QMP 7.1 D/F

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Artificial Intelligence and Data Science

MICROCONTROLLER AND EMBEDDED SYSTEMS

PRACTICAL COMPONENT OF IPCC

(Academic year 2022 -2023)

SEMESTER – IV

21CS43

Lab Manual

Name :____

USN :____

Batch :___________________Section : _______________

CHANNABASAVESHWARA INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

LABORATORY MANNUAL

Microcontroller and Embedded Systems Laboratory/21CS43

(Effective from the academic year 2022 -2023)

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SYLLABUS

MICROCONTROLLER AND EMBEDDED SYSTEMS [PRACTICAL COMPONENT OF IPCC]

SEMESTER – IV

Subject Code: 21CS43	CIE Marks: 50
Number of Lecture Hours/Week: 03 L + 02 P	SEE Marks: 50
Total Number of Contact Hours: 40T +20 P	Exam Hours: 03

CREDITS – 04

Course Learning Objectives: This course (21CS43) will enable students to:

CLO 1: Understand the fundamentals of ARM-based systems, including programming modules with registers and the CPSR.

CLO 2: Use the various instructions to program the ARM controller.

CLO 3: Program various embedded components using the embedded C program.

CLO 4: Identify various components, their purpose, and their application to the embedded system's applicability.

CLO 5: Understand the embedded system's real-time operating system and its application in IoT.

Programs List:

PART A

Conduct the following experiments by writing program using ARM7TDMI/LPC2148 using an evaluation board/simulator andsoftware tool.

.

1. Sample Programs using Keil Compiler
2. Write a program to find the sum of the first 10 integer numbers.
3. Write a program to find the factorial of a number.
4. Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM.
5. Write a program to find the square of a number (1 to 10) using a look-up table.
6. Write a program to find the largest or smallest number in an array of 32 numbers.
7. Write a program to arrange a series of 32 bit numbers in ascending/descending order.
8. Write a program to count the number of ones and zeros in two consecutive memory locations.
9. Display “Hello World” message using Internal UART.

PART –B

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

1. Interface and Control a DC Motor.
2. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.
3. Determine Digital output for a given Analog input using Internal ADC of ARM controller.
4. Interface a DAC and generate Triangular and Square waveforms.
5. Interface a 4x4 keyboard and display the key code on an LCD.
6. Demonstrate the use of an external interrupt to toggle an LED On/Off.
7. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.
8. Demonstration of IoT applications by using Arduino and Raspberry Pi.

Course Outcomes

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

CO 1. Explain C-Compilers and optimization

CO 2. Describe the ARM microcontroller's architectural features and program module.

CO 3. Apply the knowledge gained from programming on ARM to different applications.

CO 4. Program the basic hardware components and their application selection method.

CO 5. Demonstrate the need for a real-time operating system for embedded system applications.

Graduate Attributes

- Engineering Knowledge
- Problem Analysis
- Modern Tool Usage
- Conduct Investigations of Complex Problems
- Design/Development of Solutions

ARM7 based LPC2148 Microcontroller

The full form of an ARM is an advanced reduced instruction set computer (RISC) machine, and it is a 32-bit processor architecture expanded by ARM holdings. The applications of an ARM processor include several microcontrollers as well as processors. The architecture of an ARM processor was licensed by many corporations for designing ARM processor-based SoC products and CPUs.

LPC2148 Microcontroller

The LPC2148 microcontroller is designed by Philips (NXP Semiconductor) with several in-built features & peripherals. Due to these reasons, it will make more reliable as well as the efficient option for an application developer. LPC2148 is a 16-bit or 32-bit microcontroller based on ARM7 family.

Programmer's Model

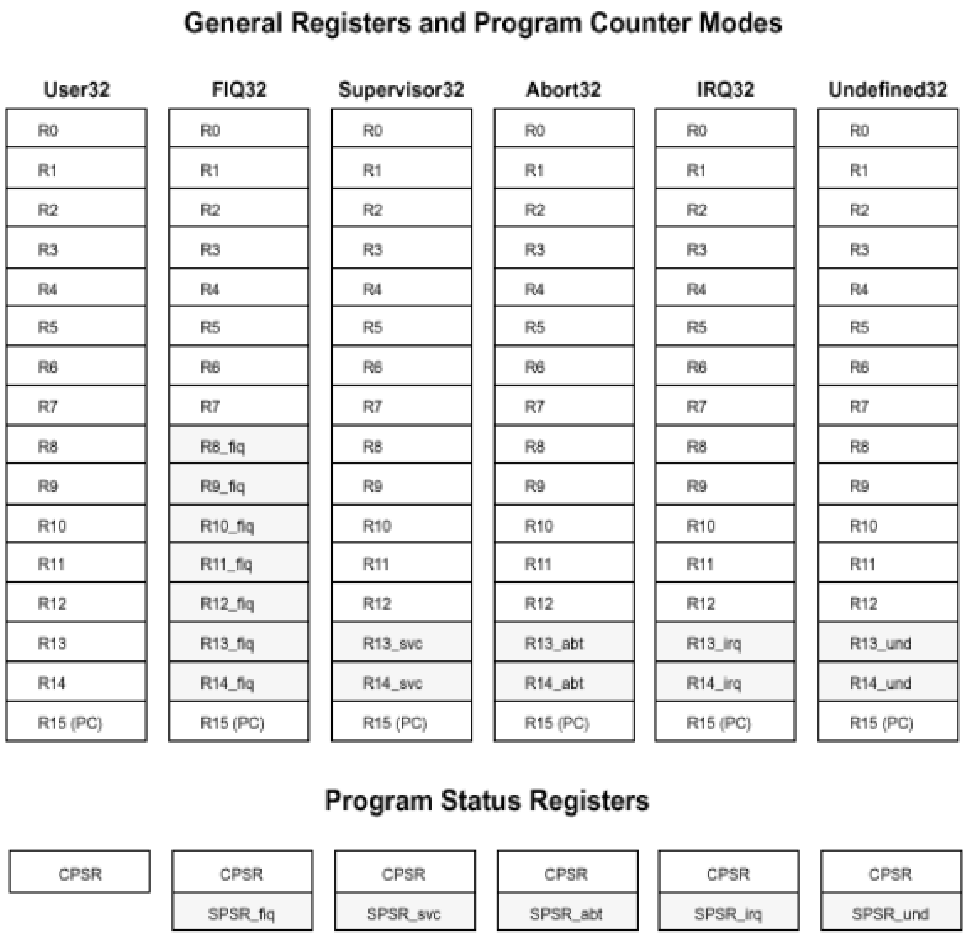
ARM has a 32-bit data bus and a 32-bit address bus. The data types the processor supports are Words (32 bits), where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations (e.g. ADD) are only performed on word quantities. Load and store operations can transfer words.

Registers

The processor has a total of 37 registers made up of 31 general 32 bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode and the other registers (the banked registers) are switched in to support IRQ, FIQ, Supervisor, Abort and undefined mode processing. The register bank organization is shown in below figure. The banked registers are shaded in the diagram.

In all modes 16 registers, R0 to R15, are directly accessible. All registers except R15 are general purpose and may be used to hold data or address values. Register R15 holds the Program Counter (PC). When R15 is read, bits [1:0] are zero and bits [31:2] contain the PC. A seventeenth register (the CPSR - Current Program Status Register) is also accessible. It contains condition code flags and the current mode bits and may be thought of as an extension to the PC. R14 is used as the subroutine link register and receives a copy of R15 when a Branch and

Link instruction is executed. It may be treated as a general- purpose register at all other times. R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.



General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Basic and Commonly used Instruction Set of ARM in programming

Data Processing Instructions

Move instructions
Arithmetic instructions
Logical instructions
Comparison instructions
Multiply instructions

MOV : move

MOV r0, r1; r0 = r1
MOV r0, #5; r0 = 5
MVN : move (negated)
MVN r0, r1; r0 = NOT (r1) =~ (r1)

Example 1

PRE: r5 = 5, r7 = 8;
MOV r7, r5, LSL #2; r7 = r5 << 2 = r5*4
POST: r5 = 5, r7 = 20

LSL: logical shift left

x << y, the least significant bits are filled with zeroes
LSR: logical shift right:
(unsigned) x >> y, the most significant bits are filled with zeroes

ASR: arithmetic shift right

(signed) x >> y, copy the sign bit to the most significant bit

ROR: rotate right

((unsigned) x >> y) | (x << (32-y))

RRX: rotate right extended

c flag <<31 | ((unsigned) x >> 1)
Performs 33-bit rotate, with the CPSR's C bit being inserted above sign bit of the word

Example 2

PRE: r0 = 0x00000000, r1 = 0x80000004
MOV r0, r1, LSL #1 ; r0 = r1 *2
POST r0 = 0x00000008, r1 = 0x80000004

Arithmetic Instructions

Syntax: <instruction> {<cond>} {S} Rd, Rn, N

N: a register or immediate value

ADD : add

ADD r0, r1, r2; r0 = r1 + r2

ADC : add with carry

ADC r0, r1, r2; r0 = r1 + r2 + C

SUB : subtract

SUB r0, r1, r2; r0 = r1 - r2

SBC : subtract with carry

SUC r0, r1, r2; r0 = r1 - r2 + C -1

RSB : reverse subtract

RSB r0, r1, r2; r0 = r2 – r1

RSC : reverse subtract with carry

RSC r0, r1, r2; r0 = r2 – r1 + C -1

MUL : multiply

MUL r0, r1, r2; r0 = r1 x r2

MLA : multiply and accumulate

MLA r0, r1, r2, r3; r0 = r1 x r2 + r3

Logical Operations

Syntax: <instruction> {<cond>} {S} Rd, RN, N

N: a register or immediate value

AND : Bit-wise and

ORR : Bit-wise or

EOR : Bit-wise exclusive-or

BIC : bit clear

BIC r0, r1, r2; r0 = r1 & Not (r2)

Example 3:

PRE: r1 = 0b1111, r2 = 0b0101

BIC r0, r1, r2; r0 = r1 AND (NOT (r2))

POST: r0=0b1010

Comparison Instructions

Compare or test a register with a 32-bit value Do not modify the registers being compared or tested But only set the values of the NZCV bits of the CPSR register. Do not need to apply to S suffix for comparison instruction to update the flags in CPSR register

Syntax: <instruction> {<cond>} {S} Rd, N

N: a register or immediate value

CMP: compare

CMP r0, r1; compute (r0 - r1) and set NZCV

CMN: negated compare

CMP r0, r1; compute (r0 + r1) and set NZCV
TST: bit-wise AND test
TST r0, r1; compute (r0 AND r1) and set NZCV
TEQ: bit-wise exclusive-or test
TEQ r0, r1; compute (r0 EOR r1) and set NZCV

Example 4
PRE: CPSR = nzcvqiFt_USER, r0 = 4, r9 = 4
CMP r0, r9
POST: CPSR = nZcvqiFt_USER

Multiply Instruction
Syntax:
MLA{<cond>} {S} Rd, Rm, Rs, Rn
MUL{<cond>} {S} Rd, Rm, Rs
MUL : multiply
MUL r0, r1, r2; r0 = r1*r2
MLA : multiply and accumulate
MLA r0, r1, r2, r3; r0 = (r1*r2) + r3

Syntax: <instruction>{<cond>} {S} RdLo, RdHi, Rm, Rs
Multiply onto a pair of register representing a 64-bit value
UMULL : unsigned multiply long
UMULL r0, r1, r2, r3; [r1,r0] = r2*r3
UMLAL : unsigned multiply accumulate long
UMLAL r0, r1, r2, r3; [r1,r0] = [r1,r0]+(r2*r3)
SMULL: signed multiply long
SMULL r0, r1, r2, r3; [r1,r0] = r2*r3
SMLAL : signed multiply accumulate long
SMLAL r0, r1, r2, r3; [r1,r0] = [r1,r0]+(r2*r3)
Branch Instructions (Cont.)
Syntax
B{<cond>} lable
BL{<cond>} lable
B : branch
B label; pc (program counter) = label Used to change execution flow
BL : branch and link
BL label; pc = label, lr = address of the next address after the BL
Similar to the B instruction but can be used for subroutine
Call Overwrite the link register (lr) with a return address

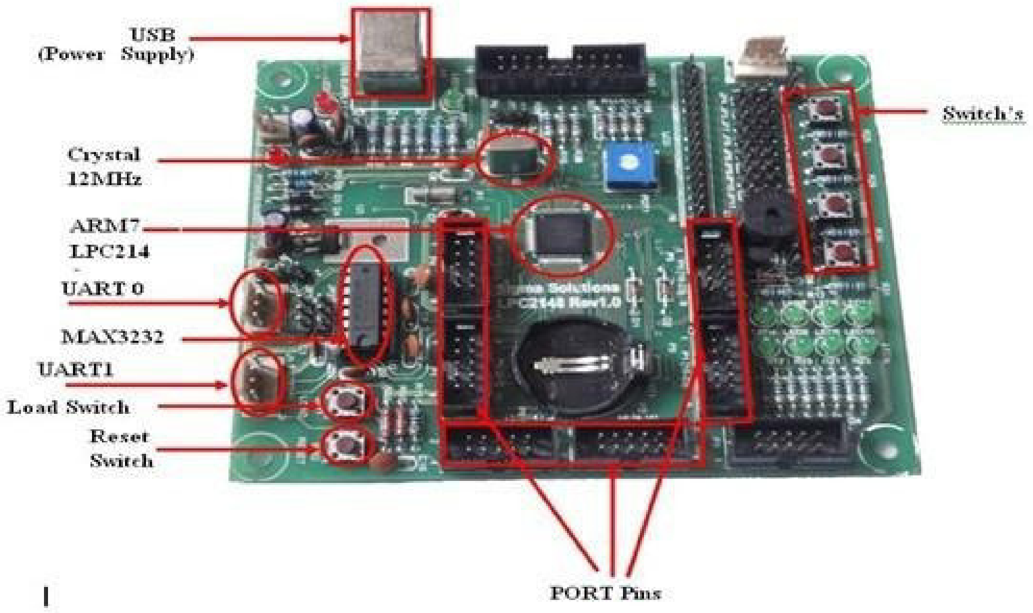
Example 5
B forward
    ADD r1, r2, #4
    ADD r0, r6, #2
    ADD r3, r7, #4
Forward
    SUB r1, r2, #4
Backward
SUB r1, r2, #4
    B backward

Load-Store Instructions
    Transfer data between memory and processor registers
    Three types
    Single-register transfer
    Multiple-register transfer
    Swap

Moving a single data item in and out of register Data item can be
A word (32-bits), Halfword (16-bits), Bytes (8-bits)
Syntax
    <LDR|STR>{<cond>}{B} Rd, addressing1
    LDR{<cond>}SB|H|SH Rd, addressing2
    STR{<cond>} H Rd, addressing2
    LDR : load word into a register from memory
    LDRB : load byte
    LDRSB : load signed byte
    LDRH : load half-word
    LSRSH : load signed halfword
    STR: store word from a register to memory
    STRB : store byte
    STRH : store half-word
Example 7
LDR r0, [r1] ;= LDR r0, [r1, #0] ;r0 = mem32[r1]
STR r0, [r1] ;= STR r0, [r1, #0] ;mem32[r1]= r0 Register r1 is called the
base address register

ARM LPC 2148 FEATURES:

- 16-bit/32-bit ARM7TDMI-S Microcontroller.
- 40 kB of on-chip static RAM and 512 kB of on-chip flash memory.
- In-System Programming/In-Application Programming (ISP/IAP) via on-chip boot loader software.
- Embedded ICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high-speed tracing of instruction execution.
- USB 2.0 Full-speed compliant device controller with 2 kB of endpoint RAM.
- Two 10-bit ADCs provide a total of 14 analog inputs
- 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)
- Watchdog Timer.
- Low power Real-Time Clock (RTC) with independent power and 32 kHz clock input.
- Multiple serial interfaces including two UARTs, two Fast I^{2}C-bus (400 kbit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored Interrupt Controller (VIC) with configurable priorities and vector addresses.
- 60 MHz maximum CPU clock available from programmable on-chip PLL with settling time of 100 us.
- On-chip integrated oscillator operates with an external crystal from 1 MHz to 25 MHz
- Power saving modes include Idle and Power-down.
- Individual enable/disable of peripheral functions as well as peripheral clock scaling for additional power optimization.



LPC 2148 TECHNICAL SPECIFICATIONS:

- Microcontroller: LPC2148 with 512K on chip memory
- Crystal for LPC2148: 12Mhz
- Crystal for RTC: 32.768KHz
- 6 – 10pin Berg headers for external interfacing(GPIOs)
- No separate programmer required (Program with Flash Magic usingon-chip boot loader)
- No Separate power adapter required (USB port as power source)
- 20pin(2X10) FRC JTAG connector for Programming and debugging
- 16 Pin Berg Header for LCD Interfacing
- Two RS-232 Interfaces (UART0 and UART1)
- Real-Time Clock with Battery Holder
- 1 Analog Potentiometer connected to ADC
- 4 USER Switches
- 8 USER LEDs
- Reset and Boot loader Switches
- On Board Buzzer Interface

HOW TO USE KEIL μVISION4
For ARM7 (LPC2148)

Step By Step

Keil is on the tool which is widely used in Industry, KEIL has tools for ARM, Cortex-M, Cortex-R, 8051, C166, and 251 processor families. In this article we are going to discuss KEIL tools for ARM. The development tools of for ARM include following...

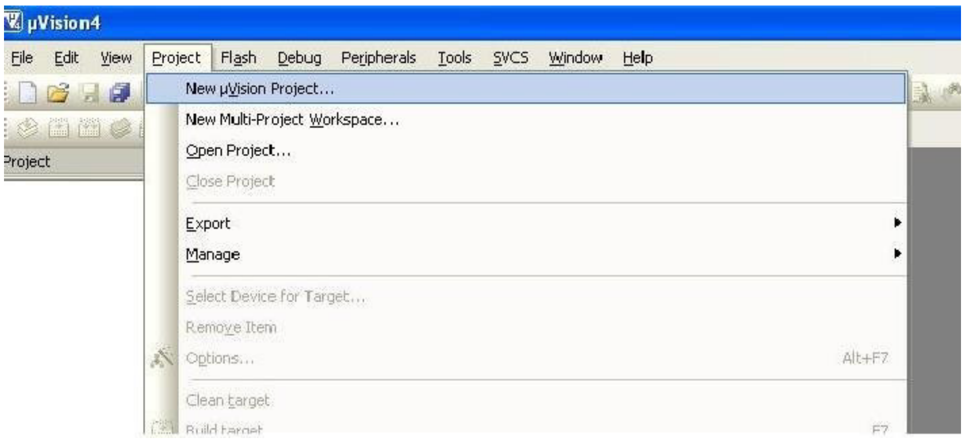
1. μvision IDE v4
2. Compiler for ARM (armcc)
3. MicroLib (C library)
4. Assembler for ARM (armasm)
5. Linker For ARM (armLink)

Step1: Click for KEIL μVISION4 Icon . Which appears after Installing Keil KEIL μVISION4. This will open uvison IDE.

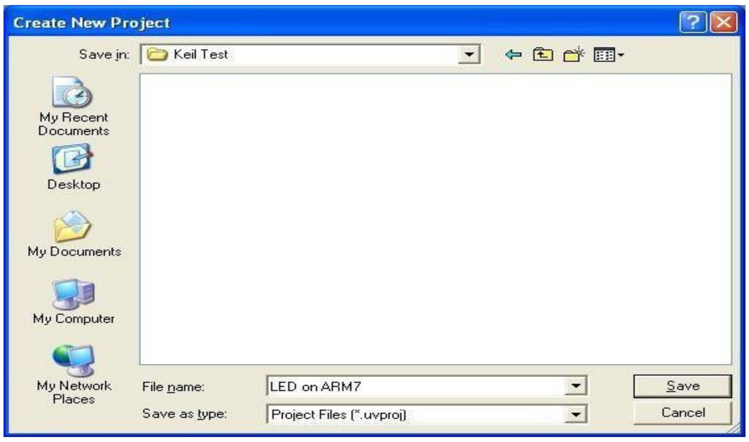
KEIL™
Tools by ARM
μVision®4
Integrated Development Environment
Copyright © 1997 - 2005 Keil Software, 2005 - 2009 ARM Ltd. All rights reserved.
This product is protected by US and international laws.

