ST JOSEPH ENGINEERING COLLEGE

VAMANJOOR, MANGALURU– 575028.

LABORATORY MANUAL (2019-20)

IV SEMESTER B.E.

MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY (18CSL48)

USN	
NAME	
BATCH	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ST JOSEPH ENGINEERING COLLEGE, MANGALURU– 575028.

IV Semester B.E.

MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY

LAB MANUAL (2019-20)

INSTRUCTIONS

1. This manual is an interim record of experiments conducted during regular laboratory sessions.
2. Students are expected to code the algorithm/perform the steps given, write the precise assembly language/Embedded C code and execute/debug the same to obtain the solution for the problem specified in the aim of the experiment.
3. Students are required to familiarize the steps for executing assembly/embedded c programs using Keil/Flash magic software and view the results in register/memory/stack.
4. All software & hardware programs must be executed individually and interfacing kits must be connected through appropriate I/O ports/connectors. Proper care must be taken to handle the hardware kits.
5. Students must be well prepared with theory, Problem Description, Methodology, Illustrations and Algorithm before attending the lab session. The manual may not give complete details about the procedure, examples, inputs, outputs (results) and language constructs.
6. Each student is responsible for maintaining his/her own Manual/Laboratory Observation Notebook. Each student is required to perform pre-lab work and enter it into his/her notebook.
7. Lab experiments are checked by the faculty. Check-out will be used to confirm that the actual lab work as recorded in the lab manual/notebook has been completed.
8. Students must ensure that their manual/notebook is signed by the faculty as proof of execution/conduction of the experiment.

9. Each experiment must be completed within the lab session. If a student is unable to complete the same, he/she may complete the same during the break time, if granted permission by the instructor and take signature as a proof for completing the experiment.

10. When the students utilize the lab facility beyond regular lab hours, they need to enter the lab usage details in the book kept inside the lab.

11. **In case a student is ABSENT for a particular lab session, he/she has to compulsorily finish the experiment/execution before the next lab session and get the observation signed.**

12. Internal Marks (CIE) break up :

Performance	-	10 Marks
Regular lab viva	-	06 Marks
Record	-	08 Marks
Internal Test	-	16 Marks

Total	-	40 Marks
		=======

13. **Internal test will be conducted for 100 Marks. Distribution of marks is done as per the breakup given in the syllabus and reduced to 16 Marks.**

14. A student need to **score minimum 20 marks** (50% of maximum marks**) in CIE** (Internal Marks) to get eligibility **to appear for External Practical Examination**.

The students are required to refer the following books :

a)	ARM system developers guide	-	Andrew N Sloss, Dominic Symes and Chris Wright
b)	Introduction to Embedded Systems	-	Shibu K V
c)	Microcontroller (ARM) and Embedded System	-	Raghunandan G H
d)	The Insider's Guide to the ARM7 Based Microcontrollers	-	Hitex Ltd
e)	ARM System-on-Chip Architecture	-	Steve Furber
f)	Embedded System	-	Raj Kamal

15. Maintaining Laboratory Record Book:

(i) Write your name, USN and subject on the outside front cover of the record. Fill the certificate page with the relevant information.

(ii) Update the index page every time a new experiment is conducted. Index must clearly reflect the aim of the experiment.

(iii) Use scale and pencil to draw the diagrams.

(iv) Start each new experiment in a new page on the right-side. Plan the record writing so that one record can accommodate all experiments.

(v) Date and experiment no. should be written on the left side of the title of the experiment. Write the page nos. on each page of the record and index page.

Every experiment must contain the following :

Right side :

- Aim
- Topic Level Objectives
- Program with comments
- Experiment Outcome
- Conclusion if any

Left side :

- Diagrams if any
- Design/Algorithm
- Output (Register/Memory contents)
- Expected wave forms if any

Note :1) Use labels and captions for figures and tables if any.

MES LAB - DO'S AND DON'TS

Be regular to the Lab.	Do not come late to the Lab.
Follow proper Dress Code.	Do not operate the ARM trainer kits without Permission
Wear your College ID card.	Don't try to remove any component from the kit
Avoid unnecessary talking while doing the Experiment.	
Take the signature of the faculty before taking the components.	
Handle the ARM kit properly.	
Connect/Disconnect the interfacing devices carefully. Observe that connectors connect in only one direction.	
Keep your work area clean after completing the Experiment.	
After completion of the experiment switch off the power supply and return the components.	
Keep the chairs in place before leavingthe laboratory.	

