



SVR ENGINEERING COLLEGE

Approved by AICTE & Permanently Affiliated to JNTUA

Ayyalurmetta, Nandyal – 518503. Website: www.svrec.ac.in

Department of Electronics and Communication Engineering



MICROPROCESSORS AND MICROCONTROLLERS

III B.Tech (ECE) II Semester

2020-21



STUDENT NAME	
ROLL NUMBER	
SECTION	



SVR ENGINEERING COLLEGE

Approved by AICTE & Permanently Affiliated to JNTUA

Ayyalurmetta, Nandyal – 518503. Website: www.svrec.ac.in

Department of Electronics and Communication Engineering

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

ACADEMIC YEAR: 2020-21

This is to certify that the bonafide record work done by

Mr./Ms. _____ bearing

H.T.No. _____ of II B.Tech II Semester in the

Microprocessors and Microcontrollers.

Faculty In-Charge

Head of the Department

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
ANANTAPUR**

B.Tech – III-II Sem

**L T P C
0 0 4 2**

15A04607 MICROPROCESSORS AND MICROCONTROLLERS LABORATORY

LIST OF EXPERIMENTS

Part A : 8086 Microprocessor Programs using NASM/8086 microprocessor kit.

1. Introduction to MASM Programming.
2. Programs using arithmetic and logical operations
3. Programs using string operations and Instruction prefix: Move Block, Reverse string, Sorting, String comparison
4. Programs for code conversion
5. Multiplication and Division programs
6. Sorting and multi byte arithmetic
7. Programs using CALL and RET instructions

Part B Embedded C Experiments using MSP430 Microcontroller

1. Interfacing and programming GPIO ports in C using MSP430 (blinking LEDs , push buttons)
2. Usage of Low Power Modes: (Use MSPEXP430FR5969 as hardware platform and demonstrate the low power modes and measure the active mode and standby mode current)
3. Interrupt programming examples through GPIOs
4. PWM generation using Timer on MSP430 GPIO
5. Interfacing potentiometer with MSP430
6. PWM based Speed Control of Motor controlled by potentiometer connected to MSP430 GPIO
7. Using ULP advisor in Code Composer Studio on MSP430
8. Low Power modes and Energy trace++:
 - a. Enable Energy Trace and Energy Trace ++ modes in CCS
 - b. Compute Total Energy, and Estimated lifetime of an AA battery.

Note : Any six experiment from Part A and Six experiments from Part B are to be conducted

Equipment & Software Required:

Software:

1. Computer Systems with latest specifications
2. Connected in LAN (Optional)
3. Operating system (Windows XP)
4. Operating system (Windows 7)
5. Code composer (CC Studio)
6. MASM Software (Dos Box)

Equipment:

1. 8086 Trainer kit.
2. MSP 430 Launch pad MSP-EXP430G2.
3. CC3100Booster pack.
4. MSP 430 F5529 Launch pad.
5. Patch cords.

ECE DEPT VISION & MISSION PEOs and PSOs

Vision

To produce highly skilled, creative and competitive Electronics and Communication Engineers to meet the emerging needs of the society.

Mission

- Impart core knowledge and necessary skills in Electronics and Communication Engineering through innovative teaching and learning.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society
- Cultivate the students with all-round competencies, for career, higher education and self-employability

I. PROGRAMME EDUCATIONAL OBJECTIVES (PEOS)

- PEO1: Graduates apply their knowledge of mathematics and science to identify, analyze and solve problems in the field of Electronics and develop sophisticated communication systems.
- PEO2: Graduates embody a commitment to professional ethics, diversity and social awareness in their professional career.
- PEO3: Graduates exhibit a desire for life-long learning through technical training and professional activities.

II. PROGRAM SPECIFIC OUTCOMES (PSOS)

- PSO1: Apply the fundamental concepts of electronics and communication engineering to design a variety of components and systems for applications including signal processing, image processing, communication, networking, embedded systems, VLSI and control system
- PSO2: Select and apply cutting-edge engineering hardware and software tools to solve complex Electronics and Communication Engineering problems.

III. PROGRAMME OUTCOMES (PO'S)

- 1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- 3. Design/development of solutions:** 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: Use 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. Modern tool usage: 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. The engineer and society: 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. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

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

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: 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, and give and receive clear instructions.

11. Project management and finance: 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. Life-long learning: 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.

IV. COURSE OBJECTIVES

- To Understand the students with basic programming of 8086 microprocessors.
- To Understand concepts of Intel x86 series of processors.
- Program MSP 430 for designing any basic Embedded System.
- Design and implement some specific real time applications Using MSP 430 low power microcontroller.
- To design and implement different Interfacing concept techniques.

V. COURSE OUTCOMES

After the completion of the course students will be able to

Course Outcomes	Course Outcome statements	BTL
CO1	Understand Assembly Language Programming using MASM / DOSBOX.	L1
CO2	Execute 8086 ALPs for arithmetic calculations, strings, Test and Debug 8086 ALPs.	L3
CO3	Understand Programming of MSP430 using Embedded C.	L4
CO4	Observe the performance of Low Power modes and Energy trace++	L2
CO5	Observe the performance of the various interfacing devices.	L5

VI. COURSE MAPPING WITH PO'S AND PEO'S

Course Title	P01	P02	P03	P04	P05	P06	P07	P08	P09	P010	P011	P012	PE01	PE02	PE03
Microprocessors and Microcontrollers	3	3	3	2	3	3	3	3	3	3	3	2	2	3	3

V MAPPING OF COURSE OUTCOMES WITH PEO'S AND PO'S

Course Title	P01	P02	P03	P04	P05	P06	P07	P08	P09	P010	P011	P012	PE01	PE02	PE03
CO1	3	3	3	2	3	3	3	2	3	3	3	2	3	3	2
CO2	2	3	3	2	2	3	2	3	2	2	3	2	3	2	3
CO3	3	3	3	3	2	3	2	3	3	3	3	2	3	2	2
CO4	3	3	2	2	2	3	2	3	3	2	3	3	2	3	3
CO5	3	3	3	3	2	3	2	3	2	3	2	3	3	2	3

LABORATORY INSTRUCTIONS

1. While entering the Laboratory, the students should follow the dress code. (Wear shoes and White apron, Female Students should tie their hair back).
2. The students should bring their observation book, record, calculator, necessary stationery items and graph sheets if any for the lab classes without which the students will not be allowed for doing the experiment.
3. All the Equipments and components should be handled with utmost care. Any breakage or damage will be charged.
4. If any damage or breakage is noticed, it should be reported to the concerned in charge immediately.
5. The theoretical calculations and the updated register values should be noted down in the observation book and should be corrected by the lab in-charge on the same day of the laboratory session.
6. Each experiment should be written in the record note book only after getting signature from the lab in-charge in the observation notebook.
7. Record book must be submitted in the successive lab session after completion of experiment.
8. 100% attendance should be maintained for the laboratory classes.

Precautions.

1. Check the connections before giving the supply.
2. Observations should be done carefully.

INDEX

S.NO.	Name of the experiment	Page No.	Performed Date	Date of submission	Marks	Faculty Signature
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

INDEX

Mr./Ms. _____

Roll Number: _____

S. No	Date	Title of the Experiment	Page No.	Marks	Signature of the Staff
PART-A					
1		Introduction To MASM8086	1		
2		Introduction ToMSP430	9		
PART - B: LIST OF EXPERIMENTS					
1		Programs using Arithmetic and logicaloperations: Addition, Subtraction, Logical	19		
2		Programs using string operations and instruction prefix: Move Block, Reverse string,String Comparison	26		
3		Programs for code conversion: BCD to Seven Segment Code Conversion, Binary to Gray Code conversion,BCD to ASCII	27		
4		Multiplication and division programs: Multiplication, Division	34		
5		Sorting and MultiByte arithmetic: Ascendingorder / Descending order, MultiByte addition / subtraction	40		
6		Programs using CALL and RET instructions: Factorial of a number using procedure, Finding prime number or not, Fibonacci series generation.	46		

7		Interfacing and Programming GPIO Ports	55		
8		Usage of Low Power Modes	61		
9		Interrupt programming through GPIOs	67		
10		PWM generation using Timer on MSP430 GPIO	71		
11		Interfacing Potentiometer with MSP430	75		
12		PWM based Speed Control of Motor by Potentiometer	79		
13		Using ULP Advisor in CCS on MSP430	84		
14		Low Power modes and Energy Trace++	88		
Additional Experiment Beyond the Curriculum					
15		Serial Communication using MSP430	93		
PART C					
16		Lab Based Mini Project	99		

PART-A
Introduction to MASM8086



Department of Electronics and Communication Engineering SVR
Engineering College: Nandyal

**Approved by AICTE, New Delhi & Affiliated to JNTUA,
Anantapuramu (An ISO 9001:2007 Certified Institution)**

Near Ayyalur (V), Allagadda Rd, Nandyal, Kurnool (Dt), Andhra Pradesh 518503

Part – A**Module-1****Introduction to MASM 8086 Software**

There are two ways to execute 8086 programs using a computer. One is using the DOS tool called DEBUG and the second way is using the Assembler MASM. Hence both the ways are to be learned.

DEBUG

The DOS utility called DEBUG allows entering assembly language programs, to execute these programs, to view memory locations and also to trace the program execution.

Starting and Quitting DEBUG

The simple way of starting the DEBUG is to type the command DEBUG at the DOS prompt. After the DEBUG is loaded the DEBUG prompt '-' is displayed. Then the DEBUG is ready to accept any DEBUG commands.

To quit DEBUG the command is Q. This command takes control back to DOS. DEBUG can be started along with a file also. To load DEBUG with a program say test1.com at the DOS prompt type DEBUG test1.com.

DEBUG Commands

The commands which are required to run / enter the programs are the following.

A	Assemble
D	Display / Dump
E	Enter data
F	Fill memory
G	Run the program
L	Load
N	Name a program
P	Proceed
Q	Quit
R	Register
T	Trace
U	Un Assemble
W	Write

The DEBUG does not detect errors until Enter key is pressed. If there is any syntax mistake then the command line is redisplayed with the word error added at the point at which the error was detected. If a command line contains more than one error, only the first error will be detected and indicated and execution will stop at that point.

