONS */
#INCLUDE "SERIAL.H"
VOID DELAY_MS(INT COUNT)     /*DELAY FUNCTION*/
{
   INT J=0,I=0;
   FOR(J=0;J<COUNT;J++)
   {
     FOR(I=0;I<35;I++);
   }
}
INT MAIN (VOID)
{
   UART0_INIT();                                          // INITIALIZE UART0
   DELAY_MS(100000);
   WHILE (1)
   {
     UART0_PUTS ("\N\RHELLO WORLD\N\R");
    DELAY_MS(1000000);
  }
}
```

10. Interface and Control a DC Motor.

```
#INCLUDE<LPC214X.H>
VOID DELAY(UNSIGNED INT X)
{
        UNSIGNED INT I,J;
        FOR(I=0;I<X;I++)
        FOR(J=0;J<1275;J++)
      ;
}
VOID MAIN()
{
        IODIR1|= 0XC0000000;  // TO ENABLE PORT1 PIN 30 & 31
        WHILE(1)
        {
         IOSET1=0X80000000;  // PUT 1 ON PIN 31(IN2) FOR CLOCKWISE
         DELAY(5000);
         IOCLR1=0XC0000000;
         DELAY(5000);
  IOSET1=0X40000000; // PUT 1 ON PIN 30(IN1) FOR ANTICLOCKWISE
         DELAY(5000);
         IOCLR1=0XC0000000;
         DELAY(5000);
         }
 }
```

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

```
#INCLUDE<LPC214X.H>
VOID DELAY(UNSIGNED INT X)
{
        UNSIGNED INT I,J;
        FOR(I=0;I<X;I++)
        FOR(J=0;J<1275;J++)
         ;
}
VOID MAIN()
{
        INT I;
 IODIR0|=0X000F0000;
 IOCLR0=0X000F0000;
 FOR(I=0;I<50;I++)
        {
         IOCLR0=0X000F0000;
         IOSET0=0X00080000; // PIN16=1 TO APPLY PULSE TO I1
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00040000; // PIN17=1 TO APPLY PULSE TO I2
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00020000; // PIN18=1 TO APPLY PULSE TO I3
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00010000; // PIN19=1 TO APPLY PULSE TO I4
         DELAY(100);
        }
        FOR(I=0;I<50;I++)
        {

         IOCLR0=0X000F0000;
         IOSET0=0X00010000; // PIN16=1 TO APPLY PULSE TO I1
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00020000; // PIN17=1 TO APPLY PULSE TO I2
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00040000; // PIN18=1 TO APPLY PULSE TO I3
         DELAY(100);
         IOCLR0=0X000F0000;
         IOSET0=0X00080000; // PIN19=1 TO APPLY PULSE TO I4
         DELAY(100);
        }
}
```

12. Determine Digital output for a given Analog input using Internal ADC of ARM controller.

```
#INCLUDE<LPC214X.H>
#DEFINE RS 0X00400000
#DEFINE RW 0X20000000
#DEFINE EN 0X10000000
UNSIGNED INT RESULT;
FLOAT VOLTAGE;
CHAR VOLT[18];
VOID DELAY(UNSIGNED INT X)
{
        UNSIGNED INT I,J;
        FOR(I=0;I<X;I++)
        FOR(J=0;J<1275;J++)
        ;
}
VOID CMD( CHAR C)
{
  IOCLR0=0X00003FC0;
  IOSET0=C<<6;
  IOCLR0=RW;
  IOCLR0=RS;
  IOSET0=EN;
  DELAY(100);
  IOCLR0=EN;
}
VOID DATA( CHAR C)
{
  IOCLR0=0X00003FC0;
  IOSET0=C<<6;
  IOCLR0=RW;
  IOSET0=RS;
  IOSET0=EN;
  DELAY(100);
  IOCLR0=EN;
}
VOID LCD_STR(CHAR  *S)
{
   WHILE(*S)
  {
    DATA(*S);
    S++;
   DELAY(20);
  }
}
VOID ADC_INIT()
{
        AD0CR=0X00210308;
        PINSEL1=0X10000000;
}
```