MICROCONTROLLER AND EMBEDDED SYSTEMS LABORATORY

[Outcome Based Education (OBE) and Choice Based Credit System (CBCS)]

(Effective from the academic year 2018-19)

Semester : IV **Laboratory Code : 18CSL48**

CIE Marks : 40 **Number of Contact Hours/Week:2(T)+2(P)**

SEE Marks : 60 **Total Number of Lab Contact Hours: 36**

Exam Hours : 03 **Credits : 02**

Course objectives:

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

Descriptions (if any) :

PROGRAMS LIST	
PART A	
Conduct the following experiments by writing program using ARM7TDMI/LPC2148 using an evaluation board/simulator and the required software tool.	
1	Write a program to multiply two 16 bit binary numbers.
2	Write a program to find the sum of first 10 integer numbers.
3	Write a program to find 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 look-up table.
6	Write a program to find the largest/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.
PART B	
Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.	
9	Display “Hello World” message using Internal UART.
10	Interface and Control a DC Motor.
11	Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.
12	Determine Digital output for a given Analog input using Internal ADC of ARM controller.

13	Interface a DAC and generate Triangular and Square waveforms.
14	Interface a 4x4 keyboard and display the key code on an LCD.
15	Demonstrate the use of an external interrupt to toggle an LED On/Off.
16	Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.

Laboratory Outcomes:

The student should be able to :

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

Conduction of Practical Examination :

Experiment distribution

- For laboratories having only one part :
  Students are allowed to pick one experiment from the lot with equal opportunity.
- For laboratories having PART A and PART B` :
  Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.

Marks Distribution : (Subject to change in accordance with university regulations)

For laboratories having only one part : Procedure + Execution + Viva-Voce: 15+70+15=100

For laboratories having PART A and PART B :
Part A : Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
Part B : Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

VISION OF THE DEPARTMENT

To be recognized as a centre of excellence in computer and allied areas with quality learning and research environment.

MISSION OF THE DEPARTMENT

1. Prepare competent professionals in the field of computer and allied fields enriched with ethical values.
2. Contribute to the Socio-economic development of the country by imparting quality education in computer and Information Technology.
3. Enhance employability through skill development.

Undergraduate Programme in Computer Science and Engineering (B.E.)

Programme Educational Objectives (PEOs)

I. To impart to students a sound foundation and ability to apply engineering fundamentals, mathematics, science and humanities necessary to formulate, analyze, design and implement engineering problems in the field of computer science.

II. To develop in students the knowledge of fundamentals of computer science and engineering to work in various related fields such as network, data, web and system engineering.

III. To develop in students the ability to work as a part of team through effective communication on multidisciplinary projects.

IV. To train students to have successful careers in computer and information technology industry that meets the needs of society enriched with professional ethics.

V. To develop in students the ability to pursue higher education and engage in research through continuous learning.

Programme Outcomes (POs)

By the end of the undergraduate programme in CSE, graduates will be able to:

1) Apply knowledge of mathematics, science, engineering fundamentals, computer science and engineering to solve complex engineering problems.

2) Identify, formulate, research literature, and analyze complex engineering problems in reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3) Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4) Conduct investigations of complex problems using research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5) Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6) Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7) Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.

8) Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9) Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10) Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.

11) Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12) Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes (PSOs)

By the end of the undergraduate programme in CSE, graduates will be able to:

1. Understand the principles underlying entrepreneurship, freelancing and the requirements to initiate a startup in the IT or related domains.

2. Participate effectively in competitive examinations related to certification, career growth and admission to higher studies.

Mapping of COs with POs and PSOs

Keywords (PO/PSO)	Apply Knowledge	Solve Problems	Design/ Development of Solution	Conduct Investigations	Use Modern Tools	Engineer and Society	Environment and Sustainability	Professional Ethics	Individual and Team Work	Communicate Effectively	Project Management and Finance	Lifelong Learning	Entrepreneurship	Competitive Exams and Higher Studies
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
18CSL48.1								**2**						
18CSL48.2								**2**						
18CSL48.3										**2**				
18CSL44.4										**2**				
18CSL44.5										**2**				
18CSL44.6										**2**				

1: Slight (Low), 2: Moderate (Medium), 3: Substantial (High);

Course Outcomes

CO No.	Course Outcomes (COs)	Bloom's Taxonomy Level	Target Attainment Level
18CSL48.1	Design and implement structured, well-commented, understandable programs in assembly language to solve 16-bit binary multiplication, factorial, square of a number	L3	2
18CSL48.2	Design and implement structured, well-commented, understandable programs in assembly language using memory concept to solve addition of array of 16-bit binary numbers, to count number of zeros and ones in two consecutive memory locations	L3	2
18CSL48.3	Design and implement structured, well-commented, understandable C programs to interface and control DC motor, stepper motor	L3	2
18CSL48.4	Design and implement structured, well-commented, understandable C programs to determine digital output for a given analog signal using ADC of ARM controller	L3	2
18CSL48.5	Design and implement structured, well-commented, understandable C programs to interface a 4x4 keyboard and display the key code on LCD.	L3	2
18CSL48.6	Design and implement structured, well-commented, understandable C program to demonstrate the working of LED.	L3	2