Part – A**Module-1****Introduction to Assembly Language Programming****LEVELS OF PROGRAMMING:**

There are three levels of programming

1. Machine language
2. Assembler language
3. High level language

Machine language programs are programs that the computer can understand and execute directly. Assembly language instructions match machine language instructions, but are written using character strings so that they are more easily understood and High-level language instructions are much closer to the English language and are structured.

Ultimately, an assembly language or high level language program must be converted into machine language by programs called translators. If the program being translated is in assembly language, the translator is referred to as an assembler, and if it is in a high level language the translator is referred to as a compiler or interpreter.

ASSEMBLY LANGUAGE PROGRAM DEVELOPMENT TOOLS:

EDITOR: An editor is a program, which allows you to create a file containing the assembly language statements for your program.

ASSEMBLER: An assembler program is used to translate the assembly language Mnemonic instructions to the corresponding binary codes. The second file generated by assembler is called the assembler List file.

LINKER: A Linker is a program used to join several object files in to one large object file. The linkers produce link files with the .EXE extension.

DEBUGGER: If your program requires no external hardware, then you can use a debugger to run and debug your program. A debugger is a program, which allows you to load your object code program into system memory, execute the program, and troubleshoot or “debug” it.

Part – A**Module-1****Introduction to 8086 - Assembler Directives**

The logical errors or other programming errors are not found by the assembler. For completing all these tasks, an assembler needs some hints from the programmer. These types of hints are given to the assembler using some predefined alphabetical strings called assembler directives, which helps the assembler to correctly understand the assembly language program to prepare the codes.

Another type of hint which helps the assembler to assign a particular constant with a label or initialize particular memory locations or labels with constants is an operator.

- **DB: Define Byte:** The DB directive is used to reserve byte or bytes of memory locations in the available memory.
- **DW: Define Word:** The DW directive serves the same purposes as the DB directive, but it makes the assembler reserves the number of memory words (16bit) instead of bytes.
- **DQ: Define Quad word:** This directive is used to direct the assembler to reserve 4 words (8bytes) of memory for the specified variable and may initialize it with the specified values.
- **DT: Define Ten Bytes:** The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialize the 10-bytes with the specified values.
- **ASSUME: Assume Logical Segment Name:** The ASSUME directive is used to inform the assembler, the names of the logical segments to be assumed for different segments used in the program.
- **END: END Of Program:** The END directive marks the end of an assembly language program.
- **ENDP: END Of Procedure:** The ENDP directive is used to indicate the end of a procedure.
- **ENDS: END Of Segment:** This directive marks the end of a logical segment.

DATA SEGMENT

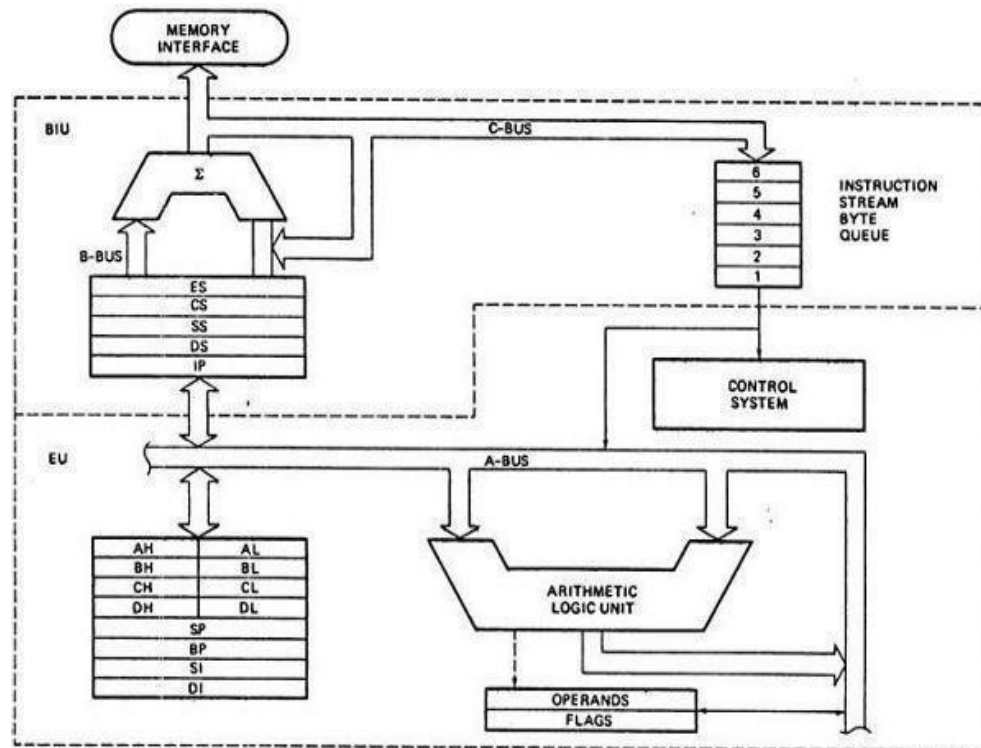
:

- **EVEN: Align On Even Memory Address:** The EVEN directive updates the location counter to the next even address, if the current location counter contents are not even, and assigns the following routine or variable or constant to that address. If the content of the location counter is already even, then the procedure will be assigned with the same address.
- **EQU: Equate:** The directive EQU is used to assign a label with a value or symbol. The use of this directive is just to reduce the recurrence of the numerical values or constants in the program code.

LABEL EQU 0500H

- **EXTERN:** External and **PUBLIC:** Public: The directive **EXTERN** informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language module.
- **GROUP:** Group the Related Segments: This directive is used to form logical groups of segments with similar purpose or type.
- **PROGRAM GROUP CODE, DATA, STACK** //this statement directs the loader/linker to prepare an EXE file such that CODE, DATA, STACK segment must lie within a 64byte memory segment that is named as PROGRAM
- **LABEL:** The LABEL directive is used to assign a name to the current content of the location counter. A LABEL directive can be used to make a FAR jump. The label directive can be used to refer to the data segment along with the data type, byte or word.
- **DATA SEGMENT**
- **DATAS DB 50H DUP (?)**
- **DATA-LAST LABEL BYTE FAR**
- **DATA ENDS**
- After reserving 50H locations for DATAS, the next location will be assigned a label DATA-LAST and its type will be byte and far.
- **LENGTH:** Byte Length of a Label: This directive is used to refer to the length of a data array or a string. Not available in MASM.
- **LOCAL:** The label, variables, constants or procedures declared LOCAL in a module are to be used only by that particular module.
- **NAME:** Logical Name of a Module: The NAME directive is used to assign a name to an assembly language program module.
- **OFFSET:** Offset of a Label: When the assembler comes across the OFFSET operator along with a label, it first computes the 16-bit displacement of the particular label, and replaces the string 'OFFSET LABEL' by the computed displacement.
- **ORG:** Origin: The ORG directive directs the assembler to start the memory allotment for the particular segment, block or code from the declared address in the ORG statement.
- **PROC:** Procedure: The PROC directive marks the start of a named procedure in the statement. Also the types FAR and NEAR specifies the type of the procedure.
- **PTR:** Pointer: The POINTER operator is used to declare the type of a label, variable or memory operand. The operator PTR is prefixed by either BYTE (8-bit quantity) or WORD (16-bit quantity).
- **SEGMENT:** Logical Segment: The SEGMENT directive marks the starting of a logical segment. The started segment is also assigned a name, i.e. label, by this statement.
- **SHORT:** The SHORT operator indicates the assembler that only one byte is required to code the displacement for a jump. This method of specifying jump address saves memory.

- **TYPE:** The TYPE operator directs the assembler to decide the data type of the specified label and replaces the TYPE label by the decided data type.
- **GLOBAL:** The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the program.
- **'+' & '-' Operators:** These operators represent arithmetic addition and subtraction respectively. And are typically used to add or subtract displacements (8 or 16 bit) to base or index registers or stack or base pointers.
- **FAR PTR:** This directive indicates the assembler that the label following FARPTR is not available within the same segment and the address of the bit is of 32 bits i.e. 2 bytes offset followed by 2 bytes.
- **NEAR PTR:** This directive indicates that the label following NEAR PTR is in the same segment and need only 16 bit i.e. 2 byte offset to address it. A NEAR PTR label is considered as default if a label is not preceded by NEAR PTR or FAR PTR.

Part – A**Module-1****8086 Architecture****ARCHITECTURE OF 8086**

The 8086 CPU is organized as two separate processors, called the Bus Interface Unit (BIU) and the Execution Unit (EU). The BIU handles all transactions of data and addresses on the buses for EU. The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue. EU executes instructions from the instruction system byte queue. Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance. BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder. EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register. Features of 8086 Microprocessor:

- Intel 8086 was launched in 1978.
- It was the first 16-bit microprocessor.
- This microprocessor had major improvement over the execution speed of 8085.
- It is available as 40-pin Dual-Inline-Package (DIP).

DOS / BIOS INTERRUPTS**1. DOS Interrupt 21 – Service 01**(Read Character From Standard Input - Keyboard, With Echo)**Registers used:**

AH = 1

AL = character read from keyboard

Ex:

MOV AH, 1

INT 21H

2. DOS Interrupt 21 - Service 02 (Write Character To Standard Output - Monitor)**Registers used:**

AH = 2

DL = the character to be displayed.

Ex:

MOV AH, 2

MOV DL, 'A'

INT 21H

3. DOS Interrupt 21 – Service 09 (Write String To Standard Output – Monitor & terminated by a \$ to the monitor)**Registers used:**

AH = 09

DX = the offset address of the data to be displayed.

Ex:

MOV AH, 09

MOV DX, OFFSET MESS1

INT 21H

4. DOS Interrupt 21 - Service 0Ah (Buffered Input.)**Registers used:**

AH = 0AH

DX = the offset address of the input string (Entered by Keyboard).

Ex:

MOV AH, 0AH

LEA DX, BUFF

INT 21H

5. DOS Interrupt 21 - Service 4Ch (Terminates a process, by returning control to a parent process or to DOS.)

Registers used:

AH = 4CH

Ex:

MOV AH, 4CH

INT 21H

Part – B**Module-2****Introduction to MSP430****MSP430:**

The **MSP430** is a mixed-signal microcontroller family from Texas Instruments. Built around a 16-bit CPU, the MSP430 is designed for low cost and, specifically, low power consumption embedded applications.

