



Department of Electronics and Communication Engineering

III Year / VI Semester

20ECL67 – Embedded System Design Laboratory

LAB MANUAL

Academic Year : 2020 - 2021

Semester : EVEN



Department of Electronics and Communication Engineering
EMBEDDED SYSTEM DESIGN LAB

Course Code	: 20ECL67	Credits	: 1.5
L:P:T:S	: 0:0:1.5	CIE Marks	: 50
Exam Hours	: 03	SEE Marks	: 50

LIST OF EXPERIMENTS

CYCLE-1

- 1) Study of ARM- Cortex M4 processor development board
- 2) Program involving instructions for transferring data within the processor.
- 3) ALP to demonstrate memory access instruction for various data sizes and addressing modes.
- 4) Program involving logic operations.
- 5) Program involving data conversion operations (extend and reverse ordering).
- 6) Program involving shift and rotate operations.
- 7) Program to illustrate bit field processing instruction.
- 8) Program to illustrate program flow instruction.
- 9) Program to illustrate saturation operation
- 10) Programs involving floating point operations

CYCLE-2

- 11) Interfacing and programming GPIO ports in embedded C ARM Development Board
 - A) Program for Blinking of a LED without delay
 - B) Write a Program for Blinking of a LED with delay
 - C) Program to turn the LED ON when the button is pressed and OFF when it is released
- 12) Generation of PWM signals for different duty cycles
- 13) Embedded C program to demonstrate serial communication using ARM Cortex development board.
- 14) Demonstrate interrupt operations using Embedded C program
 - a) Timers b) Stop Watch

Note: 1) Programming to be done using keiluvision 4 or 5 and download the program on to a M4.Evaluation board such as STM32F nucleon boards, Tiva C series board.

2) Experiments from 1 to 10 should include at least 2 to 4 programs each.

Lab In-charge

HOD-ECE



Department of Electronics and Communication Engineering
EMBEDDED SYSTEM DESIGN LAB

Course Code : 20ECL67 Credits : 1.5
L:P:T:S : 0:0:1.5 CIE Marks : 50
Exam Hours : 03 SEE Marks : 50

Expt. No	Topics	Course Outcomes
1	Program involving instructions for transferring data within the processor.	CO1
2	ALP to demonstrate memory access instruction for various data sizes and addressing modes.	CO1
3	Program involving arithmetic data operations.	CO1
4	Program involving logic operations.	CO2
5	Program involving data conversion operations.	CO2
6	Program involving shift and rotate operations.	CO3
7	Program to illustrate bit field processing instruction.	CO3
8	Program to illustrate program flow instruction.	CO3
9	Program to illustrate saturation operation.	CO4
10	Programs involving floating point operations.	CO4
11	Interfacing and programming GPIO ports in embedded C ARM Development Board a) Program for Blinking of a LED without delay. b) Write a Program for Blinking of a LED with delay. c) Program to turn the LED ON when the button is pressed and OFF when it is released.	CO5
12	Generation of PWM signals for different duty cycles.	CO5
13	Embedded C program to demonstrate serial communication using ARM Cortex development board.	CO5
14	Demonstrate interrupt operations using Embedded C program. a) Timers b) Stop Watch	CO6

CO1	Conduct experiments to understand data transfer and memory access instructions.
CO2	Construct the code for data processing using ARM instructions
CO3	Write code for given applications using bit field and process control instructions.
CO4	Develop code for DSP applications using saturation and floating point operations.
CO5	Use embedded C code to demonstrate peripheral interfacing with ARM development board.
CO6	Construct interrupt operations for specific applications.

Expt. No.	Name of the Experiment	Page NO
	INTRODUCTION ABOUT ARM CORTEX M4	
1.	Study of ARM- Cortex M4 processor development board	7
2	PROGRAM INVOLVING GENERAL DATA TRANSFER INSTRUCTIONS WITHIN THE PROCESSOR	20-24
A)	To write an Assembly Language Program to transfer contents within the registers.	20
B)	Write an ALP for adding two 32 bit number.	21
C)	Write an ALP for Addition of first ten number.	22
D)	Write an ALP for reverse Substraction of two 32 numbers.	23
E)	Write an ALP to find Factorial of a number.	24
3	PROGRAMS INVOLVING MEMORY ACCESS INSTRUCTION FOR VARIOUS DATA SIZES AND ADDRESSING MODES.	25-28
A)	Write an ALP to transfer data from one memory location to another using various addressing modes.	25
B)	Write an ALP to perform data exchange between memory locations.	27
4	PROGRAMS INVOLVING LOGICAL OPERATIONS	29-31
A)	Write a ALP to perform Logical Operations.	29
B)	Write an ALP for swapping the contents of the registers using logical operations.	31
5	PROGRAM INVOLVING DATA CONVERSION OPERATIONS	32
	Write an ALP for conversion of a number from ASCII to HEX and HEX to ASCII.	32
6	PROGRAM INVOLVING SHIFT AND ROTATE OPERATIONS	33-34
A)	Write and Assembly Language Programming to perform different types of shift and rotate instructions	33
B)	Write an Assembly Language Program to perform swapping of content within a register using rotate instruction.	34
7	PROGRAM TO ILLUSTRATE BIT FIELD PROCESSING INSTRUCTION	35-38
A)	Write an ALP to find the no's of zero's and one's in an 32 bit integer.	35
B)	Write an ALP to check if the given number is POSITIVE/NEGATIVE	36
C)	Write an ALP to check if the given number is ODD/EVEN	37
8	PROGRAM TO ILLUSTRATE PROGRAM FLOW (BRANCHING) INSTRUCTION.	39-43
A)	Write a ALP to Search a 32 bit number in given array, if found display ffffffff in r0.	39
B)	Write a ALP to find the Largest/Smallest number in an array of five numbers.	41
C)	Write a Assembly language Program for sorting of an 32 bit numbers in ascending/descending order.	42-43

9	PROGRAM TO ILLUSTRATE SATURATION OPERATION (Thumb Instructions)	44-46
A)	Write an AL P to illustrate unsigned saturation instruction and clearing the Q flag.	44
B)	To write an ALP to illustrate signed saturation instruction and clearing the Q flag.	45
10	PROGRAMS INVOLVING FLOATING OPERATIONS	-47
	To Write an Assembly language Program to perform Floating Point Calculations.	47
11	CYCLE-II :INTERFACING USING ARM STM32F401xx.	48
A)	Write a C Program for Blinking of a LED without delay.	50
B)	Write a C Program for Blinking of a LED with delay (setout fn) /PWM.	51
C)	Write a C Program for Blinking of a LED with delay.	53
D)	Write a Program for Serial communication using usart0 to display a message in usart window.	54
E)	Write a Program for Multiply and Accumulate among 2arrays and display output in USART window.	54
	ARM Cortex M4Viva Questions.	56

Scheme of Evaluation

Evaluation Criteria	Weightage
CIE – I	50 Marks
CIE – II (2 – Lab Internals)	
SEE	50 Marks

Prepared By	Verified By	Approved By
Daliya.V.K.	Dr. Rajesh.G	Dr. Sanjeev Sharma
Senior Assistant Professor / ECE	Senior Assistant Professor / ECE	Professor / HoD -ECE

CYCLE –I (ASSEMBLY LEVEL PROGRAM USING MDK ARM uKEIL5)**INTRODUCTION ABOUT ARM CORTEX M4**

The Cortex-M4 processor is developed to address digital signal control markets that demand an efficient, easy-to-use blend of control and signal processing capabilities. The combination of high-efficiency signal processing functionality with the low-power, low cost and ease-of-use benefits of the Cortex-M family of processors is designed to satisfy the emerging category of flexible solutions specifically targeting the motor control, automotive, power management, embedded audio and industrial automation markets.

arm CORTEX[®]-M4

Nested vectored
interrupt controller

Wake-up interrupt
controller

CPU
Armv7-M

Memory protection unit

DSP

FPU

3x
AHB-Lite

ITM trace

Data
watchpoint

JTAG

ETM trace

Breakpoint
unit

Serial wire

Cortex-M4 core peripherals

Nested Vectored Interrupt Controller: The NVIC is an embedded interrupt controller that supports low latency interrupt processing

System Control Block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

The system timer SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

The Memory Protection Unit (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

The Floating-Point Unit (FPU) provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values

Programmer's model:

Processor mode and privilege levels for software execution processor modes are: Thread mode Used to execute application software. The processor enters Thread mode when it comes out of reset. Handler mode Used to handle exceptions. The processor returns to thread mode when it has finished all exception processing.