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, SJEC

Assessment Rubric – Performance & Viva

Criteria	Needs Improvement (1)	Satisfactory (2)	Good (3)	Accomplished (4)	Total Maximum marks per criteria
Execution of Program (1)	Not executed / partially executed programs after one week of the session (1)	Executed programs within one week post the session (2)	Executed programs within two days of the session (3)	Executed all programs within the session (4)	**04**
Writing programs for algorithms (1)	Pre-lab work not done / partially done without any logic / copy (1)	Pre-lab work done with partial working logic / copy (2)	Pre-lab work done with working logic (3)	Pre-lab work done with own working logic or improved logic (4)	**04**
Recording of Results (0.5)	Submitted manual with complete recording of outputs after 7 days (0.5)	Submitted manual with complete recording of outputs within 3-7 days (1)	Submitted manual with complete recording of outputs within 2 days (1.5)	Submitted manual with complete recording of outputs within the session (2)	**02**
Criteria	**Needs Improvement (0)**	**Satisfactory (1)**	**Good (2)**	**Accomplished (3)**	
Daily viva (2)	Not answered any questions and demonstrated lack of understanding of concepts taught (0)	Answered less than 50% of the questions and demonstrated lack of understanding of concepts taught (1)	Answered up to 50%-75% of the questions and demonstrated a satisfactory understanding of the concepts learnt (4)	Answered 75%-100% questions and demonstrated a clear understanding of the concepts learnt (6)	**06**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, SJEC

Assessment Rubric & Marks Sheet – MES Record

Criteria	Needs Improvement (1)	Satisfactory (2)	Good (3)	Accomplished (4)	Total Maximum marks per criteria
Timeliness (0.25)	Delayed by more than 2 days with complete or incomplete header and or index entry (0.25)	Delayed by 2 days with complete or incomplete header and or index entry (0.5)	Delayed by a daytime with complete index and header entry or Submitted on time with incomplete header and or index entry (0.75)	Submitted on time with complete index and header entry (1)	**01**
Completeness (0.5)	Record is incomplete, with or incomplete program, results and documentation (0.5)	Record is partially complete, with complete program, but with missing results and documentation (1.0)	Record is complete with complete program and results but is missing documentation (1.5)	Record is complete having complete and results, written neatly with appropriate documentation (2)	**02**
Program correctness, structure and Indentation (1)	Program is partially correct and produces wrong/inaccurate results. Indentation is not followed (1)	Program is correct but results documented are either wrong, or do not match the input or are plagiarized. Indentation is followed throughout (2)	Program is correct and produces accurate results. Indentation is partially followed (3)	Program is correct and produces accurate results. Indentation is followed throughout (4)	**04**
Outcome specification (0.25)	Program outcomes and bloom's taxonomy level arenot specified. (0.25)	Program outcomes are specified but bloom's taxonomy level is missing (0.5)	Program outcomesare specified with inappropriate bloom's taxonomy level (0.75)	Program outcomesare specified with appropriate bloom's taxonomy level (1)	**01**
Marks Awarded (in words):					**08**

COURSE COVERAGE & MARKS

Exp. No.	Date	Experiment Title	Marks			Initial of the Staff & Date
			Perf. (10)	Viva (6)	Total (16)	
PART-A						
1		Write a program to multiply two 16 bit binary numbers				
2		Write a program to find the sum of first 10 integer numbers				
3		Write a program to find 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 look-up table				
6		Write a program to find the largest/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				
PART-B						
9		Display “Hello World” message using Internal UART.				
10		Interface and Control a DC Motor.				
11		Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction				
12		Determine Digital output for a given Analog input using Internal ADC of ARM controller				
13		Interface a DAC and generate Triangular and Square waveforms				
14		Interface a 4x4 keyboard and display the key code on an LCD				
15		Demonstrate the use of an external interrupt to toggle an LED On/Off				
16		Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between				
Total Marks (Performance & Regular Viva)						
Average Marks (Performance & Regular Viva) (16)						
Record Marks (8)						
Internal Test (16)						
Final Internal Marks (Performance + Regular Viva + Record + Internal Test) (40)						

Steps to execute Assembly / Embedded C Program in Kiel Software

1. Student has to create a folder with his/her USN to save the files (Eg : 18CS020) and sub folder for each program. Name the sub folders with proper program name.
2. Click on the Keil software icon on the desktop.
3. Click on the Project tab and select New µVision Project.
4. A dialog box appears asking the user to enter project name. Enter the project name in File name text field and click on the Save button.
5. In the dialog box that appears next, select expand the NXP option and select LPC2148. Next click on the OK button.
6. A dialog box appears next asking “Copy ‘Startup.s’ to project folder and Add File to project”. Here click Yes for C program & no for Assembly program. If another dialog box appears next, click Yes.
7. Next create a new file and write the program here. Save the program with the .c or .asm extension.
8. Next in the project window, expand Target1 and later right click on the Source Group 1 icon and select the option Add Existing Files to Source Group 1… Select the file that you have saved and click on Add button.
9. Right click on the Target 1 and select Options for Target 1.
10. Click on the **Target tab** and enable the **Use Microlib** option. In the **Output tab**, enable **Create Hex** file option. In the **Linker** tab, enable **Use Memory Layout from Target Dialog** option. Then click on the OK button.
11. Right Click on the file added to Source Group 1. Select Build target.
12. Click on the Debug tab and select the Start/Stop debug session to debug the program. Click OK on the dialog box that appears next. The window displays the contents of Registers, Memory Layout and the disassembled source code. Use F11 key to step over each instruction.
13. To stop the debug session, click on Debug tab again and select the Start/Stop debug session option.

