Shri Madhwa Vadiraja Institute of Technology and Management

Department of Computer Science and Engineering

4th semester BE

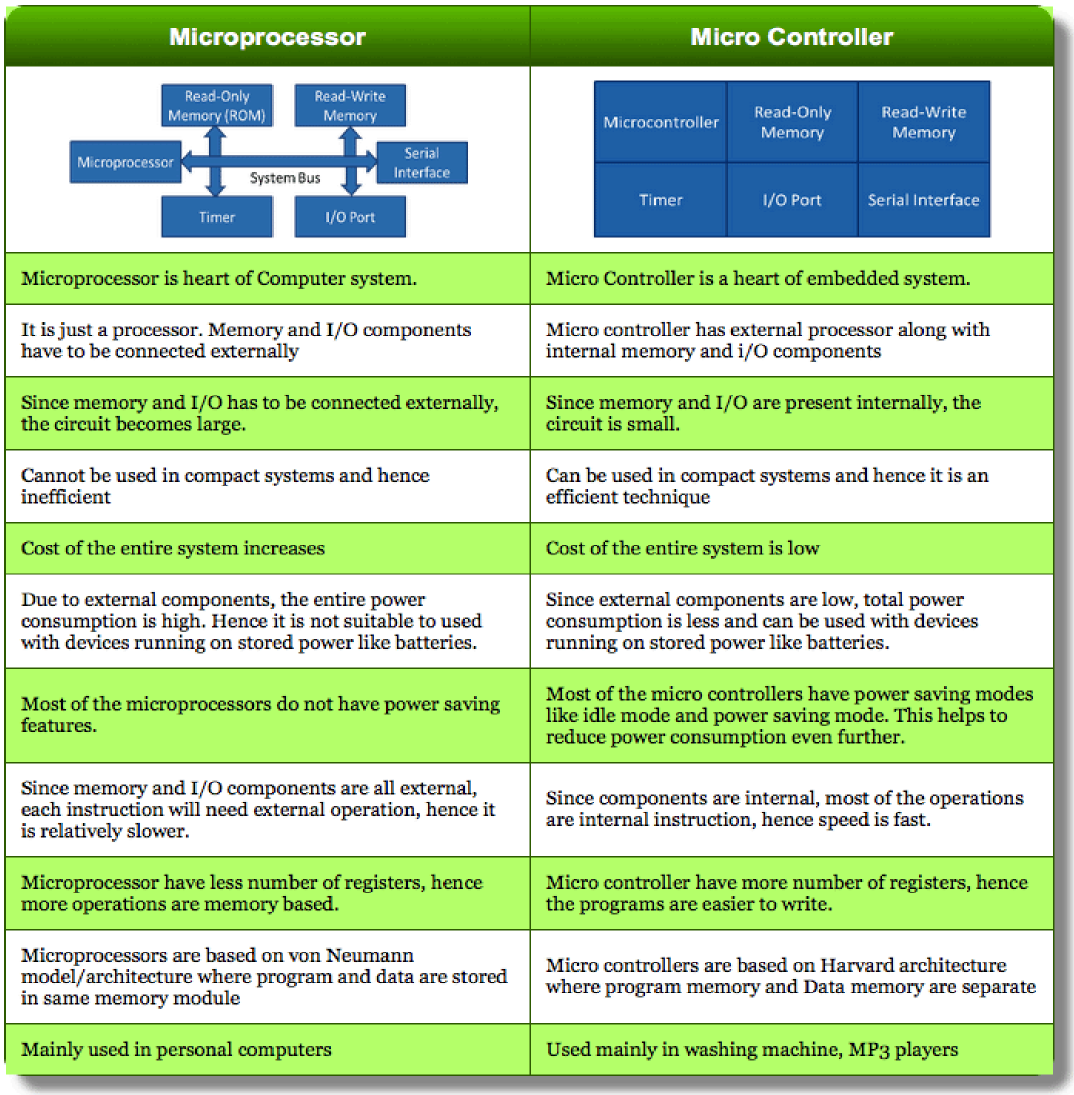
Microcontrollers Laboratory – BCS402

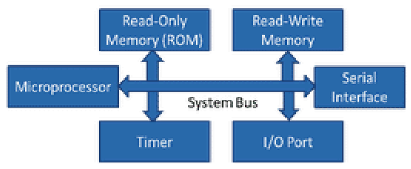
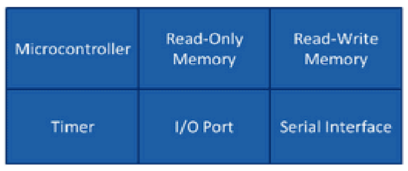
Tutorial - I

Aim:

1. To understand the difference between Microprocessor and Microcontroller
2. To understand the difference between RISC and CISC architecture
3. To understand the ARM Hardware Architecture.

1.1 Difference between Microcontroller and Microprocessor



Microprocessor	Micro Controller
	
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

1.2 Difference between RISC and CISC Architecture

CISC	RISC
A large number of instructions are present in the architecture.	Very fewer instructions are present. The number of instructions; generally less than 100.
Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory.	No instruction with a long execution time due to very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions.
Variable-length encodings of the instructions. **Example:** IA32 instruction size can range from 1 to 15 bytes.	Fixed-length encodings of the instructions are used. **Example:** In IA32, generally all instructions are encoded as 4 bytes.
Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index registers.	Simple addressing formats are supported. Only base and displacement addressing is allowed.
CISC supports array.	RISC does not supports array.
Arithmetic and logical operations can be applied to both memory and register operands.	Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions, i.e. reading from memory into a register and writing from a register to memory respectively.
Implementation programs are hidden from machine level programs. The ISA provides a clean abstraction between programs and how they get executed.	Implementation programs exposed to machine level programs. Few RISC machines do not allow specific instruction sequences.
Condition codes are used.	No condition codes are used.
The stack is being used for procedure arguments and return addresses.	Registers are being used for procedure arguments and return addresses. Memory references can be avoided by some procedures.