Keil Setup to Generate .Hex File

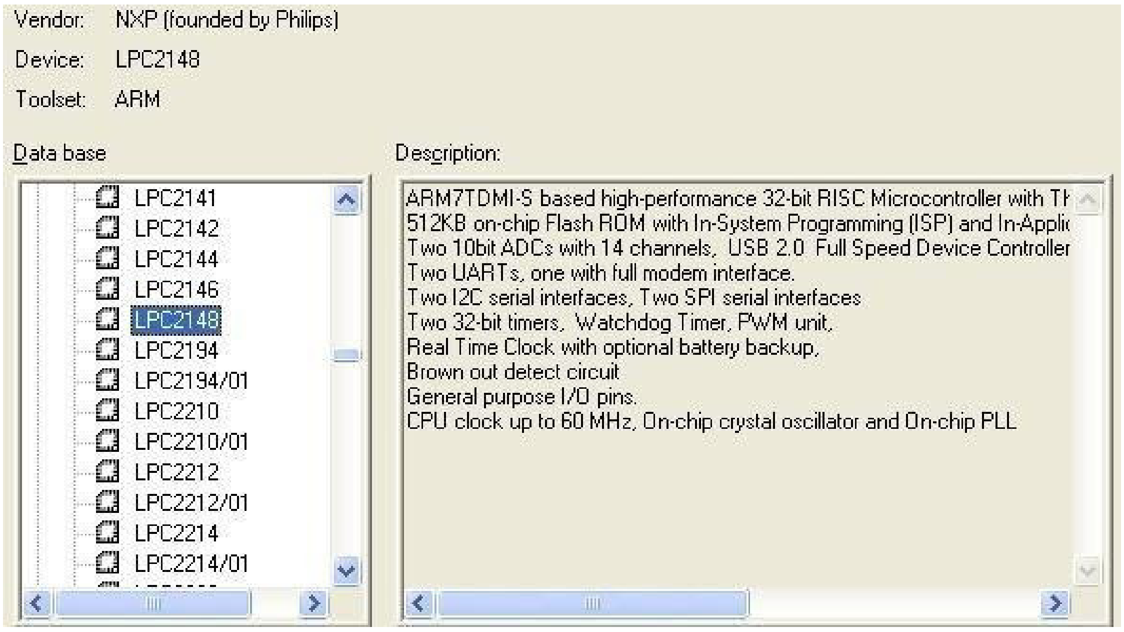
Step2: Click on Project Menu, Then New μVison Project.



Step3: Create New Project Folder named as "Keil Test".



Step 4: Select Target Device



Vendor: NXP (founded by Philips)
Device: LPC2148
Toolset: ARM

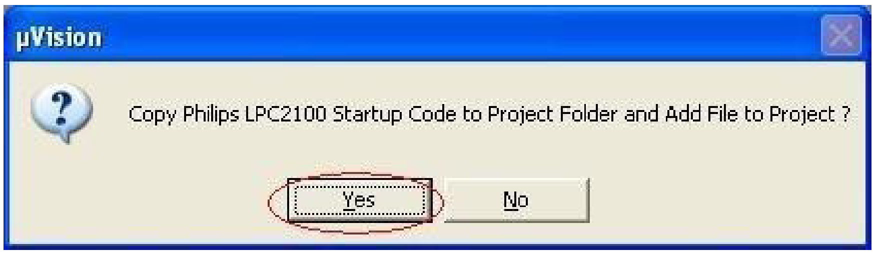
Data base

Description:

ARM7TDMI-S based high-performance 32-bit RISC Microcontroller with Th
512KB on-chip Flash ROM with In-System Programming (ISP) and In-Applic
Two 10bit ADCs with 14 channels, USB 2.0 Full Speed Device Controller
Two UARTs, one with full modem interface.
Two I2C serial interfaces, Two SPI serial interfaces
Two 32-bit timers, Watchdog Timer, PWM unit,
Real Time Clock with optional battery backup,
Brown out detect circuit
General purpose I/O pins.
CPU clock up to 60 MHz, On-chip crystal oscillator and On-chip PLL

Step 5: Then select specific chip i.e. LPC2148.

For ALP program, CLICK "NO", For "C" program click on "YES"

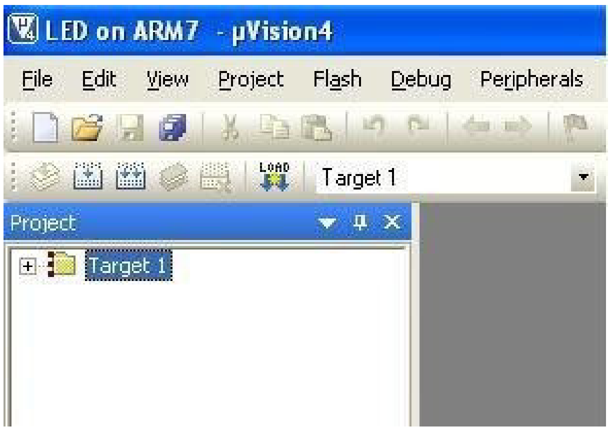


µVision

Copy Philips LPC2100 Startup Code to Project Folder and Add File to Project ?

[✓ Yes] No

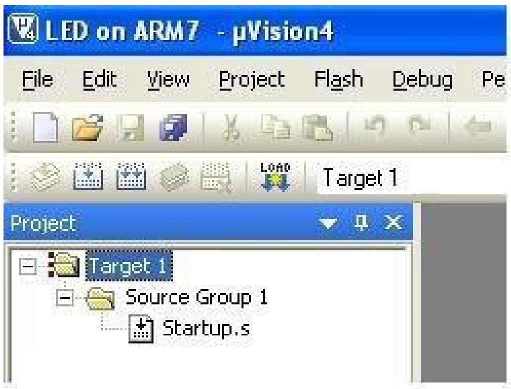
Step 6: Then you will see following window



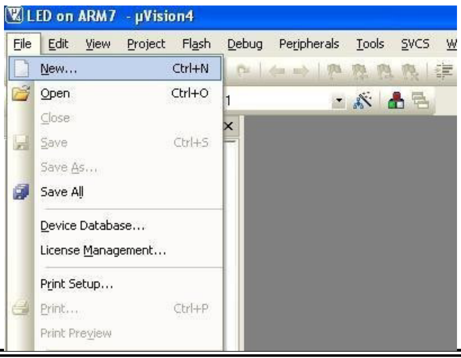
Step 7: Now you see Startup.s is already added which is necessary for running code for Keil.

Note: Code wills Not Run without Startup.s

Startup.s is available in C:\Keil\ARM\Startup\Philips.

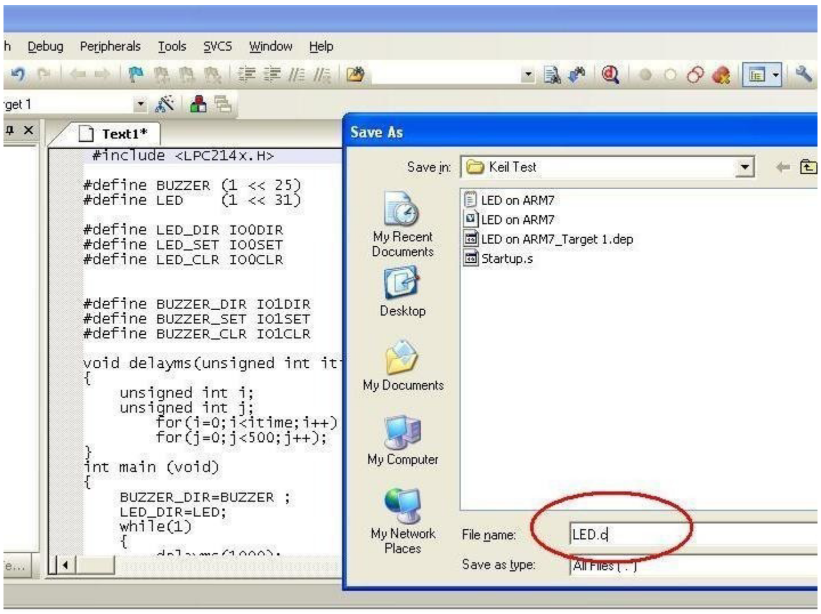


Step 8: Now Click on File Menu and Click on New.



Step 09: Write Code for Blink LED in C OR ASM and FileName.c/ASM Save.

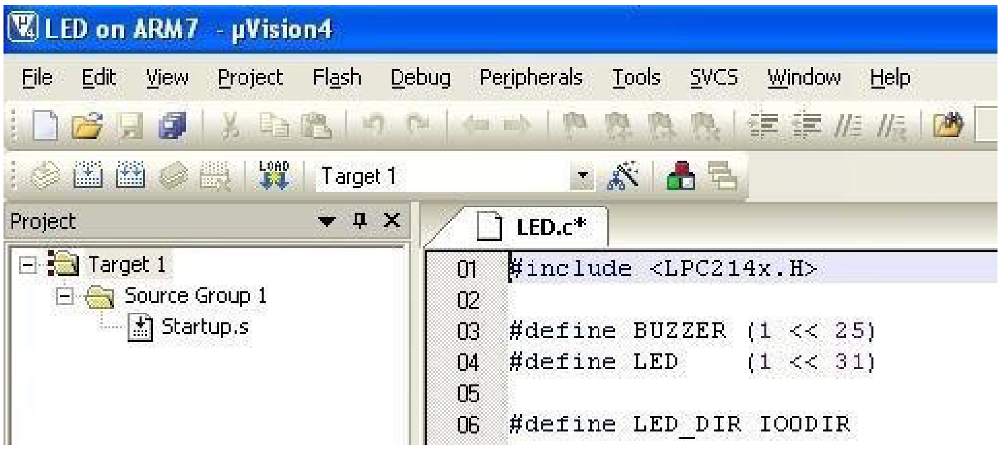
Note: Don’t forget to save .c/ASM Extension.



```
#include <LPC214x.H>

#define BUZZER (1 << 25)
#define LED    (1 << 31)

#define LED_DIR IO0DIR
#define LED_SET IO0SET
#define LED_CLR IO0CLR


#define BUZZER_DIR IO1DIR
#define BUZZER_SET IO1SET
#define BUZZER_CLR IO1CLR

void delayms(unsigned int it
{
    unsigned int i;
    unsigned int j;
        for(i=0;i<itime;i++)
        for(j=0;j<500;j++);
}
int main (void)
{
    BUZZER_DIR=BUZZER ;
    LED_DIR=LED;
    while(1)
    {
```

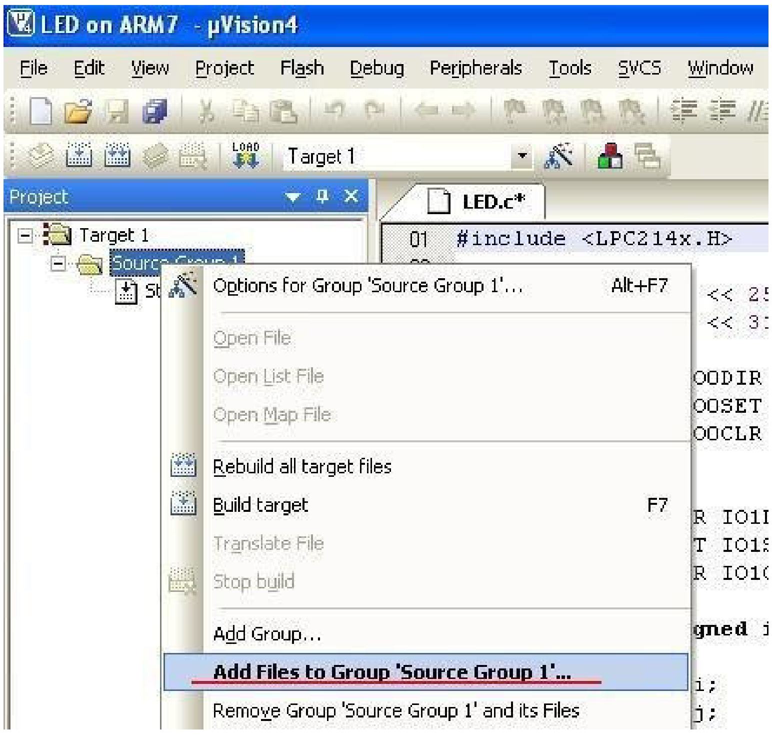
File name: LED.c

Step 10: Now you Window in C Syntax.



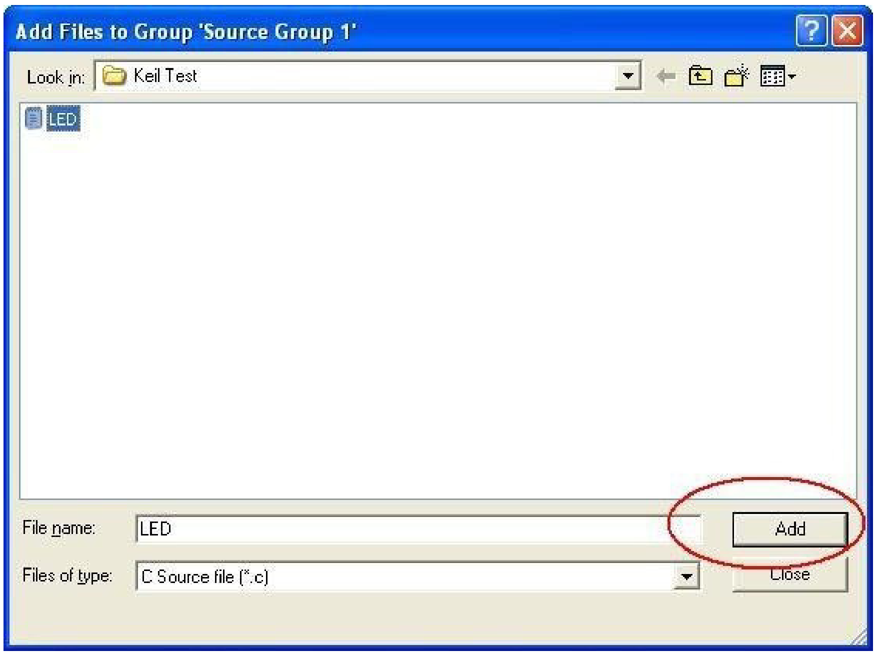
```
#include <LPC214x.H>

#define BUZZER (1 << 25)
#define LED    (1 << 31)

#define LED_DIR IO0DIR
```

Step 11: Now you add LED.c file by adding Source Group 1 Add files to Group 'Source Group 1'.


LED on ARM7 - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window
Target 1
Project
LED.c*
Target 1
Source Group 1
Options for Group 'Source Group 1'... Alt+F7
Open File
Open List File
Open Map File
Rebuild all target files
Build target F7
Translate File
Stop build
Add Group...
Add Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files

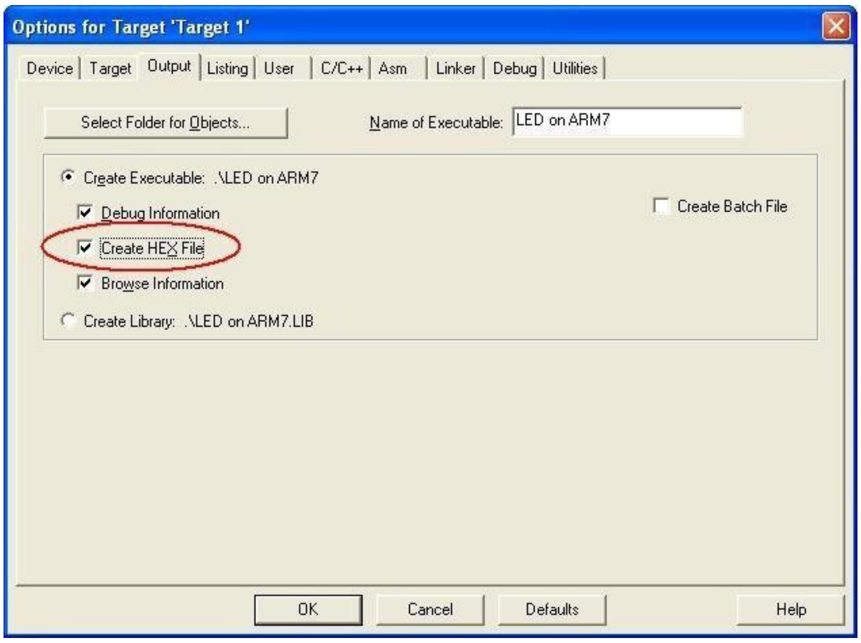
```
#include <LPC214x.H>
```

Step 12: Add LED.C file.


Add Files to Group 'Source Group 1'
Look in: Keil Test
LED
File name: LED
Files of type: C Source file (*.c)
Add
Close

Step 13: Now Click on Options for Target 'Target 1'.

Step 14: Go to Options for Target 'Target 1'. Click on Check Box Create HEX File.


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Select Folder for Objects...
OK
Cancel
Defaults
Help

Name of Executable: LED on ARM7

(●) Create Executable: .\LED on ARM7

[✓] Debug Information

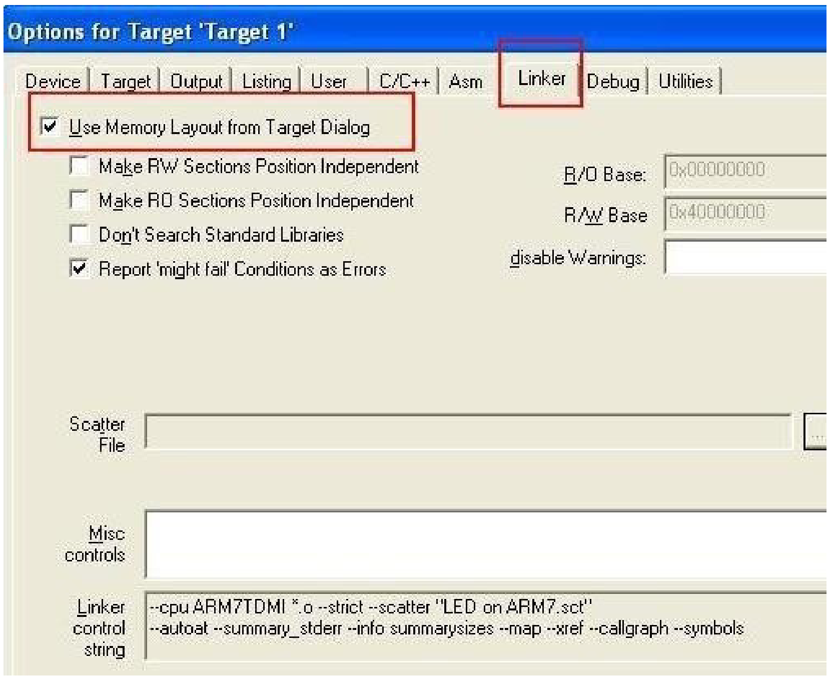
[✓] Create HEX File

[✓] Browse Information

( ) Create Library: .\LED on ARM7.LIB

[ ] Create Batch File

Step 15: Then go to Linker. Click on Use Memory Layout for Target Dialog.