INTRODUCTION TO ARM PROGRAMMING

Structure Of Arm Assembly Language Program :

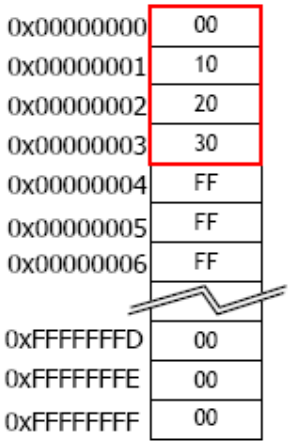
ARM programmer Model :

- The state of an ARM system is determined by the content of visible registers and memory.
- A user-mode program can see 15 32-bit general purpose registers (R0-R14), program counter (PC) and CPSR.
- Instruction set defines the operations that can change the state.
- ARM instructions are all 32-bit long (except for Thumb mode). There are 232 possible machine instructions.

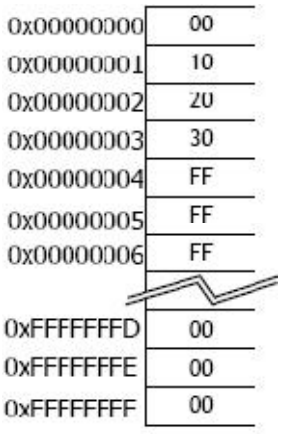
Memory system :

Memory is a linear array of bytes addressed from 0 to 2^{32}-1. Memory can be accessed as Word, half-word & byte.

Byte ordering :



Big Endian`
Least significant byte has highest address
Word address 0x00000000
Value : 00102030



Little Endian
Least significant byte has lowest address
Word address 0x00000000
Value : 30201000

Features of ARM instruction set :

- Load-store architecture
- 3-address instructions
- Conditional execution of every instruction
- Possible to load / store multiple register at once
- Possible to combine shift and ALU operations in a single instruction

Layout of assembly language source file :

The general form of source lines in assembly language is:

{label} {instruction|directive|pseudo-instruction} {;comment}

Instructions, pseudo-instructions, and directives must be preceded by white space, such as a space or a tab, even if there is no label. Some directives do not allow the use of a label. All three sections of the source line are optional. You can use blank lines to make your code more readable.

Case rules :

Instruction mnemonics, directives, and symbolic register names can be written in uppercase or lowercase, but not mixed.

Line length :

To make source files easier to read, a long line of source can be split onto several lines by placing a backslash character (\) at the end of the line. The backslash must not be followed by any other characters (including spaces and tabs). The assembler treats the backslash followed by end-of-line sequence as white space.

Do not use the backslash followed by end-of-line sequence within quoted strings.

The limit on the length of lines, including any extensions using backslashes, is 4095 characters.

Labels :

Labels are symbols that represent addresses. The address given by a label is calculated during assembly.

The assembler calculates the address of a label relative to the origin of the section where the label is defined. A reference to a label within the same section can use the PC plus or minus an offset. This is called program-relative addressing.

Addresses of labels in other sections are calculated at link time, when the linker has allocated specific locations in memory for each section.

Local labels :

Local labels are a subclass of label. A local label begins with a number in the range 0-99. Unlike other labels, a local label can be defined many times. Local labels are useful when you are generating labels with a macro. When the assembler finds a reference to a local label, it links it to a nearby instance of the local label.

The scope of local labels is limited by the AREA directive. You can use the ROUT directive to limit the scope more tightly.

Comments :

The first semicolon on a line marks the beginning of a comment, except where the semicolon appears inside a string constant. The end of the line is the end of the comment. A comment alone is a valid line. The assembler ignores all comments.

Constants :

Constants can be Numbers. Numeric constants are accepted in the following forms:

Decimal, for example, 123
Hexadecimal, for example, 0x7B

n_xxx where : n is a base between 2 and 9, xxx is a number in that base

floating-point, for example, 0.02, 123.0, or 3.14159.

Floating-point numbers are only available if your system has VFP.

Boolean :
The Boolean constants TRUE and FALSE must be written as {TRUE} and {FALSE}.

Characters :

Character constants consist of opening and closing single quotes, enclosing either a single character or an escaped character, using the standard C escape characters.