1.3 ARM7 Based LPC2148 Microcontroller Architecture

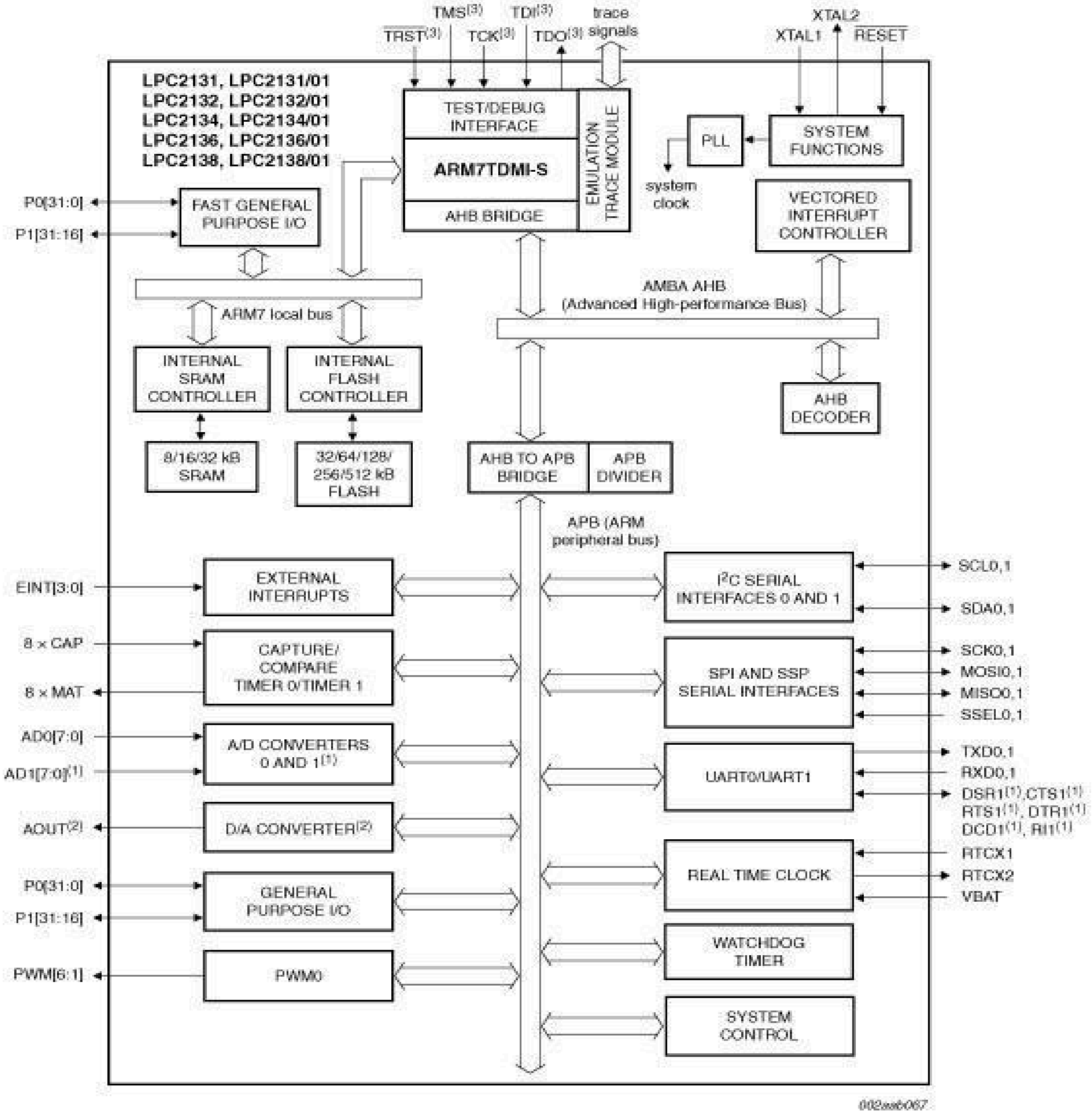


Figure 1: ARM Architecture

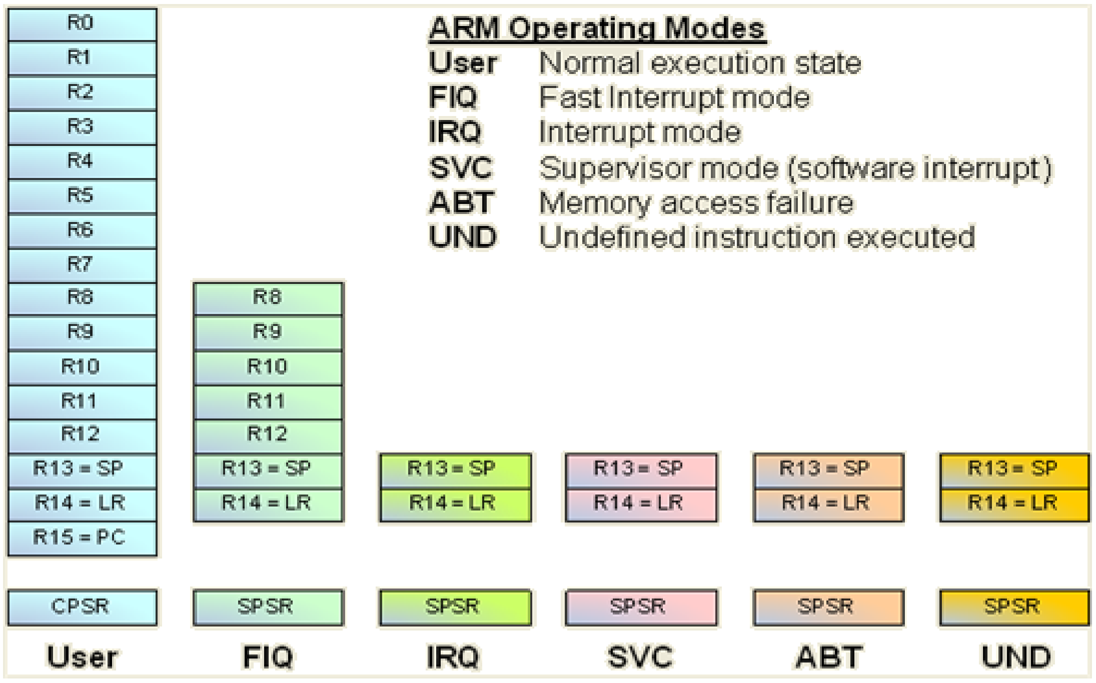


Figure 2: ARM Register Bank

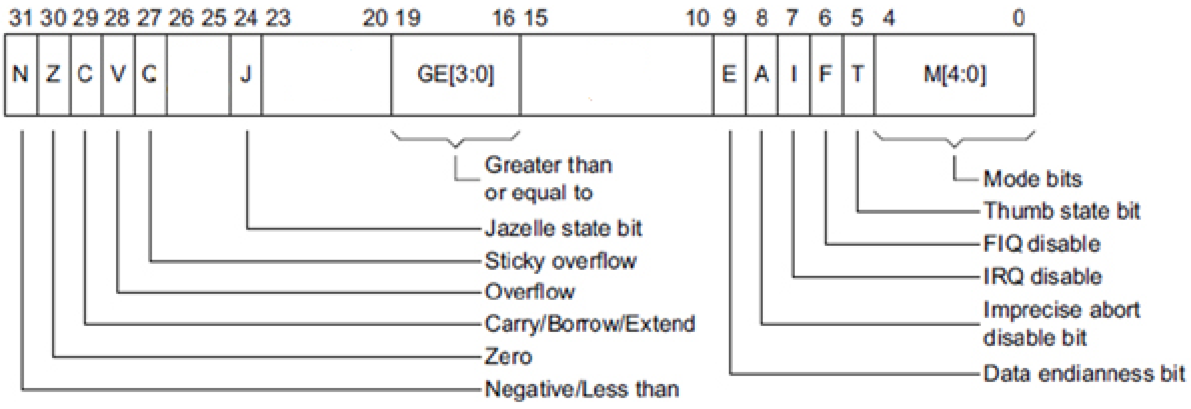


Figure 3: ARM CPSR Register

The principal feature of the ARM 7 microcontroller is that it is a register based load-and-store architecture with a number of operating modes. While the ARM7 is a 32 bit microcontroller, it is also capable of running a 16-bit instruction set, known as ―THUMB‖. While all of the register-to-register data processing instructions are single-cycle, other instructions such as data transfer instructions, are multi-cycle. To increase the performance of these instructions, the ARM 7 has a three-stage pipeline. To assist the developer, the ARM core has a built-in JTAG debug port and on-chip ―embedded ICE that allows programs to be downloaded and fully debugged in-system. ARM processors are typical of RISC (**R**educed **I**nstruction **S**et **C**omputers) processors in that they implement a load and store architecture. Only load and store instructions can access memory. Data processing instructions operate on register contents only.

The ARM (Advanced RISC Machine) has launched several processors which have different features as well as the different cores for a wide variety of applications. The first ARM architecture design had 26-bit processors, but now it has reached 64-bit processors. The general expansion of ARM products cannot be categorized on some particular information. But ARM products can be understood based on its architecture. The standard ARM series processors available in the market are starting from ARM7 to ARM11. These processors have several features like cache, Data Tightly Coupled memory, MPU, MMU, etc. Some of the widely known ARM processor series are ARM926EJ-S, ARM7TDMI, and ARM11 MPCore. This article is particularly intended for ARM7 based LPC2148 microcontroller architecture overview which will give you a brief information about the microcontroller architecture.

ARM7 Based LPC2148 Microcontroller Architecture

The ARM7 is a 32-bit general purpose microprocessor, and it offers some of the features like little power utilization, and high performance. The architecture of an ARM depends on the principles of RISC. The associated decode mechanism, as well as the RISC- instructions set are much easier when we compare with micro-programmed CISC-Complex Instruction Set Computers.

The Pipeline method is used for processing all the blocks in architecture. In general, a single instruction set is being performed, then its descendant is being translated, & a 3rd-instruction is being obtained from the memory.

An exclusive architectural plan of ARM7 is called Thumb, and it is perfectly suitable for high volume applications where the compactness of code is a matter The ARM7 also uses an exclusive architecture namely Thumb. It makes it perfectly suitable for different applications by memory limitations where the density of code is a matter.

Interrupt sources

Every peripheral device consists of a single interrupt line allied to the VIC (vector interrupt controller), although it can have various interrupt flags inside. Individual interrupt flags can also signify one or more interrupt resources.