The privilege levels for software execution are

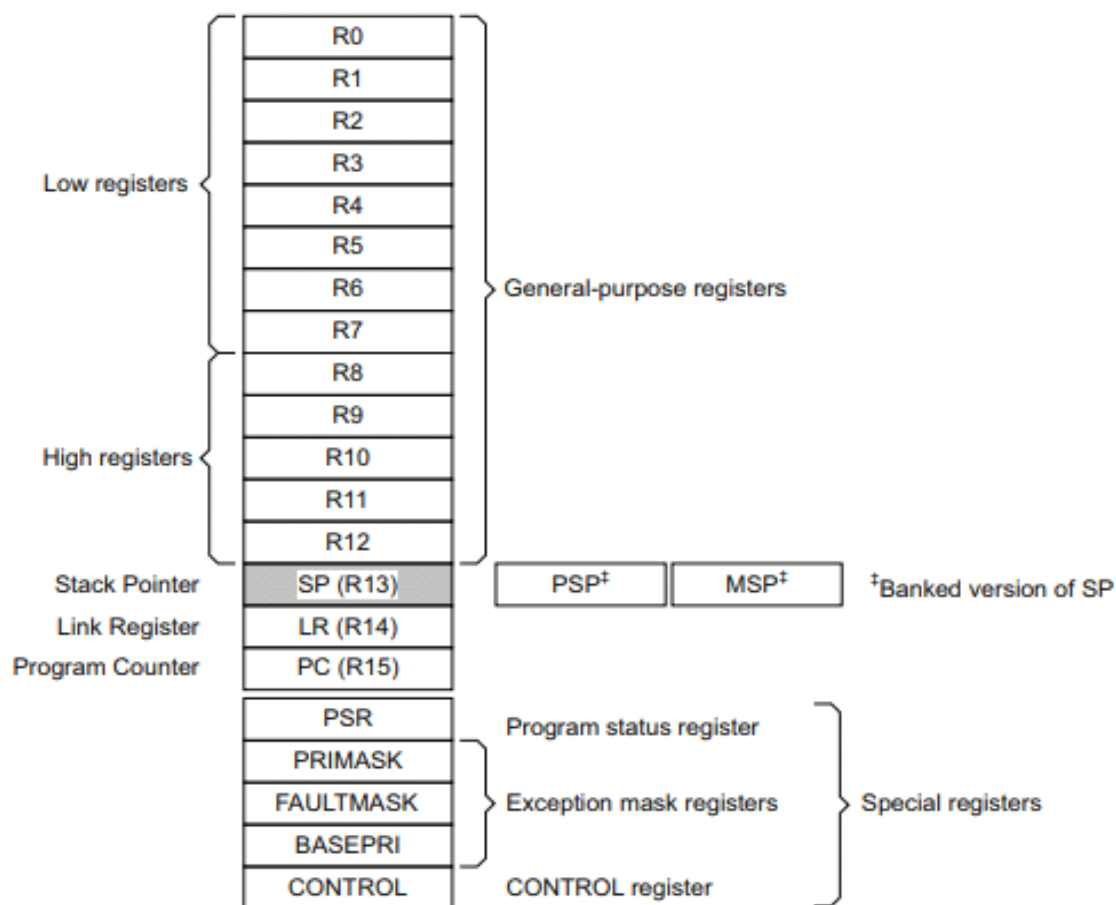
Unprivileged the software: has limited access to the MSR and MRS instructions, and cannot use the CPS instruction cannot access the system timer, NVIC or system control block might have restricted access to memory or peripherals. Unprivileged software executes at the unprivileged level.

Privileged The software can use all the instructions and has access to all resources. Privileged software executes at the privileged level.

The processor uses a full descending stack. This means the stack pointer holds the address of the last stacked item in memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location.

Core registers

The processor core registers are:



General-purpose registers R0-R12 are 32-bit general-purpose registers for data operations.

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use: 0 = Main Stack Pointer (MSP). This is the reset value. 1 = Process Stack Pointer (PSP). On reset, the processor loads the MSP with the value from address 0x00000000.

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor sets the LR value to 0xFFFFFFFF.

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

The Program Status Register (PSR) combines:

Application Program Status Register (APSR)

Interrupt Program Status Register (IPSR)

Execution Program Status Register (EPSR)

Key benefits

- Gain the advantages of a microcontroller with integrated DSP, SIMD, and MAC instructions that simplify overall system design, software development and debug.
- Accelerate single precision floating-point operations up to ten times over the equivalent integer software library with the optional Floating-Point Unit (FPU)
- Develop solutions for a large variety of markets with a full-featured Armv7-M instruction set that has been proven across a broad set of embedded applications.
- Achieve exceptional 32-bit performance with low dynamic power, delivering leading system energy efficiency due to integrated software controlled sleep modes, extensive clock gating and optional state retention.

Key features**SIMD, saturating arithmetic, fast MAC**

Powerful instruction set for accelerating DSP applications, built right into the processor. A highly optimized DSP library built using these instructions is available free-of-charge from the Arm website.

Powerful debug and non-obtrusive real-time trace

Comprehensive debug and trace features dramatically improve developer productivity. It is extremely efficient to develop embedded software with proper debug.

Memory Protection Unit (MPU)

Software reliability improves when each module is allowed access only to specific areas of memory required for it to operate. This protection prevents unexpected access that may overwrite critical data.

Integrated nested vectored interrupt controller (NVIC)

There is no need for a standalone external interrupt controller. Interrupt handling is taken care of by the NVIC removing the complexity of managing interrupts manually via the processor.

Thumb-2 code density

On average, the mix between 16bit and 32bit instructions yields better code density when compared to 8bit and 16bit architectures. This has significant advantages in terms of reduced memory requirements and maximizing the usage of precious on-chip Flash memory.

Applications

The Cortex-M4 has been specifically developed for partners to design high-performance low-cost devices for a broad range of digital signal control embedded market segments

- Connectivity
- IoT
- Audio
- Sensor fusion and wearable
- Signal processing such as power and motor control
- Smart home/enterprise/building/planet

ARM Cortex M4 Instruction set summary

Mnemonic	Operands	Brief description
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry
ADD, ADDS	{Rd,} Rn, Op2	Add
ADD, ADDW	{Rd,} Rn, #imm12	Add
ADR	Rd, label	Load PC-relative Address
AND, ANDS	{Rd,} Rn, Op2	Logical AND
ASR, ASRS	Rd, Rn, <Rs #n>	Arithmetic Shift Right
B	label	Branch
BFC	Rd, #lsb, #width	Bit Field Clear
BFI	Rd, Rn, #lsb, #width	Bit Field Insert
BIC, BICS	{Rd,} Rn, Op2	Bit Clear
BKPT	#imm	Breakpoint
BL	label	Branch with Link
BLX	Rm	Branch indirect with Link
BX	Rm	Branch indirect
CBNZ	Rn, label	Compare and Branch if Non Zero
CBZ	Rn, label	Compare and Branch if Zero
CLREX	-	Clear Exclusive
CLZ	Rd, Rm	Count Leading Zeros
CMN	Rn, Op2	Compare Negative
CMP	Rn, Op2	Compare
CPSID	i	Change Processor State, Disable Interrupts
CPSIE	i	Change Processor State, Enable Interrupts

DMB	-	Data Memory Barrier
DSB	-	Data Synchronization Barrier
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR
ISB	-	Instruction Synchronization Barrier
IT	-	If-Then condition block
LDM	Rn{!}, reglist	Load Multiple registers, increment after
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after
LDR	Rt, [Rn, #offset]	Load Register with word
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes
LDREX	Rt, [Rn, #offset]	Load Register Exclusive
LDREXB	Rt, [Rn]	Load Register Exclusive with Byte
LDREXH	Rt, [Rn]	Load Register Exclusive with Halfword
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with Halfword
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with Signed Byte
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with Signed Halfword
LDRT	Rt, [Rn, #offset]	Load Register with word
LSL, LSLS	Rd, Rm, <Rs #n>	Logical Shift Left
LSR, LSRS	Rd, Rm, <Rs #n>	Logical Shift Right
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result
MOV, MOVs	Rd, Op2	Move
MOVT	Rd, #imm16	Move Top
MOVW, MOV	Rd, #imm16	Move 16-bit constant
MRS	Rd, spec_reg	Move from Special Register to general register
MSR	spec_reg, Rm	Move from general register to Special Register
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result
MVN, MVNS	Rd, Op2	Move NOT
NOP	-	No Operation
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT
ORR, ORRS	{Rd,} Rn, Op2	Logical OR
PKHTB, PKHBT	{Rd,} Rn, Rm, Op2	Pack Halfword