Strings :

Strings consist of opening and closing double quotes, enclosing characters and spaces. If double quotes or dollar signs are used within a string as literal text characters, they must be represented by a pair of the appropriate character. For example, you must use $$ if you require a single $ in the string. The standard C escape sequences can be used within string constants.

Assembler Directives :

AREA - Instructs the assembler to assemble a new code or data area/section. Areas are independent, named, indivisible chunks of code or data that are manipulated by the linker

The syntax of the AREA directive is : AREA name {,attr}{,attr}...
Where name - is the name that the area is to be given

```
Eg :  AREA  MyCode, CODE, READONLY  ; Code section
      AREA  MyData, DATA, READWRITE  ; Data Section
```

CODE - Contains machine instructions. READONLY is the default.

DATA - Contains data, not instructions. READWRITE is the default

COMMON - Is a common data section. You must not define any code or data in it. It is initialized to zeros by the linker. All common sections with the same name are overlaid in the same section of memory by the linker. They do not all have to be the same size. The linker allocates as much space as is required by the largest common section of each name.

DCD - DCD (Define Constant Data) allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory

{label} DCD{U} expr{,expr}
Where: Expr is either: A numeric expression or A PC-relative expression.

Eg : data1 DCD 1,5,20 ; Defines 3 words containing decimal values 1, 5, and 20

data2 DCD mem06 + 4 ; Defines 1 word containing 4+ the address of the label mem06

MEMORY DCD 0x40000000 ; Stores the address of 16/32 bit number in MEMORY

DCW - The DCW directive allocates one or more halfwords of memory, aligned on two-byte boundaries, and defines the initial runtime contents of the memory.

{label} DCW{U} expr{,expr}...where: expr is a numeric expression that evaluates to an integer in the range –32768 to 65535.

Eg : data DCW -225,2*number ; number must already be defined
DCWU number+4

ENTRY - The code starts executing from this point.

END - The END directive informs the assembler that it has reached the end of a source file

Software Interrupt Instruction :

A software interrupt instruction (SWI) causes a software interrupt exception, which provides a mechanism for applications to call operating system routines.

The Software Interrupt instruction (SWI) is used to enter Supervisor mode, usually to request a particular supervisor function. The SWI handler reads the opcode to extract the SWI function number. This action restores the PC and CPSR, and returns to the instruction following the SWI.

SWI {<cond} SWI_number

Eg : SWI &11

Software Programs - (Using Assembly Level Language)

PART - A

Program No. 1 : Write a program to multiply two 16 bit binary numbers.

16 bit Multiplication

Aim: To Write a assembly level program to multiply two 16 bit binary numbers.

Topic Learning Objective:

Demonstrate the multiplication of two 16 bit binary numbers stored in memory or registers and store the result in memory.

Instructions :

MOV - Move constant or register to register. This instruction is also used for shift operations.

Syntax : MOV{cond}{S} Rd, Op2

Description : Copies the value of Op2 into Rd. Condition Flags If S is specified, N, Z flags are updated. C flag may be updated by calculation of Op2.

Example :
```
MOV R5, #0x20        // load R5 with the constant 0x20
MOV R2, R3           // load R2 with the value in R3
MOV R4, R5, SHL #4 // load R4 with the value in R5 shift left by 4 bits
```

MUL : Multiply (32-bit by 32-bit, bottom 32-bit result)

Syntax : MUL{ cond}{S} Rd, Rm, Rs

Description : Multiplies the values from Rm and Rs, and places the least significant 32 bits of The result in Rd. Condition Flags If S is specified. N and Z flags according to the result.

Example :
```
MUL R10, R2, R5          // R10:= R2*R5
```

LDR : Load 32-bit word to Memory

Syntax :
```
LDR{ cond} Rd, [Rn]
LDR{ cond} Rd, [Rn, offset]
LDR{ cond} Rd, [Rn, offset]!
LDR{ cond} Rd, label
LDR{ cond} Rd, [Rn], offset
```

Description : LDR{ cond} Rd, [Rn] (zero offset) // Rn is used as address value

LDR { cond} Rd, [Rn, offset] (Pre-indexed offset) // Rn and offset are added and used as address value.
LDR{ cond} Rd, [Rn, offset]{!} (Pre-indexed offset with update) // Rn and offset are added and used as address value. The new address value is written to Rn.
LDR{ cond} Rd, label (Program-relative) // The assembler calculates the PC offset and generates LDR{ cond} Rd, [R15, offset].

DR{ cond} Rd, [Rn], offset (Post-indexed offset) // Rn is used as address value. After memory transfer, the offset is added to Rn.

Example : LDR R8,[R10] //loads r8 from the address in r10.
LDRNE R2,[R5,#960]! // (conditionally) loads r2 from a word 960 bytes above the address in r5, and increments r5 by 960.
LDR R0 ,localdata //loads a word located at label localdata

LDRH : Load register 16-bit half word value to Memory. The address must be even for Hal fword transfers.