On-chip Flash Program Memory

The microcontroller LPC2141/42/44/46/48 includes a flash memory like 32-kilobytes, kilobytes, 128-kilobytes, 256-kilobytes respectively. This flash memory can be used for both data storage as well as code. The flash memory programming can be done in the system through the serial port.

The program application may also erase while the application of the program is running, permitting a flexibility of data storage field firmware improvements, etc. Because of the selection of architectural solution for an on-chip bootloader, the available memory for the microcontrollers LPC2141/42/44/46/48 is 32-kilobytes, kilobytes, 128-kilobytes, 256-kilobytes, & 500-kilobytes. The flash memory of these microcontrollers' offers 1, 00,000 erase per cycle and data preservation for many years.

Pin Connect Block

This block permits chosen pins of the LPC2148 microcontroller for having several functions. The multiplexers can be controlled by the configuration registers for allowing the link between the pin as well as on-chip peripherals.

Peripherals must be coupled with the suitable pins previous to being triggered, and previous to any connected interrupts being permitted. The microcontroller functionality can be defined by the pin control module by its pin selection of registers in a given hardware environment.

After rearranging all pins of ports (port 0 & port 1) are arranged as i/p by the given exceptions. If debug is allowed, the pins of the JTAG will guess the functionality of JTAG. If a trace is allowed, then the Trace pins will guess the functionality of trace. The pins connected to the I2C0 and I2C1 pins are open drain.

GPIO- General Purpose Parallel Input/output

GPIO registers control the device pins which are not linked to a particular peripheral function. The device pins can be arranged as i/p[s or o/ps. Individual registers allow for clearing any number of o/p's concurrently. The output register value can be read back, & the present condition of the port pins. These microcontrollers begin an accelerated function over LPC200 devices.

General purpose input/output registers are moved to the processor bus used for the best probable I/O time.

These registers are addressable by bytes.
The total value of a port can be
The complete value of the port can be written in the only instruction

10-bit ADC (Analog to Digital Converter)

The microcontrollers like LPC2141 or 42 include two ADC converters, and these are only 10-bit have one & the LPC2144/46/48 have two ADC's, and these are only 10-bit straight approximation ADC's. Although ADC0 includes 6-channels and ADC1 has 8-channels. Thus, the number of accessible ADC i/ps for LPC2141 or 42 is 6 & 14 for LPC2141 or 42.

10-bit DAC (Digital to Analog Converter)

The DAC allows these microcontrollers to produce a changeable analog o/p, and VREF is the utmost output of a digital to an analog voltage.

Device Controller-USB 2.0

The universal serial bus consists of 4-wires, and that gives the support for communication in between a number of peripherals and the host. This controller allows the bandwidth of USB for connecting devices using a protocol based on the token. The bus supports unplugging, hot plugging and dynamic collection of the devices. Every communication is started through the host-controller. These microcontrollers are designed with a universal serial bus apparatus controller that allows 12 Mbit/sec data replaced by a host controller of USB.

UARTs

These microcontrollers include two UARTs for standard transmit & get data-lines. Contrasted to earlier microcontrollers (LPC2000), UARTs in microcontrollers LPC2141/ LPC2142/ LPC2144/ LPC2146/ LPC2148 initiate a partial baud rate generator used for both UARTs, allowing these types of microcontrollers for achieving typical baud rates like 115200 by every crystal frequency over 2 MHz. Additionally, the control functions like CTS/RTS are completely executed in hardware.

Serial I/O Controller of I2C-bus

Each microcontroller from LPC2141/ LPC2142/ LPC2144/ LPC2146/ LPC2148 includes two I2C bus controllers, and this is bidirectional. The inter-IC control can be done with the help of two wires namely an SCL and SDA. Here the SDA & SCL are serial clock line and the serial data line

Every apparatus is identified by an individual address. Here, transmitters and receivers can work in two modes like master mode/slave mode. This is a multi-master bus, and it can be managed by one or more bus masters linked to it. These microcontrollers' supports up to-400 kbit/s bit rates.

SPI Serial Input/ Output Controller

These microcontrollers include a single SPI controller and intended to handle numerous masters & slaves associated with a specified bus.

Simply a master & a slave can converse over the interface throughout a specified data transmit. During this, the master constantly transmits a byte-of-data toward the slave, as well as the slave constantly transmits data toward the master.

SSP Serial Input/ Output Controller

These microcontrollers contain a single SSP, and this controller is capable of processing on an SPI, Microwire bus or 4-wire SSI. It can communicate with the bus of several masters as well as slaves

But, simply a particular master, as well as slave, can converse on the bus throughout a specified data transmit. This microcontroller supports full-duplex transfers, by 4-16 bits data frames used for the flow of data from the master- the slave as well as from the slave-the master.

Timers/Counters

Timers and counters are designed for counting the PCLK (peripheral clock) cycles & optionally produce interrupts based on 4-match registers.

And it comprises four capture i/ps to catch the value of a timer when an i/p signal changes. Several pins could be chosen to execute a particular capture. These microcontrollers can calculate exterior events on the inputs of capture if the least exterior pulse is equivalent. In this arrangement, idle capture lines can be chosen as usual timer capture i/ps.

Watchdog Timer

The watchdog timer is used for resetting the microcontroller in a reasonable sum of time. When it is allowed then the timer will produce a reset of a system if the consumer program does not succeed to reload the timer in a fixed sum of time.

RTC-Real-time Clock

The RTC is intended for providing counters to calculate the time when idle or normal operating method is chosen. The RTC uses a small amount of power and designed for appropriate battery power-driven arrangements where the central processing unit is not functioning constantly

Power Control

These microcontrollers support two condensed power modes such as power down mode and idle mode. In Idle mode, instruction execution is balanced until an interrupt or RST occurs. The functions of the peripheral maintain operation throughout idle mode & can produce interrupts to cause the CPU to restart finishing. Idle mode removes the power utilized by the CPU, controllers, memory systems and inner buses.

In power down mode, the oscillator is deactivated and the IC gets no inner clocks. The peripheral registers, processor condition with registers, inner SRAM values are conserved during Power-down mode & the chip logic levels output pins stay fixed.

This mode can be finished and the common process restarted by specific interrupts that are capable to work without clocks. Because the chip operation is balanced, Power-down mode decreases chip power utilization to almost zero.

PWM -Pulse Width Modulator

The PWMs are based on the normal timer-block & also come into all the features, though simply the pulse width modulator function is fixed out on the microcontrollers like LPC2141/42/44/46/48.

The timer is intended to calculate PCLK (peripheral clock) cycles & optionally produce interrupts when particular timer values arise based on 7-match registers, and PWM function also depends on match register events.

The capability of individually controlling increasing & decreasing boundary positions allows the pulse width modulation to be utilized for several applications. For example, the typical motor control with multi-phase uses 3-non-overlapping outputs of PWM by separate control of every pulse widths as well as positions.

VPB Bus

The VPB divider resolves the association between the CCLK (processor clock) and the PCLK (clock used by peripheral devices). This divider is used for two purposes. The first use is to supply peripherals by the preferred PCLK using VPB bus so that they can work at the selected speed of the ARM processor. In order to accomplish this, this bus speed can be reduced the clock rate of the processor from 1⁄2 -1⁄4.

Because this bus must work accurately at power-up, and the default state at RST (reset) is for the bus to work at 1⁄4th of the processor clock rate. The second use of this is to permit power savings whenever an application doesn't need any peripherals to work at the complete processor rate. Since the VPB-divider is associated with the output of PLL, this remains active throughout an idle mode.

Emulation & Debugging

The microcontroller (LPC2141/42/44/46/48) holds emulation & debugging through serial port-JTAG.A trace-port permit tracing the execution of the program. Trace functions & debugging concepts are multiplexed with port1 and GPIOs.

Code Security

The code security feature of these microcontrollers LPC2141/42/44/46/48 permits a function to control whether it can be protected or debugged from inspection.

Thus, this is all about ARM7 based LPC2148 microcontroller architecture. From the above article, finally, we can conclude that ARM is an architecture used in numerous processors as well as microcontrollers.