POP	reglist	Pop registers from stack
PUSH	reglist	Push registers onto stack
QADD	{Rd,} Rn, Rm	Saturating double and Add
QADD16	{Rd,} Rn, Rm	Saturating Add 16
QADD8	{Rd,} Rn, Rm	Saturating Add 8
QASX	{Rd,} Rn, Rm	Saturating Add and Subtract with Exchange
QDADD	{Rd,} Rn, Rm	Saturating Add
QDSUB	{Rd,} Rn, Rm	Saturating double and Subtract
QSAX	{Rd,} Rn, Rm	Saturating Subtract and Add with Exchange
QSUB	{Rd,} Rn, Rm	Saturating Subtract
QSUB16	{Rd,} Rn, Rm	Saturating Subtract 16
QSUB8	{Rd,} Rn, Rm	Saturating Subtract 8
RBIT	Rd, Rn	Reverse Bits
REV	Rd, Rn	Reverse byte order in a word
REV16	Rd, Rn	Reverse byte order in each halfword
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend
ROR, RORS	Rd, Rm, <Rs #n>	Rotate Right
RRX, RRXS	Rd, Rm	Rotate Right with Extend
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract
SADD16	{Rd,} Rn, Rm	Signed Add 16
SADD8	{Rd,} Rn, Rm	Signed Add 8
SASX	{Rd,} Rn, Rm	Signed Add and Subtract with Exchange
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract
SDIV	{Rd,} Rn, Rm	Signed Divide
SEL	{Rd,} Rn, Rm	Select bytes

SEV	-	Send Event
SHADD16	{Rd,} Rn, Rm	Signed Halving Add 16
SHADD8	{Rd,} Rn, Rm	Signed Halving Add 8
SHASX	{Rd,} Rn, Rm	Signed Halving Add and Subtract with Exchange
SHSAX	{Rd,} Rn, Rm	Signed Halving Subtract and Add with Exchange
SHSUB16	{Rd,} Rn, Rm	Signed Halving Subtract 16
SHSUB8	{Rd,} Rn, Rm	Signed Halving Subtract 8
STM	Rn{!}, reglist	Store Multiple registers, increment after
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after
STR	Rt, [Rn, #offset]	Store Register word
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive
STREXB	Rd, Rt, [Rn]	Store Register Exclusive Byte
STREXH	Rd, Rt, [Rn]	Store Register Exclusive Halfword

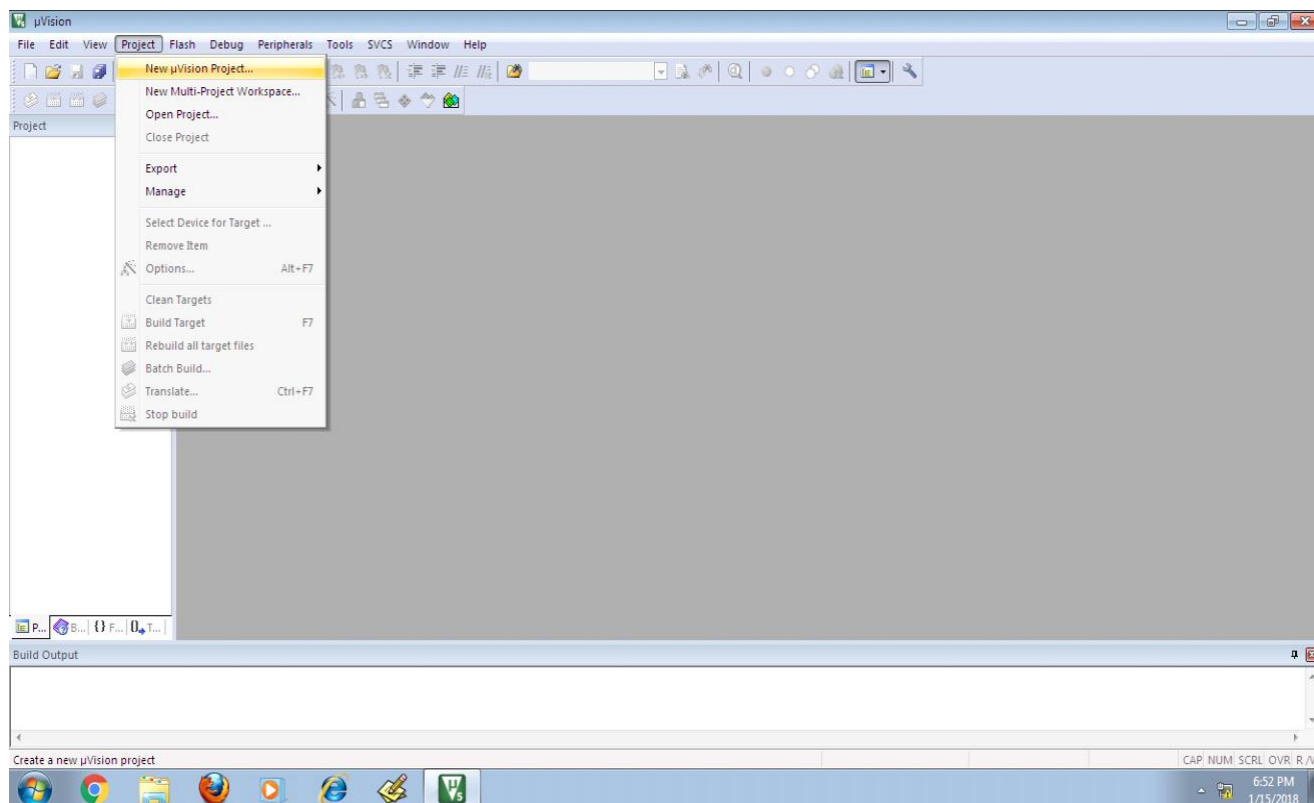
STRH, STRHT	Rt, [Rn, #offset]	Store Register Halfword
STRT	Rt, [Rn, #offset]	Store Register word
SUB, SUBS	{Rd,} Rn, Op2	Subtract
SUB, SUBW	{Rd,} Rn, #imm12	Subtract
SVC	#imm	Supervisor Call
SXTAB	{Rd,} Rn, Rm, {, ROR #}	Extend 8 bits to 32 and add
SXTAB16	{Rd,} Rn, Rm, {, ROR #}	Dual extend 8 bits to 16 and add
SXTAH	{Rd,} Rn, Rm, {, ROR #}	Extend 16 bits to 32 and add
SXTB16	{Rd,} Rm {, ROR #n}	Signed Extend Byte 16
SXTB	{Rd,} Rm {, ROR #n}	Sign extend a byte
SXTH	{Rd,} Rm {, ROR #n}	Sign extend a halfword
TBB	[Rn, Rm]	Table Branch Byte
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword
TEQ	Rn, Op2	Test Equivalence
TST	Rn, Op2	Test
UADD16	{Rd,} Rn, Rm	Unsigned Add 16
UADD8	{Rd,} Rn, Rm	Unsigned Add 8
USAX	{Rd,} Rn, Rm	Unsigned Subtract and Add with Exchange
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract

UDIV	{Rd,} Rn, Rm	Unsigned Divide
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long (32 x 32 + 32 + 32), 64-bit result
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8
VDIV.F32	{Sd,} Sn, Sm	Floating-point Divide
VFMA.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Accumulate
VFNMA.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Accumulate
VFMS.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Subtract

UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	-
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	-
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	-
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	-
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences	-
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate	-
USAT	Rd, #n, Rn {, shift #s}	Unsigned Saturate	Q
USAT16	Rd, #n, Rm	Unsigned Saturate 16	Q
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE
UXTAB	{Rd,} Rn, Rm, {, ROR #}	Rotate, extend 8 bits to 32 and Add	-
UXTAB16	{Rd,} Rn, Rm, {, ROR #}	Rotate, dual extend 8 bits to 16 and Add	-
UXTAH	{Rd,} Rn, Rm, {, ROR #}	Rotate, unsigned extend and Add Halfword	-
UXTB	{Rd,} Rn {, ROR #n}	Zero extend a Byte	-
UXTB16	{Rd,} Rn {, ROR #n}	Unsigned Extend Byte 16	-
UXTH	{Rd,} Rn {, ROR #n}	Zero extend a Halfword	-
VABS.F32	Sd, Sn	Floating-point Absolute	-
VADD.F32	{Sd,} Sn, Sm	Floating-point Add	-
VCMP.F32	Sd, <Sn #0.0>	Compare two floating-point registers, or one floating-point register and zero	FPSCR
VCMP.E.F32	Sd, <Sn #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check	FPSCR
VCVT.S32.F32	Sd, Sn	Convert between floating-point and integer	-
VCVT.S16.F32	Sd, Sd, #fbits	Convert between floating-point and fixed point	-
VCVTR.S32.F32	Sd, Sn	Convert between floating-point and integer with rounding	-
VCVT<B H>.F32.F16	Sd, Sn	Converts half-precision value to single-precision	-

VFNMS.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Subtract
VLDM.F<32 64>	Rn{!}, list	Load Multiple extension registers
VLDR.F<32 64>	<Dd Sd>, [Rn]	Load an extension register from memory
VLMA.F32	{Sd,} Sn, Sm	Floating-point Multiply Accumulate
VLMS.F32	{Sd,} Sn, Sm	Floating-point Multiply Subtract
VMCV.F32	Sd, #imm	Floating-point Move immediate
VMCV	Sd, Sn	Floating-point Move register
VMCV	Sn, Rt	Copy ARM core register to single precision
VMCV	Sn, Sn1, Rt, Rt2	Copy 2 ARM core registers to 2 single precision
VMCV	Dd[x], Rt	Copy ARM core register to scalar
VMCV	Rt, Dn[x]	Copy scalar to ARM core register
VMRS	Rt, FPSCR	Move FPSCR to ARM core register or APSR
VMSR	FPSCR, Rt	Move to FPSCR from ARM Core register
VMUL.F32	{Sd,} Sn, Sm	Floating-point Multiply
VNEG.F32	Sd, Sn	Floating-point Negate
VNMLA.F32	Sd, Sn, Sm	Floating-point Multiply and Add
VNMLS.F32	Sd, Sn, Sm	Floating-point Multiply and Subtract
VNMUL	{Sd,} Sn, Sm	Floating-point Multiply
VPOP	list	Pop extension registers
VPUSH	list	Push extension registers
VSQRT.F32	Sd, Sn	Calculates floating-point Square Root
VSTM	Rn{!}, list	Floating-point register Store Multiple
VSTR.F<32 64>	Sd, [Rn]	Stores an extension register to memory
VSUB.F<32 64>	{Sd,} Sn, Sm	Floating-point Subtract
WFE	-	Wait For Event
WFI	-	Wait For Interrupt

Procedure to create a project



Step1: Open a new project in Keilµvision 5

Step2: Select the device for the target

Device: Legacy Device Database

Select ST microelectronics STM32F401RE

Make some necessary changes if required in startup file and add it in the SOURCE GROUP

In the new file write the program in assembly (Save as .s) or C.

Add the file in SOURCE GROUP

Step 3: Build the file and check for any errors

Step 4: DEBUG the file and RUN the program

1. PROGRAM INVOLVING GENERAL DATA PROCESSING INSTRUCTIONS

A) Aim : To write an Assembly Language Program **to transfer contents within the registers.**

Software Required: Keil μ Vision 5 Software

AREA PROG1, CODE, READONLY

ENTRY

EXPORT START

START

MOV R1, #0

MOV R2, #05

MOV R3, #04

MOVS R0, R1

MOVEQ R0, R2 .

MOVS R3, #0

MOVEQ R0, R3

MVN R2, R2

STOP B STOP

END

Algorithm:

- 1) Initialize the Code Area.
- 2) Link the Program Code to the processor.
- 3) Load the digits in R1, R2, R3 registers.
- 4) Move the value from R1 to R0 and activate the flags.
- 5) Move content of R2 to R0 if R0 has value 0.
- 6) Move the value 0 to R3 and activate flags.
- 7) Move the content of R3 to R0 if R3 contains value 0.

8) Move negative of R2 to R2.

9) End the Code Area.

B) Aim: To Write a Assembly Language Program for adding two 32 bit number

Software Required: KeilµVision 5 Software

AREA PROG1, CODE, READONLY

ENTRY

EXPORT START

START

LDR R0, =10

LDR R1, =20

ADD R2, R0, R1

STOP B STOP

END

RESULT:

R0 = 0X0000000A

R1 = 0X00000014

R2= 0X0000001E

Algorithm:

- 1) Initialize the Code Area
- 2) Link the Program Code to the processor
- 3) Load the Value of R1& R0 (Decimal)
- 4) Add R1& R0 and Store the result in R2
- 5) End the Code Area

C) Aim: To Write a Assembly Language Program **for Addition of first Ten Number.**

Software Required: Keil μ Vision 5

```
AREA PROG2, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R0, =10
    LDR R1, =0
    LDR R2, =1
LOOP
    ADD R1, R2, R1
    ADD R2, R2, #1
    SUBS R4, R0, R2
    BNE LOOP
    ADD R1, R2, R1
STOP B STOP
END
```

RESULT:

R0 = 0X0000000A

R1 = 0X00000037

R2 = 0X0000000A

Algorithm:

- 1) Initialize the Code Area.
- 2) Link the Program Code to the processor.
- 3) Load the Count in R0 and give initial values to R1& R2 (Decimal).
- 4) Add R1& R2 and Store the result in R1.
- 5) Increment to next number by adding 1 to R2.
- 6) Decrement the count and keep repeating the loop unless the count equals 0.
- 7) Perform the addition of last number with the result and store the final result in R1.
- 8) End the Code Area.

D) Aim: To Write a Assembly Language Program for reverse Subtraction of two 32 numbers

Software Required: Keil μ Vision 5

```
AREA MYCODE, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R1, =0X11111111
    LDR R2, =0X22222222
    RSB R0, R1, R2
STOP B STOP
END
```

RESULT:

R2 =0X22222222

R1 =0X11111111

R1 =0X11111111

Algorithm:

- 1) Initialize the Code Area
- 2) Link the Program Code to the processor
- 3) Load the Value of R1& R2
- 4) Reverse Subtraction of R1& R2 and Store the result in R0(R0=R2-R1)
- 5) End the Code Area

E) Aim: To Write a Assembly Language Program **to find factorial of a number**

Software Required: KeilµVision 5

```
AREA PROG5, CODE, READONLY
ENTRY
EXPORT
START
    MOV R6, #10
    MOV R7, #1
LOOP CMP R6, #0
    MULGT R7, R6, R7
    SUBGT R6, R6, #1
    BGT LOOP
STOP B STOP
END
```

RESULT

R7=0X00375F00

Algorithm

- 1) Initialize the Code Area and allocate Memory.
- 2) Load R6 with the number.
- 3) Load R7 with value 1.
- 4) Compare R6 with value 0.
- 5) If content of R6 is greater than 0, multiply values of R6 and R7 and store the results in R7.
- 6) Subtract R6 by 1.
- 7) Repeat the loop till R6 contains the value 0.
- 8) End the Code Area

2. PROGRAMS INVOLVING MEMORY ACCESS INSTRUCTION FOR VARIOUS DATA SIZES AND ADDRESSING MODES

A. Aim: To write an assembly language program to transfer data from one memory location to another using various addressing modes.

AREA DATATRANSFER, CODE, READONLY

ENTRY

EXPORT START

START

LDR R0, =0x20000000

LDR R1, =0x20000016

LDR R2, =0x20000042

LDRH R4,[R2]

UP LDRB R3, [R0]

STRB R3, [R1]

ADD R0, R0, #1

ADD R1, R1, #1

ADD R3,R3,#1

STRB R3,[R0]

ADD R4, R4, #-1

CMP R4, #0

BEQ NEXT

B UP

NEXT LDR R5, =0X12345678

STRB R5,[R1]

STRH R5,[R1,#1]

STR R5,[R1,#4]

STOP B STOP

END

Algorithm

1. Initialise the code area and allocate the memory.
2. Load R0,R1 and R2 with memory locations
3. Load register R4 with the lower half word (16 bits) of the content of memory location pointed by R2.
4. Load R3 with lower order byte of the content of memory location pointed by R0.
5. Store the byte available in R3 to memory location pointed by R1.