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

[✓] Use Memory Layout from Target Dialog

[ ] Make RW Sections Position Independent

[ ] Make RO Sections Position Independent

[ ] Don't Search Standard Libraries

[✓] Report 'might fail' Conditions as Errors

R/O Base: 0x00000000

R/W Base 0x40000000

disable Warnings:

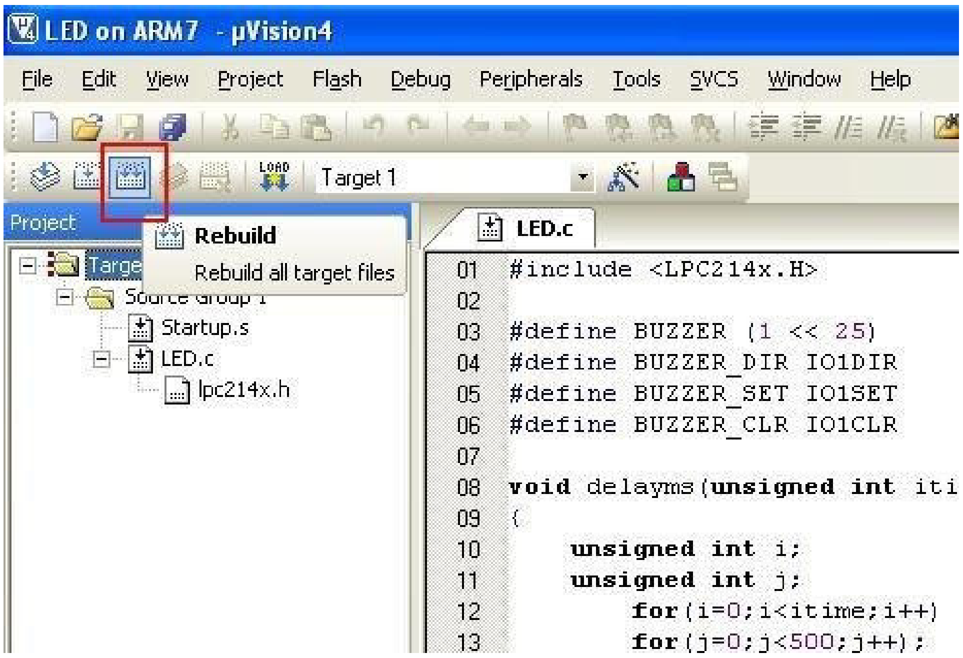
Scatter File

Misc controls

Linker control string

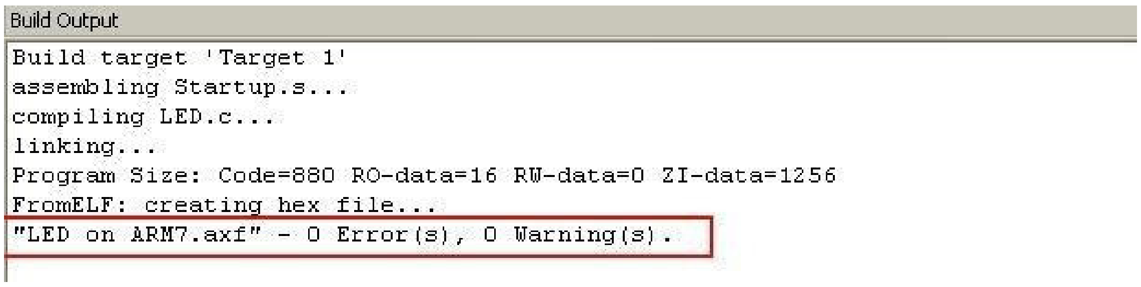
```
--cpu ARM7TDMI *.o --strict --scatter "LED on ARM7.sct"
--autoat --summary_stderr --info summarysizes --map --xref --callgraph --symbols
```

Step 16: Then Click on Rebuild All Target Files



```
#include <LPC214x.H>

#define BUZZER (1 << 25)
#define BUZZER_DIR IO1DIR
#define BUZZER_SET IO1SET
#define BUZZER_CLR IO1CLR

void delayms(unsigned int iti
{
    unsigned int i;
    unsigned int j;
        for(i=0;i<itime;i++)
        for(j=0;j<500;j++);
```

Step 17: Now you see 0 Error(s), 0 Warning (s). Then Hex File will create in Specific Folder. Now to download it for you target hardware.



```
Build target 'Target 1'
assembling Startup.s...
compiling LED.c...
linking...
Program Size: Code=880 RO-data=16 RW-data=0 ZI-data=1256
FromELF: creating hex file...
"LED on ARM7.axf" - 0 Error(s), 0 Warning(s).
```

Part A

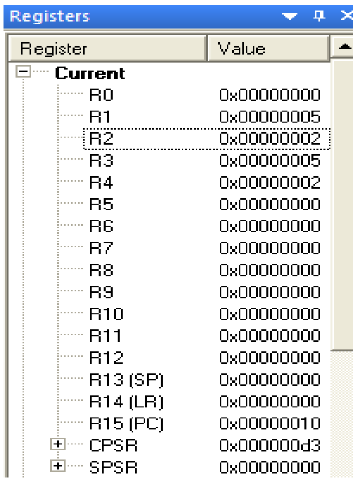
Program1-Sample Programs using Keil Compiler:

<u>AIM: To write and simulate ARM assembly language programs for data transfer, arithmetic and logical operations (Demonstrate with the help of a suitable program).</u>

1. Data Transfer.

The below assembly level program moves the 32 bit data from register to register.

```
        area movt, code, readonly
        entry
        mov r1,#0005    ; Mov immediate 32 bit data to r1
        mov r2,#0002 ; Mov immediate 32 bit data to r1
        mov r3,r1             ; Register-Register movement
        mov r4,r2               ; Register-Register movement

stop b stop                       ; End of the program
         end
```

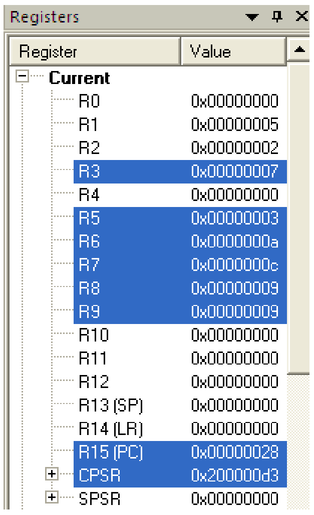


```
Registers
Register        Value
Current
  R0            0x00000000
  R1            0x00000005
  R2            0x00000002
  R3            0x00000005
  R4            0x00000002
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000010
  CPSR          0x000000d3
  SPSR          0x00000000
```

2. Arithmetic Operations

   A. Addition, Subtraction and Multiplication:

```
        area addt, code, readonly
        entry
        mov r1,#0005          ; Mov immediate 32 bit data to r1
        mov r2,#0002          ; Mov immediate 32 bit data to r2
        add r3,r2,r1          ; Add the contents present in r2 with the
                                contents of r1 and store in r3
        sub r5,r1,r2          ; Subtract; r5 = r1-r2
        mul r6,r1,r2          ; Multiply
        mov r7,r6
        add r7,#2             ; Add immediate data
        mov r8,r7
        sub r8,#3             ; Subtract immediate data
        mov r9,r8

stop b stop
```

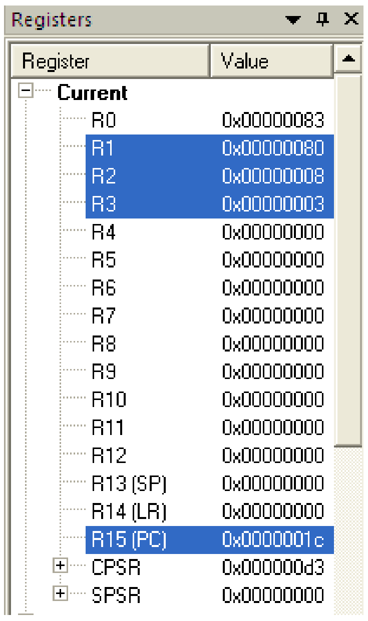


```
Registers
Register        Value
Current
  R0            0x00000000
  R1            0x00000005
  R2            0x00000002
  R3            0x00000007
  R4            0x00000000
  R5            0x00000003
  R6            0x0000000a
  R7            0x0000000c
  R8            0x00000009
  R9            0x00000009
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000028
  CPSR          0x200000d3
  SPSR          0x00000000
```

```
            end
```

3. Logical operations : To perform AND, Logical Shift operations,

```
area dis,code,readonly
entry
mov r0,#0x83
mov r1,r0
and r1, # 0Xf0  ; Perform Logical AND operation
mov r2,r1
lsr r2, #4        ; Perform Logical right Shift operation
mov r3, r0
and r3, # 0X0f
and r1,r0
orr r2,r1         ; Logical OR Operation
lsr  r2, #3       ; Logical shift right r2 by 3 bit positions

stop b stop
        end
```



Registers

Register	Value
Current	
R0	0x00000083
R1	0x00000080
R2	0x00000008
R3	0x00000003
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001c
CPSR	0x000000d3
SPSR	0x00000000

Write an ALP using ARM to execute the following instructions

• ADD r1, r0, r0, LSL #3
; r1 = r0 + r0 << 3 = r0 + 8 × r0

• ADD r1, r0, r0, LSR #3
; r1 = r0 + r0 >> 3 = r0 + r0/8 (unsigned)

• ADD r1, r0, r0, ASR #3
; r1 = r0 + r0 >> 3 = r0 + r0/8 (signed)

The state of the system after loading the code for Program 1

- ✓ The semicolon indicates a user-supplied comment.
- ✓ Anything following a semicolon on the same line is ignored by the assembler.
- ✓ The first line is AREA Example1, CODE, READONLY is an assembler directive and is required to set up the program. It is a feature of the development system and not the ARM assembly language.
- ✓ An assembler from a different company may have a different way of defining the start of a program. In this case, AREA refers to the segment of code, Example1 is the name we've given it, CODE indicates executable code rather than data, and READONLY state that it cannot be modified at run time.

- ✓ Anything starting in column 1 (in this case Stop) is a label that can be used to refer to that line.
- ✓ The instruction Stop B Stop means 'Branch to the line labeled Stop' and is used to create an infinite loop. This is a convenient way of ending programs in simple examples like these.
- ✓ The last line END is an assemble directive that tells the assembler there is not more code to follow. It ends the program.

Graded ARM assembly language Examples

ADDITION

The problem: P = Q + R + S
Let Q = 2, R = 4, S = 5. Assume that r1 = Q, r2 = R, r3 = S. The result Q will go in r0.

The Code

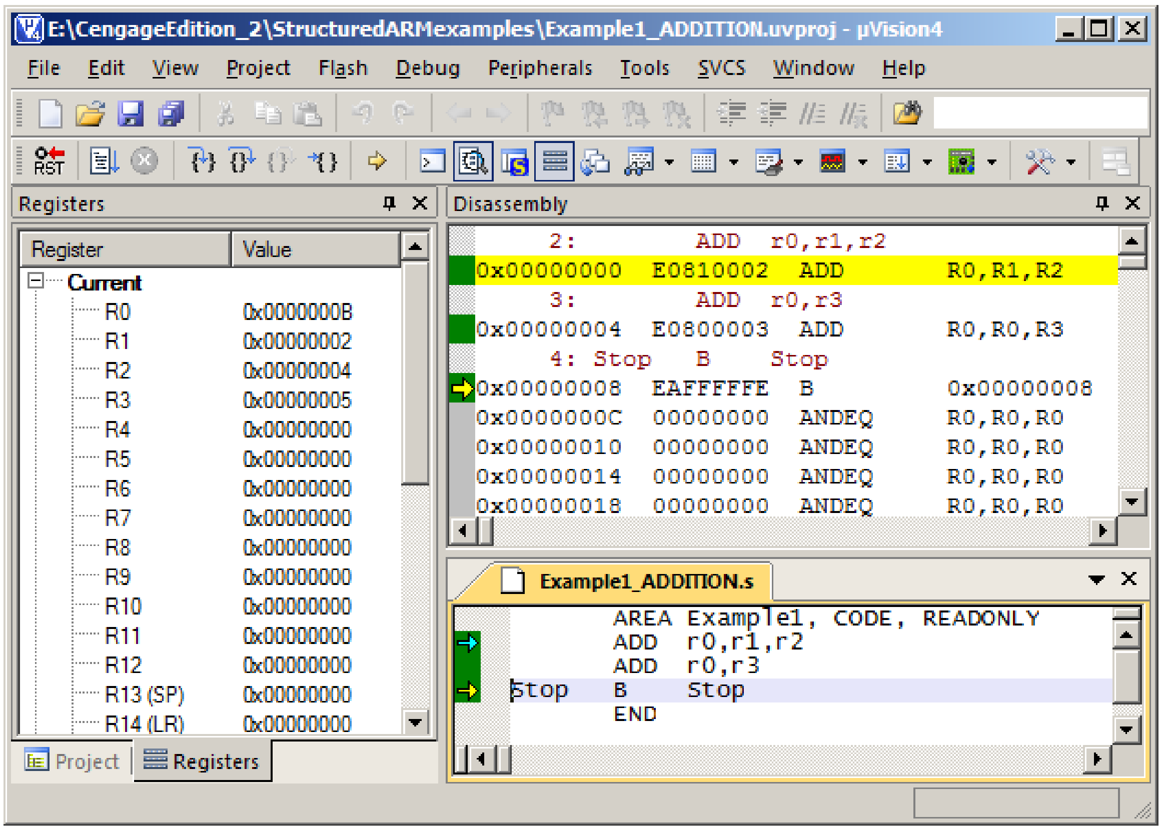
```
ADD r0,r1,r2 ;add Q to R and put in P
ADD r0,r0,r3 ;add S to P and put the result in P
```

The program

```
AREA Example1, CODE, READONLY
ADD r0,r1,r2
ADD r0,r3
Stop B Stop
END
```

Notes:

The semicolon indicates a user-supplied comment. Anything following a semicolon on the same line is ignored by the assembler.
The first line is AREA Example1, CODE, READONLY is an assembler directive and is required to set up the program. It is a feature of the development system and not the ARM assembly language. An assembler from a different company may have a different way of defining the start of a program. In this case, AREA refers to the segment of code, Example1 is the name we've given it, CODE indicates executable code rather than data, and READONLYstate that it cannot be modified at run time.
Anything starting in column 1 (in this case Stop) is a label that can be used to refer to that line.
The instruction Stop B Stop means 'Branch to the line labelled Stop' and is used to create an infinite loop. This is a convenient way of ending programs in simple examples like these.
The last line ENDis an assemble directive that tells the assembler there is not more code to follow. It ends the program.


E:\CengageEdition_2\StructuredARMexamples\Example1_ADDITION.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
Example1_ADDITION.s
Project
Registers

Register	Value
Current	
R0	0x0000000B
R1	0x00000002
R2	0x00000004
R3	0x00000005
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000

```
     2:          ADD  r0,r1,r2
0x00000000  E0810002  ADD       R0,R1,R2
     3:          ADD  r0,r3
0x00000004  E0800003  ADD       R0,R0,R3
     4: Stop    B     Stop
0x00000008  EAFFFFFE  B         0x00000008
0x0000000C  00000000  ANDEQ     R0,R0,R0
0x00000010  00000000  ANDEQ     R0,R0,R0
0x00000014  00000000  ANDEQ     R0,R0,R0
0x00000018  00000000  ANDEQ     R0,R0,R0
```

```
        AREA Example1, CODE, READONLY
        ADD  r0,r1,r2
        ADD  r0,r3
Stop    B    Stop
        END
```

Figure Example 1.1 The state of the system after loading the code for Example 1

Note that the contents of r0 are 2 + 4 + 5 = 11 = 0x0B. This is the result we expected.

Example 2 ADDITION

This problem is the same as Example 1. P = Q + R + S
Once again, let Q = 2, R = 4, S = 5 and assume r1 = Q, r2 = R, r3 = S. In this case, we will put the data in memory in the form of constants before the program runs.
The Code

```
MOV r1,#Q ;load Q into r1
MOV r2,#R ;load R into r2
MOV r3,#S ;load S into r3
ADD r0,r1,r2 ;Add Q to R
ADD r0,r0,r3 ;Add S to (Q + R)
```

Here we use the instruction MOV that copies a value into a register. The value may be the contents of another register or a literal. The literal is denoted by the # symbol.
We can write, for example, MOV r7,r0, MOV r1,#25 or MOV r5,#Time

We have used symbolic names Q, R and S. We have to relate these names to actual values. We do this with the EQU (equate) assembler directive; for example,
Q EQU 2
Relates the name Q to the value 5. If the programmer uses Q in an expression, it is exactly the same as writing 2. The
purpose of using Q rather than 2 is to make the program more readable.
The program

```
AREA Example2, CODE, READONLY
MOV r1,#Q ;load r1 with the constant Q
MOV r2,#R
MOV r3,#S
ADD r0,r1,r2
ADD r0,r0,r3
Stop B Stop
Q EQU 2 ;Equate the symbolic name Q to the value 2
R EQU 4 ;
S EQU 5 ;
END
```

Example 3 ADDITION
The problem once again is P = Q + R + S. As before, Q = 2, R = 4, S = 5 and we assume that r1 = Q, r2 = R, r3 = S.
In this case, we will put the data in memory as constants before the program runs. We first use the load register,
LDR r1,Q instruction to load register r1 with the contents of memory location Q. This instruction *does not exist* and is not part of the ARM's instruction set. However, the ARM assembler automatically changes it into an actual instruction.
We call LDR r1,Q a *pseudoinstruction* because it behaves like a real instruction. It is indented to make the life of a programmer happier by providing a shortcut.
The Code

```
LDR r1,Q ;load r1 with Q
LDR r2,R ;load r2 with R
LDR r3,S ;load r3 with S
ADD r0,r1,r2 ;add Q to R
ADD r0,r0,r3 ;add  in  S
STR r0,Q ;store result in Q
```

The program

```
AREA Example3, CODE, READWRITE
LDR r1,Q ;load r1 with Q
LDR r2,R ;load r2 with R
LDR r3,S ;load r3 with S
```

```
ADD r0,r1,r2 ;add Q to R
ADD r0,r3 ;add in S
STR r0,Q ;store result in Q
Stop B Stop
AREA Example3, CODE, READWRITE
P SPACE 4 ;save one word of storage
Q DCD 2 ;create variable Q with initial value 2
R DCD 4 ;create variable  R with initial value  4
S DCD 5 ;create variable S with initial value 5
END
```

Note how we have to create a data area at the end of the program. We have reserved spaces for P, Q, R, and S. We use the SPACE directive for S to reserve 4 bytes of memory space for the variable S. After that we reserve space for Q, R, and S. In each case we use a DCD assembler directive to reserve a word location (4 bytes) and to initialize it. For example,

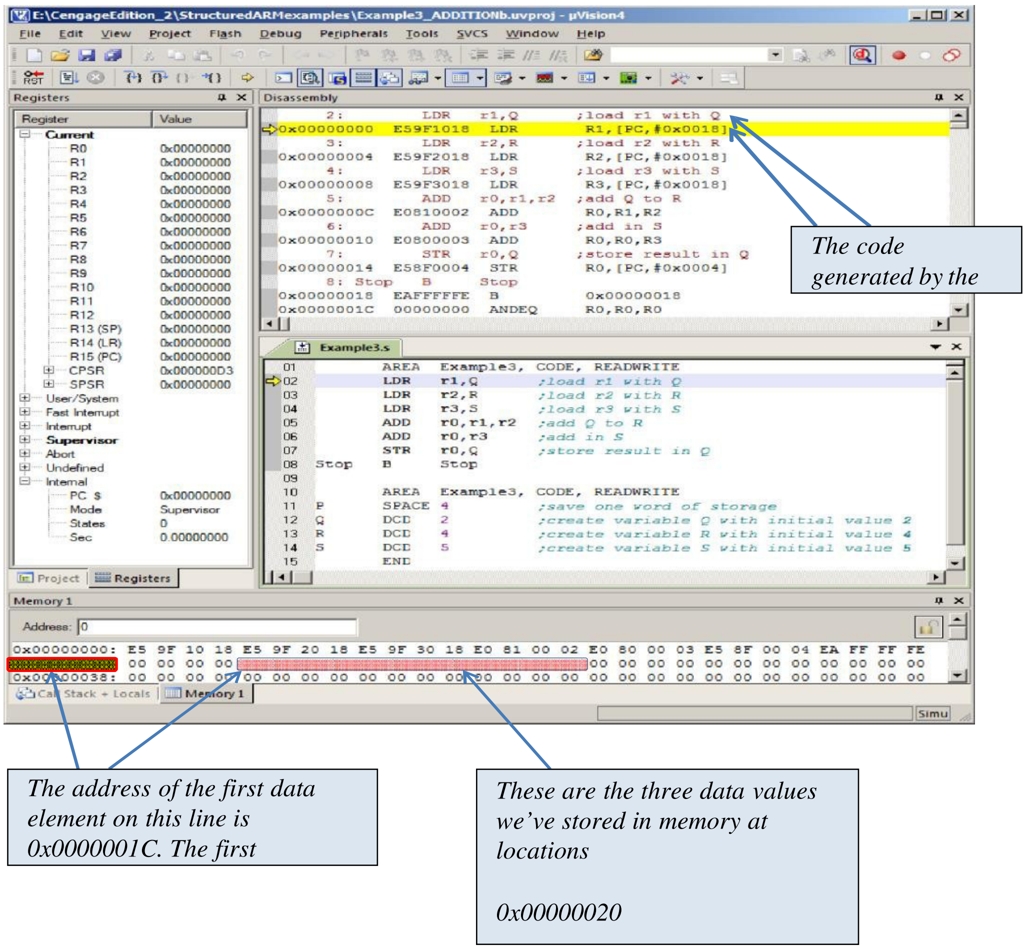
Q DCD 2 ;create variable Q with initial value 2 means 'call the current line Q and store the word 0x00000002 at that location.

Figure Example 3.1 shows the state of the program after it has been loaded. In this case we've used the view memory command to show the memory space. We have highlighted the three constants that have been pre-loaded into memory.

Take a look at the disassembled code. The pseudoinstruction LDR r1,Q was actually translated into the real ARM instruction LDR r1,[PC,#0x0018]. This is still a load instruction but the addressing mode is register indirect. In this case, the address is the contents of the program counter, PC, plus the hexadecimal offset 0x18. Note also that the program counter is always 8 bytes beyond the address of the current instruction. This is a feature of the ARM's pipeline.

Consequently, the address of the operand is [PC] + 0x18 + 8 = 0 + 18 + 8 = 0x20.

If you look at the memory display area you will find that the contents of 0x20 are indeed 0x00000002.