Tutorial - II

Aim:

1. To understand the ARM instruction set
2. To understand ARM condition mnemonics
3. Introduction to KEIL Software
4. Learn to create and debug a program KEIL

2.1 Mnemonic Instruction Set:

Sl.No.	Mnemonic	Instruction	Action
1	ADC	Add with carry	Rd := Rn + Op2 + Carry
2	ADD	Add	Rd := Rn + Op2
3	AND	AND	Rd := Rn AND Op2
4	B	Branch	R15 := address
5	BIC	Bit Clear	Rd := Rn AND NOT Op2
6	BKPT	break point instructions	
7	BL	Branch with Link	R14 := R15, R15 := address
8	BLX	branch with link and exchange	
9	BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]
10	CPD CPD2	Coprocessor Data Processing	(Coprocessor-specific)
11	CLZ	count leading zeros	
12	CMN	Compare Negative	CPSR flags := Rn + Op2
13	CMP	Compare	CPSR flags := Rn - Op2
14	EOR	Exclusive OR	Rd := (Rn AND NOT Op2)
15	LDC LDC2	Load coprocessor from memory	Coprocessor load
16	LDM	Load multiple registers	Stack manipulation (Pop)
17	LDR	Load register from memory	Rd := (address)
18	MCR MCR2 MCRR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}
19	MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn
20	MOV	Move register or constant	Rd : = Op2
21	MRC MRC2 MRRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}
22	MRS	Move PSR status/flags to register	Rn := PSR
23	MSR	Move register to PSR status/flags	PSR := Rm
24	MUL	Multiply	Rd := Rm * Rs
25	MVN	Move negative register	Rd := 0xFFFFFFFF EOR Op2
26	ORR	OR	(op2 AND NOT Rn)
27	PLD	preload hint instruction	
28	QADD	signed saturated 32 - bit add	
29	QDADD	signed saturated double and 32 - bit add	
30	QDSUB	signed saturated double and 32 - bit subtract	
31	QSUB	signed saturated 32 - bit subtract	
32	RSB	Reverse Subtract	Rd := Op2 - Rn

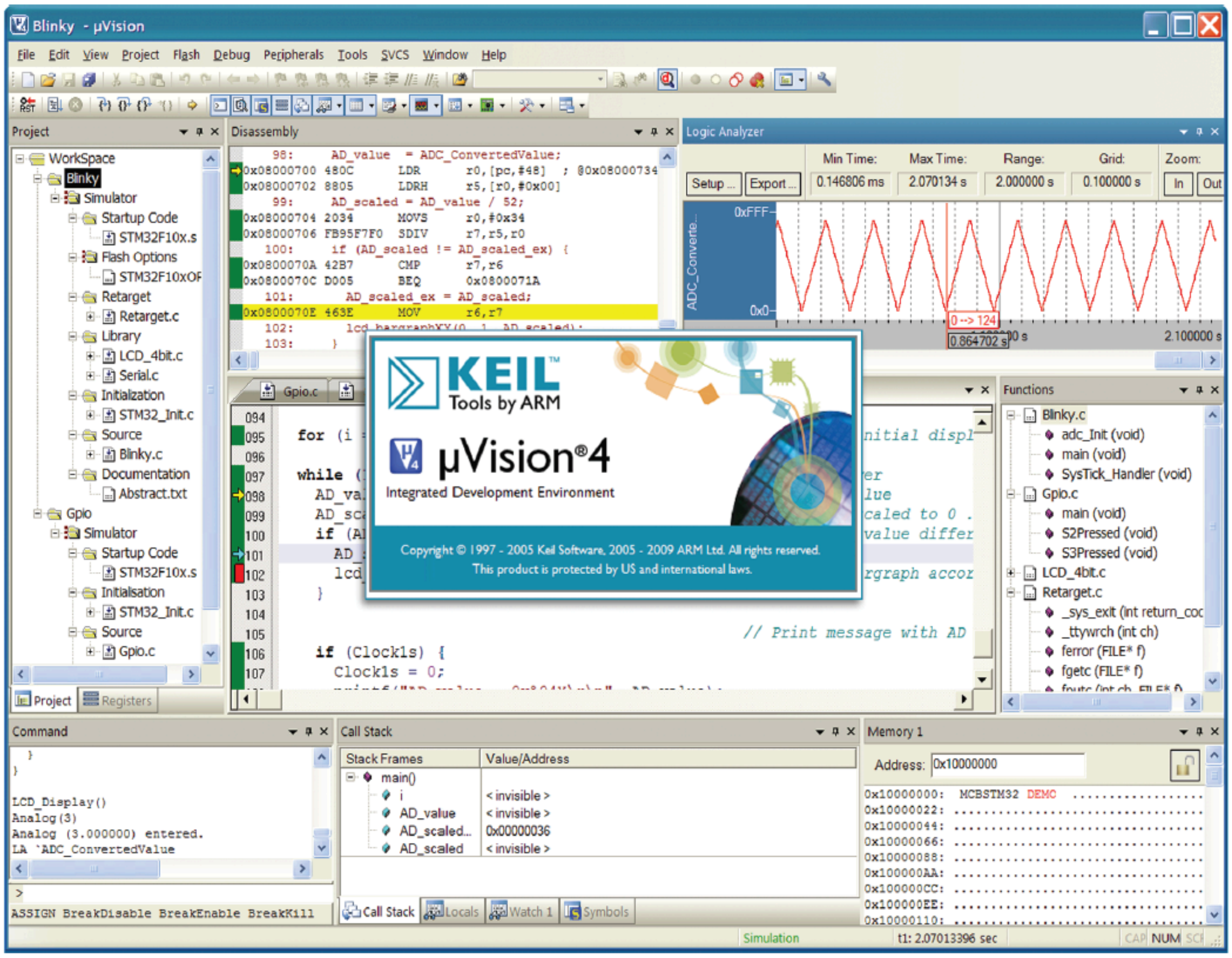
33	RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry
34	SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry
35	SMLAxy	signed multiply accumulate instructions ((16 x 16) + 32 = 32-bit)	
36	SMALL	signed multiply accumulate long ((32 x 32) + 64 = 64-bit))	
37	SMLALxy	signed multiply accumulate long ((16 x 16) + 32 = 64-bit))	
38	SMLAWy	signed multiply accumulate instruction ((32 x 16) ^ 16) + 32 = 32 - bit))	
39	SMULL	signed multiply long(32 x 32) = 64 - bit)	
40	SMULxy	signed multiply instructions(16 x 16) = 32 - bit)	
41	SMULWy	signed multiply instructions((32 x 16) ^ 16 = 32 - bit)	
42	STC STC2	store to memory single or multiple 32 - bit values from coprocessor	
	STC	Store coprocessor register to memory	address := CRn
43	STM	Store Multiple	Stack manipulation (Push)
44	STR	Store register to memory	<address> := Rd
45	SUB	Subtract	Rd := Rn - Op2
46	SWI	Software Interrupt	OS call
47	SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm
48	TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2
49	TST	Test bits	CPSR flags := Rn AND Op2
50	UMLAL	unsigned multiply accumulate long ((32 x 32) + 64 = 64-bit)	
51	UMULL	unsigned multiply long ((32 x 32) = 64 -bit)	

2.2 Condition mnemonics:

Mnemonic	Name
EQ	Equal
NE	Not equal
CS/HS	Carry set / unsigned higher or same
CC/LO	Carry clear / unsigned lower
MI	Minus / negative
PL	Plus / positive or zero
VS	Overflow / overflow set
VC	No overflow / overflow clear
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL (or none)	Always / unconditional

Introduction KEIL Software

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



```
    98:      AD_value  = ADC_ConvertedValue;
0x08000700 480C      LDR      r0,[pc,#48]  ; @0x08000734
0x08000702 8805      LDRH     r5,[r0,#0x00]
    99:      AD_scaled = AD_value / 52;
0x08000704 2034      MOVS     r0,#0x34
0x08000706 FB95F7F0  SDIV     r7,r5,r0
   100:      if (AD_scaled != AD_scaled_ex) {
0x0800070A 42B7      CMP      r7,r6
0x0800070C D005      BEQ      0x0800071A
   101:        AD_scaled_ex = AD_scaled;
0x0800070E 463E      MOV      r6,r7
```

```
  }
}

LCD_Display()
Analog(3)
Analog (3.000000) entered.
LA `ADC_ConvertedValue
```

```
ASSIGN BreakDisable BreakEnable BreakKill
```

Stack Frames	Value/Address
main()	
i	< invisible >
AD_value	< invisible >
AD_scaled...	0x00000036
AD_scaled	< invisible >

Copyright © 1997 - 2005 Keil Software, 2005 - 2009 ARM Ltd. All rights reserved.
This product is protected by US and international laws.