Applications

The MSP430 can be used for low powered embedded devices. The current drawn in idle mode can be less than 1 μ A. The top CPU speed is 25 MHz. It can be throttled back for lower power consumption. The MSP430 also uses six different low-power modes, which can disable unneeded clocks and CPU. Additionally, the MSP430 is capable of wake-up times below 1 microsecond, allowing the microcontroller to stay in sleep mode longer, minimizing its average current consumption. The device comes in a variety of configurations featuring the usual peripherals: internal oscillator, timer including PWM, watchdog, USART, SPI, I²C, 10/12/14/16/24-bit ADCs, and brownout reset circuitry. Some less usual peripheral options include comparators (that can be used with the timers to do simple ADC), on-chip op-amps for signal conditioning, 12-bit DAC, LCD driver, hardware multiplier, USB, and DMA for ADC results. Apart from some older EPROM (MSP430E3xx) and high volume mask ROM (MSP430Cxxx) versions, all of the devices are in-system programmable via JTAG (full four-wire or Spy-Bi-Wire) or a built in bootstrap loader (BSL) using UART such as RS232, or USB on devices with USB support.

There are, however, limitations that preclude its use in more complex embedded systems. The MSP430 does not have an external memory bus, so it is limited to on-chip memory (up to 512 KB flash memory and 66 KB RAM) which may be too small for applications that require large buffers or data tables. Also, although it has a DMA controller, it is very difficult to use it to move data off the chip due to a lack of a DMA output strobe.

MSP430 Nomenclature

An **MSP430** part number such as "**MSP430F2618ATZQWT-EP**" consists of the following pieces:

- **MSP430**: Standard prefix.
- **F**: Indicates a memory type or specialized application. "**F**" indicating flash memory is by far the most popular. Other options for memory type include "**C**" for masked ROM, "**FR**" for FRAM, "**G**" for Flash Value Line, and "**L**" as in the MSP430L09x series, which indicates a RAM-only part; it must remain continuously powered to retain its programming. A second letter (except for "FR") indicates a specialized application for the part. For example, "**G**" is an optional specialization letter indicating hardware support for a specialized use. "**E**" indicates special electricity meter functions, "**G**" devices are designed for medical instrumentation, and "**W**" devices include a special "scan interface" designed for flow meters. An exception is the MSP430FG2xx devices, which are considered a separate generation.

- **2:** The generation of device. There can be significant changes to core peripherals (clock generators, UARTs, etc.) in different generations. These are not in chronological order, but rather higher values roughly indicate greater size, complexity and cost. For example, generations **3** and **4** include LCD controllers which the others do not.
- **6:** The model within the generation. This indicates the mixture of on-board peripheral devices and number of pins.
- **18:** One or two digits indicating the amount of memory on the device. The numbering is (mostly) consistent throughout the MSP430 series. Not all suffixes are valid with all models; most models are available in 3–6 memory sizes, chosen to match the other capabilities of the device. Larger numbers indicate increasing amounts of memory, but sometimes one type of memory (RAM or ROM) is sacrificed to fit more of the other.

MSP430 Memory configurations

Suffix	RAM	ROM	Suffix	RAM	ROM
0	128	1 K	10	5 K	32 K
1	128	2 K	11	10 K	48 K
2	256	4 K	12	5 K	55/56 K
3	256	8 K	13		
4	512	12 K	14		
5	512	16 K	15		
6	1 K	24 K	16	4 K	92 K
7	1 K	32 K	17	8 K	92 K
8	2 K	48K	18	8 K	116 K
9	2 K	60K	19	4 K	120 K

- An optional suffix digit indicating a variant device, adding or deleting some analog peripherals. For example, a "1" suffix may indicate the addition of a comparator or deletion of an ADC. If the memory size is "1", this suffix can be confused with part of the memory size, but no single model is available in both "1" and "10" (or greater) memory sizes.
- An optional "A" suffix indicating an upward-compatible revised version. The MSP430F11x1A has an additional 256 bytes of data flash not present in the plain 'F11x1.

Trailing suffix letters indicate options not visible to software:

- **T:** Indicates a temperature range of –40 °C to +105 °C.
- **ZQW:** Indicates the package the part is kept in. "ZQW" is a TI-specific name for a ball grid array.
- **T:** Indicates that the parts are shipped in small reel (7-inch) packaging.
- **-EP:** Indicates an additional feature. "**-Q1**" specifies that the part is automotive qualified. "**-EP**" and "**-HT**" indicate extended temperature products. Enhanced products, "**-EP**", have a temperature range from –40 °C to 125 °C, and extreme temperature parts, "**-HT**", have a temperature range from –56 °C to 150 °C.

MSP430 generations

There are six general generations of MSP430 processors. In order of development, they were the '3xx generation, the '1xx generation, the '4xx generation, the '2xx generation, the '5xx generation, and the '6xx generation. The digit after the generation identifies the model (generally higher model numbers are larger and more capable), the third digit identifies the amount of memory on board, and the fourth, if present, identifies a minor model variant. The most common variation is a different on-chip analog-to-digital converter.

The 3xx and 1xx generations were limited to a 16-bit address space. In the later generations this was expanded to include '430X' instructions that allow a 20-bit address space. As happened with other processor architectures (e.g. the processor of the PDP-11), extending the addressing range beyond the 16-bit word size introduced some peculiarities and inefficiencies for programs larger than 64 Kbytes.

In the following list, it helps to think of the typical 200 mA Hr capacity of a CR2032 lithium coin cell as 200,000 μA Hr, or 22.8 μA year. Thus, considering only the CPU draw, such a battery could supply a 0.7 μA current draw for 32 years. (In reality, battery self-discharge would reduce this number.)

The significance of the 'RAM retention' vs the 'real-time clock mode' is that in real time clock mode the CPU can go to sleep with a clock running which will wake it up at a specific future time. In RAM retention mode, some external signal is required to wake it, e.g. I/O pin signal or SPI slave receive interrupt.

MSP430G2xx series

The MSP430G2xx Value Series features flash-based Ultra-Low Power MCUs up to 16 MIPS with 1.8–3.6 V operation. Includes the Very-Low power Oscillator (VLO), internal pull-up/pull-down resistors, and low-pin count options, at lower prices than the MSP430F2xx series.

- Ultra-Low Power, as low as (@2.2 V):
 - 0.1 μA RAM retention
 - 0.4 μA Standby mode (VLO)
 - 0.7 μA real-time clock mode
 - 220 μA / MIPS active
 - Ultra-Fast Wake-Up From Standby Mode in $<1 \mu\text{s}$
- Device parameters
 - Flash options: 0.5–56 KB
 - RAM options: 128 B–4 KB
 - GPIO options: 10, 16, 24, 32 pins
 - ADC options: Slope, 10-bit SAR
 - Other integrated peripherals: Capacitive Touch I/O, up to 3 16-bit timers, watchdog timer, brown-out reset, USI module (I²C, SPI), USCI module, Comparator_A+, Temp sensor



PART-A

8086 MICROPROCESSOR PROGRAMS

Department of Electronics and Communication Engineering SVR Engineering College, Nandyal

**Approved by AICTE, New Delhi & Affiliated to JNTUA,
Anantapuramu (An ISO 9001:2007 Certified Institution)**

Near Ayyalur (V), Allagadda Rd, Nandyal, Kurnool (Dt), Andhra Pradesh 518503

Experiment No _____

2. Programs using Arithmetic and logical operation

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To Perform addition of n-numbers using ALP
- 2) To evaluate an expression $W=X+Y-Z$
- 3) To perform conditional addition using logical operations

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) Differentiate ADD and ADC instructions?

- 2) List different arithmetic instructions in 8086?

- 3) List various logical instructions in 8086?

4) What is an assembler directive and mention some assembler directives in 8086?

5) Explain about CMP instruction?

PROGRAM

A) 16 BIT ADDITION

ASSUME CS: CODE, DS: DATA

DATA SEGMENT

OPERN1 DW 1234H

OPERN2 DW 5678H

RESLT DW 01H DUP(?)

DATA ENDS

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV AX,OPERN1

MOV BX,OPERN2

ADD AX,BX

MOV RESLT,AX

INT 03H

CODE ENDS

END START

INPUT:**OUTPUT:****THEORITICAL CALCULATIONS****B) 16 BIT SUBTRACTION**

ASSUME CS: CODE, DS: DATA

DATA SEGMENT

OPERN1 DW 5678H

OPERN2 DW 1234H

RESLT DW 01H DUP(?)

DATA ENDS

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV AX,OPERN1

MOV BX,OPERN2

SUB AX,BX

MOV RESLT,AX

INT 03H

CODE ENDS

END START

INPUT:

OUTPUT:

THEORITICAL CALCULATIONS

C) 16 BIT LOGICAL OR OPERATION

ASSUME DS:DATA, CS:CODE

OPD1 DW 1234H

OPD2 DW 5678H

RES DW 01H DUP(?)

DATA ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AX,OPD1

MOV BX,OPD2

OR AX,BX

MOV RES,AX

INT 03H

CODE ENDS

END START

```
ASSUME DS:DATA, CS:CODE
OPD1 DW 1234H
OPD2 DW 5678H
RES  DW 01H DUP(?)
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPD1
MOV BX,OPD2
AND AX,BX
MOV RES,AX
INT 03H
CODE ENDS
END START
```

```
ASSUME DS:DATA, CS:CODE
OPD1 DW 1234H
OPD2 DW 5678H
RES  DW 01H DUP(?)
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPD1
MOV BX,OPD2
XOR AX,BX
```

```
MOV RES,AX  
INT 03H  
CODE ENDS  
END START
```

INPUT:

OUTPUT:

THEORITICAL CALCULATIONS

POST LAB QUESTIONS

- 1) What do you understand the importance of OFFSER assembler directive in above program?

- 2) What happens if LOOP instruction is executed in above program?

- 3) Which addressing mode is used for addition and subtraction in above program?

4) Identify the branch instructions in above programs?

5) What are the major two flags that effects arithmetic and logical operations?