Syntax : LDR{ cond}H Rd, [Rn]
LDR{ cond}H Rd, [Rn, offset]
LDR{ cond}H Rd, [Rn, offset]!
LDR{ cond}H Rd, label
LDR{ cond}H Rd, [Rn], offset

Description :

LDR{ cond}H Rd, [Rn] (Zero offset) // Rn is used as address value.
LDR{ cond}H Rd, [Rn, offset] (Pre-indexed offset) // Rn and offset are added and used as address value.
LDR{ cond}H Rd, [Rn, offset]! (Pre-indexed offset with update) // Rn and offset are added and used as address value. The new address value is written to Rn.
LDR{ cond}H Rd, label (Program-relative)
The assembler calculates the PC offset and generates LDR{ cond}H Rd, [R15, offset]

STR : Store register 32-bit words to Memory. The address must be 32-bit word-aligned.

Syntax : STR{ cond} Rd, [Rn]
STR{ cond} Rd, [Rn, offset]
STR{ cond} Rd, [Rn, offset]!

STR{ cond} Rd, label
STR{ cond} Rd, [Rn], offset

Description : STR{ cond} Rd, [Rn] (zero offset) Rn is used as address value.
STR{ cond} Rd, [Rn, offset] (Pre-indexed offset)
STR{ cond} Rd, [Rn, offset]! (Pre-indexed offset with update)
STR{ cond} Rd, label (Program-relative)

The assembler calculates the PC offset and generates STR{cond} Rd, [R15], offset.

STR{ cond} Rd, [Rn], offset (Post-indexed offset) // Rn is used as address value. After memory transfer, the offset is added to Rn.

Example :

```
LDR r8,[r10]   // loads r8 from the address in r10.
LDRNE r2, [r5,#960]! // (conditionally) loads r2 from a word 960 bytes above the
                     address in r5, and increments r5 by 960.
STR r2,[r9,#consta-struc] // consta-struc is an expression evaluating to a constant in
                          the range 0-4095.
STRB r0,[r3,-r8,ASR #2] // stores the least significant byte from r0 to a byte at an
                        address equal to contents(r3) minus contents(r9)/4. r3 and
                        r8 are not altered.
STR r5,[r7],#-8 //stores a word from r5 to the address in r7, and then decrements r7 by 8
LDR r0,localdata //loads a word located at label localdata
```

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) & **DATA** (read write).

Step 2 : Define constant data using **DCD** (32 bit) or **DCW** (16 bits) directives.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Load the first 16 bit value into Register **R0** from memory (LDRH)

Step 5 : Load the second 16 bit value into Register **R1** from memory.

Step 6 : Multiply values stored in **R0** and **R1** and store the 32 bit resulting value in **R2**.

Step 7 : Store the Result in **R2** into memory location **RESULT**. Ensure that RESULT points to a memory location which lies in the read/write area and starts at the address 40000000.

Step 8 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Note : Any General Purpose Register from R0 to R14 can be used.

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			1	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 2 : Write a program to find the sum of first 10 integer numbers.

Sum of first 10 integer numbers

Aim: To write a program to find the sum of first 10 integer numbers.

Topic Learning Objective:

Demonstrate the addition of first 10 numbers in memory and store the result back to memory location.

.

Instructions :

ADD - Add values and store result to register.

Syntax : ADD{ cond}{S} Rd, Rn, Op2

Description : Add Rn and Op2 and store result to Rd. Condition Flags If S is specified update flags: N, Z, C, V.

Example : ADDS R0,R2,R4 // Add R2 and R4 and store result to R0, update flags
ADD R4,R4,#0xFF00 // Add value in 0xFF00 and R4 and store result in R4

SUB - Subtract registers.

Syntax : SUB{ cond}{S} Rd, Rn, Op2

Description : subtracts the value of Op2 from the value in Rn. Condition Flags If S is specified update flags: N, Z, C, V.

Example : SUBS R8,R6,#240 // R8=R6-240

CMP – Compare - Used in combination with conditional branch instructions.

Syntax : CMP { cond} Rn, Op2

Description : subtracts the value of Op2 from the value in Rn (equals to the SUBS instruction with a discarded result).This instruction updates the condition flags, but do not place a result in a register. Condition Flags N, Z, C and V flags are updated.

Example : CMP R2, R9 // Subtract value of R9 from R2

B (Branch) : The B instruction is used to branch to a label. The branching decision is taken based on the conditional flag bits set/reset during previous instruction.

Syntax : B{cond}{.W} label where: cond is an optional condition code.

BNE ("Branch if Not Equal") is the mnemonic for a machine language instruction which branches, or "jumps", to the address specified if, and only if the zero flag is clear.

Eg : B LOOP (unconditional branching)
BNE UP (Branch to label up if zero flag is 0)

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) & **DATA** (read write).

Step 2 : Define constant data using **DCD** (32 bit) directive.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Copy **R5** with count (**10**), (immediate number).

Step 5 : Copy **R0** with **0** (Sum=0)

Step 6 : Copy **R1** with **1** (increment).