```
VOID DISPLAY(UNSIGNED INT N)
{
   IF(N==0)
     DATA(N+0X30);
  IF(N)
  {
     DISPLAY(N/10);
     DATA((N%10)+0X30);
   }
}
VOID INIT()
{
            CMD(0X38);
            CMD(0X0E);
            CMD(0X80); // STARTING ADDRESS OF THE FIRST LINE
         CMD(0X01);
}
VOID MAIN()
{
           IODIR0|=0X30403FC0;
           INIT();
           ADC_INIT();
           WHILE(1)
           {
          CMD(0X01);
             WHILE(AD0DR3 & (0X80000000)==0);
           RESULT=(AD0DR3 & (0X3FF <<6)); // TO STORE DATA IN RESULT BITS(6-15)
           RESULT=RESULT >> 6; //TO PUSH THE RESULTS TO DATA BITS
           LCD_STR("ADC:");
       CMD(0X84);
       DISPLAY(RESULT);
       VOLTAGE = ((RESULT/1023.0)*3.3); //VOLTAGE WILL HAVE FLOAT VALUES
       SPRINTF(VOLT,"VOLTAGE:%.2F V",VOLTAGE);
       CMD(0XC0);
       LCD_STR(VOLT);
      DELAY(1000);
    }
}
```

13. Interface a DAC and generate Triangular and Square waveforms.

```
/* TRIANGLE WAVE */
#INCLUDE "LPC214X.H"
UNSIGNED INT  VALUE;
INT MAIN()
{
    PINSEL1|=0X00080000;
    WHILE(1)
   {
     VALUE = 0;
     WHILE ( VALUE != 1023 )
     {
          DACR = ( (1<<16) | (VALUE<<6) );
          VALUE++;
     }
    WHILE ( VALUE != 0 )
    {
      DACR = ( (1<<16) | (VALUE<<6) );
      VALUE--;
    }
  }
}
/* SQUARE WAVE /*
#INCLUDE "LPC214X.H"
UNSIGNED INT  RESULT=0X00000040,VAL;
INT MAIN()
{
   PINSEL1|=0X00080000;
   WHILE(1)
  {
     WHILE(1)
    {
          VAL =0XFFFFFFFF;
          DACR=VAL;
        {
          BREAK;
        }
    }
   WHILE(1)
   {
     VAL =0X00000000;
    DACR=VAL;
     {
      BREAK;
     }
   }
  }
}
```

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

```
#include <LPC214x.H>                    /* LPC214x definitions */
#include "lcd.h"
// Matrix Keypad Scanning Routine
// COL1 COL2 COL3 COL4
// 0   1   2   3   ROW 1
// 4   5   6   7   ROW 2
// 8   9   A   B   ROW 3
// C   D   E   F   ROW 4
#define SEG7_CTRL_DIR              IO0DIR
#define SEG7_CTRL_SET      IO0SET
#define SEG7_CTRL_CLR      IO0CLR
#define COL1            (1 << 16)
#define COL2            (1 << 17)
#define COL3            (1 << 18)
#define COL4            (1 << 19)
#define ROW1            (1 << 20)
#define ROW2            (1 << 21)
#define ROW3            (1 << 22)
#define ROW4            (1 << 23)
#define COLMASK                         (COL1 | COL2 | COL3 | COL4)
#define ROWMASK                         (ROW1 | ROW2 | ROW3 | ROW4)
#define KEY_CTRL_DIR    IO1DIR
#define KEY_CTRL_SET      IO1SET
#define KEY_CTRL_CLR      IO1CLR
#define KEY_CTRL_PIN      IO1PIN
/////////////// COLUMN WRITE //////////////////////
Void col_write( unsigned char data )
{
  Unsigned int temp=0;
   Temp=(data << 16) & COLMASK;
  KEY_CTRL_CLR |= COLMASK;
  KEY_CTRL_SET |= temp;
}
//////////////////////////////// MAIN  ////////////////////////////////////
Int main (void)
{
Unsigned char key, i;
Unsigned char rval[] = {0x7,0xb,0xd,0xe,0x0};
Unsigned char keypadmatrix[] =
{
   '4','8','B','F',
   '3','7','A','E',
   '2','6','0','D',
   '1','5','9','C'
};
  Init_lcd();
  KEY_CTRL_DIR |= COLMASK;    //Set cols as Outputs
  KEY_CTRL_DIR &= ~(ROWMASK); //  Set ROW lines as Inputs
```