OUT COMES

Upon completion of the experiment, the student will be able to

1. Analyze & understand the different Arithmetic and logical Operations.
2. Verified theoretical values with practical values

GRADING

DATE OF SUBMISSION						
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (2M)	Calculations (2M)	Post-Lab Questions (2M)	Viva (2M)	Total (10M)
Remarks						
Signature of the Evaluator with Date						

NOTES

Experiment No _____

3.Move Block, Reverse string, String Comparison

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To move a block of data from source to destination using ALP
- 2) To reverse a given string using ALP
- 3) To compare two strings using ALP

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) What is the importance of SI and DI registers in strings?

- 2) List different string instructions in 8086?

- 3) What is the importance of REP in string instructions?

4) Explain about STD and CLD?

5) List various DOS interrupts?

PROGRAM

A) MOVE BLOCK

ASSUME DS:DATA,ES:EXTRA,CS:CODE

DATA SEGMENT

STR1 DB 'SVREC'

DATA ENDS

EXTRA SEGMENT

STR2 DB 08H DUP(0)

EXTRA ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AX,EXTRA

MOV ES,AX

MOV SI,OFFSET STR1

MOV DI,OFFSET STR2

MOV CL,08H

CLD

REP: MOVSB

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT: SOURCE= "SVREC"

OUTPUT: DEST (ADDRESS) =

B) COMPARISION STRING

ASSUME DS:DATA,ES:EXTRA,CS:CODE

DATA SEGMENT

STR1 DB 'SVREC'

DATA ENDS

EXTRA SEGMENT

STR2 DB 'SVREC'

EXTRA ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AX,EXTRA

MOV ES,AX

MOV SI,OFFSET STR1

MOV DI,OFFSET STR2

MOV CL,08H

CLD

L2: CMPSB

JNE L1

LOOP L2

MOV BX,1111H

JMP L3

L1: MOV BX,0000H

L3:INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT: S1= "THIS IS FIRST STRING"
 S2="THIS IS SECOND STRING"

OUTPUT: DEST (ADDRESS) =

POST LAB QUESTIONS

1) What happens if we execute STD before MOVSB instruction?

2) What is the need of ADD SI, 02 IN reverse string program?

3) Which addressing mode is used for comparison of a string in above program?

4) What is the importance of EQU assembler directive in above program?

5) What are the two flags that effects string operations?

OUT COMES

Upon completion of the experiment, the student will be able to

1. Analyze the string instructions
2. Analyze the DOS interrupts

GRADING

DATE OF SUBMISSION					
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No

3

Programs for code conversion

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To convert BCD to seven segment code conversion using ALP
- 2) To convert Binary to gray code conversion using ALP
- 3) To convert BCD to ASCII code conversion using ALP

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) Give an example for BCD to seven segment code conversion?

- 2) Give an example for Binary to gray code conversion?

- 3) Give an example for BCD to ASCII code conversion?

4) Explain about ROR and XLAT instruction in 8086?

5) Differentiate SAL and SHL?

PROGRAM

A) ASCII TO BCD CONVERSION

ASSUME CS:CODE

CODE SEGMENT

MOV AH,35H

MOV AL,34H

MOV CL,04H

AND AL,0FH

SHL AH,CL

OR AH,AL

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT:

OUTPUT: ANS (ADDRESS) =

THEORITICAL CALCULATIONS

B) BCD TO ASCII CODE CONVERSION

```
ASSUME CS: CODE
CODE SEGMENT
START: MOV AL,56H
MOV AH,AL
AND AL,0FH
MOV CL,04H
SHR AH,CL
OR AX,3030H
INT 03H
CODE ENDS
END START
```

OBSERVATIONS

INPUT: = 56H

OUTPUT: ANS (ADDRESS) =

THEORITICAL CALCULATIONS

POST LAB QUESTIONS

- 1) What happens if XLAT instruction is executed in BCD to seven segment code conversion program?

- 2) What happens if this instruction AND AL,80 is executed in above binary to gray code conversion program?

- 3) What happens if this instruction ROR AH,04 is executed in above BCD to ASCII code conversion program?

- 4) What is the value present in AH after executing the SAL AH,1 and AH=05h?

- 5) What is the addressing mode of OR AL, AH instruction?

OUT COMES

Upon completion of the experiment, the student will be able to

1. Analyze & understand the different code conversions.

2. Verified theoretical values with practical values

GRADING

DATE OF SUBMISSION						
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (2M)	Calculations (2M)	Post-Lab Questions (2M)	Viva (2M)	Total (10M)
Remarks						
Signature of the Evaluator with Date						

NOTES

Experiment No

4

Programs for Multiplication and Division

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To perform multiplication operation using ALP
- 2) To perform division operation using ALP

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) In DIV instruction the numerator and denominator values are stored in which register?

- 2) Explain about BYTE PTR in 8086?

- 3) Give an example for 8-bit multiplication operation?

- 4) Which flag is affected when we execute IMUL instruction?

5) Differentiate MUL and IMUL?

PROGRAM

A) 16 MULTIPLICATION

ASSUME CS: CODE, DS: DATA

DATA SEGMENT

OPERN1 DW 5678H

OPERN2 DW 1234H

RESULTL DW 01H DUP(?)

RESLTH DW 01H DUP(?)

DATA ENDS

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV AX,OPERN1

MOV BX,OPERN2

MUL BX

MOV RESULTL,AX

MOV RESLTH,DX

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT:

OUTPUT: ANS (ADDRESS) =

THEORITICAL CALCULATIONS

B) 16 BIT DIVISION

ASSUME CS: CODE, DS: DATA

DATA SEGMENT

OPERN1 DW 000AH

OPERN2 DB 03H

RESLT DW 01H DUP(?)

DATA ENDS

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV AX, OPERN1

MOV BL,OPERN2

DIV BL

MOV RESLT,AX

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT:

OUTPUT: ANS (ADDRESS) =

THEORITICAL CALCULATIONS

POST LAB QUESTIONS

1) What happens if we execute LOOP L1 instruction in above program?

2) What is the status of the flag after unsigned addition?

3) Which addressing mode is used for DIV instruction in above program?

4) Is immediate addressing mode is allowed or not in MUL instruction in above program?

5) What is the status of flags after execution of DIV BL instruction in above program?

OUT COMES

Upon completion of the experiment, the student will be able to

1. Understand and analyze the multiplication and division operations
2. Verify the theoretical and practical values

GRADING

DATE OF SUBMISSION						
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (2M)	Calculations (2M)	Post-Lab Questions (2M)	Viva (2M)	Total (10M)
Remarks						
Signature of the Evaluator with Date						

NOTES

Experiment No

5

Programs for Sorting and MultiByte Arithmetic

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To perform ascending order/ descending order operations using ALP
- 2) To perform MultiByte addition/subtraction operation using ALP

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) Explain about XCHG instruction in 8086?

- 2) Give an example for ADC instruction in 8086?

- 3) Give an example for INC and DEC instructions in 8086?

- 4) What are the address storage registers in 8086?

5) Differentiate DB and DW?

PROGRAM

A) ASCENDING ORDER

ASSUME CS : CODE, DS : DATA

DATA SEGMENT

LIST DB 01h,05h,09h,02h

COUNT EQU 04H

DATA ENDS

CODE SEGMENT

START:

MOV AX, DATA

MOV DS, AX

MOV DL, COUNT - 1

L3 : MOV CL, DL

MOV SI,OFFSET LIST

L2 : MOV AL, [SI]

CMP AL, [SI + 1]

JC L1

XCHG AL, [SI + 1]

XCHG AL, [SI]

L1:INC SI

DEC CL

JNZ L2

DEC DL

JNZ L3

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT:

OUTPUT: BX (ADDRESS) =

THEORITICAL CALCULATIONS

B) MULTIBYTE ADDITION

ASSUME CS:CODE

CODE SEGMENT

START:

XOR AL,AL

MOV SI, 2000H

MOV BX,3000H

MOV DI,4000H

MOV CL,04H

UP:MOV AL,[SI]

ADD AL,[BX]