Step 7 : while (**R5** ≠ 0)
do
{ ADD **R0** and **R1** and store the result in **R0** (R0=R0+R1)
Increment **R1** by 1 (R1=R1+1)
Decrement count (**R5**) by 1 (R5=R5-1)
}

Step 8 : Store the value in **R0** into memory location **SUM**.

Step 9 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Experiment No.			2	
Date of Conduction				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 3 : Write a program to find factorial of a number.

Factorial of a Number

Aim: To write a program to find factorial of a number.

Topic Learning Objective:

Demonstrate the program to find factorial of a number stored in register or memory.

.

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) & **DATA** (read write)

Step 2 : Define constant data using **DCD** directive.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Copy **R0** with a number (N) whose factorial has to be calculated (Eg : **7**)

Step 5 : Copy the number in **R0** to **R1**.

Step 6 : Decrement the number in **R1 by 1** (N-1).

Step 7 : Repeat the following till (**R1 = 1**)

{ Decrement **R1** by 1 (R1=R1-1)
if R1=1, Go to step 8
Multiply **R0** and **R1** and store the result in **R3** (R3=R0*R1)
Copy the content of **R3** to **R0**
}

Step 8 : Store the value in **R3** into memory location **FACT**.

Step 9 : Terminate the program and change the control to Operating System by calling the Appropriate Interrupt ((or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			3	
Date of Exeuction				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 4 : Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM.

Addition 16 bit numbers stored in an Array

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

Topic Learning Objective:

Demonstrate the addition of 16 bit numbers stored in an array.

.

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) & **DATA** (read write)

Step 2 : Define constant data using **DCD & DCW** (16 bit data) directive.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Load **R5** with the counter.

Step 5 : Load **R0** with **0**. (Sum=0)

Step 6 : Load the starting address of array **(VALUE1**) into **R1**.

Step 7 : Repeat the following till count (**R5**) becomes 0

{

Load the value of data (16 bit no) whose address is in **R1** into **R2** and increment the offset by 2. (Now R1 points to next 16 bit No.)
Add **R0** and **R2**, Store the result in **R0** (0+1st Num)
Decrement the count (R5=R5-1)

}

Step 8 : Store the value in **R0** into memory location **SUM**.

Step 9 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			4	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 5 : Write a program to find the square of a number (1 to 10) using look-up table.

Square of nos from 1 to 10 using look-up table

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

Topic Learning Objective:

Demonstrate the program to find the square of the numbers from 1 to 10 using look-up table.

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) & DATA (read write)

Step 2 : Create a look-up table using **DCD** directive which holds the squares of the numbers from 1 to 10 in Hexadecimal.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Load starting address of Lookup table (**TABLE1**) in **R0**.

Step 5 : Load the no (1-10), whose square is to be found in **R1**. (say 9)

Step 6 : Generate address corresponding to square of the given no in **R1** (Left shift R1 BY 2 bits).

Step 7 : Add the starting address of lookup-table (**R0)** and the offset address corresponding to the square of the given no. (R0=R0+R1)

Step 8 : Load the square of value in **R3**.

Step 9 : Store the value in **R0** into memory location **SQUARE**

Step 10 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			5	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 6 : Write a program to find the largest/smallest number in an array of 32 bit numbers.

Largest/Smallest in an Array of 32 bit Numbers

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

Topic Learning Objective:

Demonstrate the program to find the find the largest/smallest number in an array of 32 bit numbers.

Instructions :

BLT : Branch on Lower Than. The destination operand will be added to the PC, and the 68k will continue reading at the new offset held in PC, if the following conditions are met. The N flag is clear, but the V flag is set. The N flag is set, but the V flag is clear

```
Eg : SUBS R7, R6, R4        ; Performs r7 = r6 - r4 and sets condition register
     BLT LABEL              ; Branches to LABEL if r7 < 0 (in which case r6 < r4)
```

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) and **DATA** (read write)

Step 2 : Define constant data using **DCD** directive which holds an array of 32 bit numbers.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Initialize the **R5** with counter (number of comparisons) (Eg : R5=6 when N =7).

Step 5 : Load starting address of the Array **(VALUE1**) in **R1** (R1 points to 1st no).

Step 6 : Load the contents of **R1** to **R2** and increment the address by 4, so that R1 now points to 2nd no. Now **R2** holds first value.

Step 7 : Repeat the following till count (**R5**) becomes 0

{

Load the contents of **R1** to **R4** and increment the address by 4, so that R1 now points to 3rd no. **R4** holds second value.

Compare the numbers in **R2** and **R4**.
If **R2** > **R4**, Go to Step 8
Copy the second no to first no. (copy **R4** to **R2**)
Step 8 : Decrement the counter
Go to Step 7

}

Step 9 : Store the value of **R2** in **SMALL**.

Step 10 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			6	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 7 : Write a program to arrange a series of 32 bit numbers in ascending/descending order.