6. Increment the memory locations pointed by R0, R1 and R3.
7. Store the byte in R3 to address location pointed by R0.
8. Decrement R4 by 1.
9. Repeat the loop until content of R4 becomes 0.
10. Load R5 with a 32 bit number.
11. Store the lower byte of R5 to memory location pointed by R1.
12. Store Lower Half Word of R5 to the location pointed by R1, by incrementing it to the next location.
13. Store the 32 bit number in R5 to the location pointed by R1 by incrementing it by 4 positions.
14. End the code area.

B. Aim: To write an Assembly Language Program **to perform data exchange between memory locations.**

```
AREA WORD,CODE,READONLY
```

```
NUM EQU 8
```

```
EXPORT START
```

```
START
```

```
    LDR R0,=0X20000000
```

```
    LDR R1,=0X2000002C
```

```
    MOV R2,#NUM
```

```
    MOV R5,#0
```

```
WORDCOPY
```

```
    LDR R3,[R0,R5]
```

```
    LDR R4,[R1,R5]
```

```
    STR R3,[R1,R5]
```

```
    STR R4,[R0,R5]
```

```
    SUBS R2,R2,#4
```

```
    ADD R5,R5,#4
```

```
    BNE WORDCOPY
```

```
    STOP B STOP
```

```
    END
```

RESULT

Before Execution

R0 =0X20000000	11	22	33	44
----------------	----	----	----	----

R1 =0X20000050	01	02	03	04
----------------	----	----	----	----

After Execution

R3 `	11	22	33	44
------	----	----	----	----

R4	01	02	03	04
----	----	----	----	----

R0 =0X20000000	01	02	03	04
----------------	----	----	----	----

R1 =0X20000050	11	22	33	04
----------------	----	----	----	----

Algorithm

1. Load R0 and R1 with address of memory locations.
2. Load R2 with the count of bytes used.
3. Load R5 with value 0.
4. Load R3 with the memory location pointed by addition of content of R0 with R5.
5. Load R4 with the memory location pointed by addition of content of R0 with R5.
6. Store content of R3 to the memory location pointed by addition of content of R1 with R5.
7. Store content of R4 to the memory location pointed by addition of content of R0 with R5.
8. Subtract the count of bytes in R2 by 4.
9. Add the memory location pointed by R5 by 4.
10. Repeat the loop till the count in R2 reaches 0.
11. End the code area.

3. PROGRAM INVOLVING LOGICAL OPERATIONS

A. Aim: To write a Assembly language Program **to perform Logical Operations.**

Software Required: Keil μ Vision 5

```
AREA MYCODE,CODE,READONLY
ENTRY
EXPORT START
START
    LDR R0, =0X00000000
    LDR R1, =0X00000005
    LDR R2, =0X0000000F
    AND R0,R1,R2
    ORR R3,R1,R2
    EOR R4,R1,R2
    BIC R7,R2
    BIC R5, R2,R1
STOP B STOP
END
```

RESULT :

```
R0= 0X00000000
R1= 0X00000005
R3= 0X0000000F
R4 = 0X0000000A
R7 = 0X00000000
R5 = 0X00000005
```

Algorithm

1. Initialise the code area and allocate the memory.
2. Load R0,R1 and R2 with data.
3. Perform AND operation between the contents of R1 and R2 and store the results in R0.
4. Perform OR operation between the contents of R1 and R2 and store the results in R3.
5. Perform EXOR operation between the contents of R1 and R2 and store the results in R4.

6. Perform bitwise AND between the contents of R2 and R7 and store the results in R7.
7. Perform bitwise AND operation between the contents of R1 and R2 and store the results in R5
8. End of Code area.

B. To write an Assembly Language Program for swapping the contents of the registers using logical operations.

Software Required: **KeilµVision 5**

```
AREA PROG4, CODE, READONLY
ENTRY
EXPORT
START
    LDR R0,=0XF631024C
    LDR R1,=0X17539ABD
    EOR R0, R0, R1
    EOR R1, R0, R1
    EOR R0, R0, R1
STOP B STOP
END
```

RESULT

R1=0XF631024C
R0=0X17539ABD

Algorithm

1. Initialise the code area and allocate the memory.
2. Load R0 and R1 with 32 bit data.
3. Perform EXOR operation between the contents of R0 and R1 and store the results in R0.
4. Perform EXOR operation between the contents of R0 and R1 and store the results in R1.
5. Perform EXOR operation between the contents of R0 and R1 and store the results in R0.
6. End of Code area.

4. PROGRAMS INVOLVING DATA CONVERSION OPERATIONS.

To Write an Assembly language Program for conversion of a number from ASCII to HEX and HEX to ASCII.

Software Required: Keil μ Vision 5

```

AREA DATA1, DATA, READONLY
DIGIT DCD & 0F
AREA DATA2, DATA, READWRITE
RESULT DCD 0
AREA PROGRAM, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R0, DIGIT
    LDR R1, =RESULT
    CMP R0, #0XA      ; JUST CHANGE INTO 3A FOR HEX TO ASCII
    BLT ADD_0
    ADD R0, R0, #07    ; CHANGE ADD TO SUB FOR HEX TO ASCII
ADD_0 ADD R0, R0, #"0" ; CHANGE ADD TO SUB FOR HEX TO ASCII
    STR R0, [R1]
    STOP B STOP
    END

```

RESULT :

R0= 0X00000046 (ASCII TO HEX)

R0= 0X0000000F (HEX TO ASCII)

Algorithm:

- 1) Initialize the Code Area and allocate Memory.
- 2) Load the given ASCII value to R0.
- 3) Load R1 with the memory location for storing the result.
- 4) Compare R0 with 0A (hex).
- 5) If the result is less than 0A, add '0' (number 30 in hex) and store the results in R0.
- 6) Else, if the result is more than 0A, add 7 and then add '0' and store the results in R0.
- 7) Store the obtained number from R0 to memory location pointed by R1.
- 8) End the Code Area.

5. PROGRAMS INVOLVING SHIFT AND ROTATE INSTRUCTIONS

- A. Aim: To Write an Assembly Language Programming to perform different types of shift and rotate instructions.**

AREA ROTATESHIFT, CODE, READONLY

ENTRY

EXPORT START

START

LDR R0, =0XF0000001

LSLS R1, R0, #1

ASRS R1, R0, #1

RORS R2, R0, #1

RRXS R3, R0

STOP

B STOP

END

Algorithm:

- 1) Initialize the Code Area and allocate Memory.
- 2) Load R0 with a number.
- 3) Logically left shift the content of R0 by 1 bit and load the value to R1.
- 4) Logically right shift the content of R0 by 1 bit and load the value to R2.
- 5) Arithmetically shift right the contents of R0 by 1 and store the results in R1.
- 6) Rotate right the content of R0 by 1 bit and store the results in R2.
- 7) Rotate the content of R0 through carry and store the results in R3.
- 8) End the Code Area.

- B. Aim: To Write an Assembly Language Program to perform swapping of content within a register using rotate instruction.**

AREA PROG, CODE, READONLY

ENTRY

EXPORT START

START

LDR R0,=0XF631024C

ROR R1,R0,#16

LDR R2,=0X20000000

STRH R1,[R2]

LDRH R3,[R2]

ROR R4,R3,#16

MOV R5,#2

STR R4,[R2,R5]

STOP B STOP

END

Algorithm:

- 1) Initialize the Code Area and allocate Memory.
- 2) Load R0 with a 32 bit number.
- 3) Rotate right, the content of R0 by 16 bits and store the results in R1.
- 4) Load R2 with the address of memory location.
- 5) Store the lower order half word of the content of register R1 to the memory location pointed by R2.
- 6) Load the half word from the memory location pointed by R2 to R3.
- 7) Rotate the content of R3 right by 16 bits and store the results in R4.
- 8) Load R5 with value 2.
- 9) Store the value in R4 to the memory location obtained after adding the content of R2 with R5.
- 8) End the Code Area.