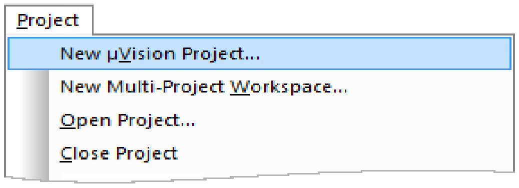
Fig 2: KIEL Software Layout

Creating Application using KEIL µVision5

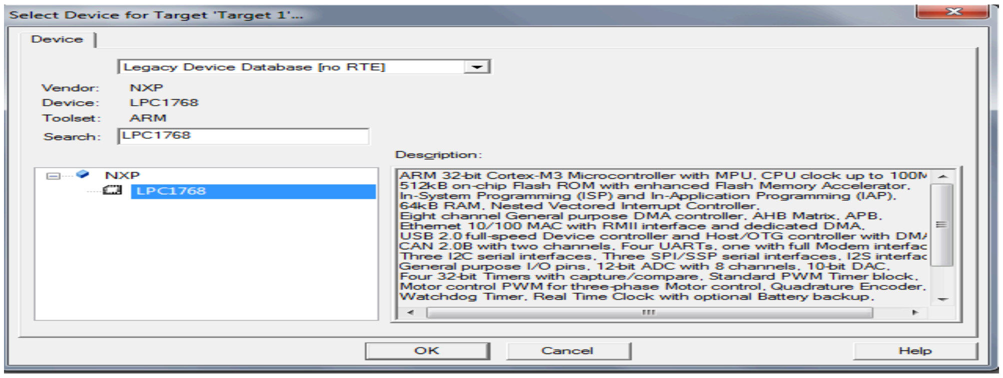
The required steps for creating application programs are listed below:

1. 1Select **Project - New Project** from the µVision5 menu.

This opens a standard Windows dialog, which prompts you for the new project file name.



2. Create a new folder with a name **SMVITM** and while selecting target Select the device LPC2148 and click OK as shown below and press no for LPC2148.s file option


Select Device for Target 'Target 1'...
Device
Legacy Device Database [no RTE]
NXP
LPC1768
OK
Cancel
Help

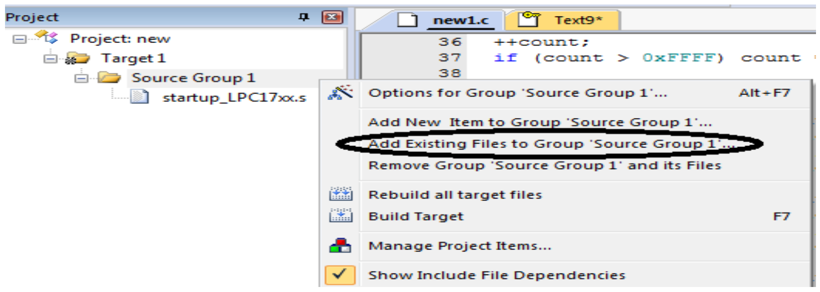
Vendor: NXP
Device: LPC1768
Toolset: ARM
Search: LPC1768

Description:

ARM 32-bit Cortex-M3 Microcontroller with MPU, CPU clock up to 100M
512kB on-chip Flash ROM with enhanced Flash Memory Accelerator,
In-System Programming (ISP) and In-Application Programming (IAP),
64kB RAM, Nested Vectored Interrupt Controller,
Eight channel General purpose DMA controller, AHB Matrix, APB,
Ethernet 10/100 MAC with RMII interface and dedicated DMA,
USB 2.0 full-speed Device controller and Host/OTG controller with DM/
CAN 2.0B with two channels, Four UARTs, one with full Modem interfac
Three I2C serial interfaces, Three SPI/SSP serial interfaces, I2S interfac
General purpose I/O pins, 12-bit ADC with 8 channels, 10-bit DAC,
Four 32-bit Timers with capture/compare, Standard PWM Timer block,
Motor control PWM for three-phase Motor control, Quadrature Encoder,
Watchdog Timer, Real Time Clock with optional Battery backup.

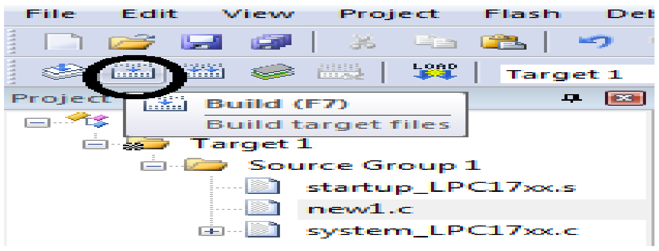
3. Go to the fileNewNow edit the programsavefile_name.c (given extension '.c')

4. Add your file_name.c : right click on source group1 as shown below:


Project
Project: new
Target 1
Source Group 1
startup_LPC17xx.s
new1.c
Text9*
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Rebuild all target files
Build Target F7
Manage Project Items...
Show Include File Dependencies

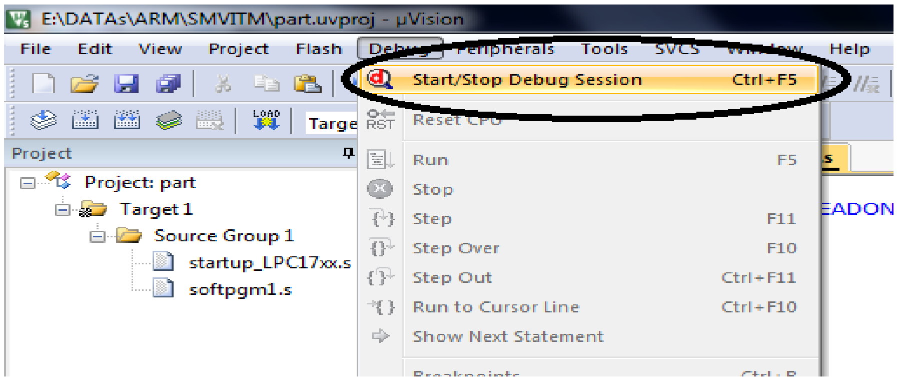
```
++count;
if (count > 0xFFFF) count
```

5. Save the program and build the target as shown below



6. Errors or warnings are displayed in the **Build Output Window**. Double-click on a message to jump to the line where the incident occurred.

7. **For Part-A program execution use Debug option**


E:\DATAs\ARM\SMVITM\part.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Start/Stop Debug Session Ctrl+F5
Reset CPU
Run F5
Stop
Step F11
Step Over F10
Step Out Ctrl+F11
Run to Cursor Line Ctrl+F10
Show Next Statement
Project
Project: part
Target 1
Source Group 1
startup_LPC17xx.s
softpgm1.s

8. **For single step execution press f11 and observe the output**

Course Code	BCS402	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours T + 8-10 Lab Slots	Total Marks	100
Credits	04	Exam Hours	03

Course Objectives:
CLO 1: Understand the fundamentals of ARM-based systems and basic architecture of CISC and RISC.
CLO 2: Familiarize with ARM programming modules along with registers, CPSR and Flags.
CLO 3: Develop ALP using various instructions to program the ARM controller.
CLO 4: Understand the Exceptions and Interrupt handling mechanism in Microcontrollers.
CLO 5: Discuss the ARM Firmware packages and Cache memory policies

Experiments

1. Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).
2. Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program).
3. Develop an ALP to multiply two 16-bit binary numbers.
4. Develop an ALP to find the sum of the first 10 integer numbers.
5. Develop an ALP to find the largest/smallest number in an array of 32 numbers.
6. Develop an ALP to count the number of ones and zeros in two consecutive memory locations.
7. Simulate a program in C for ARM microcontroller using KEIL to sort the numbers in ascending/descending order using bubble sort.
8. Simulate a program in C for an ARM microcontroller to find the factorial of a number.
9. Simulate a program in C for an ARM microcontroller to demonstrate case conversion of characters from upper to lowercase and lower to uppercase.
10. Demonstrate enabling and disabling of Interrupts in ARM.
11. Demonstrate the handling of divide by zero, Invalid Operation and Overflow exceptions in ARM.