E:\CengageEdition_2\StructuredARMexamples\Example3_ADDITIONb.uvproj - µVision4

Registers

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

Disassembly

```
      2:          LDR    r1,Q       ;load r1 with Q
0x00000000  E59F1018  LDR        R1,[PC,#0x0018]
      3:          LDR    r2,R       ;load r2 with R
0x00000004  E59F2018  LDR        R2,[PC,#0x0018]
      4:          LDR    r3,S       ;load r3 with S
0x00000008  E59F3018  LDR        R3,[PC,#0x0018]
      5:          ADD    r0,r1,r2   ;add Q to R
0x0000000C  E0810002  ADD        R0,R1,R2
      6:          ADD    r0,r3      ;add in S
0x00000010  E0800003  ADD        R0,R0,R3
      7:          STR    r0,Q       ;store result in Q
0x00000014  E58F0004  STR        R0,[PC,#0x0004]
      8: Stop     B      Stop
0x00000018  EAFFFFFE  B          0x00000018
0x0000001C  00000000  ANDEQ      R0,R0,R0
```

Example3.s

```
        AREA   Example3, CODE, READWRITE
        LDR    r1,Q        ;load r1 with Q
        LDR    r2,R        ;load r2 with R
        LDR    r3,S        ;load r3 with S
        ADD    r0,r1,r2    ;add Q to R
        ADD    r0,r3       ;add in S
        STR    r0,Q        ;store result in Q
Stop    B      Stop

        AREA   Example3, CODE, READWRITE
P       SPACE  4           ;save one word of storage
Q       DCD    2           ;create variable Q with initial value 2
R       DCD    4           ;create variable R with initial value 4
S       DCD    5           ;create variable S with initial value 5
        END
```

Memory 1

Address: 0

```
0x00000000: E5 9F 10 18 E5 9F 20 18 E5 9F 30 18 E0 81 00 02 E0 80 00 03 E5 8F 00 04 EA FF FF FE
            00 00 00 00                                     00 00 00 00 00 00 00 00 00 00 00 00
0x00000038: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project
Registers
Call Stack + Locals
Memory 1
Simu
The code generated by the
The address of the first data element on this line is 0x0000001C. The first
These are the three data values we've stored in memory at locations
0x00000020

Figure The state of the system after loading the program

Example 4 ADDITION

The problem P = Q + R + S where Q = 2, R = 4, S = 5. In this case we are going to use register indirect addressing to access the variables. That is, we have to set up a pointer to the variables and access them via this pointer.

The Code

```
ADR r4,TheData ;r4 points to the data area
LDR r1,[r4,#Q] ;load Q into r1
LDR r2,[r4,#R] ;load R into r2
LDR r3,[r4,#S] ;load S into r3
ADD r0,r1,r2 ;add  Q  and  R
ADD r0,r0,r3 ;add S to the total
STR r0,[r4,#P] ;save the result in memory
```

The program

```
AREA  Example4,  CODE,  READWRITE
ENTRY
ADR r4,TheData ;r4 points to the data area
LDR r1,[r4,#Q] ;load Q into r1
LDR r2,[r4,#R] ;load R into r2
LDR r3,[r4,#S] ;load S into r3
ADD r0,r1,r2 ;add  Q  and  R
ADD r0,r0,r3 ;add S to the total
STR r0,[r4,#P] ;save the result in memory
Stop B Stop
P EQU 0 ;offset for P
Q EQU 4 ;offset for Q
R EQU 8 ;offset for R
S EQU 12 ;offset for S
AREA  Example4,  CODE,  READWRITE
TheData SPACE 4 ;save one word of storage for P
DCD 2 ;create variable  Q  with  initial value  2
DCD 4 ;create variable  R  with  initial value  4
DCD 5 ;create variable S with  initial  value  5
END
```

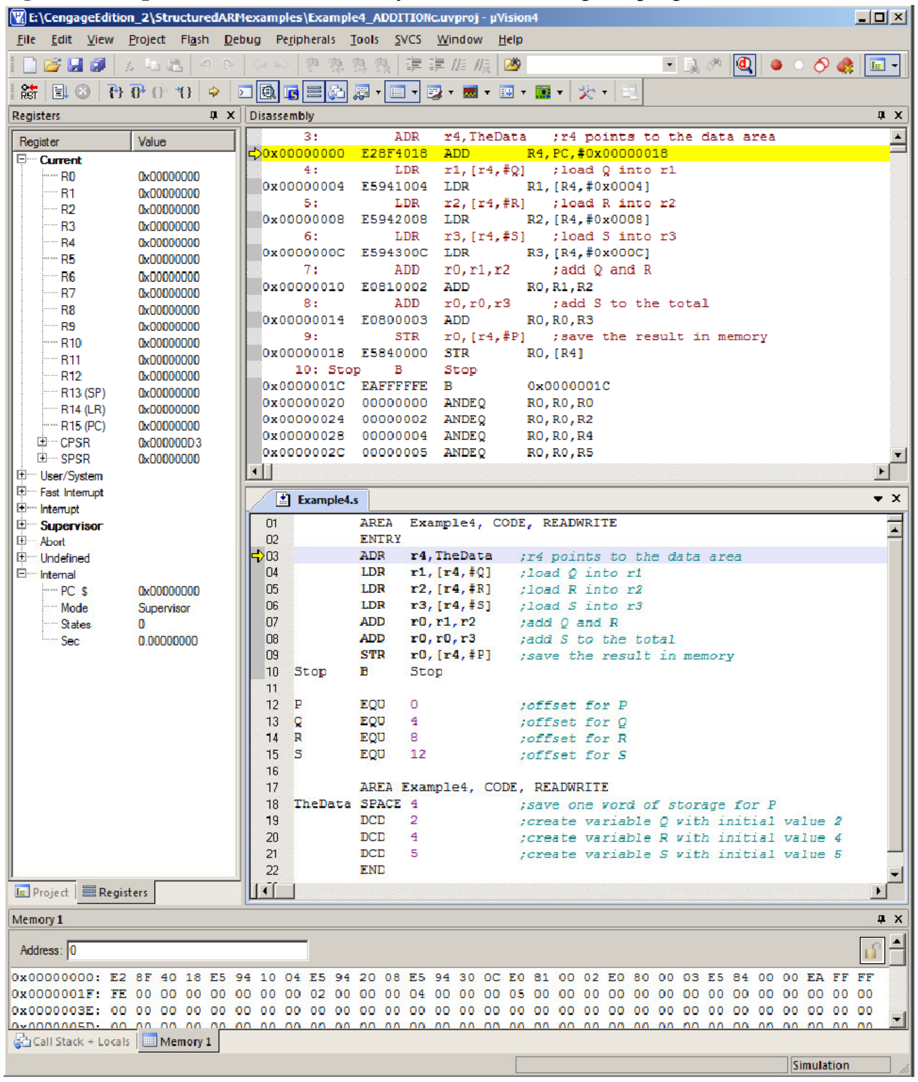
Figure Example 4.1 shows the state of the system after the program has been loaded. I have to admit, that I would not write this code as it is presented. It is far too verbose. However, it does illustrate several concepts.

First, the instruction ADR r4,TheData loads the address of the data region (that we have labelled TheData into register r4. That is, r4 is pointing at the data area. If you look at the code, we have reserved four bytes for P and then have loaded the values for Q, R and S into consecutive word location. Note that we have not labelled any of these locations.

The instruction ADR (load an address into a register) is a pseudoinstruction. If you look at the actual disassembled code in Figure Example 4.1 you will see that this instruction is translated into ADD r4,pc,#0x18. Instead of loading the actual address of TheData into r4 it is loading the PC plus an offset that will give the appropriate value. Fortunately, programmers can sleep soundly without worrying about how the ARM is going to translate an ADR into actual code – that's the beauty of pseudoinstructions.

When we load Q into r1 we use LDR r1,[r4,#Q]. This is an ARM load register instruction with a literal offset; that is, Q. If you look at the EQU region, Q is equated to 4 and therefore register r1 is loaded with the data value that is 4 bytes on from where r4 is pointing. This location is, of course, where the data corresponding to Q has been stored.

Figure Example 4.1 The state of the system after loading the program


E:\CengageEdition_2\StructuredARMexamples\Example4_ADDITIONc.uvproj - µVision4

Registers

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

Disassembly

```
     3:            ADR     r4,TheData    ;r4 points to the data area
0x00000000  E28F4018  ADD        R4,PC,#0x00000018
     4:            LDR     r1,[r4,#Q]    ;load Q into r1
0x00000004  E5941004  LDR        R1,[R4,#0x0004]
     5:            LDR     r2,[r4,#R]    ;load R into r2
0x00000008  E5942008  LDR        R2,[R4,#0x0008]
     6:            LDR     r3,[r4,#S]    ;load S into r3
0x0000000C  E594300C  LDR        R3,[R4,#0x000C]
     7:            ADD     r0,r1,r2      ;add Q and R
0x00000010  E0810002  ADD        R0,R1,R2
     8:            ADD     r0,r0,r3      ;add S to the total
0x00000014  E0800003  ADD        R0,R0,R3
     9:            STR     r0,[r4,#P]    ;save the result in memory
0x00000018  E5840000  STR        R0,[R4]
    10: Stop     B     Stop
0x0000001C  EAFFFFFE  B          0x0000001C
0x00000020  00000000  ANDEQ      R0,R0,R0
0x00000024  00000002  ANDEQ      R0,R0,R2
0x00000028  00000004  ANDEQ      R0,R0,R4
0x0000002C  00000005  ANDEQ      R0,R0,R5
```

Example4.s

```
        AREA   Example4, CODE, READWRITE
        ENTRY
        ADR    r4,TheData     ;r4 points to the data area
        LDR    r1,[r4,#Q]     ;load Q into r1
        LDR    r2,[r4,#R]     ;load R into r2
        LDR    r3,[r4,#S]     ;load S into r3
        ADD    r0,r1,r2       ;add Q and R
        ADD    r0,r0,r3       ;add S to the total
        STR    r0,[r4,#P]     ;save the result in memory
Stop    B      Stop

P       EQU    0              ;offset for P
Q       EQU    4              ;offset for Q
R       EQU    8              ;offset for R
S       EQU    12             ;offset for S

        AREA Example4, CODE, READWRITE
TheData SPACE  4              ;save one word of storage for P
        DCD    2              ;create variable Q with initial value 2
        DCD    4              ;create variable R with initial value 4
        DCD    5              ;create variable S with initial value 5
        END
```

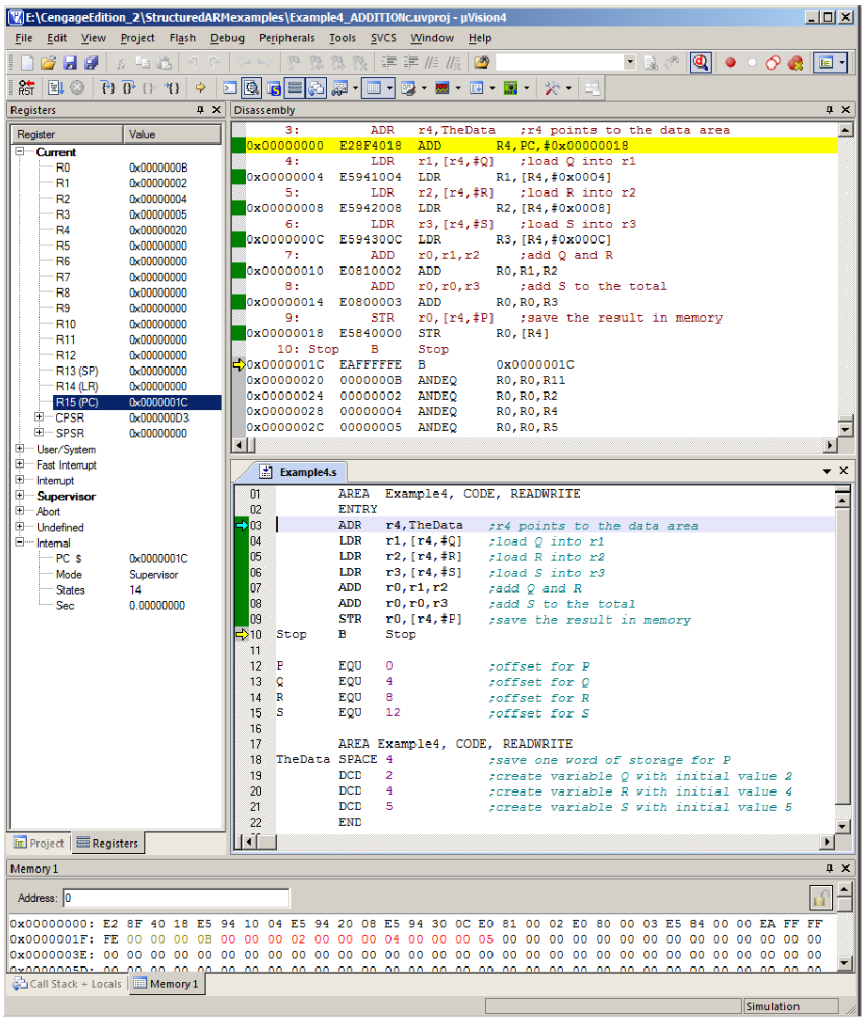
Memory 1

Address: 0

```
0x00000000: E2 8F 40 18 E5 94 10 04 E5 94 20 08 E5 94 30 0C E0 81 00 02 E0 80 00 03 E5 84 00 00 EA FF FF
0x0000001F: FE 00 00 00 00 00 00 00 02 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000003E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000005D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Simulation

Figure Example 4.2 The state of the system after executing the program


E:\CengageEdition_2\StructuredARMexamples\Example4_ADDITIONc.uvproj - µVision4

```
Register        Value
Current
  R0            0x0000000B
  R1            0x00000002
  R2            0x00000004
  R3            0x00000005
  R4            0x00000020
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x0000001C
  CPSR          0x000000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
  PC $          0x0000001C
  Mode          Supervisor
  States        14
  Sec           0.00000000
```

Disassembly

```
     3:          ADR     r4,TheData    ;r4 points to the data area
0x00000000  E28F4018  ADD        R4,PC,#0x00000018
     4:          LDR     r1,[r4,#Q]    ;load Q into r1
0x00000004  E5941004  LDR        R1,[R4,#0x0004]
     5:          LDR     r2,[r4,#R]    ;load R into r2
0x00000008  E5942008  LDR        R2,[R4,#0x0008]
     6:          LDR     r3,[r4,#S]    ;load S into r3
0x0000000C  E594300C  LDR        R3,[R4,#0x000C]
     7:          ADD     r0,r1,r2      ;add Q and R
0x00000010  E0810002  ADD        R0,R1,R2
     8:          ADD     r0,r0,r3      ;add S to the total
0x00000014  E0800003  ADD        R0,R0,R3
     9:          STR     r0,[r4,#P]    ;save the result in memory
0x00000018  E5840000  STR        R0,[R4]
    10: Stop     B       Stop
0x0000001C  EAFFFFFE  B          0x0000001C
0x00000020  0000000B  ANDEQ      R0,R0,R11
0x00000024  00000002  ANDEQ      R0,R0,R2
0x00000028  00000004  ANDEQ      R0,R0,R4
0x0000002C  00000005  ANDEQ      R0,R0,R5
```

Example4.s

```
         AREA   Example4, CODE, READWRITE
         ENTRY
         ADR    r4,TheData    ;r4 points to the data area
         LDR    r1,[r4,#Q]    ;load Q into r1
         LDR    r2,[r4,#R]    ;load R into r2
         LDR    r3,[r4,#S]    ;load S into r3
         ADD    r0,r1,r2      ;add Q and R
         ADD    r0,r0,r3      ;add S to the total
         STR    r0,[r4,#P]    ;save the result in memory
Stop     B      Stop

P        EQU    0             ;offset for P
Q        EQU    4             ;offset for Q
R        EQU    8             ;offset for R
S        EQU    12            ;offset for S

         AREA Example4, CODE, READWRITE
TheData  SPACE 4              ;save one word of storage for P
         DCD    2             ;create variable Q with initial value 2
         DCD    4             ;create variable R with initial value 4
         DCD    5             ;create variable S with initial value 5
         END
```

Memory 1 — Address: 0

```
0x00000000: E2 8F 40 18 E5 94 10 04 E5 94 20 08 E5 94 30 0C E0 81 00 02 E0 80 00 03 E5 84 00 00 EA FF FF
0x0000001F: FE 00 00 00 0B 00 00 00 02 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000003E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Example 5 ADDITION

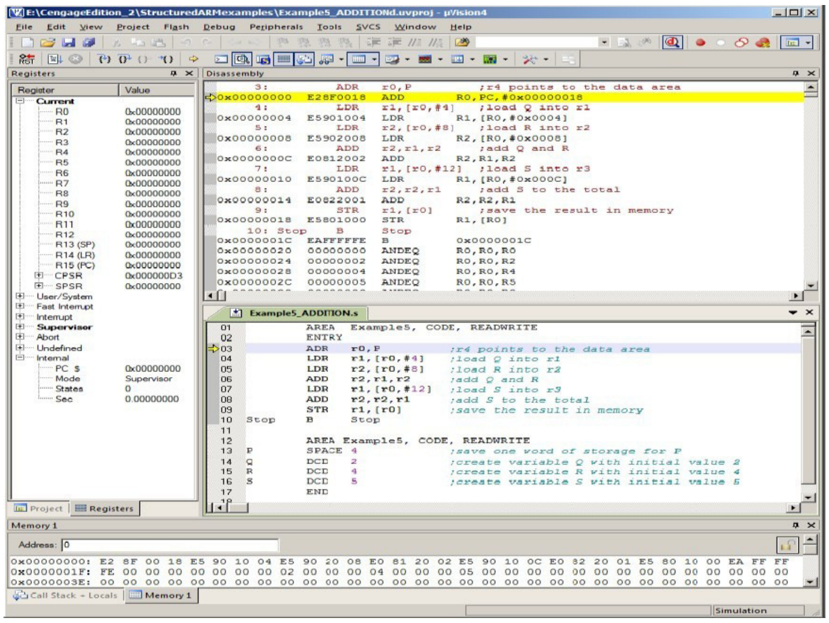
We’re going to repeat the same example once again. This time we will write the program in a more compact fashion,
still using the ADR (load register with address instruction).
To simplify the code, we’ve used simple numeric offsets (because there is relatively little data and the user comments
tell us what’s happening. Note that we have used labels Q, R, and S for the data. These labels are redundant and are not
needed since they are not referred to anywhere else in the program. There’s nothing wrong with this. These labels just
serve as a reminder to the programmer.