6. PROGRAMS TO ILLUSTRATE BIT FIELD PROCESSING INSTURCTIONS.

A. Aim: To write an Assembly Language Program **to find the no's of zero's and one's in a 32 bit integer.**

Software Required: **Keil μ Vision 5**

```
AREA DATA1, DATA, READONLY
    DCB 0X0000008F
    AREA DATA2, DATA, READWRITE
ONES  DCB 0
ZEROS DCB 0
    AREA PROGRAM, CODE, READONLY
    ENTRY
    EXPORT START
START
    LDR R0,=DATA1
    EOR R5, R5, R5
    EOR R6, R6, R6
    MOV R1, #32
    LDR R3,[R0]
TOP   TST R3,#01
    BEQ INC_ZERO
    ADD R6,#1
    B DN
INC_ZERO
    ADD R5,#1
DN    LSR R3, #1
    SUB R1, #1
    CMP R1,#0
    BNE TOP
    LDR R0,=ZEROS
    STRB R6,[R0]
    LDR R0,=ONES
    STRB R5,[R0]
STOP B STOP
    END
```

RESULT:

R5= 0X00000005

R6= 0X0000001B

Algorithm:

- 1) Initialize the Code Area
- 2) Load R0 with data,R1 indicates the length of array
- 3) Load R3 with the number available in R0.
- 4) Check the LSB of the number .
- 5) If it is 1, increment the count of 1's , by incrementing the counter represented by R6.
- 6) Else increment the count of 0's , by incrementing the counter represented by R5
- 7) Logically shift the data in R3, subtract content of R1 by 1 and compare with 0.
- 8) If the count in R1 is not zero, repeat the loop.
- 9) Store the values in R6 and R5 in memory locations pointed by R0.
- 6) End the Code Area

B. Aim: To write an Assembly Language Program to check if the given number is POSITIVE/NEGATIVE

```
AREA PROG2, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R0, = -11
    TST R0, #80000000
    BEQ num_is_pos
    BNE num_is_neg
num_is_pos
    MOV R3, #0X00
    B STOP
num_is_neg
    MOV R3, #0XFF
    STOP B STOP
END
```

Algorithm

1. Initialise code area
2. Load R0 with the number
3. Test the MSB of the number by using Bitwise AND with 80000000
4. If the MSB =0 , Branch to loop and Store 0 to R3 register and Stop.
5. Else if MSB =1 Branch to loop2 and store R3 with FF.
6. End code area

C. Aim: To write an Assembly Language Program to check if the given number is ODD/EVEN

```
AREA PROG2, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R0, =11
    TST R0, #1
    BEQ num_is_even
    BNE num_is_odd
num_is_even
    MOV R3, #0X00
    B STOP
num_is_odd
    MOV R3, #0XFF
    STOP B STOP
END
```

Algorithm

1. Initialise code area
2. Load the number in R0
3. Check LSB of the number
4. If it is zero, branch to the loop 1
5. Load R3 with 00 and stop
6. Else branch to loop 2
7. Load R3 with FF
8. End the code area

7. PROGRAMS TO ILLUSTRATE PROGRAM FLOW (BRANCHING) INSTRUCTION.

A) Aim: To Write a Assembly Language Program to Search a 32 bit number in given array, if found display ffffffff in r0.

Software Required: **Keil μ Vision 5**

```
AREA DATA1, DATA, READONLY
ARRY DCD 0X08F, 0X012, 0X55, 0X03, 0X088
AREA DATA2, DATA, READONLY
KEY DCD 0X088
LEN EQU 05
AREA PROGRAM, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R0,=ARRY
    LDR R2,=LEN
    LDR R3,=KEY
    LDR R4,[R3]
LOOP
    LDR R5,[R0]
    CMP R4,R5
    BEQ FOUND
    ADD R0,R0,#4
    SUBS R2,R2,#0X1
    BNE LOOP
    MOV R0,#0
    B STOP
FOUND MOV R0,#0xFFFFFFFF
STOP B STOP
END
```

RESULT :

R0= 0xFFFFFFFF (Found)

Algorithm

1. Initialise code area.
2. Load R0 with the memory location allocated for the array of numbers.
3. Load R3 with the number which has to be searched in the array and R2 with the length of the array.
4. Move the content of R3 to R4.
5. Load the first number from the location pointed by R0 to R5.
6. Compare the contents of R5 with that of R4.
7. If equal, branch to the loop found and load R0 with ffffffff. and stop.
8. Else go to the next memory location and decrement the count of numbers indicated by R2.
9. Repeat the loop till R2 equals to 0.
10. If not found, load R0 with 0.
11. End the code area.

B. Aim: To Write an Assembly Language Program to find the Largest/Smallest number in an array of five numbers.

Software Required: **Keil μ Vision 5**

;Largest in an array

```
AREA SRC, DATA, READONLY
DCD 0x0000008F, 0x00000012, 0x00000024, 0x00000023, 0x00000011
ALIGN
AREA DST, DATA, READWRITE
SPACE 02
LEN EQU 05
AREA PROGRAM, CODE, READONLY
ENTRY
EXPORT START
START
    LDR R6,=LEN
    LDR R0,=SRC
    SUB R6,R6,#01
    LDR R7,[R0],#04
TOP
    LDR R8,[R0],#04
    CMP R8, R7
    MOVGE R7, R8; Change to MOVLE for finding the smallest number
    SUB R6, R6,#1
    CMP R6, #00
    BNE TOP
    LDR R0,=DST
    STR R7,[R0]
STOP B STOP
END
```

RESULT :

R7 = 0X0000008F

Algorithm:

- 1) Initialize the Code Area.
- 2) Load R0 with data, R6 indicates the length of array.
- 3) Load R7 and R8 with contents of memory locations pointed by R0 and increment pointer by 4.
- 4) Compare R8 and R7 if R8>R7, Move Content of R8 to R7, Decrement R6 by 1.
- 5 Else if R8<R7, Decrement R6 by 1.
- 5) If R6=0; Store R7value to R0, Else GoTo Loop to Load the next data.
- 6) End the Code Area.

C) Aim: To Write a Assembly language Program **for sorting of an 32 bit numbers in ascending/descending order.**

Software Required: **Keil µ Vision 5**

```
AREA LIST, DATA, READONLY
DCD 0x0000008F, 0x00000012, 0x00000024, 0x00000023, 0x00000011
ALIGN
AREA DST, DATA, READWRITE
SPACE 10
LEN EQU 05
AREA PROGRAM, CODE, READONLY
ENTRY
EXPORT START
START LDR R0,=LEN
      LDR R5,=LIST
      LDR R6,=DST
UP    LDR R1,[R5]
      STR R1,[R6]
      ADD R5,#04
      ADD R6,#04
      SUB R0,#1
      CMP R0,#00
      BNE UP
      LDR R0,=LEN
TOP2  LDR R6,=DST
```

```
LDR R1,=LEN
SUB R1,#1
TOP LDR R2,[R6]
ADD R6,#4
LDR R3,[R6]
CMP R2,R3 ; CHANGE CMP R3,R2 FOR DESCENDING
BLE DN
STR R2,[R6]
SUB R6,#4
STR R3,[R6]
ADD R6,#4
DN SUB R1,#01
CMP R1,#00
BLE DN1
B TOP
DN1 SUB R0,#1
CMP R0,#00
BLE DN2
B TOP2
DN2 B DN2 END
```

Algorithm:

- 1) Initialize the Code Area.
- 2) Load the Data in R5 pointed to memory location.
- 3) Load R0 with first data and Load R1 with Second data.
- 4) Compare R1 & R0.
- 5) If $R1 > R0$, Keep the value without exchanging and decrement R4 or if $R0 > R1$, Exchange the value and decrement R4.
- 6) Next check if $R5 = 0$ decrement R5 else load the next values.
- 7) Continue the process, till $R5 = 0$.
- 8) End the Code Area.