MOV [DI],AL

INC SI

INC BX

INC DI

DEC CL

JNZ UP

INT 03H

CODE ENDS

END START

OBSERVATIONS

INPUT: X = 35H,67H,45H,23H,11H
 Y = 11H,22H,65H,44H,37H

OUTPUT: ANS (ADDRESS) =

C) ODD AND EVEN

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
LIST DB 01H,03H,04H,05H,06H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV SI,OFFSET LIST
MOV CL,COUNT
L3:MOV AL,[SI]
CLC
RCR AL,01H
JC L1
INC BL
JMP L2
L1:INC DL
L2:INC SI
DEC CL
JNZ L3
INT 03H
CODE ENDS
END START

```

OBSERVATIONS

INPUT:

OUTPUT:

THEORITICAL CALCULATIONS

POST LAB QUESTIONS

- 1) What is the status of the flag after execution of ADC AL, [SI] instruction in above multi byte addition program?

- 2) What is the result stored in BX if MOV BX, OFFSET X instruction is executed?

- 3) Explain about SI and DI instruction in above programs?

- 4) Which register is important for executing LOOP instruction?

OUT COMES

Upon completion of the experiment, the student will be able to

1. Understand and analyze the sorting and Multibyte addition operations using ALP
2. Verify the theoretical and practical values

GRADING

DATE OF SUBMISSION						
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (2M)	Calculations (2M)	Post-Lab Questions (2M)	Viva (2M)	Total (10M)
Remarks						
Signature of the Evaluator with Date						

NOTES

Experiment No _____

6

Programs using CALL and RET instructions

Regd. No: _____

Date: _____

OBJECTIVES

- 1) To find factorial of a number using ALP
- 2) To find whether the given number is prime or not using ALP
- 3) To perform Fibonacci series Using ALP

APPARATUS

Computer System Installed with EMULATOR 8086 Software.

PRE LAB QUESTIONS

- 1) Differentiate procedure and macro arithmetic instructions in 8086?

- 2) Explain about RET instruction in 8086?

- 3) What is addressing mode used for MUL instruction ?

4) Explain about JE and JNE instructions in 8086?

5) What is the difference between SUB and CMP instructions in 8086?

PROGRAM

A) PERFORM ALL LOGICAL OPERATION BY USING CALL AND RET

ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE

CODE SEGMENT

ORG 1000H

START: MOV DI, 2000H

CALL INPUTS

AND AX, BX

CALL OUTPUTS

CALL INPUTS

OR AX, BX

CALL OUTPUTS

CALL INPUTS

XOR AX, BX

CALL OUTPUTS

INPUTS: MOV AX, 1234H

MOV BX, 5678H

RET

OUTPUTS: MOV [DI], AX

INC DI

INC DI

NOT AX

MOV [DI], AX

INC DI

INC DI

RET

INT 3

CODE ENDS

END

OBSERVATIONS

INPUT:

OUTPUT: ANS (ADDRESS) =

THEORITICAL CALCULATIONS

POST LAB QUESTIONS

1) How many times the loop executes if CX=08h instruction is executed in above program?

2) What is the importance of CALL instruction in above program?

3) Explain about FIBONACCI PROC NEAR instructions in above program?

4) Explain about LEA instruction in above program?

5) Differentiate END and ENDP in above program?

OUTCOMES

Upon completion of the experiment, the student will be able to

1. Understand and analyze the CALL and RET instructions
2. Verify the theoretical and practical values

GRADING

DATE OF SUBMISSION						
MARKS AWARDED (Max. Marks - 10M)	Pre-lab Questions (2M)	Observations (2M)	Calculations (2M)	Post-Lab Questions (2M)	Viva (2M)	Total (10M)
Remarks						
Signature of the Evaluator with Date						

NOTES

PART-B
EMBEDDED C EXPERIMENTS USING
MSP430 MICROCONTROLLER



Experiment No. _____

7. Interfacing and Programming GPIO Ports

Regd. No: _____

Date: _____

Objectives:

- (A) To blink the RED LED using C language.
- (B) To blink the GREEN LED using C language.
- (C) To blink the RED & GREEN LED using C language.
- (D) To control the on-board GREEN LED by taking the input from switch

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launch Pad plugged into your PC via USB

Pre Lab Questions:

1. Define Embedded System.

2. What are the features of MSP430?

3. Define Watchdog Timer.

Procedure:

1. Open code composer studio
2. Select new CCS project
3. Write embedded C program
4. Build the program
5. Load and run the code
6. Observe the outputs on MSP430 launch pad

(A) Program:

```

#include<msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    P1DIR |= 0x01; // Set P1.0 to output direction
    while(1) {
        volatile unsigned long i; // Volatile to prevent
        //optimization
        P1OUT ^= 0x01; // Toggle P1.0 using XOR
        i = 50000; // SW Delay
        doi--;
        while(i != 0);
    }
}

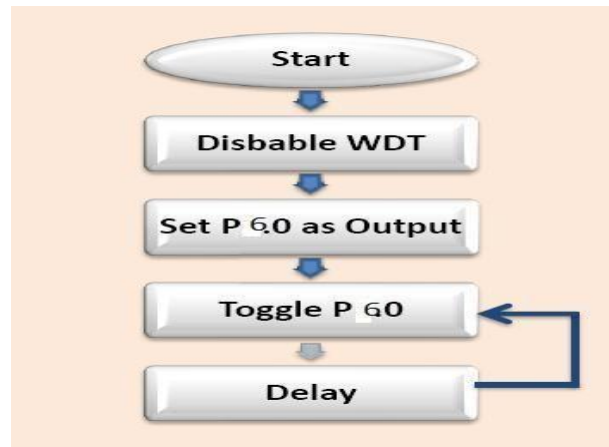
```

Flow Chart:**(B) Program:**

```

#include<msp430.h>
int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    P1DIR |= 0x40; // Set P6.0 to output direction
    while(1) {
        volatile unsigned long i; // Volatile to prevent
        //optimization
        P1OUT ^= 0x40; // Toggle P6.0 using XOR
        i = 50000; // SW Delay
        do i--;
        while(i != 0);
    }
}

```

Flow Chart:**(C) Program:**

```

#include<msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    P1DIR |= 0x41; // Set P6.0 to output direction
    while(1)
    {

        volatile unsigned long i; // Volatile to prevent
        //optimization
        P1OUT ^= 0x41; // Toggle P6.0 using XOR
        i = 50000; // SW Delay
        do i--;
        while(i != 0);
    }
}
  
```

(D) Program:

```

#include<msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
  
```

```

P1DIR |= 0x40; // Set P1.6 to output direction
P1REN |= 0x08;
P1OUT |= 0x08;
while(1)
{
if((P1IN & BIT3)) { // If button is open(P1.3 HIGH)
P1OUT = P1OUT & ~BIT6; // ... else turn it off.
} // or P1OUT |= BIT0;
Else
{
P1OUT = P1OUT | BIT6; // ... turn on LED
// or P1OUT &= ~BIT0
}
}
}

```

Post lab Questions:

1. What is the need of Pull up and Pull down Resistors for GPIO lines?

2. What are I/O operations in MSP430?

3. List the registers available in MSP430?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

8. Usage of Low Power Modes

Regd. No: _____

Date: _____

Objectives:

To measure active mode and standby mode current using MSPEXP430FR5969 as hardware platform.

Apparatus:

1. Code Composer Studio
2. MSPEXP430FR5969 plugged into your PC via USB
3. Digital Multimeter.

Pre Lab Questions:

1. What is Active Mode?

2. List the Low Power Modes?

3. What is LPM2 & LPM3?

Procedure:

Measurement of Active Current:

1. Copy the code **main_active.cin** the CCS new project.
2. Build, load, and run the code. The green LED will blink once every three or four seconds.
3. When done, halt the code and click the Terminate button to return to the "CCS Edit".
4. Remove all five jumpers on header J3.
5. The red lead of the multimeter should connect to the top (emulation side) Vcc pin on header J3 and the black lead of the multimeter should connect to the bottom (target side) Vcc pin on header J3.
6. Press the Reset button on the Launchpad board.
7. Measure the current drawn by the MSP430.

Measurement of Standby Current:

1. Copy the code **main_standby.cin** the CCS new project.
2. Build, load, and run the code. The green LED will blink once every three or four seconds.
3. When done, halt the code and click the Terminate button to return to the "CCS Edit".
4. Remove all five jumpers on header J3.
5. The red lead of the multimeter should connect to the top (emulation side) Vcc pin on header J3 and the black lead of the multimeter should connect to the bottom (target side) Vcc pin on header J3.
6. Press the Reset button on the Launchpad board.
7. Measure the current drawn by the MSP430.

Program:

Main_active.c

```
#include<msp430.h>
int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    P1DIR |= 0x01; // Set P1.0 to output direction
    while(1) {
        volatile unsigned long i; // Volatile to prevent
        //optimization
        P1OUT ^= 0x01; // Toggle P1.0 using XOR
        i = 50000; // SW Delay
        do i--;
        while(i != 0);
    }
}
```

Main_standby.c:

```

#include <mcp430g2553.h>
#ifndef TIMER0_A1_VECTOR
#define TIMER0_A1_VECTOR TIMERA1_VECTOR
#define TIMER0_A0_VECTOR TIMERA0_VECTOR
#endif
volatile long tempRaw;
//volatile unsigned int i;
void FaultRoutine(void);
void ConfigWDT(void);
void ConfigClocks(void);
void ConfigPins(void);
void ConfigADC10(void);
void ConfigTimerA2(void);
void main(void)
{
    ConfigWDT();
    ConfigClocks(); ConfigPins();
    ConfigADC10();
    ConfigTimerA2();
    // _BIS_SR(GIE);
    while(1)
    {
        _bis_SR_register(LPM3_bits + GIE); // Enter LPM3 with interrupts
    }
} void ConfigWDT(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
}
void ConfigClocks(void)
{
    if(CALBC1_1MHZ == 0xFF || CALDCO_1MHZ == 0xFF)
        FaultRoutine(); // If calibration data is erased
    // run FaultRoutine()
    BCSCTL1 = CALBC1_1MHZ; // Set range
    DCOCTL = CALDCO_1MHZ; // Set DCO step + modulation
    BCSCTL3 |= LFXT1S_2; // LFXT1 = VLO
    IFG1 &= ~OFIFG; // Clear OSCFault flag
    BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3; // MCLK = DCO/8, SMCLK = DCO/8
}

```