```
AREA Example5, CODE, READWRITE
ENTRY
ADR r0,P ;r4 points to the data area
LDR r1,[r0,#4] ;load Q into r1
LDR r2,[r0,#8] ;load R into r2
```

```
ADD r2,r1,r2 ;add Q and R
LDR r1,[r0,#12] ;load S into r3
ADD r2,r2,r1 ;add S to the total
STR r1,[r2] ;save the result in memory
Stop B Stop
AREA Example5, CODE, READWRITE
P SPACE 4 ;save one word of storage for P
Q DCD 2 ;create variable Q with initial value 2
R DCD 4 ;create variable  R with initial value  4
S DCD 5 ;create variable S with initial value 5
END
```

Note also that we have reused registers to avoid taking up so many. This example uses only r0, r1, and r2. Once a register has been used (and its value plays no further part in a program, it can be reused. However, this can make debugging harder. In this example at one point r1 contains Q and at another point it contains S. Finally, it contains the result S. Figure Example 5.1 gives a snapshot of the system after the program has been loaded, and Figure Example 5.2 shows the state after the program has been executed.

Figure Example 5.1 The state of the system before executing the program


E:\CengageEdition_2\StructuredARMexamples\Example5_ADDITIONd.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
Example5_ADDITION.s
Project
Registers
Memory 1
Call Stack + Locals
Simulation

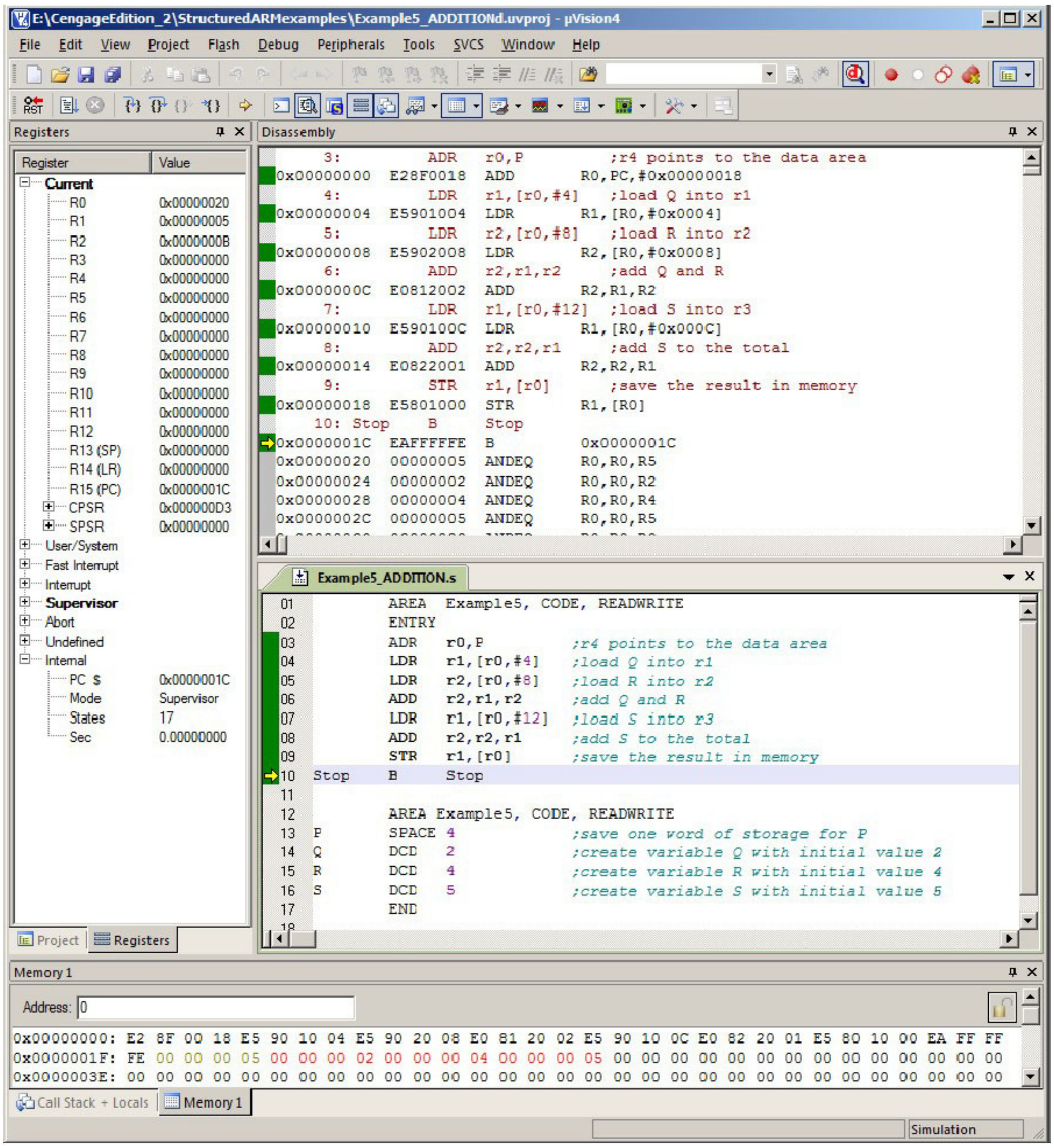
Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

```
      3:           ADR    r0,P          ;r4 points to the data area
0x00000000  E28F0018  ADD       R0,PC,#0x00000018
      4:           LDR    r1,[r0,#4]    ;load Q into r1
0x00000004  E5901004  LDR       R1,[R0,#0x0004]
      5:           LDR    r2,[r0,#8]    ;load R into r2
0x00000008  E5902008  LDR       R2,[R0,#0x0008]
      6:           ADD    r2,r1,r2      ;add Q and R
0x0000000C  E0812002  ADD       R2,R1,R2
      7:           LDR    r1,[r0,#12]   ;load S into r3
0x00000010  E590100C  LDR       R1,[R0,#0x000C]
      8:           ADD    r2,r2,r1      ;add S to the total
0x00000014  E0822001  ADD       R2,R2,R1
      9:           STR    r1,[r0]       ;save the result in memory
0x00000018  E5801000  STR       R1,[R0]
     10: Stop      B      Stop
0x0000001C  EAFFFFFE  B         0x0000001C
0x00000020  00000000  ANDEQ     R0,R0,R0
0x00000024  00000002  ANDEQ     R0,R0,R2
0x00000028  00000004  ANDEQ     R0,R0,R4
0x0000002C  00000005  ANDEQ     R0,R0,R5
```

```
        AREA   Example5, CODE, READWRITE
        ENTRY
        ADR    r0,P             ;r4 points to the data area
        LDR    r1,[r0,#4]       ;load Q into r1
        LDR    r2,[r0,#8]       ;load R into r2
        ADD    r2,r1,r2         ;add Q and R
        LDR    r1,[r0,#12]      ;load S into r3
        ADD    r2,r2,r1         ;add S to the total
        STR    r1,[r0]          ;save the result in memory
Stop    B      Stop

        AREA Example5, CODE, READWRITE
P       SPACE  4                ;save one word of storage for P
Q       DCD    2                ;create variable Q with initial value 2
R       DCD    4                ;create variable R with initial value 4
S       DCD    5                ;create variable S with initial value 5
        END
```

```
Address: 0
0x00000000: E2 8F 00 18 E5 90 10 04 E5 90 20 08 E0 81 20 02 E5 90 10 0C E0 82 20 01 E5 80 10 00 EA FF FF
0x0000001F: FE 00 00 00 00 00 00 00 02 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000003E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure Example 5.2 The state of the system after executing the program



E:\CengageEdition_2\StructuredARMexamples\Example5_ADDITIONd.uvproj - µVision4

Registers

Register	Value
Current	
R0	0x00000020
R1	0x00000005
R2	0x0000000B
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001C
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x0000001C
Mode	Supervisor
States	17
Sec	0.00000000

Disassembly

```
     3:          ADR     r0,P          ;r4 points to the data area
0x00000000  E28F0018  ADD       R0,PC,#0x00000018
     4:          LDR     r1,[r0,#4]    ;load Q into r1
0x00000004  E5901004  LDR       R1,[R0,#0x0004]
     5:          LDR     r2,[r0,#8]    ;load R into r2
0x00000008  E5902008  LDR       R2,[R0,#0x0008]
     6:          ADD     r2,r1,r2      ;add Q and R
0x0000000C  E0812002  ADD       R2,R1,R2
     7:          LDR     r1,[r0,#12]   ;load S into r3
0x00000010  E590100C  LDR       R1,[R0,#0x000C]
     8:          ADD     r2,r2,r1      ;add S to the total
0x00000014  E0822001  ADD       R2,R2,R1
     9:          STR     r1,[r0]       ;save the result in memory
0x00000018  E5801000  STR       R1,[R0]
    10: Stop     B       Stop
0x0000001C  EAFFFFFE  B         0x0000001C
0x00000020  00000005  ANDEQ     R0,R0,R5
0x00000024  00000002  ANDEQ     R0,R0,R2
0x00000028  00000004  ANDEQ     R0,R0,R4
0x0000002C  00000005  ANDEQ     R0,R0,R5
```

Example5_ADDITION.s

```
        AREA   Example5, CODE, READWRITE
        ENTRY
        ADR    r0,P             ;r4 points to the data area
        LDR    r1,[r0,#4]       ;load Q into r1
        LDR    r2,[r0,#8]       ;load R into r2
        ADD    r2,r1,r2         ;add Q and R
        LDR    r1,[r0,#12]      ;load S into r3
        ADD    r2,r2,r1         ;add S to the total
        STR    r1,[r0]          ;save the result in memory
Stop    B      Stop

        AREA Example5, CODE, READWRITE
P       SPACE 4                 ;save one word of storage for P
Q       DCD   2                 ;create variable Q with initial value 2
R       DCD   4                 ;create variable R with initial value 4
S       DCD   5                 ;create variable S with initial value 5
        END
```

Memory 1

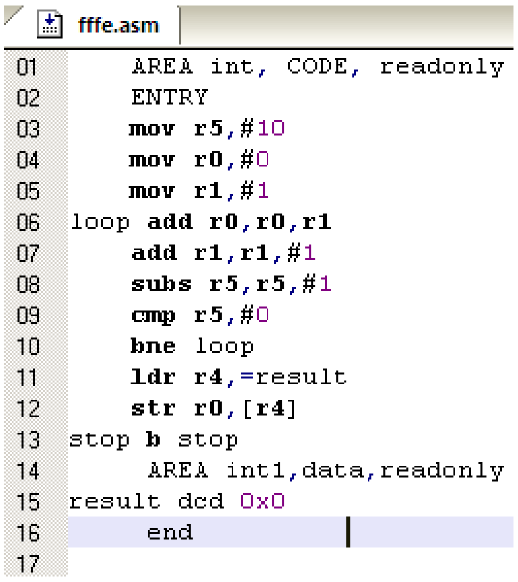
Address: 0

```
0x00000000: E2 8F 00 18 E5 90 10 04 E5 90 20 08 E0 81 20 02 E5 90 10 0C E0 82 20 01 E5 80 10 00 EA FF FF
0x0000001F: FE 00 00 00 05 00 00 00 02 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000003E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

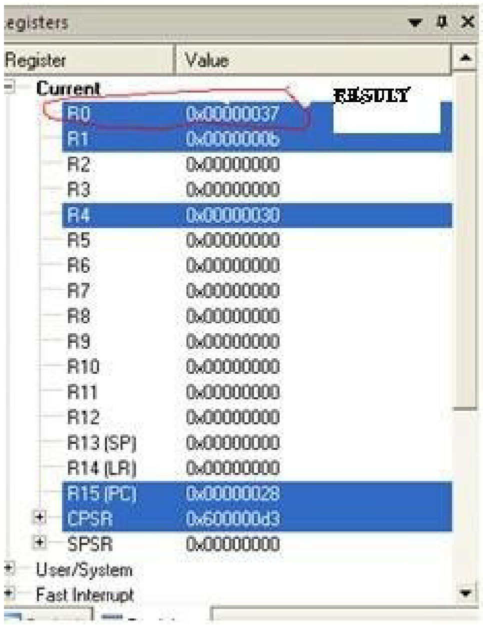
Program No .2 Date:

AIM: Write a program to find the sum of first 10 integer numbers.

```
        AREA int, CODE, readonly
        ENTRY
        mov r5,#10
        mov r0,#0
        mov r1,#1
loop add r0,r0,r1
        add r1,r1,#1
        subs r5,r5,#1
        cmp r5,#0
        bne loop
        ldr r4,=result
        str r0,[r4]
stop b stop
     AREA int1,data,readonly
result dcd 0x0
         end
```


fffe.asm

```
     AREA int, CODE, readonly
     ENTRY
     mov r5,#10
     mov r0,#0
     mov r1,#1
loop add r0,r0,r1
     add r1,r1,#1
     subs r5,r5,#1
     cmp r5,#0
     bne loop
     ldr r4,=result
     str r0,[r4]
stop b stop
      AREA int1,data,readonly
result dcd 0x0
      end
```



```
Register       Value
- Current
   R0          0x00000037
   R1          0x0000000b
   R2          0x00000000
   R3          0x00000000
   R4          0x00000030
   R5          0x00000000
   R6          0x00000000
   R7          0x00000000
   R8          0x00000000
   R9          0x00000000
   R10         0x00000000
   R11         0x00000000
   R12         0x00000000
   R13 (SP)    0x00000000
   R14 (LR)    0x00000000
   R15 (PC)    0x00000028
 + CPSR        0x600000d3
 + SPSR        0x00000000
```

Result: 1+2+3+4+5+6+7+8+9+10= 55 in decimal
The Hexa value of 55 is 37 is stored in R0 Register.

Program No .3 Date:

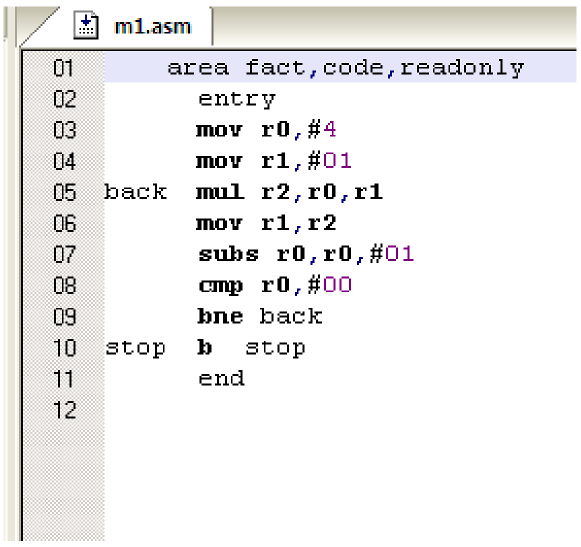
AIM: Write a program to find factorial of a number.

```
        AREA factorial, CODE, readonly
        ENTRY

        MOV  R0,#1          ;int c =1
        MOV  R1,#5          ;int fact=3
        MOV  R3,#1          ;int n=1
        BL loop
        B STOP
loop
        MUL R4,R3,R0
        MOV R3,R4
        ADD R0,R0,#1
        CMP R0,R1
        BLE loop
        MOV PC,LR
STOP B STOP
        END
```

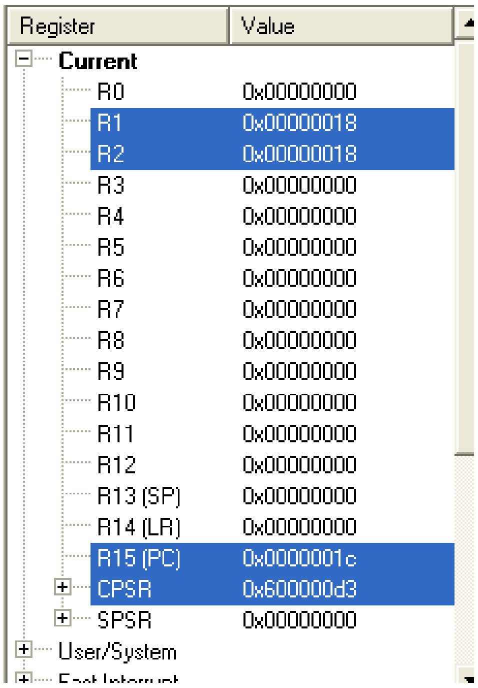
OR

```
        area fact,code,readonly
        entry
        mov r0,#4
        mov r1,#01
back mul r2,r0,r1
        mov r1,r2
        subs r0,r0,#01
        cmp r0,#00
        bne back
stop b stop
        end
```

The final result will be available in R1 register, It will be in Hexadecimal value eg: The data given hers is 5: Factorial is 5*4*3*2*1= 120d, But result in R1 will be 78, which is the hexadecimal value of 120.
For 4; 24 is Decimal and 18 in Hexa as shown in below output of Register R1.


m1.asm

```
         area fact,code,readonly
            entry
            mov r0,#4
            mov r1,#01
back        mul r2,r0,r1
            mov r1,r2
            subs r0,r0,#01
            cmp r0,#00
            bne back
stop        b  stop
            end
```

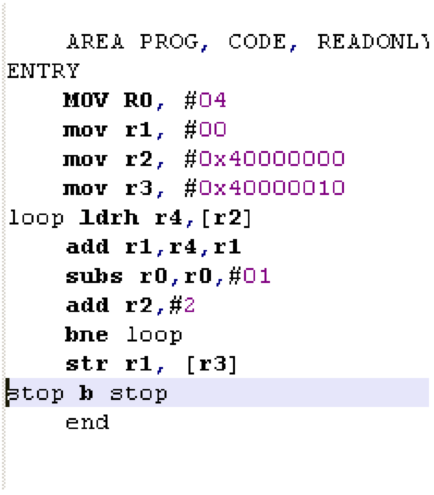


Register	Value
Current	
R0	0x00000000
R1	0x00000018
R2	0x00000018
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001c
CPSR	0x600000d3
SPSR	0x00000000
User/System	

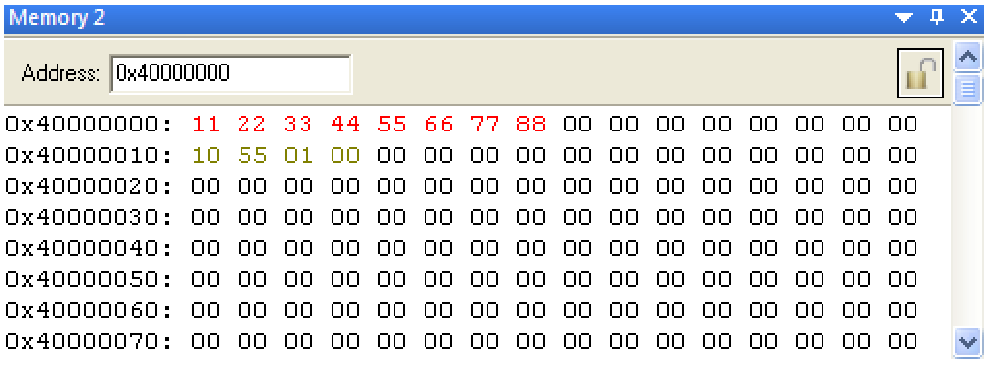
Program No. 4 Date:

Aim: Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM

```
        AREA  PROG, CODE, READONLY
        ENTRY
        MOV R0, #04
        mov r1, #00
        mov r2, #0x40000000
        mov r3, #0x40000010
loop ldrh r4,[r2]
        add r1,r4,r1
        subs r0,r0,#01
        add  r2,#2
        bne loop
        str r1, [r3]
stop b stop
        end
```



```
    AREA PROG, CODE, READONLY
ENTRY
    MOV R0, #04
    mov r1, #00
    mov r2, #0x40000000
    mov r3, #0x40000010
loop ldrh r4,[r2]
    add r1,r4,r1
    subs r0,r0,#01
    add r2,#2
    bne loop
    str r1, [r3]
stop b stop
    end
```



```
0x40000000: 11 22 33 44 55 66 77 88 00 00 00 00 00 00 00 00
0x40000010: 10 55 01 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Result:

```
      2211
+     4433
+     6655
+     8877
----------------------
  00|01|55|10   ←
```

Perform Hexa addition:

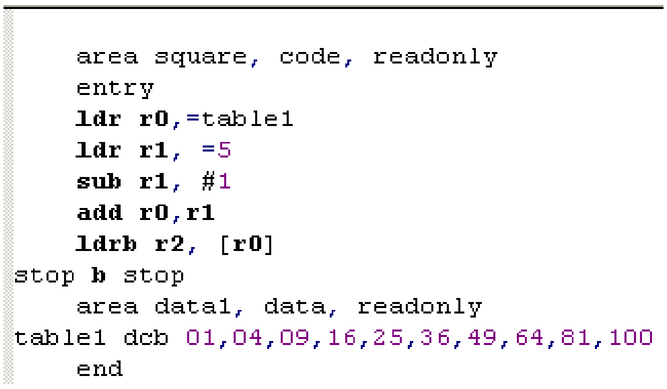
Result: Result will be stored from this side

Look for the results in location 0x40000010.

Program No. 5 Date:

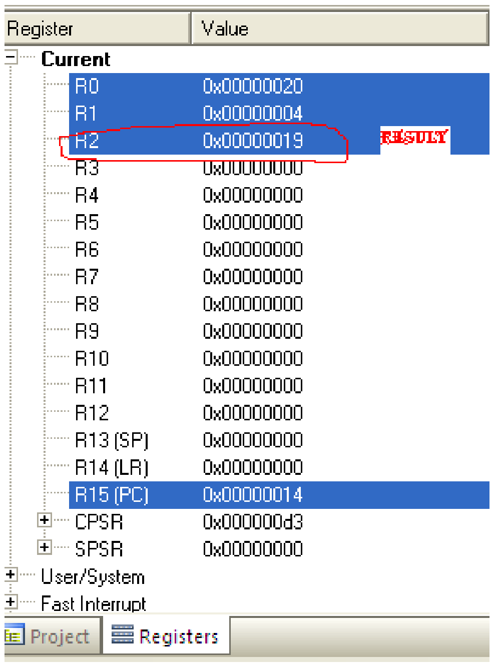
Aim: Write a program to find the square of a number (1 to 10) using look-up table.

```
        area square, code, readonly
        entry
        ldr r0,=table1
        ldr r1, =5
        sub r1, #1
        add r0,r1
        ldrb r2, [r0]
stop b stop
        area data1, data, readonly
table1 dcb 01,04,09,16,25,36,49,64,81,100
        end
```



```
    area square, code, readonly
    entry
    ldr r0,=table1
    ldr r1, =5
    sub r1, #1
    add r0,r1
    ldrb r2, [r0]
stop b stop
    area data1, data, readonly
table1 dcb 01,04,09,16,25,36,49,64,81,100
    end
```

Result:



```
Register        Value
Current
  R0            0x00000020
  R1            0x00000004
  R2            0x00000019   RESULT
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000014
  CPSR          0x000000d3
  SPSR          0x00000000
User/System
Fast Interrupt
```

Result: The given number is 5, Square of 5 is 25 in decimal, It is 18 in Hexa, The value 18 is found in R2.