Course outcomes (Course Skill Set):
At the end of the course, the student will be able to:
● Explain the ARM Architectural features and Instructions.
● Develop programs using ARM instruction set for an ARM Microcontroller.
● Explain C-Compiler Optimizations and portability issues in ARM Microcontroller.
● Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.
● Demonstrate the role of Cache management and Firmware in Microcontrollers.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The
minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum
passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/course if the student secures a minimum of
40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the practical component of the IPCC
1. 15 marks for the conduction of the experiment and preparation of laboratory record, and 10 marks for the test to be conducted after the completion of all the laboratory sessions.
2. On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
3. The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to 15 marks.
4. The laboratory test (duration 02/03 hours) after completion of all the experiments shall be conducted for 50 marks and scaled down to 10 marks.
5. Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for 25 marks.
6. The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC
Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the
course (duration 03 hours)
1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3
sub-questions), should have a mix of topics under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored by the student shall be proportionally scaled down to 50 Marks.
The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE
component only. Questions mentioned in the SEE paper may include questions from the practical component

Suggested Learning Resources:
Text Books:
1. Andrew N Sloss, Dominic Symes and Chris Wright, ARM system developers guide, Elsevier, Morgan Kaufman publishers, 2008.
Reference Books:

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

Activity Based Learning (Suggested Activities in Class)/ Practical Based Learning
Assign the group task to demonstrate the Installation and working of Keil Software.

Lab program 1 and 2

1. Using Keil software, observe the various Registers, Dump, CPSR, with a simple Assembly Language Programs (ALP).
2. Develop and simulate ARM ALP for Data Transfer, Arithmetic and Logical operations (Demonstrate with the help of a suitable program).

A) To add two words

```
 ; program to add two words
 ; R1,= 0X43210010 + R3,=0X43212102  = R4 = 0x8642212
 ; R0,= 0X1234E640 + R2,=0X12348900  = R5 = 0x24696f40

 AREA ADDITION ,CODE , READONLY
START
        LDR R0,=0x1234e640
        LDR R1,=0x43210010
        LDR R2,=0x12348900
        LDR R3,=0x43212102
        ADDS R4,R1,R3
        ADC R5,R0,R2
        BX LR
 END
```

B) subtraction

```
  ; program to subtract two words

 AREA SUBTRACTION ,CODE , READONLY
START
        LDR R0,=0x1234e640
        LDR R1,=0x43210010
        SUB R2, R1, R0
        SBC R3, R1, R0
        RSB R4, R1, R0
        BX LR
 END
```

C) logical operations

1) AND / OR / MVN instruction

```
    AREA AND , CODE , RAEDONLY
  START
          LDR R0,=0x1234e640
          LDR R1,=0x43210010
          LDR R2,=0x706f
          AND   R4, R0, R1
          ORR    R5,R0, R1
          MVN R6, R2
          BX LR
    END
```

Lab program 3

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

```
; Multiplication of Two 16 Bit binary numbers

; TTL 16 bit MUL

        AREA Program, CODE, READONLY

 ENTRY

                LDR     R0, MEMORY              ; load Address of memory

                LDRH    R1, [R0]        ; load First number

                LDRH    R2, [R0,#2]     ; load Second number

                MUL     R2, R1, R2      ; R2 = R1 x R2

                STRH    R2, [R0,#4]     ; Store the result

                SWI     &11             ; all done

 MEMORY         DCD     0x40000000      ;  Address of First 16 bit number
        END
```

; Before execution

```
; 0x40000000: 11  12   33  34   00  00  00  00
```

; After execution

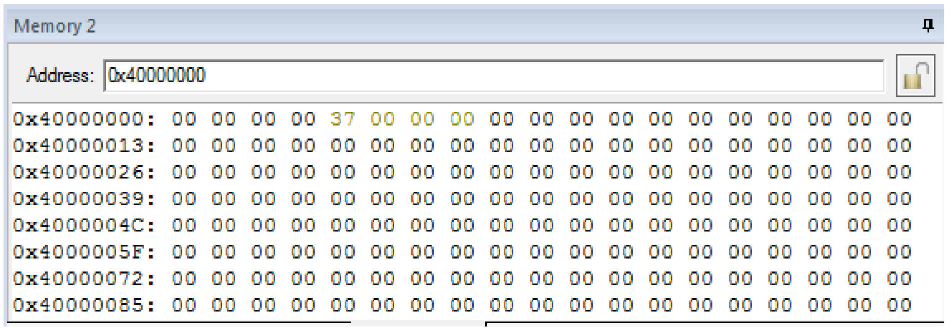
```
; 0x40000000: 11  12   33  34   63  0D  AF  03
```

Lab program 4

```
;WRITE A PROGRAM TO FIND THE SUM OF FIRST 10 INTEGER NUMBERS.

        AREA Program, CODE, READONLY

ENTRY

                LDR R0, MEMORY

                MOV R1,#0               ;load R1 with 0

                MOV R2,#10              ;load R2 with A=10

AGAIN           ADD R1,R1,R2             ;adding R1 & R2 value, AGAIN is just a label

                                        ; Jump till value Not Zero back to addition, if zero stop
                                        looping

                SUB R2,R2,#1            ;Decrement the value of R2

                CMP R2,#0               ;Compare R2 value with 0

                BNE AGAIN               ; loop until R2==0

        STR     R1, [R0, #4]            ; R2==0 store the final value in R1  to MEMORY

                SWI     &11             ;Terminate

MEMORY          DCD     0x40000000

        END
```

Output:

Sum of first 10 integers (1 to A)


Memory 2

Address: 0x40000000

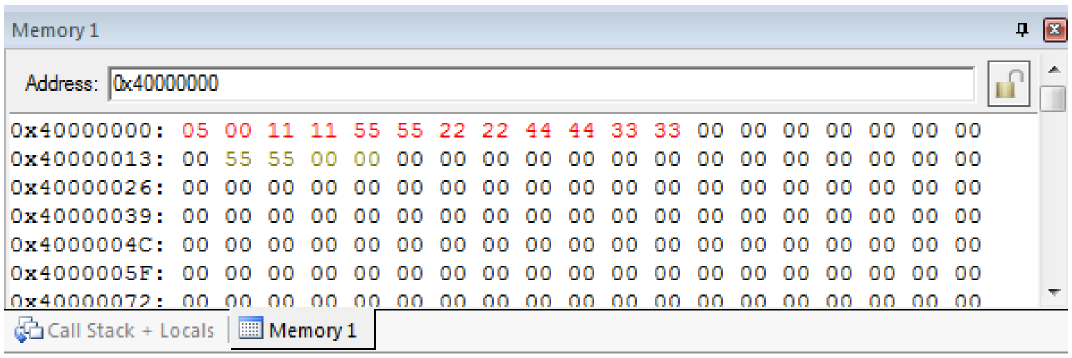
```
0x40000000: 00 00 00 00 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000026: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000039: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000004C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000005F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000072: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000085: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Lab program 5.

;Write an ARM assembly program to find the largest/smallest number in an array of 32 bit numbers.