32 bit Numbers in Ascending/Descending order

Aim: To write a program to arrange a series of 32 bit numbers in ascending/descending order.

.

Topic Learning Objective:

Demonstrate the program to arrange a series of 32 bit numbers in ascending/descending order.

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) and **DATA** (read write)

Step 2 : Define constant data using **DCD** directive which holds an array of 32 bit numbers.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Load starting address of the Array in **R1** (R1 points to 1st no).

Step 5 : Repeat till flag becomes 0

{

Step 6 : Initialize the **R5** with counter (number of comparisons) (Eg : R5=6 when N =7).

Step 7 : Initialize **R7** with **0** (flag to denote exchange has occurred).

Step 8 : Load starting address of the Array in **R1** (R1 points to 1st no).

Step 9 : Repeat the following till count (**R5**) becomes 0

{

Load the contents of **R1** to **R2** and increment the address by 4, so that **R1** now points to 2nd no. Now **R2** holds first value.

Load the contents of **R1** to **R3** so that **R3** holds second value.

Compare the numbers in **R2** and **R3**.
If **R2** < **R3**, Go to Step 8
Else
{

Exchange the contents of **R2** & **R3.** (Copy the contents of R1 (presently pointing to 2nd no. R3) to **R2** (R2=R3), then decrement the address by 4, so that R1 points to 1st no **R2.** Copy that to **R3**. Thus R2 & R3 are exchanged.

Set the Flag in R7 to 1 indicating exchange has taken place (R7=1)

Copy the second no to first no. (copy **R4** to **R2**)
Restore the pointer (Increment R1 by 4, so that it points to R3)

}

Step 10 : Decrement the counter
Go to Step 7

}

Step 11 : Verify that the 32 bit numbers stored in the array are sorted in ascending order.

Step 12: Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			7	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Program No. 8 : Write a program to count the number of ones and zeros in two consecutive memory locations.

Count the number of 0's and 1's

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

.

Topic Learning Objective:

Demonstrate the program to count the number of ones and zeros in two consecutive memory locations.

Algorithm :

Step 1 : Specify the memory area for **CODE** (read only) and **DATA** (read write)

Step 2 : Define constant data using **DCD** directive which holds two 32 bit nos.

Step 3 : Specify the Entry point of the program. The code starts executing from this point.

Step 4 : Initialize two counters to store 1's (**R2**=0) & 0's (**R3**=0)

Step 5 : Initialize third counter with 2 (counter to get two words, **R7**=2)

Step 6 : Load starting address of the Array in **R6** (R6 points to 1st no).

Step 7 : Repeat till (**R7≠0**) (Both the 32 bit words are checked)

{

Step 8 : Initialize **R1** with counter for number of bits (R1=32)
Step 9 : Load the value into **R0** and increment the address (**R6**) by 4 (LDR R0,[R6],#4)
Step10 : Repeat till (**R1≠0**) (32 bits of each word is checked)

{

Step 11 : Right Rotate **R0** by one bit and update CPSR Register. If rotated bit is 1, carry flag is set else reset (RORS R0,#1)
Step 12 : If carry flag is set goto step 14
Step 13 : Increment 0's count (R3=R3+1) and goto step 15
Step 14 : Increment 1's count (R2=R2+1)
Step 15 : Decrement the bit counter (R1=R1-1)
}
Step 16 : Decrement word count (R7=R7-1)
}

Step 17 : Now R2 contains no of 1's and R3 contains no. of 0's. Store them in the memory as **SUM0** & **SUM1**

Step 18 : Terminate the program and change the control to Operating System by calling the appropriate Interrupt (or Stop the debugger).

Program with comments :

Program	Comments

Viva / Work Area :

Output :

Performance & Viva Marks

Program No.			8	
Date of Execution				
Date of Submission				
C/M	1	2	3	4
C1 / 1				
C2 / 1				
C3 / 0.5				
Viva	0	1	2	3
C4 / 2				
Total				

Steps to use Flash Magic software (Loads the program into hardware kit)

1. Go to **Options** tab. Under Options click on **Advanced Options** and select **Hardware Configuration.**

2. In the **Hardware Configuration** check these : **Use DTR and RTS and ISP pin** and **Keep RTS asserted while COM port.**

3. Next click on **OK** button and do the following settings
   a. **Step1:** Under **Communication** see that the the following values are set
      i. **Device:** LPC2148
      ii. **COM port:** COM1 (Check and Connect)
      iii. **Interface:** Name (SP)
      iv. **Baud rate:** 9600
      v. **Oscillation:** 12 MHz
   b. **Step 2: Erase**
      i. Check erase blocks used by hex file
   c. **Step3: Hex file**
      i. Browse and select the hex file to use
   d. **Step 4: Options**
      i. Check verify after programming
   e. **Step 5: Start**
      i. Click start to transfer the hex file to the target device. The output is displayed on the target device.