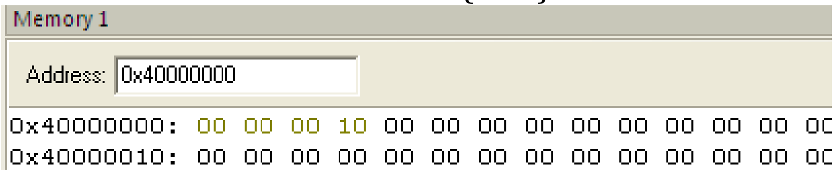
Program No. 6 Date:

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

```
AREA PROG, CODE, READONLY
             ENTRY
             ldr r0,= data1
             ldr  r3,=0x40000000
             ldr r4, = 0x05
             ldr r1, [r0],#04
             sub  r4,r4,#01
back         ldr r2, [r0]
             cmp r1,r2
             bhs/blo less
             mov r1,r2
less         add r0,r0,#04
             sub r4,r4,#01
             cmp r4,#00
             bne back
             str r1,[r3]
stop b stop
             area data, code
; data1 dcd &64,&05,&96,&10,&65 ; (Either Data can be given in
this format or as shown in the next line)
data1 dcd 0x70000000,0x80000000,0x90000000,0x10000000,0x50000000
             end
```

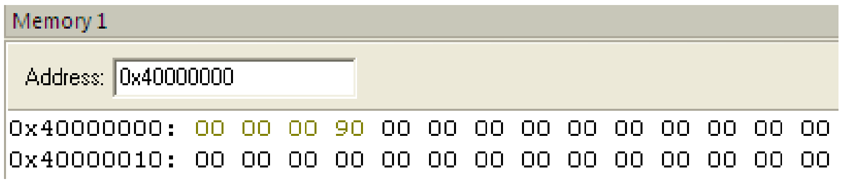
RESULTS:

LOWEST VALUE 0x10000000{BLO} IS STORED AS SHOWN BELOW



```
0x40000000: 00 00 00 10 00 00 00 00 00 00 00 00 00 0C
0x40000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 0C
```

HISHEST VALUE 0x9000000{BHS} IS STORED AS SHOWN BELOW



```
0x40000000: 00 00 00 90 00 00 00 00 00 00 00 00 00 00
0x40000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

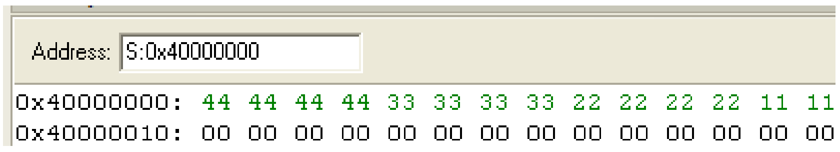
Program No. 7   Date:

AIM: Write a program to arrange a series of 32 bit numbers in ascending/descending order.

```
;/* PROGRAM TO sort in Descending order */
;/*     ARRAY OF 4 NUMBERS 0X44444444
*/,0X11111111,0X33333333,0X22222222                    */
;/* SET A BREAKPOINT AT START1 LABLE & RUN THE PROGRAM*/
;/*     CHECK THE UNSORTED NUMBERS AT LOCATION 0X40000000
NEXT                                           */
;/* SET A BREAKPOINT AT NOP INSTRUCTION,RUN THE PROGRAM &
CHECK THE RESULT */
;/* RESULT CAN BE VIEWED AT LOCATION 0X40000000 */


        AREA DESCENDING, CODE, READONLY

ENTRY                           ;Mark first instruction to execute


                MOV R8,#4          ; INTIALISE COUNTER TO 4(i.e. N=4)
                LDR R2,=CVALUE              ; ADDRESS OF CODE REGION
                LDR R3,=DVALUE          ; ADDRESS OF DATA REGION

LOOP0
                LDR R1,[R2],#4     ;  LOADING VALUES FROM CODE REGION
                STR R1,[R3],#4     ;  STORING VALUES TO DATA REGION

                SUBS R8,R8,#1     ; DECREMENT COUNTER
                CMP R8,#0          ; COMPARE COUNTER TO 0
                BNE LOOP0          ; LOOP BACK TILL ARRAY ENDS

START1          MOV R5,#3 ; INTIALISE COUNTER TO 3(i.e. N=4)
                MOV R7,#0 ; FLAG TO DENOTE EXCHANGE HAS OCCURED
                LDR R1,=DVALUE ; LOADS THE ADDRESS OF FIRST VALUE

LOOP        LDR R2, [R1],#4   ; WORD ALIGN T0  ARRAY ELEMENT
                LDR R3,[R1]              ; LOAD SECOND NUMBER
                CMP R2,R3                    ; COMPARE NUMBERS
                BGT/BLT LOOP2 ; IF THE FIRST NUMBER IS > THEN GOTO LOOP2
                STR R2,[R1],#-4          ; INTERCHANGE NUMBER R2 & R3
```

```
                        STR R3,[R1]                 ; INTERCHANGE NUMBER R2 & R3
                        MOV R7,#1 ; FLAG DENOTING EXCHANGE HAS TAKEN PLACE
                        ADD R1,#4                           ; RESTORE THE PTR

LOOP2
                        SUBS R5,R5,#1               ; DECREMENT COUNTER
                        CMP R5,#0                   ; COMPARE COUNTER TO 0
                        BNE LOOP            ; LOOP BACK TILL ARRAY ENDS
                        CMP R7,#0                           ; COMPARING FLAG
                        BNE START1; IF FLAG IS NOT ZERO THEN GO TO START1 LOOP
        NOP
        NOP
        NOP

; ARRAY OF 32 BIT NUMBERS(N=4) IN CODE REGION

CVALUE
                DCD 0X44444444                  ;
                DCD  0X11111111                         ;
                DCD  0X33333333                         ;
                DCD  0X22222222                         ;

        AREA DATA1,DATA,READWRITE       ;
; ARRAY OF 32 BIT NUMBERS IN DATA REGION
DVALUE
                DCD 0X00000000                  ;

        END                             ; Mark end of file
```

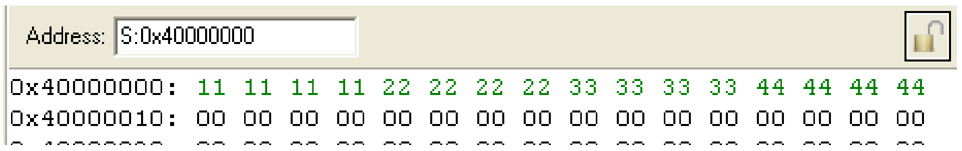
DESCENDING ORDER OUTPUT



```
Address: S:0x40000000
0x40000000: 44 44 44 44 33 33 33 33 22 22 22 22 11 11
0x40000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

<u>ASCENDING ORDER</u>

OUTPUT:

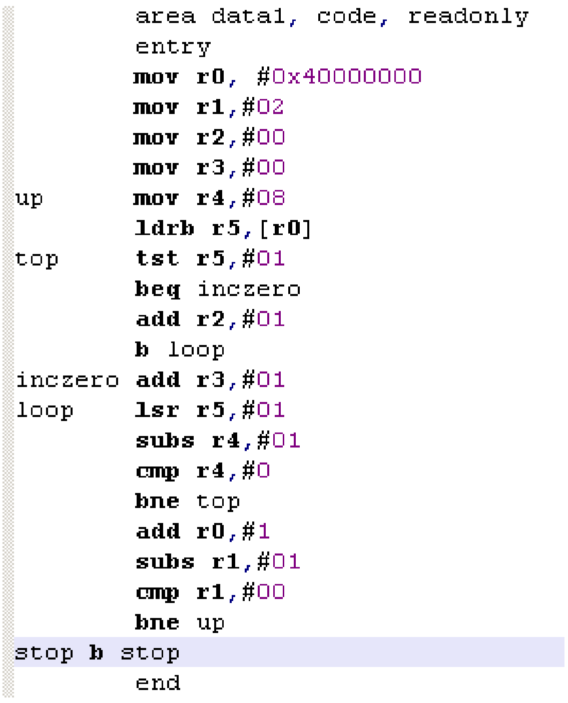


```
Address: S:0x40000000
0x40000000: 11 11 11 11 22 22 22 22 33 33 33 33 44 44 44 44
0x40000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

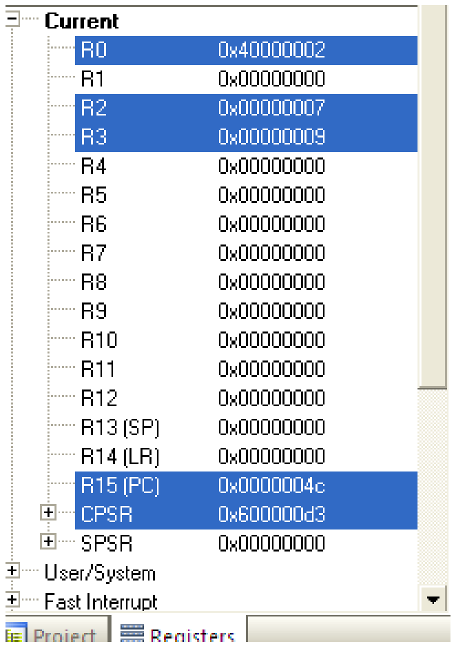
Program No. 8 Date:

Aim: Write a program to count the number of ones and zeros in two consecutive memory locations.

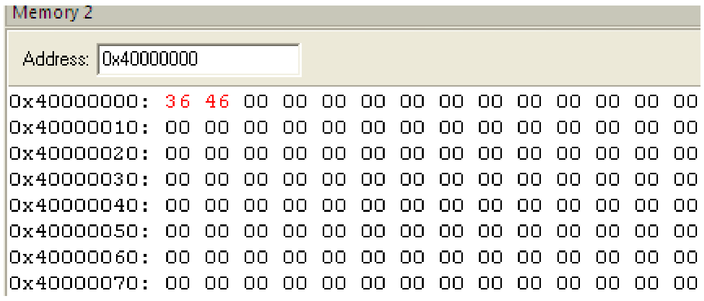
```
        area data1, code, readonly
        entry
        mov r0, #0x40000000
        mov  r1,#02
        mov  r2,#00
        mov r3,#00
 up             mov r4,#08
        ldrb r5,[r0]
top             tst r5,#01
        beq inczero
        add r2,#01
        b loop
inczero         add r3,#01
loop    lsr r5,#01
        subs r4,#01
        cmp r4,#0
        bne  top
        add r0,#1
        subs r1,#01
        cmp r1,#00
        bne up
stop b stop
        end
```



```
         area data1, code, readonly
         entry
         mov r0, #0x40000000
         mov r1,#02
         mov r2,#00
         mov r3,#00
up       mov r4,#08
         ldrb r5,[r0]
top      tst r5,#01
         beq inczero
         add r2,#01
         b loop
inczero  add r3,#01
loop     lsr r5,#01
         subs r4,#01
         cmp r4,#0
         bne top
         add r0,#1
         subs r1,#01
         cmp r1,#00
         bne up
stop b stop
         end
```



```
Current
  R0          0x40000002
  R1          0x00000000
  R2          0x00000007
  R3          0x00000009
  R4          0x00000000
  R5          0x00000000
  R6          0x00000000
  R7          0x00000000
  R8          0x00000000
  R9          0x00000000
  R10         0x00000000
  R11         0x00000000
  R12         0x00000000
  R13 (SP)    0x00000000
  R14 (LR)    0x00000000
  R15 (PC)    0x0000004c
  CPSR        0x600000d3
  SPSR        0x00000000
User/System
Fast Interrupt
```



```
Address: 0x40000000
0x40000000: 36 46 00 00 00 00 00 00 00 00 00 00 00 00
0x40000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Result:

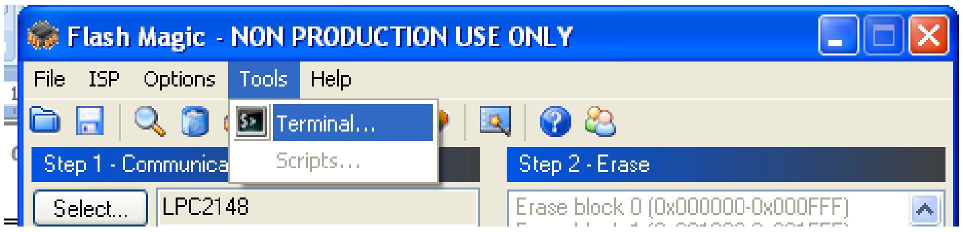
The given data is 36 and 46:
0011 0110 0100 0110
There are 9 zeros stored in R3 and 7 ones stored in R2.

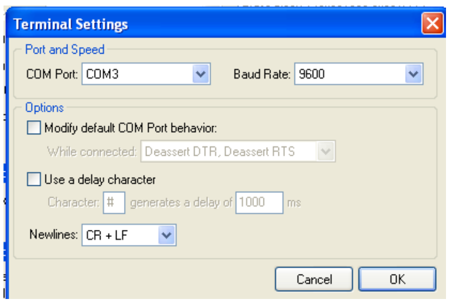
Program No. 09 Date:

Aim: Display "Hello World" message using Internal UART

PROGRAM:

```
#include <LPC21xx.H>            /* LPC21xx definitions */
#include "Serial.h"
void delay_ms(int count)
{
  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);
  }
}
```

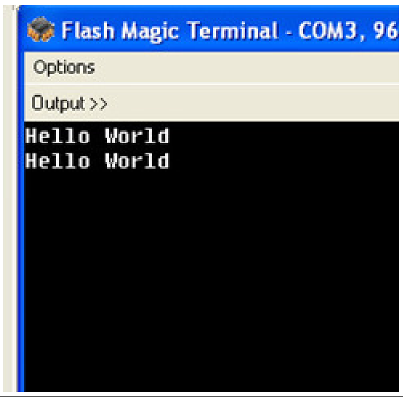



Terminal Settings

Port and Speed
COM Port: COM3 Baud Rate: 9600

Options
☐ Modify default COM Port behavior:
While connected: Deassert DTR, Deassert RTS
☐ Use a delay character
Character: # generates a delay of 1000 ms
Newlines: CR + LF

Cancel OK



```
Hello World
Hello World
```

PART-B

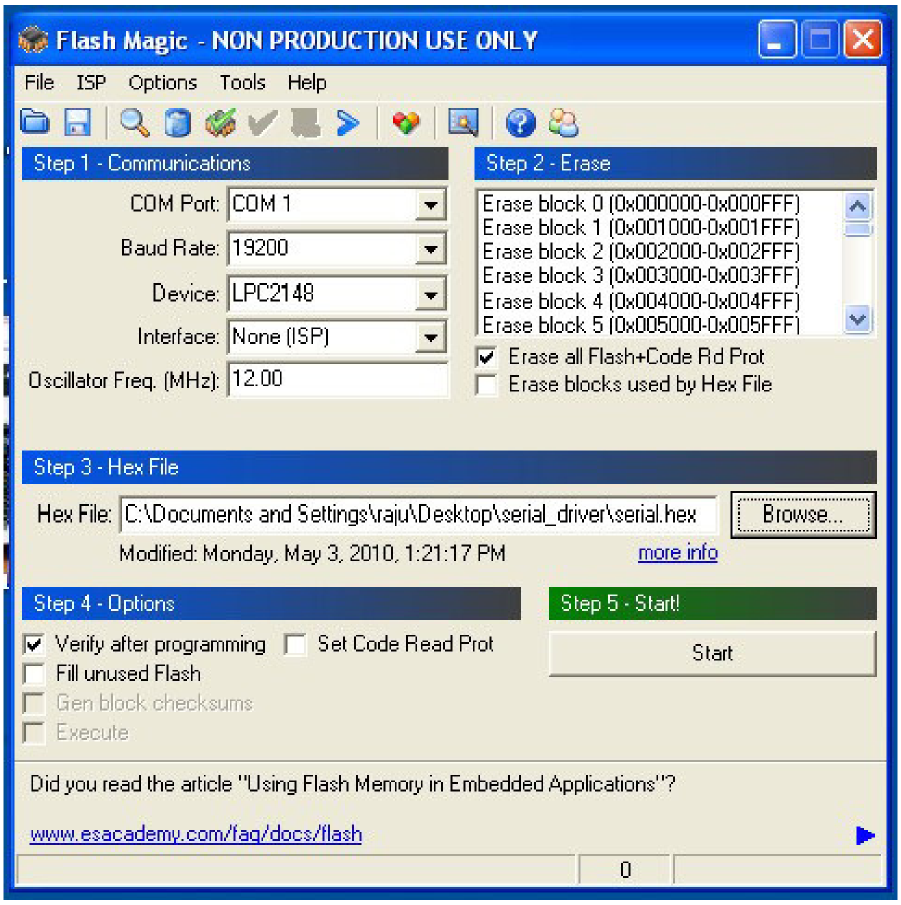
Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.

Flash Magic Tool

To program the Microcontroller, Flash Magic tool is used. Generally, the microcontroller is in one of the two modes. One is RUN mode and the other is PROGRAMMING mode. In RUN mode microcontroller executes the application present in the microcontroller flash memory. In PROGRAMMING mode, microcontroller programs its flash memory in synchronisation with Flash Magic.

To enter in to the programming mode, Hold down SW2(isp) and SW3(reset), then release SW3 first and finally SW2 . To enter in to Run Mode,press the SW3(reset) after programming is over.

Snapshot of the Flash Magic Tool.



COM Port: COM 1
Baud Rate: 19200
Device: LPC2148
Interface: None (ISP)
Oscillator Freq. (MHz): 12.00

Erase block 0 (0x000000-0x000FFF)
Erase block 1 (0x001000-0x001FFF)
Erase block 2 (0x002000-0x002FFF)
Erase block 3 (0x003000-0x003FFF)
Erase block 4 (0x004000-0x004FFF)
Erase block 5 (0x005000-0x005FFF)

[✓] Erase all Flash+Code Rd Prot
[ ] Erase blocks used by Hex File

Hex File: C:\Documents and Settings\raju\Desktop\serial_driver\serial.hex
Modified: Monday, May 3, 2010, 1:21:17 PM

[✓] Verify after programming
[ ] Set Code Read Prot
[ ] Fill unused Flash
[ ] Gen block checksums
[ ] Execute

Did you read the article "Using Flash Memory in Embedded Applications"?
www.esacademy.com/faq/docs/flash

‘

Sample programs to demonstrate with the help of a suitable program

LPC2148 Led Blinking

This C program discusses how to configure the LPC2148 ports as GPIO and then send a low/high signal on it.

The Below registers will be used for Configuring and using the GPIOs for sending and receiving the Digital signals.

1. PINSEL: GPIO Pins Select Register

Almost all the LPC1768 pins are multiplexed to support more than 1 function. Every GPIO pin has a minimum of one function and max of four functions. The required function can be selected by configuring the PINSEL register.

2. IODIR: GPIO Direction Control Register.

This register individually controls the direction of each port pin.

IOxDIR : This is the GPIO direction control register. Setting a bit to ‘0’ in this register will configure the corresponding pin to be used as input while setting it to ‘1’ will configure it as output.

Values	Direction
0	Input
1	Output

3. IOSET:Port Output Set Register.

This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register, not the physical port value.

IOxSET : This register can be used to drive an 'output' configured pin to logic 1 i.e. HIGH. Writing zero does not have any effect and hence it can't be used to drive a pin to Logic 0 i.e. LOW. For driving pins LOW IOxCLR is used which is explained as below:

Values	IOSET
0	No Effect
1	Sets High on Pin

4. IOCLR:Port Output Clear Register.

IOxCLR:This register can be used to drive an 'output' configured pin to logic 0 i.e. LOW. Writing zero does not have any effect and hence it can't be used to drive a pin to Logic 1.
This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect.

Values	IOCLR
0	No Effect
1	Sets Low on Pin

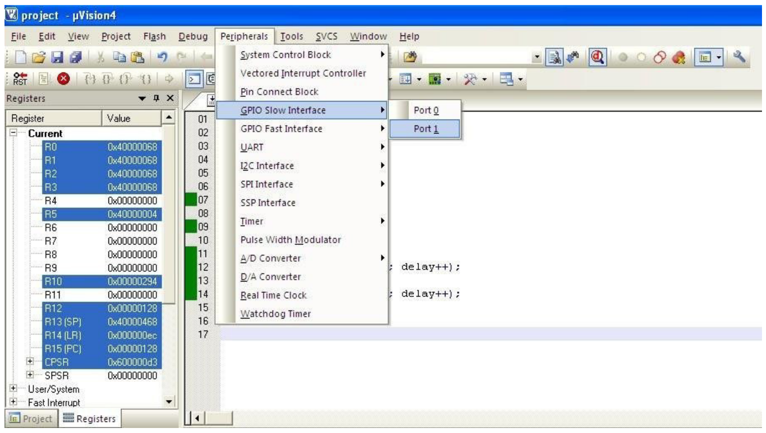
5. IOPIN: GPIO Port Pin Value Register.