```
;TTL Largest_16bit
            AREA Program, CODE, READONLY
 ENTRY

                LDR     R0, MEMORY          ; load the address of the lookup table
                LDR     R4, RESULT
                LDRH    R1, [R0]
                ADD     R0,R0,#2
                LDRH    R5, [R0]
                MOV     R2, R1              ; load the count
LOOP            CMP     R2, #1              ; is the count is zero
                BEQ     DONE
                ADD     R0,R0,#2
                LDRH    R3,[R0]
                SUB     R2,R2,#1
                CMP     R5,R3
                BHI     LOOP   (For smallest—BLS)
                MOV     R5,R3
                B       LOOP
DONE            STR     R5,[R4]
HERE            B       HERE
MEMORY                  DCD         0x40000000
RESULT                  DCD         0x40000015
                        END
```

Output:



```
Address: 0x40000000
0x40000000: 05 00 11 11 55 55 22 22 44 44 33 33 00 00 00 00 00 00 00
0x40000013: 00 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000026: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000039: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000004C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000005F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000072: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
        AREA Program, CODE, READONLY
  ENTRY
         LDR    R0, MEMORY
         LDR    R1, [R0],#04
         LDR    R2, [R0], #04
         SUB R1, #01
LOOP    LDR    R3,[R0], #04
         CMP    R2,R3
         BHI    NEXT
         MOV    R2,R3
NEXT  SUBS R1, #01
        CMP R1, #0
        BNE LOOP
        STR R2, [R0,#04]
HERE    B    HERE
MEMORY       DCD 0x40000000
         END
```

Lab program 6

;ARM assembly program to count the number of ones and zeros in two consecutive memory locations.

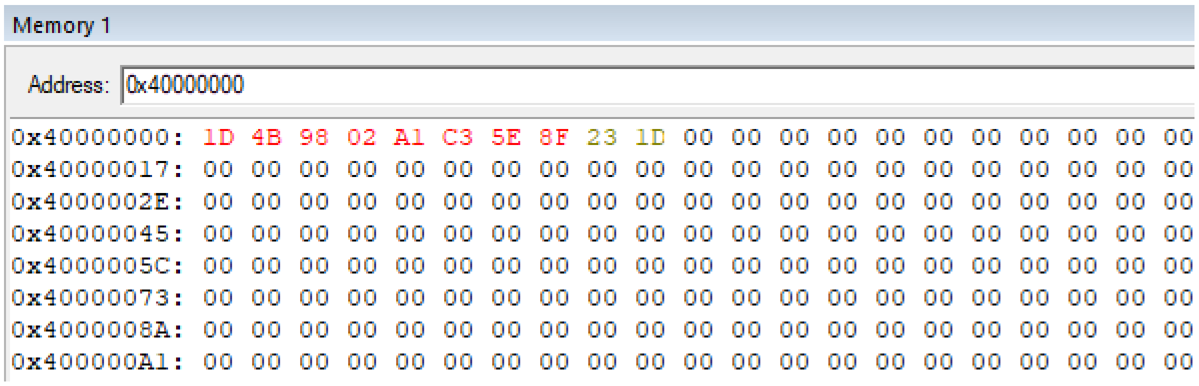
```
 AREA Program, CODE, READONLY
ENTRY
            LDR   R0, LOOKUP
            MOV   R5, #2
TWO         LDR   R1,[R0]
            MOV   R4, #32
ROTATE      RORS R1, #1
            BCS   ONES
            ADD   R3,R3,#1
            B     NEXT
ONES        ADD   R2,R2,#1
NEXT        ADD   R4,R4,#-1
            CMP   R4,#0
            BNE   ROTATE
            SUB R5, R5, #1
            CMP R5, #0
            BEQ STORE
            ADD   R0,R0,#04
            B TWO
STORE       ADD   R0,R0,#04
            STRB  R3,[R0]
            ADD   R0,R0,#1
            STRB  R2,[R0]
```

```
HERE        B       HERE
LOOKUP      DCD     0x40000000
            END
```

Output:

Count the numbers of ‘1’ and ‘0’ in the binary equivalent of “1D 4B 98 02 A1 C3 5E 8F”

OUTPUT after execution - 23 1D


Memory 1

Address: 0x40000000

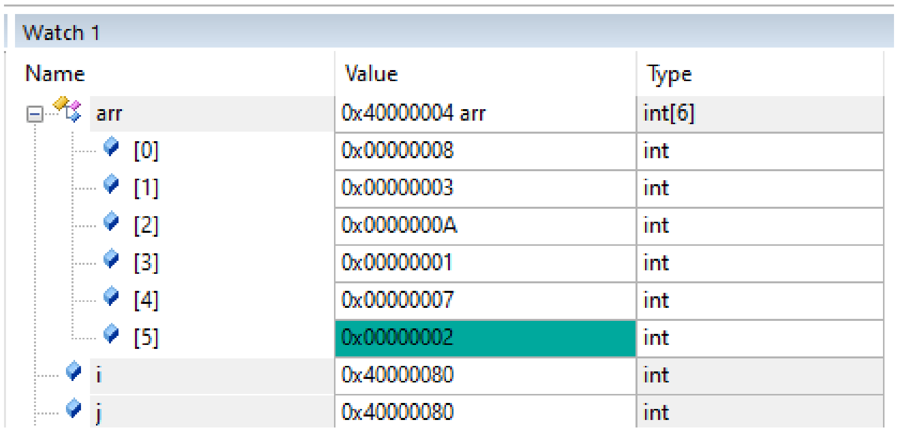
```
0x40000000: 1D 4B 98 02 A1 C3 5E 8F 23 1D 00 00 00 00 00 00 00 00 00 00 00
0x40000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000002E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000045: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000005C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000073: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000008A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x400000A1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Lab Program 7

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

```
int arr[6];

int n=6;

int main()

{

        int i, j;

 for (i = 0; i < n - 1; i++) {

        for (j = 0; j < n - i - 1; j++) {

        if (arr[j] > arr[j + 1]) {

        int temp = arr[j];

        arr[j] = arr[j + 1];

        arr[j + 1] = temp;

        }

        }

 }

        return 0;

}
```

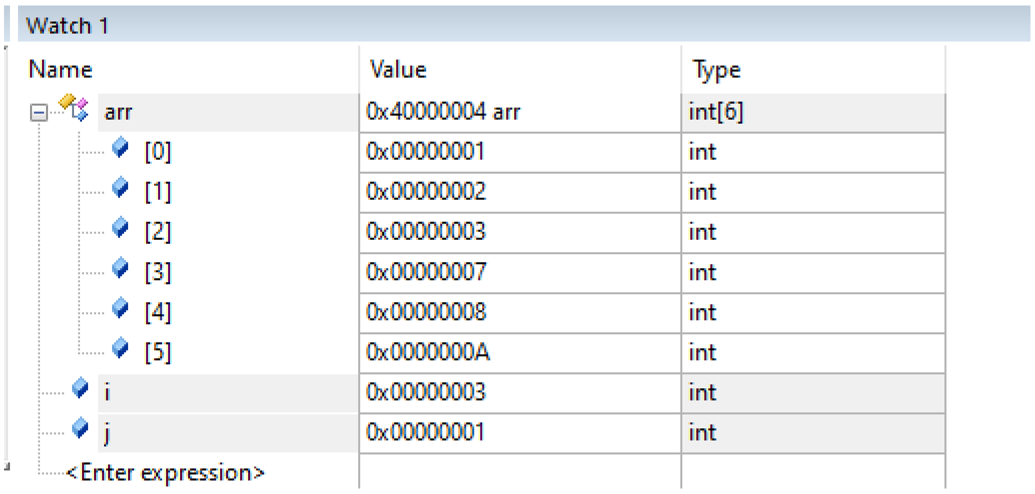
Before execution:



Watch 1

Name	Value	Type
arr	0x40000004 arr	int[6]
[0]	0x00000008	int
[1]	0x00000003	int
[2]	0x0000000A	int
[3]	0x00000001	int
[4]	0x00000007	int
[5]	0x00000002	int
i	0x40000080	int
j	0x40000080	int

After execution:


Watch 1

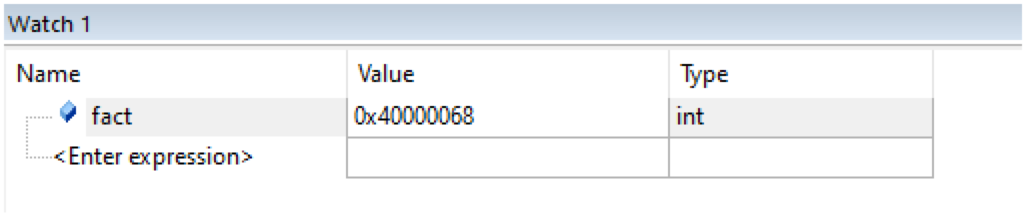
Name	Value	Type
arr	0x40000004 arr	int[6]
[0]	0x00000001	int
[1]	0x00000002	int
[2]	0x00000003	int
[3]	0x00000007	int
[4]	0x00000008	int
[5]	0x0000000A	int
i	0x00000003	int
j	0x00000001	int
<Enter expression>		