```
Lcd_putstring16(0,"Press HEX Keys..");
Lcd_putstring16(1,"Key Pressed =   ");
While (1)
{
  Key = 0;
  For( i = 0; i < 4; i++ )
  {
    // turn on COL output one by one
                Col_write(rval[i]);
     // read rows - break when key press detected
    If (!(KEY_CTRL_PIN & ROW1))
      Break;
    Key++;
    If (!(KEY_CTRL_PIN & ROW2))
      Break;
    Key++;
    If (!(KEY_CTRL_PIN & ROW3))
      Break;
    Key++;
                If (!(KEY_CTRL_PIN & ROW4))
      Break;
    Key++;
  }
        If (key == 0x10)
                Lcd_putstring16(1,"Key Pressed =   ");
        Else
                {       lcd_gotoxy(1,14);
                        Lcd_putchar(keypadmatrix[key]);
                }
}
}
```

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

```
#include <LPC214x.H>
Int i;
__irq void Ext_ISR(void) // Interrupt Service Routine-ISR
//The _irq keyword tells the compiler that the function is an interrupt routine
{
        IO1DIR |= 0x00010000;
        IO1CLR |=  0x00010000;
        For(i=0; i<3000000;i++);
        IO1SET |=  0x00010000;
        EXTINT |= 0x4;              //clear interrupt
        Vicvectaddr = 0;            // End of interrupt execution
}
Void init_ext_interrupt()  // Initialize Interrupt
{
 EXTMODE = 0x4;                    //Edge sensitive mode on EINT2
 EXTPOLAR &= ~(0x4);  //Falling Edge Sensitive
 PINSEL0 = 0x80000000; //Select Pin function P0.15 as EINT2
 /* initialize the interrupt vector */
 Vicintselect &= ~ (1<<16);        // EINT2 selected as IRQ 16
 Vicvectaddr5 = (unsigned int)Ext_ISR; // address of the ISR
 Vicvectcntl5 = (1<<5) | 16;
//  Basically Vector Address Register store the address of the function i.e. ISR and used to assign or
enable vector IRQ slot..Pointer Interrupt Function (ISR)
 Vicintenable = (1<<16);           // EINT2 interrupt enabled
 EXTINT &= (0x4);
}
Int main (void)
{
  Init_ext_interrupt();  // initialize the external interrupt
   While(1);
}
```

16. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.

```
#include <LPC214x.H>
Void delay_led(unsigned long int);
Int main(void)
{
  IO0DIR = 0x000007fc;
  While(1)
 {
  IO0CLR = 0x00000fff;
 IO0SET = 0x00000604;
 Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x000007e4;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000648;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000618;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000730;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000690;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000680;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x0000063c;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000600;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000630;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000620;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x00000780;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x000006c4;
Delay_led(150000);
IO0CLR = 0x00000fff;
```

```
IO0SET = 0x00000708;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x000006c0;
Delay_led(150000);
IO0CLR = 0x00000fff;
IO0SET = 0x000006e0;
Delay_led(150000);
IO0CLR = 0x00000fff;
  }
}
Void delay_led(unsigned long int count1)
{
   While(count1 > 0) {count1--;}
}
```