This register is used for both reading and writing data from/to the PORT. Output: Writing to this register places corresponding values in all bits of the particular PORT pins.

Input: The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC
IO0CLR=(1<<10), this is how we can make P0.10 to become LOW (LED turned ON). IO0SET= (1<<10), would make output HIGH (LED turned OFF) for Pin P0.10.

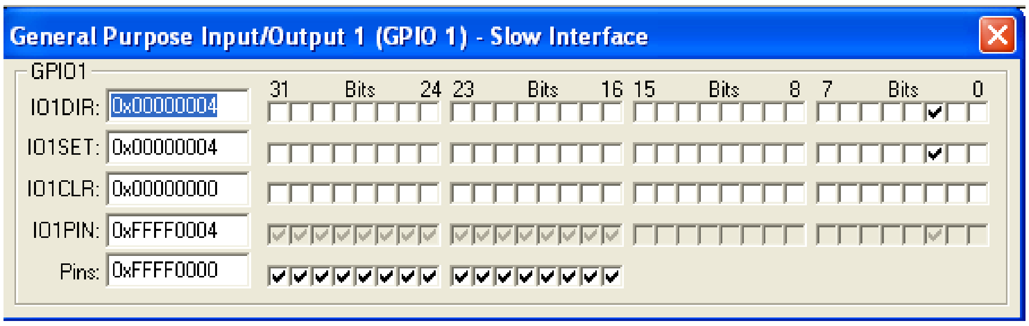
Sample 'C' Program: To write a C program to Blink a LED /Port Pin with LPC 2148 ARM 7 Microcontroller.

```
#include <lpc214x.h>                    //Header File "x"  can be wrt to
controller
 unsigned int delay;

        int  main(void)
                {
                        IO1DIR = (4);       // Bit No 4 (0100) will be activated

                        while(1)                                   // If True
                        {
                      IO1CLR = (04);                       // Clear Bit 04 of GPIO1
                    for (delay=0 ;delay<5000; delay++);    // Call Delay
                    IO1SET = (04);                  // Set Bit 04 of GPIO1
                    for (delay=0; delay<5000; delay++);   // Call Delay
                        }
                    }
```



Register	Value
Current	
R0	0x40000068
R1	0x40000068
R2	0x40000068
R3	0x40000068
R4	0x00000000
R5	0x40000004
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000294
R11	0x00000000
R12	0x00000128
R13 (SP)	0x40000468
R14 (LR)	0x000000ec
R15 (PC)	0x00000128
CPSR	0x600000d3
SPSR	0x00000000
User/System	
Fast Interrupt	

```
; delay++);
; delay++);
```


General Purpose Input/Output 1 (GPIO 1) - Slow Interface

GPIO1

		31 Bits 24	23 Bits 16	15 Bits 8	7 Bits 0
IO1DIR:	0x00000004	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐✓☐☐
IO1SET:	0x00000004	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐✓☐☐
IO1CLR:	0x00000000	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐
IO1PIN:	0xFFFF0004	✓✓✓✓✓✓✓✓	✓✓✓✓✓✓✓✓	☐☐☐☐☐☐☐☐	☐☐☐☐☐✓☐☐
Pins:	0xFFFF0000	✓✓✓✓✓✓✓✓	✓✓✓✓✓✓✓✓		

Program No. 1

Aim: Interface and Control a DC Motor.

DC Motor Control using PWM of LPC1768
In most of the applications controlling the speed of DC motor is essential where the precision and protection are the essence. Here we will use the PWM technique to control the speed of the motor
LPC 2148 has one PWM channel with six ports. PWM changes the average output voltage by fast switching. By changing the on time, the output voltage can be 0 to 100%. There are two software parameters that need a little explanation: cycle and offset. Cycle is the length of a PWM duty cycle and offset is the on time of a duty cycle.
SELECTING THE PWM FUNCTION TO GPIO
The block diagram below shows the PWM pins multiplexed with other GPIO pins. The PWM pin can be enabled by configuring the corresponding PINSEL register to select PWM function. When the PWM function is selected for that pin in the Pin Select register, other Digital signals are disconnected from the PWM input pins.
PWM REGISTERS:
The registers associated with LPC1768 PWM are

□ IR-> Interrupt Register: The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.

□ TCR-> Timer Control Register: The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.

□ PR- > Prescale Register: The TC is incremented every PR+1 cycles of PCLK.

□ MCR-> Match Control Register: The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.

□ MR0 – MR6-> Match Register: Each can be enabled in the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC.

□ PCR-> PWM Control Register: Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.

□ LCR-> Load Enable Register: Enables use of new PWM match values.
Note: for detailed description of each registers kindly refer PWM waveform section
If you need to control the speed of a DC motor you have a few options. Controlling the speed by controlling either voltage or current is inefficient. Let--s understand a bit the speed control of DC motor Using Pulse Width Modulation because controlling how long the voltage is applied with a certain frequency gives you the best control over the motor--s speed.

Conventional power supplies tend to generate lots of heat because are working as variable resistors pumping current through external circuits. The pulse width modulation circuits are digital circuits which produce pulsed current. Due to the fact that the pulsed width modulation power supplies works in a state in between on and off, the heat generated is very low compared to the conventional power supplies.
The duty cycle of the circuit can be changed by pressing the switches SW22 and SW23. If we increase the duty cycle(press SW22), the speed of the motor increases and if we decrease the duty cycle(press SW23), the speed of the motor decreases.

PROGRAM:

```
#include <LPC214x.H>
void delay_led(unsigned long int); // Delay Time Function
int main(void)
{

IO1DIR = 0xC0000000;
IO0DIR = 0x00200000;
while(1) // Loop Continue
{
IO0SET = 0x00200000;
delay_led(15000);
IO1SET = 0x80000000;
IO1CLR = 0x40000000; // Clear Pin P0.7,6,5,4 (ON LED)
delay_led(1500000); // Display LED Delay
IO1SET = 0x40000000;
IO1CLR = 0x80000000; // Set Pin P0.7,6,5,4 (OFF LED)
delay_led(1500000); // Display LED Delay
}
}
/***********************/
/* Delay Time Function */
/**********************/
void delay_led(unsigned long int count1)
{
while(count1 > 0) {count1--;} // Loop Decrease Counter
}
```

Program No. 2 Date:

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

How Stepper Motors Work?

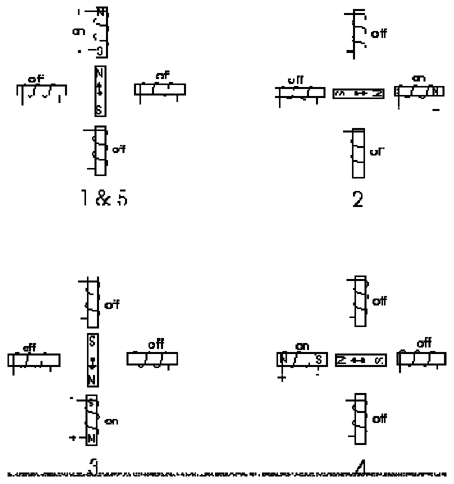
Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Figure 1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

What are stepper motors good for?

□ Positioning – Since steppers move in precise repeatable steps, they excel in applications requiring precise positioning such as 3D printers, CNC, Camera platforms and X,Y Plotters. Some disk drives also use stepper motors to position the read/write head.

□ Speed Control – Precise increments of movement also allow for excellent control of rotational speed for process automation and robotics.

□ Low Speed Torque - Normal DC motors don't have very much torque at low speeds. A Stepper motor has maximum torque at low speeds, so they are a good choice for applications requiring low speed with high precision.



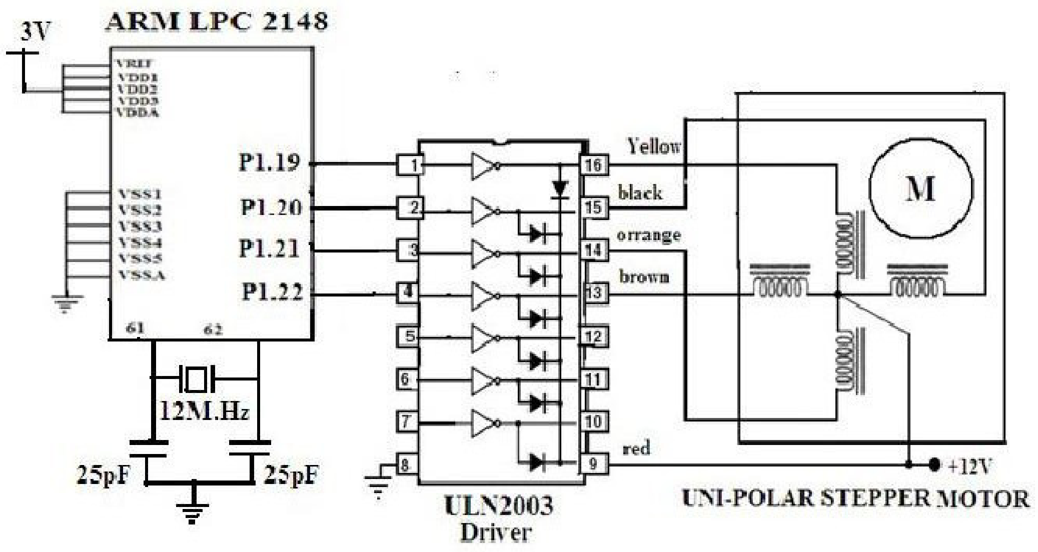
In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 1.8 degrees would move its rotor 1.8 degrees per step, thereby requiring 200 pulses (steps) to complete a full 360 degree rotation.
Here we are using 200 pole stepper motor hence it gives 360degree/200 pole=1.8 degree per step.
So for example if we need 120 degree rotation then we have to apply approximately 67 pulses to complete 120 degree rotation
120/1.8=66.66==67 steps approximately.
Here one cycle means 4 steps. So if we need 90 degree rotation then 90/1.8=50 steps.
Here one cycle means 4 steps. So 50/4=12.5 =~ 13. So we need 13 cycles to rotate 90 degree.
If we want to run 180 degree then 180/1.8=100. So 100/4=25 cycles would make a stepper motor to rotate 180 degree.

PROGRAM:

```
#include <LPC214X.h>

void delay();


void delay()
{
  int i,j;
  for (i=0; i<0xff; i++)
     for (j=0; j<0xff; j++);
}


int main()
{

  IO0DIR=0x000F0000;              //Consider ARM port Pin from 16-19
                                  //And set these pins
  while (1)
  {
  //while (IO0PIN & 0x00008000);
  //while (! (IO0PIN & 0x00008000));
IO0PIN=0x00010000;
delay ();
IO0PIN=0x00020000;
delay ();
IO0PIN=0x00040000;
delay ();
IO0PIN=0x00080000;
delay();


 }
}
```

; This is for Clock wise rotation

; For Anti- Clock wise Change the direction as 8,4,2,1



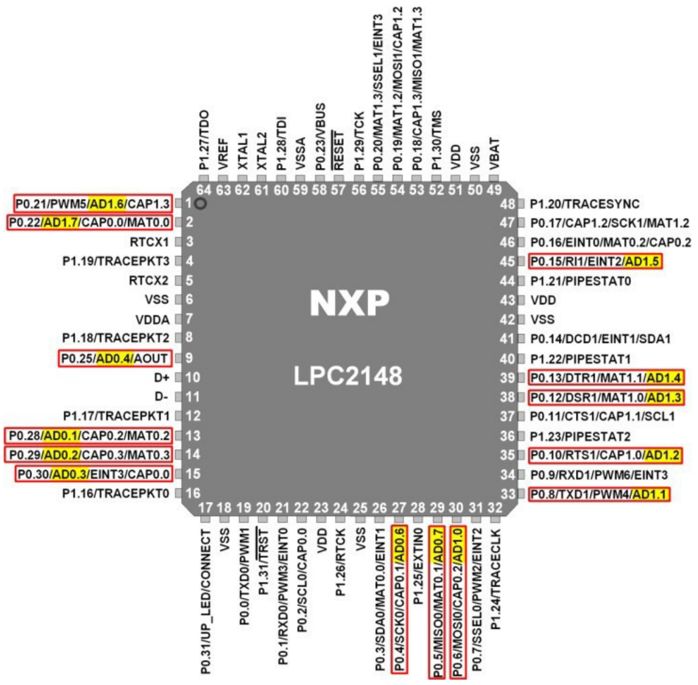
Program No. 3 Date:

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

Analog to Digital Converter(ADC) is used to convert analog signal into digital form. LPC2148 has two inbuilt 10-bit ADC i.e. ADC0 & ADC1.

- ADC0 has 6 channels &ADC1 has 8 channels.
- Hence, we can connect 6 distinct types of input analog signals to ADC0 and 8 distinct types of input analog signals to ADC1.
- ADCs in LPC2148 use Successive Approximation technique to convert analog signal into digital form.
- This Successive Approximation process requires a clock less than or equal to 4.5 MHz. We can adjust this clock using clock divider settings.
- Both ADCs in LCP2148 convert analog signals in the range of 0V to VREF (typically 3V; not to exceed VDDA voltage level).

LPC 2148 ADC Pins



AD0.1:4, AD0.6:7 & AD1.7:0 (Analog Inputs)

These are Analog input pins of ADC. If ADC is used, signal level on analog pins must not be above the level of VDDA; otherwise, ADC readings will be invalid. If ADC is not used, then the pins can be used as 5V tolerant digital I/O pins.

VREF (Voltage Reference)

Provide Voltage Reference for ADC.

VDDA& VSSA (Analog Power and Ground)

These are the power and ground pins for ADC. These should be same as VDD & VSS.

Let's see the ADC registers which are used to control and monitors the ADC operation.

Here, we will see ADC0 registers and their configurations. ADC1 has similar registers and can be configured in a similar manner.

ADC0 Registers

1. AD0CR (ADC0 Control Register)

- AD0CR is a 32-bit register.
- This register must be written to select the operating mode before A/D conversion can occur.
- It is used for selecting channel of ADC, clock frequency for ADC, number of clocks or number of bits in result, start of conversion and few other parameters.

AD0CR (ADC0 Control Register)

- Bits 7:0 – SEL

  These bits select ADC0 channel as analog input. In software-controlled mode, only one of these bits should be 1.e.g. bit 7 (10000000) selects AD0.7 channel as analog input.

- Bits 15:8 – CLKDIV

  The APB(ARM Peripheral Bus)clock is divided by this value plus one, to produce the clock for ADC.
  This clock should be less than or equal to 4.5MHz.

- Bit 16 – BURST

  0 = Conversions are software controlled and require 11 clocks
  1 = In Burst mode ADC does repeated conversions at the rate selected by theCLKS field for the analog inputs selected by SEL field. It can be terminated by clearing this bit, but the conversion that is in progress will be completed.

  When Burst = 1, the START bits must be 000, otherwise the conversions will not start.

- Bits 19:17 – CLKS
- Selects the number of clocks used for each conversion in burst mode and the number of bits of accuracy of Result bits of AD0DR.
  e.g. 000 uses 11 clocks for each conversion and provide 10 bits of result in corresponding ADDR register.

000 =	11	clocks	/	10	bits
001 =	10	clocks	/	9	bits
010 =	9	clocks	/	8	bits
011 =	8	clocks	/	7	bits
100 =	7	clocks	/	6	bits
101 =	6	clocks	/	5	bits
110 =	5	clocks	/	4	bits

111 = 4 clocks / 3 bits

- Bit 20 – RESERVED

- Bit 21 – PDN
  0 = ADC is in Power Down mode
  1 = ADC is operational
- Bit 23:22 – RESERVED
- Bit 26:24 – START
  When BURST bit is 0, these bits control whether and when A/D conversion is started

  000 = No start (Should be used when clearing PDN to 0)
  001 = Start conversion now

  010 = Start conversion when edge selected by bit 27 of this register occurs on CAP0.2/MAT0.2 pin

  011= Start conversion when edge selected by bit 27 of this register occurs on CAP0.0/MAT0.0 pin

  100 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.1 pin

  101 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.3 pin

  110 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.0 pin

  111 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.1 pin

- Bit 27 – EDGE

- This bit is significant only when the Start field contains 010-111. In these cases,
- 0 = Start conversion on a rising edge on the selected CAP/MAT signal
  1 = Start conversion on a falling edge on the selected CAP/MAT signal
- Bit 31:28 – RESERVED

2. AD0GDR (ADC0 Global Data Register)

- AD0GDR is a 32-bit register.
- This register contains the ADC's DONE bit and the result of the most recent A/D conversion.

AD0GDR (ADC0 Global Data Register)

- Bit 5:0 – RESERVED
- Bits 15:6 – RESULT
  When DONE bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- Bit 23:16 – RESERVED
- Bits 26:24 – CHN
  These bits contain the channel from which ADC value is read.
  e.g. 000 identifies that the RESULT field contains ADC value of channel 0.
- Bit 29:27 – RESERVED
- Bit 30 – Overrun
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the RESULT bits.
  This bit is cleared by reading this register.
  Bit 31 – DONE
  This bit is set to 1 when an A/D conversion completes. It is cleared when

this register is read and when the AD0CR is written. If AD0CR is written while a conversion is still in progress, this bit is set and new conversion is started.

3. ADGSR (A/D Global Start Register)

- ADGSR is a 32-bit register.
- Software can write to this register to simultaneously start conversions on both ADC.

ADGSR (A/D Global Start Register)

- BURST (Bit 16), START (Bit <26:24>) & EDGE (Bit 27) These bits have same function as in the individual ADC control registers i.e. AD0CR & AD1CR. Only difference is that we can use these function for both ADC commonly from this register.

4. AD0STAT (ADC0 Status Register)

- AD0STAT is a 32-bit register.
- It allows checking of status of all the A/D channels simultaneously.

AD0STAT (ADC0 Status Register)

- Bit 7:0 – DONE7:DONE0
  These bits reflect the DONE status flag from the result registers for A/D channel 7 - channel 0.
- Bit 15:8 – OVERRUN7:OVERRUN0
  These bits reflect the OVERRUN status flag from the result registers for A/D channel 7 - channel 0.
- Bit 16 – ADINT
  This bit is 1 when any of the individual A/D channel DONE flags is asserted and enables ADC interrupt if any of interrupt is enabled in AD0INTEN register.
- Bit 31:17 – RESERVED

5. AD0INTEN (ADC0 Interrupt Enable)

- AD0INTEN is a 32-bit register.
- It allows control over which channels generate an interrupt when conversion is completed.

AD0INTEN (ADC0 Interrupt Enable)

- Bit 0 – ADINTEN0
  0 = Completion of a A/D conversion on ADC channel 0 will not generate an interrupt
  1 = Completion of a conversion on ADC channel 0 will generate an interrupt
- Remaining ADINTEN bits have similar description as given for ADINTEN0.
- Bit 8 – ADGINTEN
  0 = Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts
  1 = Only the global DONE flag in A/D Data Register is enabled to generate an interrupt

6. AD0DR0-AD0DR7 (ADC0 Data Registers)

- These are 32-bit registers.
- They hold the result when A/D conversion is completed.
- They also include flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

AD0 Data Registers Structure

- Bit 5:0 – RESERVED
- Bits 15:6 – RESULT
  When DONE bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).

- Bit 29:16 – RESERVED
- Bit 30 – Overrun
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in theRESULT bits.
  This bit is cleared by reading this register.
- Bit 31 – DONE
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

Steps for Analog to Digital Conversion

1. Configure the ADxCR (ADC Control Register) according to the need of application.
2. Start ADC conversion by writing appropriate value to START bits in ADxCR. (Example, writing 001 to START bits of the register 26:24, conversion is started immediately).
3. Monitor the DONE bit (bit number 31) of the corresponding ADxDRy (ADC Data Register) till it changes from 0 to 1. This signals completion of conversion. We can also monitor DONE bit of ADGSR or the DONE bit corresponding to the ADC channel in the ADCxSTAT register.
4. Read the ADC result from the corresponding ADC Data Register. ADxDRy. E.g. AD0DR1 contains ADC result of channel 1 of ADC0.