Lab Program 8

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

```
int n=5;
int main()
{
int fact=1;
while(n!=0)
{
fact=fact*n;
n--;
}return 0;
}
```

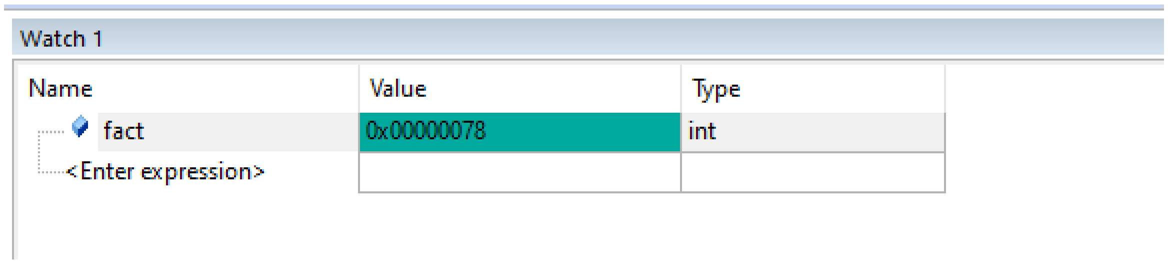
Before execution:



Watch 1

Name	Value	Type
fact	0x40000068	int
<Enter expression>		

After execution:



Watch 1

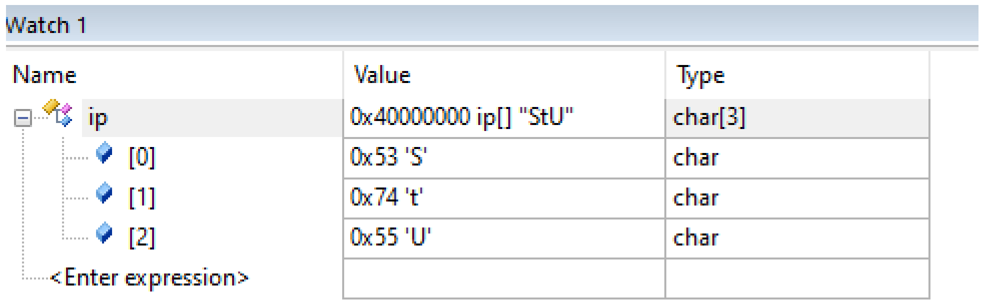
Name	Value	Type
fact	0x00000078	int
<Enter expression>		

Lab Program 9

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

```
char ip[3]={'S', 't', 'U'};
        int main()
{
        int i;
        for (i=0;i<3;i++)
        {
        if(ip[i]>=97 && ip[i]<=123)
        ip[i]-=32;
        else
        ip[i]+=32;
        }
        return 0;
}
```

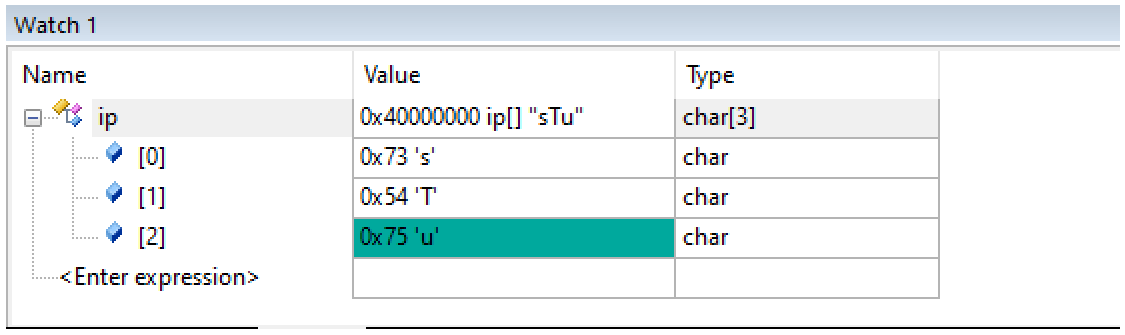
Before Execution:



Watch 1

Name	Value	Type
ip	0x40000000 ip[] "StU"	char[3]
[0]	0x53 'S'	char
[1]	0x74 't'	char
[2]	0x55 'U'	char
<Enter expression>		

After Execution:



Watch 1

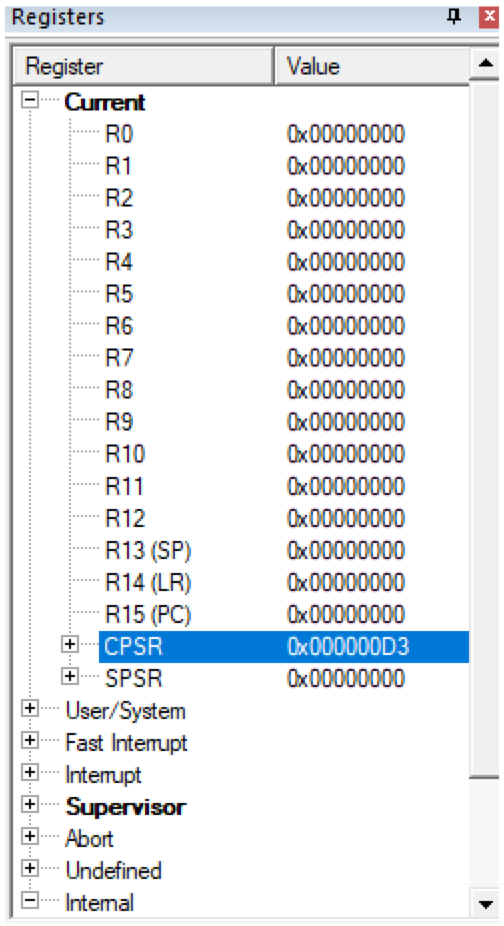
Name	Value	Type
ip	0x40000000 ip[] "sTu"	char[3]
[0]	0x73 's'	char
[1]	0x54 'T'	char
[2]	0x75 'u'	char
<Enter expression>		

Lab Program 10:

Demonstrate enabling and disabling of Interrupts in ARM.

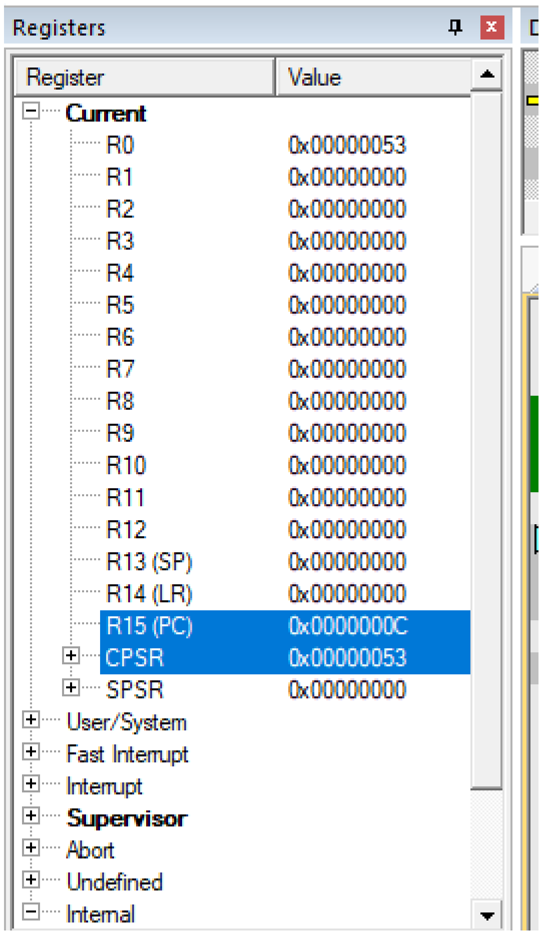
```
area demo,code,readonly
start
;disabling
        mrs r0, cpsr ;11010011=D3
        BIC r0, r0, #0x80  ;01010011=53
        msr cpsr_c, r0;
;enabling
        mrs r0, cpsr ;
        ORR r0, r0, #0x80
        msr cpsr_c, r0;
;stopping
        bx lr
        End
```

Before Execution:



```
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
```

After Execution:



Registers

Register	Value
Current	
R0	0x00000053
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)	0x0000000C
CPSR	0x00000053
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	