```

} void FaultRoutine(void)
{
P1OUT = BIT0; // P1.0 on (red LED)
while(1); // TRAP
} void ConfigPins(void)
{
P1DIR = ~BIT3; // P1.6 and P1.0 outputs
P1OUT = 0;
P2SEL = ~(BIT6 + BIT7);
P2DIR |= BIT6 + BIT7;
P2OUT = 0; // LEDs off
} void ConfigADC10(void)
{
ADC10CTL1 = INCH_10 + ADC10DIV_0; // Temp Sensor ADC10CLK
} void ConfigTimerA2(void)
{
CCTL0 = CCIE;
CCR0 = 36000;
TACTL = TASSEL_1 + MC_2;
}
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;
_delay_cycles(4); // Wait for ADC Ref to settle
ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
P1OUT |= BIT6; // P1.6 on (green LED)
_delay_cycles(100);
ADC10CTL0 &= ~ENC; // Disable ADC conversion
ADC10CTL0 &= ~(REFON + ADC10ON); // Ref and ADC10 off
tempRaw = ADC10MEM; // Read conversion value
P1OUT &= ~BIT6; // green LED off
CCR0 += 36000; // add 1 second to the timer
_bic_SR_register_on_exit(LPM3_bits); // Clr LPM3 bits from SR on exit
}

```

Observations:

The current consumption in both the active and standby modes is measured while running the same application code. The reading for both the cases for MSP430G2553 are tabulated in below Table

Table: Current Consumption for MSP430G2553

Platform	Active Mode	Standby Mode
MSP430G2553		

Post lab Questions:

1. Write the Contents of Status Register.

2. What is Standby Mode?

3. Write the contents of Status register when MSP430 is operated in LPM4?

Outcomes

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

9. Interrupt programming through GPIOs

Regd. No: _____

Date: _____

Objectives:

To configure interrupts through GPIOs.

Apparatus:

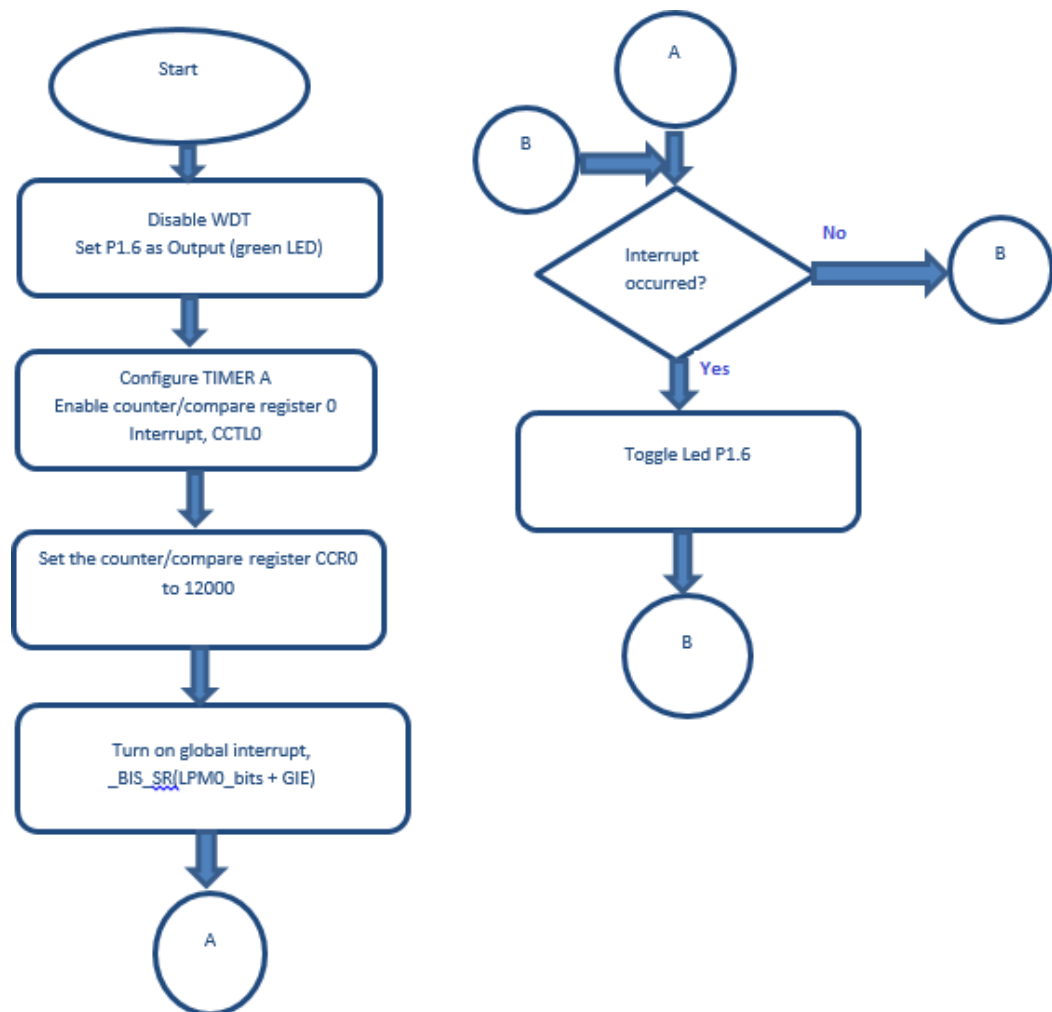
1. Code Composer Studio
2. 2MSP430G2553 Launchpad plugged into your PC via USB

Pre Lab Questions:

1. What are the various criteria to choose microcontrollers?

2. What is Interrupt and ISR?

3. What is Interrupt Latency? How can you reduce it?

Flow Chart:**Program:**

```

#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    P1DIR |= BIT6; // Set P1.6 to output direction
    P1REN |= BIT3; // Enable P1.3 internal resistance
    P1OUT |= BIT3; // Set P1.3 as pull up resistance
    P1IES |= BIT3; // P1.3 High/Low Edge
    P1IFG &= ~BIT3; // P1.3 IFG Cleared
    P1IE |= BIT3; // P1.3 Interrupt Enabled

```

```

_bis_SR_register(LPM4_bits + GIE); // Enter LPM4 w/ interrupt
_no_operation(); // For debugger
}
#pragma vector=PORT1_VECTOR
interrupt void Port_1(void)
{
P1OUT ^= BIT6; // Toggle P1.6
P1IFG &= ~BIT3; // P1.3 IFG Cleared
}

```

Post lab Questions:

1. What is P1IFG?

2. What is P1IE?

3. What is P1IES?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

10.PWM generation using Timer on MSP430 GPIO

Regd. No: _____

Date: _____

Objectives:

To generate PWM using Timer on MSP430 GPIO.

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launchpad plugged into your PC via USB

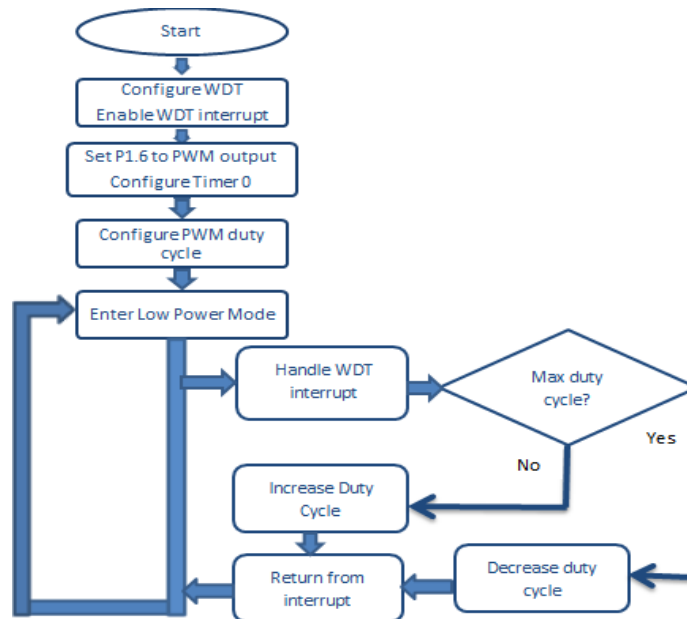
Pre Lab Questions:

1. Define PWM.

2. Define Duty Cycle and how can it be calculated.

3. What are the concerned registers used for achieving LED Brightness?

Flow Chart:



Program:

```

#include <msp430g2553.h>
int pwmDirection = 1;
void main(void){
    WDTCTL = WDT_MDLY_32; // Watchdog timer 32ms
    IE1 |= WDTIE; // enable Watchdog timer interrupts
    P1DIR |= BIT6; // Green LED for output
    P1SEL |= BIT6; // Green LED Pulse width modulation
    TA0CCR0 = 1000; // PWM period
    TA0CCR1 = 1; // PWM duty cycle, on 1/1000 initially
    TA0CCTL1 = OUTMOD_7; // TA0CCR1 reset/set-high voltage
    // below count, low voltage when past
    TA0CTL = TASSEL_2 + MC_1; // Timer A control set to SMCLK, 1MHz
    // and count up mode MC_1
    _BIS_SR(LPM0_bits + GIE); // Enter Low power mode 0
}
#pragma vector=WDT_VECTOR // Watchdog Timer ISR
__interrupt void watchdog_timer(void)

```

```

{
TAOCCR1 += pwmDirection*20; // Increase duty cycle, on vs. off time
if( TAOCCR1 > 980 ) // Pulse brighter (increasing TAOCCR1)
TAOCCR1 = 1;
}

```

Post lab Questions:

1. What is TAOCTL?

2. What is TAOCCR0?

3. What is TAOCTL1?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

11. Interfacing Potentiometer with MSP430

Regd. No: _____

Date: _____

Objectives:

To control the on-board, Red LED by the analog input from a potentiometer

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launchpad plugged into your PC via USB
3. 10k Ω Potentiometer

Pre Lab Questions:

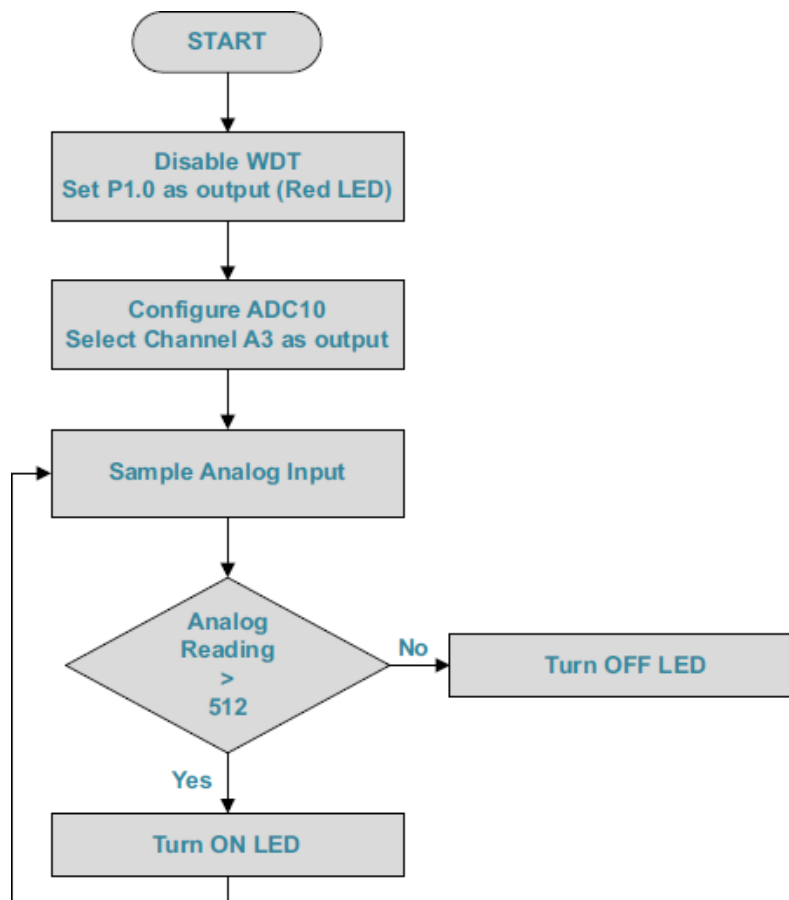
4. What is ADC?

5. What is Resolution in ADC?

6. What is Voltage Reference?

Procedure:

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.
2. Connect the positive lead of the potentiometer to the Vcc pin on the MSP-EXP430G2 Launch-
Pad and the negative lead of the potentiometer to the GND pin.
3. Connect the output lead of the potentiometer to pin P1.3 or Analog Channel A3.
4. Connect the jumpers on the MSP-EXP430G2 LaunchPad for RXD and TXD horizontally.
5. Build, program and debug the code into the LaunchPad using CCS.
6. Vary the potentiometer and observe the on board red LED.

Flow Chart:

Program:

```

int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Stop WDT
    ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10ON;
    ADC10CTL1 = INCH_3;          // input A3
    ADC10AE0 |= 0x08;            // PA.3 ADC option select
    P1DIR |= 0x01;               // Set P1.0 to output direction
    while(1)
    {
        ADC10CTL0 |= ENC + ADC10SC;    // Sampling and conversion start
        if (ADC10MEM < 512)             // ADC10MEM = A3 > 512?
            P1OUT &= ~0x01;            // Clear P1.0 LED off
        else
            P1OUT |= 0x01;              // Set P1.0 LED on
    }
}

```

Post lab Questions:

1. What is ADC10?

2. What is ADC10CTL0?

4. What is ADC10AE & ADC10MEM?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

12.PWM based Speed Control of Motor by Potentiometer

Regd. No: _____

Date: _____

Objectives:

To control the speed of a DC Motor using the potentiometer.

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launchpad plugged into your PC via USB
3. ULN2003 IC
4. DC Motor

Pre Lab Questions:

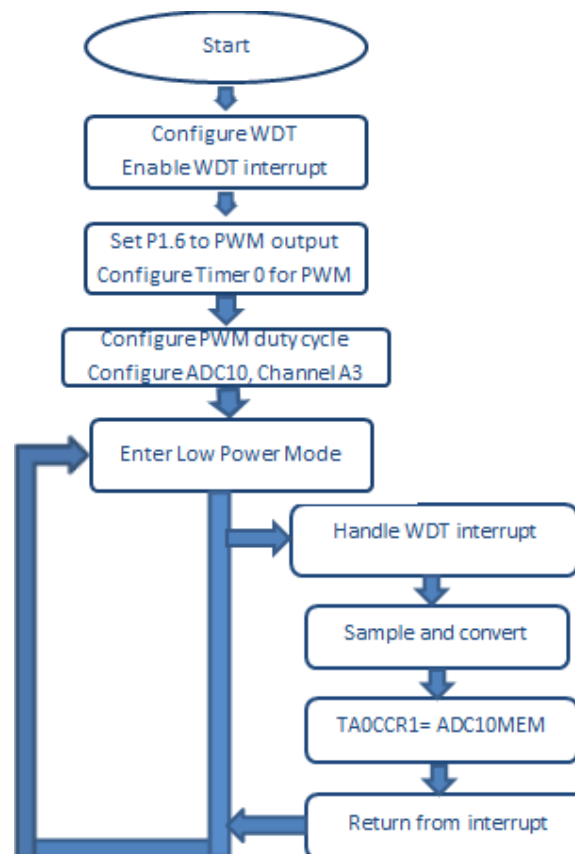
1. What is PWM?

2. What is Potentiometer?

3. What is ULN2003?

Procedure:

1. Position DIP IC ULN2003AN on a Breadboard.
2. Connect one terminal of the DC motor to the Common pin (IC Pin 9) of ULN2003.
3. Connect the junction of the above 2 terminals 5V Power from USB.
4. Connect the other terminal of the DC motor to the Drive Pin (IC Pin 16) of ULN 2003.
5. Connect the GND of the LaunchPad to the Ground Pin (IC Pin 8) of ULN2003.
6. Connect the PWM Output P1.6 to Input Signal (IC Pin 1) of ULN2003.
7. Connect one lead of the Potentiometer to VCC (J1 connector, Pin 1) on the LaunchPad.
8. Connect other lead of the Potentiometer to the GND Pin of ULN2003 (IC Pin 8).
9. Connect the center lead of the Potentiometer (variable analog output) to P1.3 which is the analog Input 0 of ADC10 module of MSP43G2553.
10. Build, program and debug the code into the LaunchPad using CCS.
11. Vary the potentiometer and observe the speed of the DC motor.
12. Also observe the corresponding digital output on the CCS window.

Flow Chart:

Program:

```

#include <msp430g2553.h>
int pwmDirection = 1;
void main(void){
    WDTCTL = WDT_MDLY_32;           // Watchdog timer 32ms
    IE1 |= WDTIE;                   // enable Watchdog timer interrupts
    ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10ON;
    ADC10CTL1 = INCH_3;              // input A3
    ADC10AE0 |= 0x08;               // PA.3 ADC option select
    P1DIR |= BIT6;                  // Green LED for output
    P1SEL |= BIT6;                  // Green LED Pulse width modulation
    TA0CCR0 = 1024;                  // PWM period
    TA0CCR1 = 1;                    // PWM duty cycle, on 1/1000 initially
    TA0CCTL1 = OUTMOD_7;            // TA0CCR1 reset/set-high voltage
                                   // below count, low voltage when past
    TA0CTL = TASSEL_2 + MC_1;       // Timer A control set to SMCLK, 1MHz
                                   // and count up mode MC_1
    _BIS_SR(LPM0_bits + GIE);      // Enter Low power mode 0
}