PROGRAM:

```
        #include<LPC214X.H>
        /*..........................................................................................
        MACRO FOR ADC
        ...........................................................................................*/
    #define ch (1 << 3)
        #define clk_div (3 << 8)
        #define bst_on (1 << 16)
//#define bst_off (0 << 16)
        #define clk_res (0 << 17)
        #define operational (1 << 21)
        #define start (0 << 24)
        #define adc_init_macro ch | clk_div | bst_on | clk_res |
operational | start
        /*...........................................................................
        MACRO FOR LCD
        ...........................................................................*/
        #define EN (1 << 28)
        #define RW (1 << 29)
        #define RS (1 << 22)
        #define DATA (0Xff << 6)
        #define port  EN | RW | RS | DATA

/*...........................................................................
FUNCTION DECLARATIONS
...........................................................................*/
                void adc_init(void);
                void delay(int count);
                void cmd(int c);
                void data(char d);
                void lcd_string(char *str);
                void display(unsigned int n);

/*...........................................................................
                GLOBAL VARIABLES
...........................................................................*/
                unsigned int result;
            float voltage;
                char volt[18];
/*...........................................................................
                FUNCTION DEFINITIONS
...........................................................................*/
                void adc_init(void)
                {
                        AD0CR = adc_init_macro;
                }
                void cmd(int c)
                {
                        IOPIN0 = c << 6;
                        IOCLR0 = RW;
                        IOCLR0 = RS;
                        IOSET0 = EN;
                        delay(100);
                        IOCLR0  =  EN;
                }
                void data(char d)
                {
                        IOPIN0 = d << 6;
                        IOCLR0 = RW;
                        IOSET0 = RS;
                        IOSET0 = EN;
                        delay(100);
                        IOCLR0 =  EN;
                }

                void lcd_string(char *str)
                {
                        while(*str)
                        {
                                data(*str);
                                str++;
                                delay(20);
                        }
                }
                void display(unsigned int n)
                {
                        if(n == 0)
                                data(n+0x30);
                        if(n)
                        {
                                display(n / 10);
                                data((n % 10) + 0x30);
                        }
                }

                void delay(int count)
                {
                        int i,j;
                        for(i = 0;i < count;i++)
                                for(j = 0;j < 5000;j++);
                }
/*------------------------------------------------------------------------------------------------
MAIN
------------------------------------------------------------------------------------------------*/
                        int main()
                        {
                                int c = 0;
                                IODIR0 |= port ;
                                PINSEL1|=0x10000000;
                                cmd(0x38);
                                cmd(0x0E);
                                cmd(0X80);
                                cmd(0X01);
                                adc_init();
                                lcd_string("ADC PROGRAM");
                                cmd(0X01);
                                while(1)
                                {
                                        cmd(0x01);
                                        while((AD0DR3 & (0x80000000)==0));
                                        result = (AD0DR3 & (0X3FF << 6));
                                        result = result >> 6;
                                        lcd_string("ADC:");
                                        cmd(0x86);
                                        display(result);
                                        voltage = ( (result/1023.0) * 3.3 );
                                        cmd(0xc0);
                                        sprintf(volt, "Voltage=%.2f V ", voltage);
                                        lcd_string(volt);
                                        //delay(1000);
                                }
                        }
```

Program No. 4

Aim: Interface a DAC and generate Triangular and Square waveforms.

Digital to Analog Converter (DAC) are mostly used to generate analog signals (e.g. sine wave, triangular wave etc.) from digital values.

- LPC2148 has 10-bit DAC with resistor string architecture. It also works in Power down mode.
- LPC2148 has Analog output pin (AOUT) on chip, where we can get digital value in the form of Analog output voltage.
- The Analog voltage on AOUT pin is calculated as ((VALUE/1024) * VREF). Hence, we can change voltage by changing VALUE(10-bit digital value) field in DACR (DAC Register).
- e.g. if we set VALUE = 512, then, we can get analog voltage on AOUT pin as ((512/1024) * VREF) = VREF/2.

AOUT (Analog Output)

This is Analog Output pin of LPC2148 DAC peripheral where we can get Analog output voltage from digital value.

VREF (Voltage Reference)

Provides Voltage Reference for DAC.

VDDA& VSSA (Analog Power and Ground)

These are the power and ground pins for DAC. These should be same as VDD& VSS.

Let's see the Register used for DAC

DACR (DAC Register)

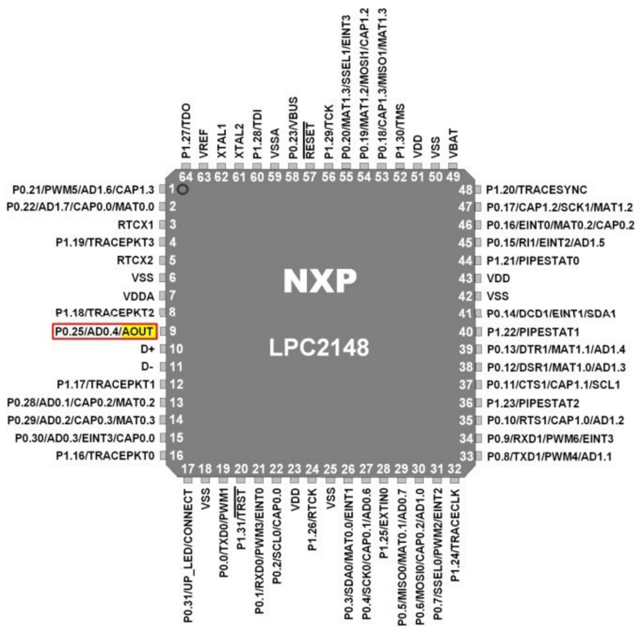
- DACR is a 32-bit register.
- It is a read-write register.

DACR (DAC Register)

- Bit 5:0 – RESERVED
- Bits 15:6 – VALUE
  This field contains the 10-bit digital value that is to be converted in to Analog voltage. We can get Analog output voltage on AOUT pin and it is calculated with the formula (VALUE/1024) * VREF.
- Bit 16 – BIAS
  0 = Maximum settling time of 1µsec and maximum current is 700µA
  1 = Settling time of 2.5µsec and maximum current is 350µA
  Note that, the settling times are valid for a capacitance load on the AOUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time.
- Bit 31:17 – RESERVED

Programming Steps

- First, configure P0.25/AOUT pin as DAC output using PINSEL Register.
- Then set settling time using BIAS bit in DACR Register.
- Now write 10-bit value (which we want to convert into analog form) in VALUE field of DACR Register.



Refer for the program https://www.electronicwings.com/arm7/lpc2148-dac-digital-to-analog-converter

PROGRAM:

SQUARE WAVE PROGRAM

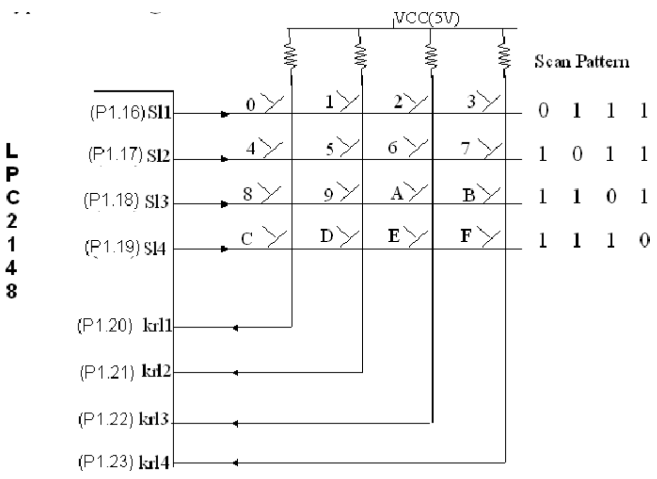
```
#include "LPC214X.h"

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;
       }
    }
 }

}
```

TRIANGLE WAVE PROGRAM

```
#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--;
                              }
 }

}
```

Program No. 5 Date:

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



PROGRAM:

```
#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]);
              }
 }

}
```

Program No. 6 Date:

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

```
#include <LPC214x.H> // LPC2148 MPU Register
/* pototype section */
void delay_led(unsigned long int); // Delay Time Function
int main(void)
{

IO1DIR = 0x00FF0000; // Set GPIO0.7,6,5,4 = Output
// Loop Blink LED on GPIO0.16 //
while(1) // Loop Continue
{
IO1CLR = 0x00FF0000; // Clear Pin P0.7,6,5,4 (ON LED)
delay_led(150000); // Display LED Delay
IO1SET = 0x00FF0000; // Set Pin P0.7,6,5,4 (OFF LED)
delay_led(150000); // Display LED Delay
}
}
/***********************/
/* Delay Time Function */
/***********************/
void delay_led(unsigned long int count1)
{
while(count1 > 0) {count1--;} // Loop Decrease Counter
}
```

OR (FIRST ONE IS EASY)

```
#include <LPC214x.H>
int i;
void init_ext_interrupt(void);
__irq void Ext_ISR(void);
int main (void)
{       init_ext_interrupt();       // initialize the external interrupt
  while (1)
  {
        }
}
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;                    //
  VICIntEnable = (1<<16);                // EINT2 interrupt enabled
  EXTINT &= (0x4);
}
__irq void Ext_ISR(void) // Interrupt Service Routine-ISR
{
        IO1DIR |= (1<<16);
        IO1SET |= (1<<16);        // Turn OFF Buzzer
        for(i=0; i<2000000;i++);
        IO1CLR |= (1<<16);        // Turn ON Buzzer
        EXTINT |= 0x4;            //clear interrupt
        VICVectAddr = 0; // End of interrupt execution
}
```

Program No. 7 Date:

Aim: 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(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x000007E4;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000648;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000618;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000730;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000690;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000680;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x0000063C;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000600;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000630;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000620;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000780;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x000006C4;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x00000708;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x000006C0;
delay_led(15000000);
IO0CLR = 0x00000FFF;
IO0SET  =  0x000006E0;
delay_led(15000000);
IO0CLR  =  0x00000FFF;
}
}
void delay_led(unsigned long int count1)
{
while(count1 > 0) {count1--;}
}
```

REFERENCES:

Textbooks:

1. Andrew N Sloss, Dominic Symes and Chris Wright, ARM system developers guide, Elsevier, Morgan Kaufman publishers, 2008.
2. Shibu K V, “Introduction to Embedded Systems”, Tata McGraw Hill Education, Private Limited, 2nd Edition.

Reference Books:
1. Raghunandan..G.H, Microcontroller (ARM) and Embedded System, Cengage learning
Publication,2019
2. The Insider’s Guide to the ARM7 Based Microcontrollers, Hitex Ltd.,1st edition, 2005.
3. Steve Furber, ARM System-on-Chip Architecture, Second Edition, Pearson, 2015.
4. Raj Kamal, Embedded System, Tata McGraw-Hill Publishers, 2nd Edition, 2008.

VIVA QUESTIONS:

1. What is the processor used by ARM7?
   a) 8-bit CISC
   b) 8-bit RISC
   c) 32-bit CISC
   d) 32-bit RISC

2. What the instruction set used by ARM7?
   a) 16-bit instruction set
   b) 32-bit instruction set
   c) 64-bit instruction set
   d) 8-bit instruction set

3. How many registers are there in ARM7?
   a. 35 register (28 GPR and 7 SPR)
   b. 37 register (31 GPR and 7 SPR)
   c. 37 register (28 GPR and 9 SPR)
   d. 35 register(30 GPR and 5 SPR)

Explanation: ARM7TDMI has 37 registers(31 GPR and 6 SPR).
All these designs use a Von Neumann architecture, thus the few versions comprising a cache do not separate data and instruction caches.

4. ARM7 has an in-built debugging device?
   a. True
   b. False
5. What is the capability of ARM7 instruction for a second?
   a. 110 MIPS
   b. 150 MIPS
   c. 125 MIPS
   d. 130 MIPS

6. We have no use of having silicon customization?
   a. True
   b. False
7. Which of the following has the same instruction set as ARM7?

a. ARMv3
b. ARM71a0
c. ARMv4T

8. What are T,D,M,I stands for in ARM7TDMI?
   a. Timer, Debug ,Multiplexer, ICE
   b. Timer, Debug, Multiplier, ICE
   c. Timer, Debug, Modulation, ICE
   d. Timer, Debug, Multiplexer, IS

9. ARM stands for ----------------
   a. Advanced RISC Machine
   b. Advanced RISC Methodology
   c. Advanced Reduced Machine
   d. Advanced Reduced Methodology

10. What are the profiles for ARM architecture?
   a) A,R
   b) A,M
   c) A,R,M
   d) R,M

11. ARM7DI operates in which mode?
   a) Big Endian
   b) Little Endian
   c) Both big and little Endian
   d) Neither big nor little Endian

12. In which of the following ARM processors virtual memory is present?
   a) ARM7DI
   b) ARM7TDMI-S
   c) ARM7TDMI
   d) ARM7EJ-S

13. How many instructions pipelining is used in ARM7EJ-S?
   a) 3-Stage
   b) 4-Stage
   c) 5-Stage
   d)2-stage

14. How many bit data bus is used in ARM7EJ-s?

    a) 32-bit

    b) 16-bit

    c) 8-bit

    d) Both 16 and 32 bit

15. What is the cache memory for ARM710T?

    a) 12Kb

    b) 16Kb

    c) 32Kb

    d) 8Kb

ADDITIONAL PROGRAMS:

1. Write ARM assembly language program to add two 32 bit numbers.

```
AREA          add32, CODE, READONLY
ENTRY
MAIN
       LDR    R0, =Value1
       LDR    R1,  [R0]
       ADD    R0, R0, #0*4
       LDR    R2,  [R0]
       ADD R1, R1, R2
       LDR    R0, =Result
       STR    R1, [R0]
       SWI    &11; TERMINATION

 Value1    DCD    &37E3C123
 Value2    DCD    &367402AA
  Result   DCD    0
```

2. Write ARM assembly language program to add two 64 bit numbers.

```
            AREA    add64, CODE, READONLY
ENTRY
MAIN
       LDR    R0, =Value1        ;pointer to first value
       LDR    R1, [R0]           ;load first part of value1
       LDR    R2, [R0, #4]        ; load lower part of value1
       LDR    R0, =Value2        ;pointer to second value
       LDR    R3, [R0]           ;load upper part of value2
       LDR    R4, [R0, #4]        ; load lower part of value2
       ADDS    R6, R2, R4         ;add lower 4 bytes and set carry flag
       ADC    R5, R1, R3        ;add upper 4 bytes including carry
       LDR    R0, =Result        ;pointer to result
       STR    R5, [R0]           ;store upper part of result
       STR    R6, [R0, #4]        ;store lower part of result
        SWI    &11
Value1    DCD    &12A2E640, &F2100123
Value2    DCD    &001019BF, &40023F51
Result    DCD    0

       END
```

DATA EXCHANGE

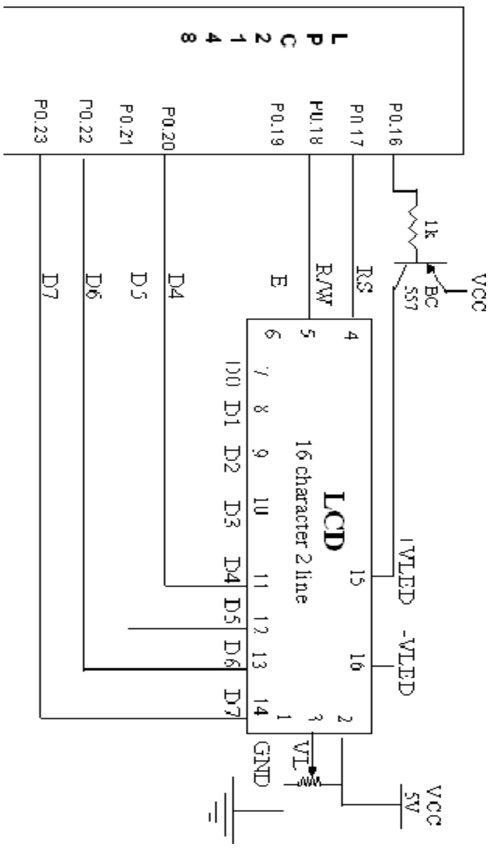
```
   area data_exch,code,readonly
entry
      ldr r0,=0x40000000
      ldr  r1,=0x40000044
      mov r4,#09

loop ldr r2,[r0]
      mov r5,r2
      ldr r6,[r1]
      str r6,[r0],#04
      str r5,[r1],#04
      subs r4,#01
      cmp r4,#00
      bne loop
stop b stop

      end
```

BLOCK OF DATA TRANSFER

```
     area data_trans,code,readonly
entry
             ldr r0,=0x40000000
      ldr r1,=0x40000044
      mov r4,#09

loop     ldr r2,[r0],#04
            str r2,[r1],#04
     subs r4,#01
     cmp r4,#00
     bne loop
stop b stop

          end
```

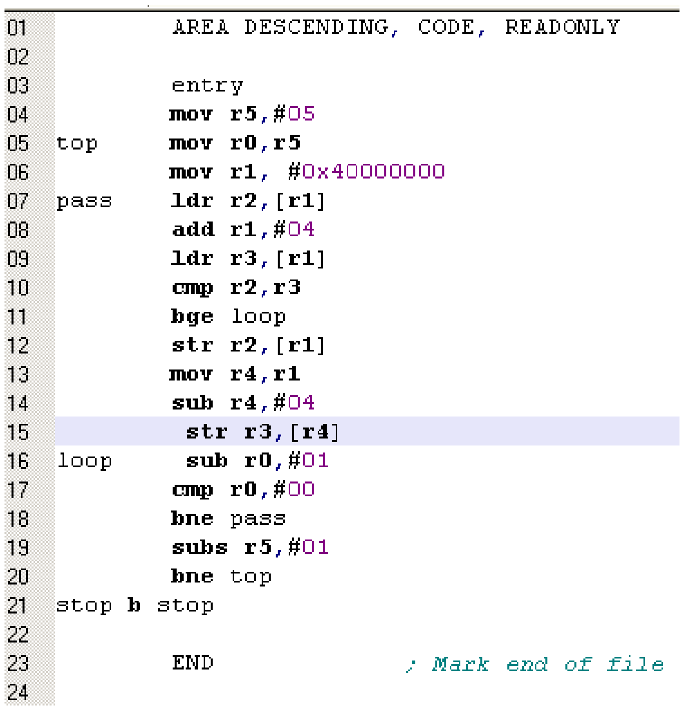
3. To interface LCD with ARM processor-- ARM7TDMI/LPC2148. Write and execute programs in C language for displaying text messages and numbers on LCD

```
#include <LPC214x.h>

void cmd(unsigned char d);
void datal(unsigned char t);
void delay (int count);

int main()
{
int i;
unsigned char name[]={"CBVAR"};

IO0DIR=0x30403C00;
delay(100);

cmd(0x02);                                          //cursor home
command
cmd(0x01);                                          //clear display
command
cmd(0x28);                                          //4-bit mode
entry command(0x38 for 8 bit mode)
cmd(0x06);                                          //entry mode
command
cmd(0x0C);                                          //display on
cursor off command
//cmd(0xC0);                                        //LCD bottom
line display command

for (i=0;i<11;i++)
{
datal(name[i]);
}
while(1);
}

void cmd(unsigned char d)
{
int a=0;
a = d | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET =(IO0SET | 0x00003c00)&a;
delay(1000);
IO0CLR = 0x10000000;

a=0x0;
d=d<<4;
a = d | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET = (IO0SET | 0x00003C00)&a;
delay(1000);
IO0CLR = 0x10000000;
}
void datal(unsigned char t)
{
int b=0;
b = t|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00)&b;
delay(1000);
IO0CLR = 0x10000000;

b=0x0;
t=t<<4;
b=t|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00)&b;
delay(1000);
IO0CLR = 0x10000000;
}
void delay(int count)
{
int j=0, i=0;
for (j=0;j<count;j++)
for (i=0;i<35;i++);
}
```



ASCENDING AND DESCENDING ORDER PROGRAM IN ANOTHER METHOD:

```
AREA DESCENDING, CODE, READONLY

            entry
            mov r5,#05
top         mov r0,r5
            mov r1, #0x40000000
pass        ldr r2,[r1]
            add r1,#04
            ldr r3,[r1]
            cmp r2,r3
            bge/ble loop
            str r2,[r1]
            mov r4,r1
            sub r4,#04
            str r3,[r4]
loop        sub r0,#01
            cmp r0,#00
            bne pass
            subs r5,#01
            bne top
      stop b stop

            END                     ; Mark end of file
```



```
        AREA DESCENDING, CODE, READONLY

        entry
        mov r5,#05
top     mov r0,r5
        mov r1, #0x40000000
pass    ldr r2,[r1]
        add r1,#04
        ldr r3,[r1]
        cmp r2,r3
        bge loop
        str r2,[r1]
        mov r4,r1
        sub r4,#04
         str r3,[r4]
loop     sub r0,#01
        cmp r0,#00
        bne pass
        subs r5,#01
        bne top
stop b stop

        END                 ; Mark end of file
```