Result: R6= 0x00000011, 0x00000012, 0x00000023, 0x00000024, 0x0000008F

8. PROGRAMS TO ILLUSTRATE SATURATION OPERATION (Thumb Instructions)

A. Aim: To write an Assembly Language Program **to illustrate unsigned saturation instruction and clearing the Q flag.**

PROGRAM

AREA PROG, CODE, READONLY

ENTRY

EXPORT START

START

LDR R1,=0xFFFF8000

USAT R2,#16,R1

MRS R0,APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

LDR R1,=0X00030000

USAT R3, #16, R1

MRS R0, APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

LDR R1,=0X00001234

USAT R4, #16,R1

MRS R0,APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

STOP B STOP

END

Algorithm

1. Load R1 with a number below the lower limit of unsigned saturation operation.
2. Saturate R1 with 16 bit unsigned saturation instruction and store the results in R2
3. Move the APSR flag contents to register R0.

4. Clear the bit corresponds to Q flag.
- 5, Load the APSR register with the cleared value from R0.
6. Load R1 with a number above the upper limit of unsigned saturation operation and repeat the same process.
7. Load R1 with a number within the limit of unsigned saturation operation and repeat the same process.
8. End the code area.

B. Aim: To write an Assembly Language Program **to illustrate signed saturation instruction and clearing the Q flag.**

PROGRAM

AREA PROG, CODE, READONLY

ENTRY

EXPORT START

START

LDR R1,=0xFFFF7FFF

SSAT R2,#16,R1

MRS R0,APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

LDR R1,=0X00030000

SSAT R3, #16, R1

MRS R0,APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

LDR R1,=0X00001234

SSAT R4, #16,R1

MRS R0,APSR

BIC R0,R0,1<<27

MSR APSR_nzcvq,R0

STOP B STOP

END

Algorithm

1. Load R1 with a number below the lower limit of signed saturation operation.
2. Saturate R1 with 16 bit signed saturation instruction and store the results in R2.
3. Move the APSR flag contents to register R0.
4. Clear the bit corresponds to Q flag.
- 5, Load the APSR register with the cleared value from R0.
6. Load R1 with a number above the upper limit of signed saturation operation and repeat the same process.
7. Load R1 with a number within the limit of signed saturation operation and repeat the same process.
8. End the code area.

9. PROGRAMS INVOLVING FLOATING POINT INSTRUCTIONS

Aim: To Write an Assembly language Program to perform Floating Point Calculations.

Software Required: Keil μ Vision 5

AREA PGM, CODE, READONLY

ENTRY

EXPORT START

START

LDR r0,=0XE000ED88

LDR r1,[r0]

ORR r1,r1,#(0XF<<20)

STR r1,[r0]

LDR R1,=0X0000000A

VMOV.F32 S7,#0X41280000

VCVT.F32.U32 S17,S17

VMOV.F32 S8,#0X40200000

VMOV.F32 S9,#0X40200000

VADD.F32 S10,S8,S9

VMUL.F32 S11,S8,S7

VSUB.F32 S12,S11,S10

VNEG.F32 S13,S12

VABS.F32 S14,S13

VDIV.F32 S15,S11,S10

VMLA.F32 S11,S8,S9

LOOP B LOOP

END

RESULT :

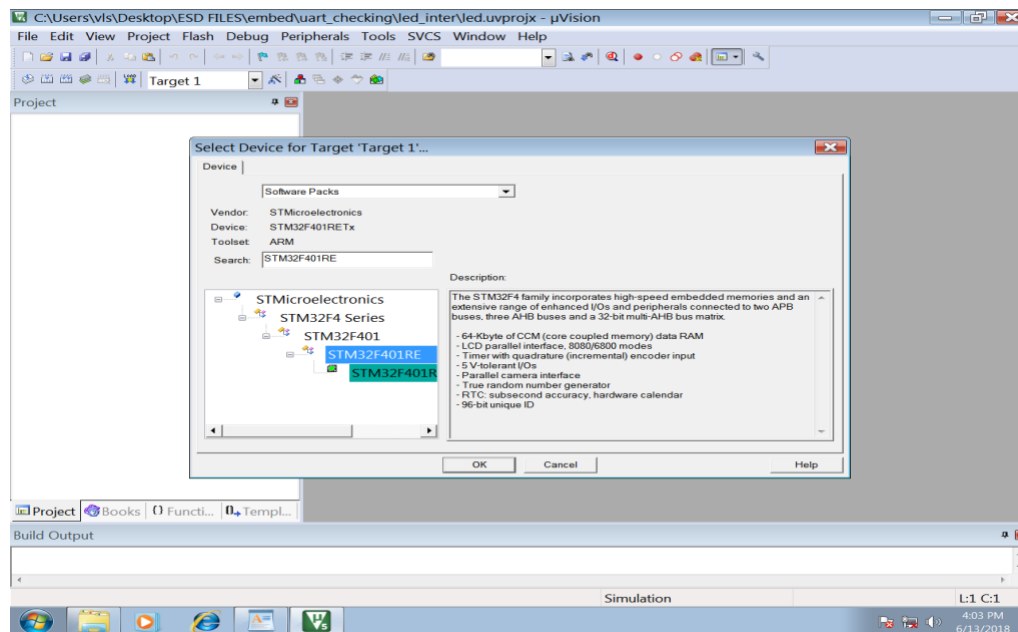
The Assembly Language Program for Floating Pont operations are Performed.

Algorithm

1. Load the register r0 with a memory location address.
2. Move the content of the location to register r1.
3. Perform OR operation of r1 with 0F left shifted by 20 bits.
4. Store the result in memory location pointed by r0.
5. Load registers R1 with 0A.
6. Move various floating point numbers to different floating point registers.
7. Perform number conversion from unsigned to floating point number.
8. Perform various Arithmetic operations of floating point number and store the results in registers assigned for floating point operations.