#pragma vector=WDT_VECTOR           // Watchdog Timer ISR
__interrupt void watchdog_timer(void) {
    ADC10CTL0 |= ENC + ADC10SC;
    TA0CCR1 = ADC10MEM;
}

```

Post lab Questions:

1. What is DC Motor?

2. What is LPM0 & GIE?

3. What is OUTMOD_7?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

13.Using ULP Advisor in CCS on MSP430

Regd. No: _____

Date: _____

Objectives:

To optimize the power efficient application on MSP430 launch pad using ULP Advisor in CCStudio.

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launchpad plugged into your PC via USB

Pre Lab Questions:

4. Mention the capabilities of ULP advisor.

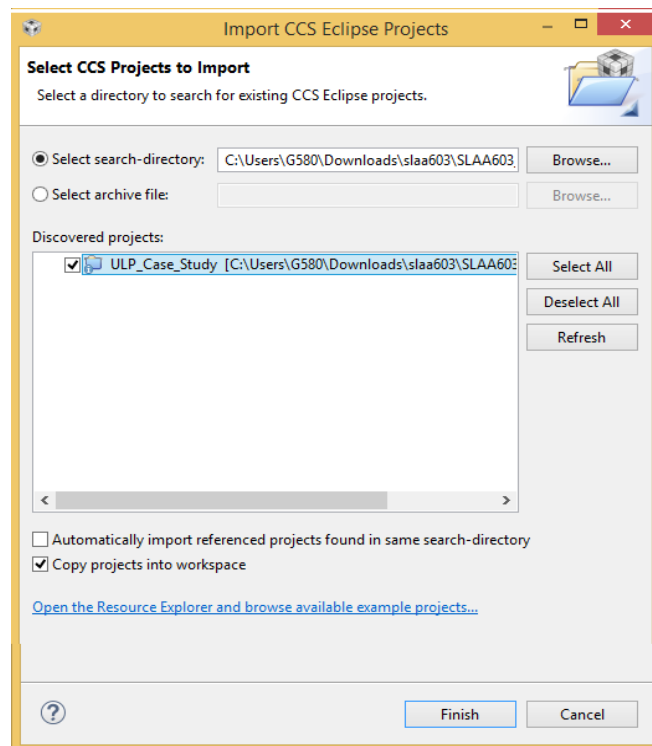
5. How does combination of functions reduce memory requirement in embedded systems?

6. What are the programming languages used in embedded systems?

Procedure:

Import the project into your CCS workspace:

1. Click Project --Import Existing CCS Eclipse Project.
2. Browse for the directory where downloaded the file.
3. Check the box next to the 'ULP_Case_Study' project in the Discovered projects window.
4. Un-check the box next to Copy projects into workspace.
5. Click Finish



6. Set as active project by left-clicking on the project name. The active build configuration should be Inefficient. If not, right click on the project name in the Project Explorer, and then click Build Configurations → Set Active → Inefficient.
7. Build the project.
8. The Advice window shows suggestions from the ULP Advisor under the Power (ULP) Advice heading. Click View → Advice
9. Click the link #1532-D, which corresponds to the first ULP violation (for using `sprintf()`), to open the ULP Advisor wiki page in a second tab titled Advice. All of the information for this particular rule violation is provided.
10. Scroll down to the Remedy section. The first suggestion is to avoid using `sprintf()` altogether and work with the raw data.

Post lab Questions:

1. How does the ULP advisor software help in designing power optimized code?

2. Which ULP rule violation helps us to detect looping counting violation?

4. Mention how I/O devices are classified for embedded system.

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

14.Low Power modes and Energy Trace++

Regd. No: _____

Date: _____

Objectives:

- (A) To Enable Energy Trace and Energy Trace++ modes in CCS.
(B) To Compute Total Energy, and Estimated lifetime of an AA battery

Apparatus:

1. Code Composer Studio
2. MSP430G2 LaunchPad
3. MSP430FR5969 Launchpad plugged into your PC via the USB cable
5. Jumper Wires

Pre Lab Questions:

1. What are the modes in Energy Capture?

2. What is Energy Trace?

3. What is Energy Trace++?

Procedure:

Hardware Connection

1. Remove the RST, TST, V+, and GND jumpers from J13 on the MSP-EXP430FR5969 LaunchPad.
2. Remove the RST, TEST, and VCC jumpers from J3 on the MSP-EXP430G2 LaunchPad.
3. Use jumper wires to connect the signals on the emulation side of the MSP-EXP430FR5969 board to the corresponding signal on the device side of the MSP-EXP430G2 LaunchPad as in below Table.

Table : Connection of Signals

MSP-EXP430FR5969 Emulation	MSP-EXP430G2XL Device
RST	RST
TST	TEST
V+	VCC
GND	GND

4. Target power must be supplied through the MSP-EXP430FR5969 LaunchPad that has the Energy Trace technology included in the on-board emulation. Plug-in the micro-USB to the MSP-EXP430FR5969.
5. Initialize the debug session for the MSP430G2553 to enable the Energy Trace mode.

Software Execution

1. Build the example code of Experiment 7: PWM Based Speed Control of Motor by Potentiometer that you want to analyze using Energy Trace technology.
2. Enable Energy Trace technology
 - a. Click Window → Preferences → Code Composer Studio → Advanced Tools → Energy Trace Technology
 - b. Make sure that the box next to Enable is checked
 - c. Select the Energy Trace++ mode
 - d. Click Apply and OK.
3. Debug the example code, the Energy Trace window will open automatically.
If the window does not open automatically, click View → Others → MSP430 Energy Trace → Energy Trace Technology
4. Run the code. The Energy Trace Technology window will update the real time data.

(A) To Enable Energy Trace and Energy Trace++ modes in CCS.

Observation:

Complete analysis of Energy Trace Technology for Experiment 7: PWM Based Speed Control of Motor by Potentiometer in terms of Energy, Power, Current, Voltage are given in below Table

Table : Analysis of Energy Trace Technology

Example Code	Energy in mJ	Mean Power in mW	Mean Current in mA	Mean Voltage in V
--------------	--------------	------------------	--------------------	-------------------

(B) To Compute Total Energy, and Estimated lifetime of an AA battery

Observation:

The Energy trace profile for active and stand by code is analyzed using Energy Trace technology. The Energy Trace profile window of the application in active mode provides us the estimated lifetime of a battery of _____ days. If the same application runs in standby mode, the estimated lifetime of a battery exceeds to _____ days.

Table : Energy measurement and Estimated lifetime of a battery

Mode	Energy in mJ for the 30sec time period	Estimated life time of a battery in days.
Active Mode		
Standby Mode		

Post lab Questions:

1. What is digital signal controller?

2. List the features of MSP430.

3. Mention the difference between firmware and software.

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading:

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

Experiment No. _____

15.Serial Communication

Regd. No: _____

Date: _____

Objectives:

To use UART of the MSP430G2553 to communicate with the computer.

Apparatus:

1. Code Composer Studio
2. MSP430G2553 Launchpad plugged into your PC via USB


Pre Lab Questions:

7. What is Synchronous serial communication?

8. What is Asynchronous serial communication?

9. What is Baud rate?

UART Terminal Setup:

1. In CCS window select **View** → **Other** and from the options given select **Terminal**.
2. When the terminal window opens, click on the **Settings** icon  and set the values as shown in below Figure

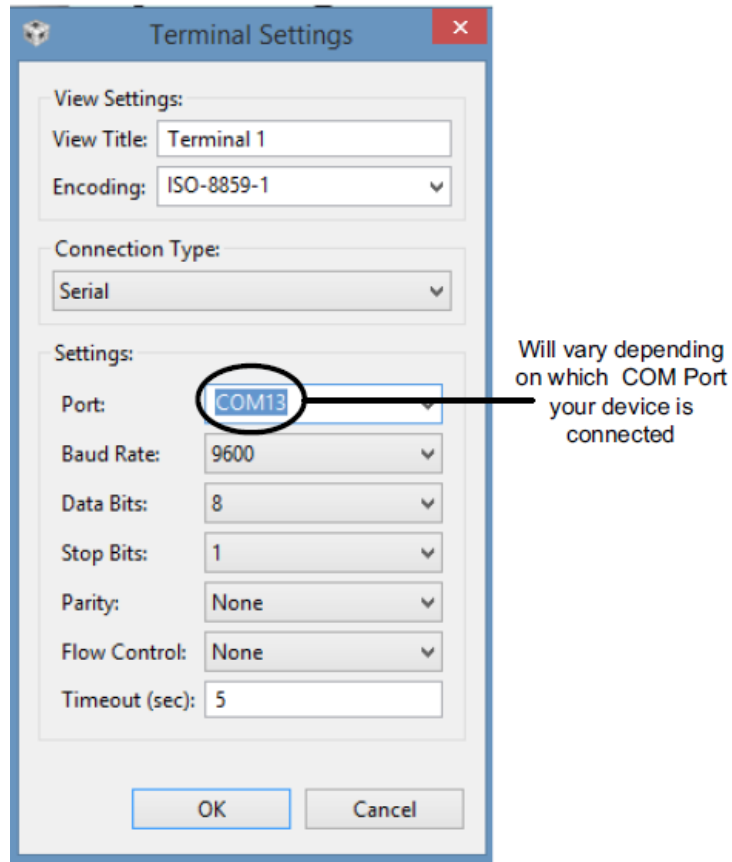
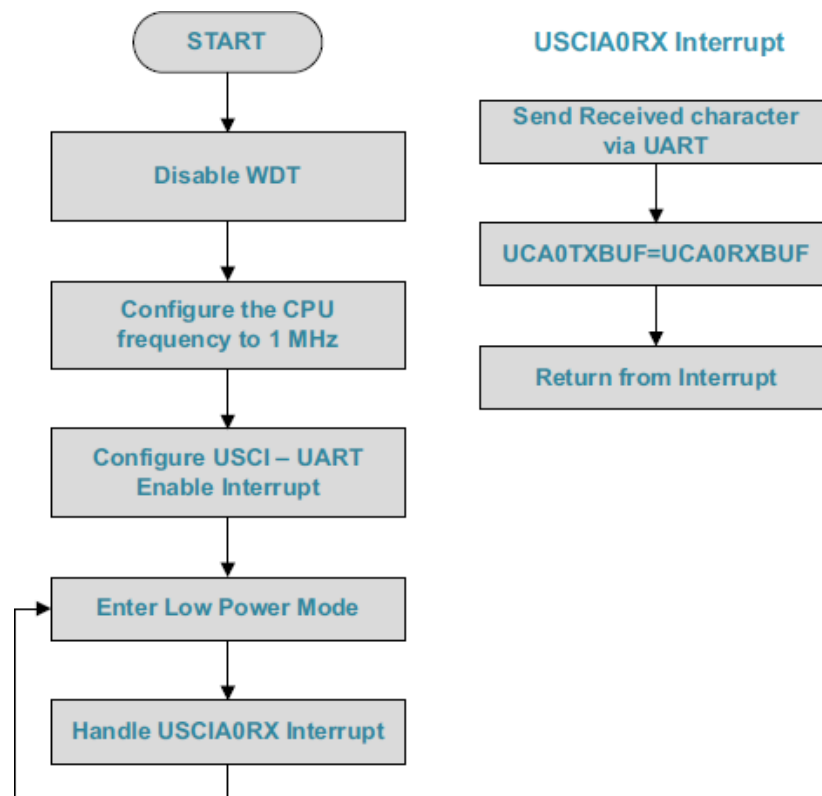


Figure: Terminal Window Settings

This Terminal will allow you to view the serial information received by the computer and will also allow you to type in the information that you wish to transmit to the MSP430G2553 via the UART.

Procedure:

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.
2. Build, program and debug the code into the LaunchPad using CCS.

Flow Chart:**Figure: Flowchart for Serial Communication Using UART****Program:**

```

#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
    P1DIR |= BIT0;
    if (CALBC1_1MHZ==0xFF)        // If calibration constant erased
    {
        P1OUT |= BIT0;            // Red LED Indication
        while(1);                 // do not load, trap CPU!!
    }
    /* Use Calibration values for 1MHz Clock DCO*/
    DCOCTL = 0;                   // Select lowest DCOx and MODx
    settings
    BCSTL1 = CALBC1_1MHZ;         // Set DCO
  
```

```

DCOCTL = CALDCO_1MHZ;
/* Configure Pin P1.1 = RXD and P1.2 = TXD */
P1SEL = BIT1 + BIT2;
P1SEL2 = BIT1 + BIT2;
/* Place UCA0 in Reset to be configured */
UCA0CTL1 = UCSWRST;
/* Configure */
UCA0CTL0 = 0x00;                                // No parity, LSB first, 8-bit data, 1
                                                // stop bit, UART, Asynchronous

UCA0CTL1 |= UCSSEL_2;                            // CLK = SMCLK
UCA0BR0 = 104;                                    // 1MHz/9600 = 104.166
UCA0BR1 = 0x00;
UCA0MCTL = UCBRS0;                                // Modulation UCBRSx = 1
/* Take UCA0 out of reset */
UCA0CTL1 &= ~UCSWRST;                            // Initialize USCI_UART
IE2 |= UCA0RXIE;                                  // Enable USCI_A0 RX interrupt
__bis_SR_register(LPM3_bits + GIE);              // Enter LPM0 with interrupt
}
/*Echo back RXed character, confirm TX buffer is ready first*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    while (!(IFG2 & UCA0TXIFG));                  // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;                        // TX RXed character
}

```

Post lab Questions:

1. What is UCA0CTL0?

2. What is UCA0CTL1?

3. What is UCA0BR0 & UCA0BR1?

Outcomes:

Upon completion of the experiment, the student will be able to

1. Develop an algorithm, the flow diagram, source code and perform the compilation.
2. Generate the required binary file which can be dumped into the controller and obtain the respective output control on the connected peripheral.
3. Verify the logic with the necessary hardware.

Grading

Date of Submission					
Marks Awarded (Max.Marks-10M)	Pre-Lab Questions (2M)	Observations (3M)	Post-Lab Questions (2M)	Viva (3M)	Total (10M)
Remarks					
Signature of the Evaluator with Date					

NOTES

MICROPROCESSORS & MICROCONTROLLERS LABORATORY

PART - C

LAB BASED MINI PROJECT



Department of Electronics and Communication
Engineering SVR Engineering College: Nandyal

**Approved by AICTE, New Delhi & Affiliated to
JNTUA, Anantapuramu (An ISO 9001:2007 Certified
Institution)**

Near Ayyalur (V), Allagadda Rd, Nandyal, Kurnool (Dt), Andhra Pradesh 518503

LAB BASED MINI PROJECT
(Format to be submitted with Mini Project Report)

Title of the Project :

Abstract :

Introduction :

Objectives :

Project Description :

Outcomes :

Conclusions :

References :