CYCLE-II : INTERFACING USING ARM STM32F401xx

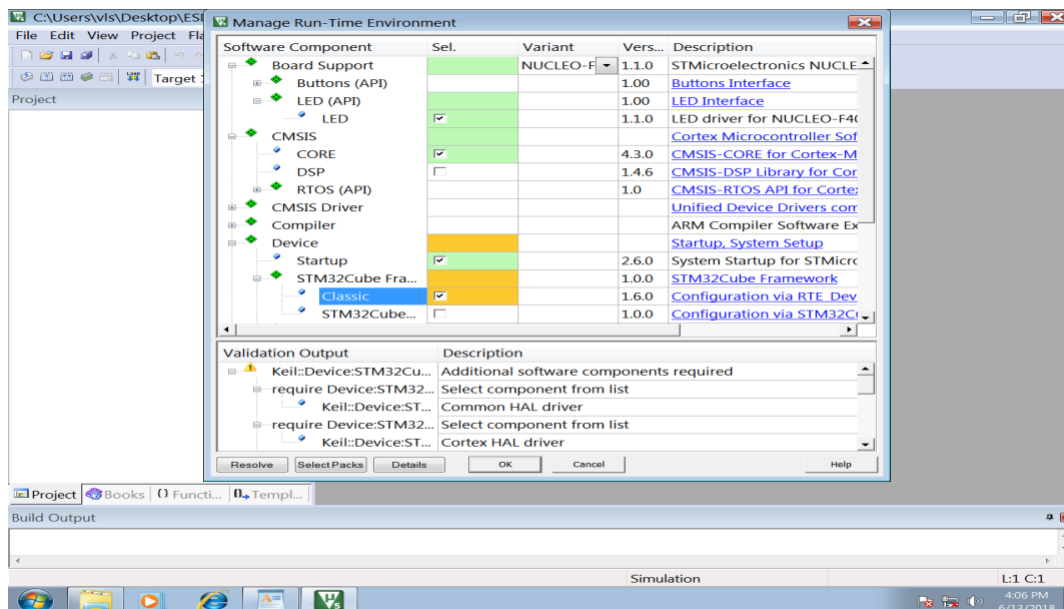
1. Blinking LED



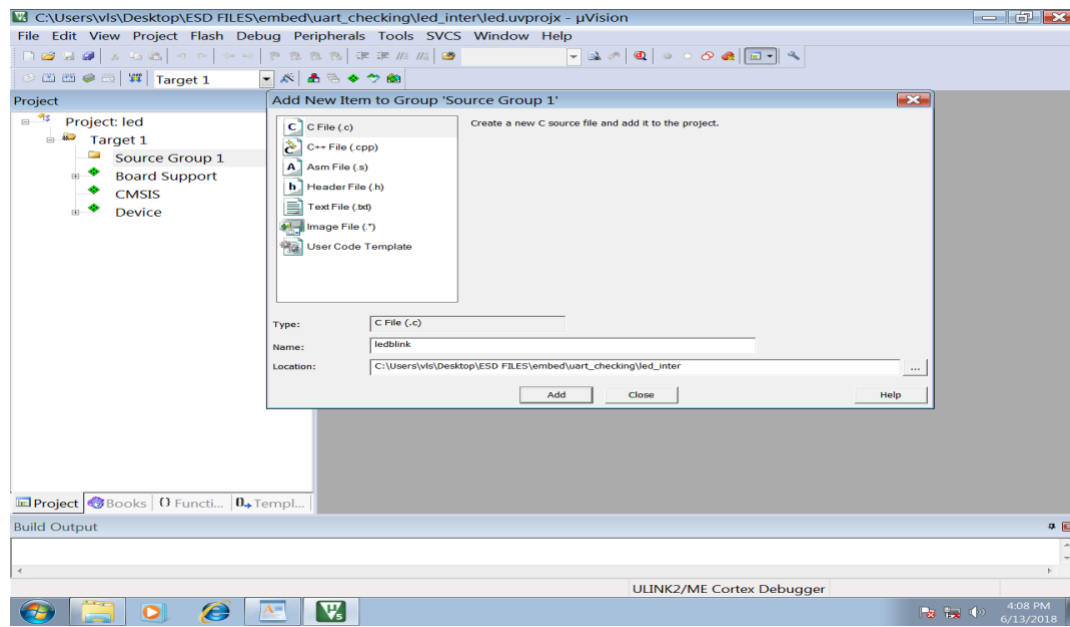
Procedure to be followed for interfacing LED.

STEP 1: Open keiluvision5, create new project select the device for the target software Packs and the device is STM32F401RE.

STEP 2: LED API has to be included by selecting a nuculeovariant and cortex microcontroller software interfacing standard core.then click on resolve then ok.



STEP 3: Add a file to the source group and save with an extension .c.



and type the program build the program without errors then connect the interfacing kit and observe the output.

A) Aim: To Write a C Program for **Blinking of an LED on and off without delay.**

Software Required: Keil μ Vision 5

```
#include"stm32F4xx_hal.h"
#include"Board_LED.h"
int main(void)
{
const unsigned intnum=0;
LED_Initialize();

while(1)
{
    LED_On(num);
    LED_Off(num);
}
}
```

RESULT :

The Program for Blinking of an LED ON and OFF without Delay is interfaced with board and outputs are verified.

Algorithm:

- 1) Indicate the Header File of STM32401RE.
- 2) Start the main function.
- 3) Declare a Variable num and initialize to Zero.
- 4) Call the Function LED Initialize.
- 5) Run a Endless Loop for On & OFF LED and Repeat.
- 6) End the Program.

B) Aim: To Write a C Program for **Blinking of an LED with delay using Set Out function.**

Software Required: Keil μ Vision 5

```
#include"stm32F4xx_hal.h"
#include"Board_LED.h"
int main(void)
{
const unsigned int off_code=0x0000;
    const unsigned int on_code=0x00ff;
    unsigned int i;
LED_Initialize();
{
for(;;)
    {
        for(i=0;i<30000;i++)
LED_SetOut(on_code);
        for(i=0;i<30000;i++)
LED_SetOut(off_code);
    }
}
}

/*int32_t LED_SetOut (uint32_t val) {
int i;
for (i = 0; i < LED_NUM; i++) {
if (val & (1<<i)) {
LED_On (i);
} else {
LED_Off(i);
}
}
return (0);
}
*/
```

RESULT :

The Program for Blinking of an LED ON and OFF with Delay is interfaced with board and outputs are verified

Algorithm:

- 1) Indicate the Header File of STM32401RE
- 2) Start the main function
- 3) Declare a Variable ON and OFF and initialize to Zero
- 4) Call the Function LED Initialize
- 5) Include a Delay using FOR Loop and Set the LED to ON State
- 6) Include a Delay using FOR Loop and Set the LED to OFF State
- 7) End the Program

C) Aim: To Write a C Program for Blinking of an LED with delay.

Software Required: Keil μ Vision 5

```
#include"stm32F4xx_hal.h"
#include"Board_LED.h"
int main(void)
{
const unsigned intnum=0;
    unsignedinti=0;
LED_Initialize();
while(1)
{
for(i=0;i<50000;i++)
LED_On(num);
for(i=0;i<50000;i++)
    LED_Off(num);
}
}
```

RESULT :

The Program for Blinking of an LED ON and OFF with Delay is interfaced with board and outputs are verified.

Algorithm:

- 1) Indicate the Header File of STM32401RE.
- 2) Start the main function.
- 3) Declare a Variable ON and OFF and initialize to Zero.
- 4) Call the Function LED Initialize.
- 5) Include a Delay using FOR Loop and Set the LED to ON State.
- 6) Include a Delay using FOR Loop and Set the LED to OFF State.
- 7) End the Program.

D) Aim: To Write a C Program for Serial communication using usart0 to display a message in USART window.

Software Required: Keil μ Vision 5

```
#include <stdio.h>          /* prototype declarations for I/O functions */
#include <stm32f10x.h>      /* STM32F10x definitions */

extern void SER_Init(void); /* see Serial.c */

/*-----
main program
*-----*/
int main (void) {          /* execution starts here */

SER_Init ();              /* initialize the serial interface */

printf ("this is akhil\n"); /* the 'printf' function call */

while (1) {               /* An embedded program does not stop and */
; /* ... */               /* never returns. We use an endless loop. */
}                          /* Replace the dots (...) with your own code.*/

}
```

Result :

The Program for Serial communication using usart0 to display a message in USART window is executed and observed the output in keil 5 IDE in serial window.

Algorithm:

- 1) Indicate the Header File of STM32401RE
- 2) Start the main function
- 3) Call the Function Serial Initialize
- 4) Run a While Loop (Endless Function)
- 5) End the Program

E) Aim: To Write a C Program **for Multiply and Accumulate among two arrays and display output in usart window.**

Software Required: Keil μ Vision 5

```
#include <stdio.h>          /* prototype declarations for I/O functions */
extern void SER_Init(void); /* see Serial.c */

/*-----
User 'strcpy' Function in In-line Assembly
*-----*/
void my_strcpy (char *dst, const char *src) {
    int ch;

    __asm {
loop: LDRB    ch, [src], #1
        STRB   ch, [dst], #1
        CMP    ch, #0
        BNE    loop
    }
}

/*-----
Multiply & Accumulate in In-line Assembly
*-----*/
#define lo64(a) (((unsigned int *)&a)[0]) /* Low 32-bits of a long long */
#define hi64(a) (((int *)&a)[1]) /* High 32-bits of a long long */

__inline __int64 mlal (__int64 sum, int a, int b) {
    __asm{ SMLAL lo64(sum), hi64(sum), a, b }
    return sum;
}

const char *txt = "Hello World!";
char buf[16];
```

```
int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
int b[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };  
__int64 sum;
```

Result : The Program for Multiply and Accumulate among two arrays and display output in usart window.is executed and observe the output in keil 5 IDE in serial window.

Algorithm:

- 1) Indicate the Header File of STM32401RE
- 2) Start the main function
- 3) Call the Inline unction
- 4) Run a While Loop (Endless Function)
- 5) End the Program

Embedded System Design Viva Questions

Explain the different classifications of embedded systems .Give an example of each

Explain the various purposes of embedded systems in detail with illustrative examples

Differentiate between RISC and CISC processors

Explain the role of Watchdog timer in Embedded System

Explain the features of Cortex M4 processors.

Explain the advantages and applications of Cortex M4 processors

Explain the architecture of Cortex M4 processors.

Explain the memory format of Cortex M4 processors.

ARM stands for _

What are t, d, m, I stands for in ARM7TDMI.

What are the profiles for ARM architecture?

Describe the Various tools found in ARM Development suite.

What is the cache memory for ARM710T

Explain the Operation Modes & Privilege Levels in ARM Cortex M4

Illustrate with diagram, The Programmer's Model for ARM Cortex M4 Processor.

Illustrate with diagram,. Memory map of cortex M4 Processor.

Number of interrupts present in Cortex-M4 processors are

What is the processor used by ARM7

How many registers are there in ARM7

What is the capability of ARM7 f instruction for a second?

