DEPARTMENT OF
ELECTRONICS & COMMUNICATION ENGINEERING
LAB MANUAL
Microprocessors & Microcontrollers Lab
II - B. Tech. II - Semester

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Autonomous, Accredited by NBA & NAAC, an ISO 9001:2008 certified institution)

(Sponsored by Siddhartha Academy of General & Technical Education)

VIJAYAWADA – 520 007,
ANDHRA PRADESH

Microprocessors & Microcontrollers Lab MANUAL

Prepared by

Mr. K. Phani Rama Krishna

&

Dr. Haji. Habibulla Md.

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
Microprocessors & Microcontrollers Lab

LIST OF EXPERIMENTS

Syllabus		
Expt. No.	**Contents**	**Mapped CO**
I	16-bit Signed and unsigned Arithmetic operations, ASCII – arithmetic operations	**CO1,CO4**
II	Arithmetic operations – Multi byte Addition and Subtraction	**CO1,CO4**
III	Logical operations, Sum of Squares, Sum of Cubes	**CO1,CO4**
IV	Write ALP to find smallest, largest number, arrange numbers in Ascending order, Descending order in a given series.	**CO1,CO4**
V	Using string operation and Instruction prefix: Move Block, Reverse string, String comparison	**CO1,CO4**
VI	Introduction to MSP430 launch pad and Programming Environment. (Study Experiment)	**CO2, CO4**
VII	Read input from switch and Automatic control/flash LED (soft-ware delay).	**CO2,CO3,CO4**
VIII	Read Temperature of MSP430 with the help of ADC.	**CO2, CO3,CO4**
IX	Interrupts Programming Example Using GPIO	**CO2, CO3,CO4**
X	Use Of Comparator To Compare The Signal Threshold Level	**CO2, CO3, CO4**

Additional Experiments:

1. Average of numbers
2. Conversion of Packed BCD to Unpacked BCD, Packed BCD to ASCII

INSTRUCTIONS TO THE STUDENTS

1. Students are required to attend all labs.

2. Students have to bring the lab manual cum observation book, record etc. along with them whenever they come for lab work.

3. Should learn the prelab questions. Read through the lab experiment to familiarize themselves with the components and assembly sequence.

4. Should utilize 3 hours’ time properly to perform the experiment and to record the readings. Do the calculations, and take signature from the instructor.

5. If the experiment is not completed in the stipulated time, the pending work has to be carried out in the leisure hours or extended hours.

6. Should submit the completed record book according to the deadlines set up by the instructor.

7. For practical subjects there shall be a continuous evaluation during the semester for 15 internal marks and 35 end examination marks.

8. Out of 15 internal marks, 10 marks shall be awarded for day-to-day work and 5 marks to be awarded by conducting an internal laboratory test.

EXPERIMENT-1

16-bit Signed and unsigned Arithmetic operations, ASCII – arithmetic operations

AIM: To perform 16-bit Signed and unsigned Arithmetic operations, ASCII – arithmetic operations using TASM.

Experimental Requirements: PC loaded with TASM software ,8086 microprocessor kit and power supply.

Procedure for doing **DEBUG** program:

Step1:

Open the dosbox icon placed in the desktop

Step 2:

type the following

mount c c:\8086

and press enter

then dos box will be mounted to the local directory.

Step 3:

type c:

and press enter

You will be getting the screen as:

c:\

Step4:

Now Type

debug

,and press enter

Now you are going to get hyphen symbol

Step5: Type

a ,and press enter

Type your program as shown in the attachment.

Step6:

Type

R IP

,and press enter

The instruction pointer should point to the starting address of the program.

If not , type the starting address of the program.For Example, 0100

Step7:

We need to do single step execution.

For single step execution:

type

t as shown in the attachment.

Till in the end of the program we need to repeat.

Procedure for **TASM**:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd... or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

Programs

8-BIT OPERATIONS

1.ADDITION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DB 78H
OPR2 DB 23H
RES DB 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AL, OPR1
MOV     BL, OPR2
ADD       AL, BL
MOV     RES, AL
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AL: BL:

OUTPUT:

AL: AH:

FLAG STATUS:

Theoretical Calculations:

2. **SUBTRACTION:**

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DB 36H
OPR2 DB 23H
RES DB 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AL, OPR1
MOV     BL, OPR2
SUB       AL, BL
MOV     RES, AL
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:
Theoretical Calculations:

3. **MULTIPLICATION:**

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DB 15H
OPR2 DB 05H
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AL, OPR1
MOV     BL, OPR2
MOV      AH,00H
MUL       BL
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AL: BL:

OUTPUT:

AL: AH:

FLAG STATUS:

Theoretical Calculations:

4. **DIVISION:**

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DB 20H
OPR2 DB 05H
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AL, OPR1
MOV     BL, OPR2
MOV      AH,00H
DIV       BL
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:
Theoretical Calculations:

1.Addition

```
-A
072A:0100
-A 400
072A:400 MOV AL,55
072A:4002 MOV BL,32
072A:4004 ADD AL,BL
072A:4006
-R IP
IP 0100
-R IP 4000
-T
-G
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:

Theoretical Calculations:

2.Subraction :

```
-A
072A:0100
-A 4000
072A:4000 MOV AL,37
072A:4002 MOV BL,36
072A:4004 SUB AL,BL
072A:4006
-R IP
IP 0100
-R IP 4000
-T
-G
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:

Theoretical Calculations:

3.Multiplication:

```
-A
072A:0100
-A 400
072A:400 MOV AL,54
072A:4002 MOV BL,21
072A:4004 MUL,BL
072A:4006 INT 03
-R IP
IP 0100
-R IP 4000
-T
-G
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:

Theoretical Calculations:

4. Divison

```
-A
072A:0100
-A 400
072A:400 MOV AL,24
072A:4002 MOV BL,4
072A:4004 DIV,BL
072A:4006
-R IP
IP 0100
-R IP 4000
-T
-G
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AL: BL:

OUTPUT:

AL: AH:

FLAG STATUS:

Theoretical Calculations:

16-bit ADDITION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 78BCH
OPR2 DW 23FEH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
ADD       AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:
Theoretical Calculations:

16-bit SUBTRACTION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 36BBH
OPR2 DW 23CCH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
SUB       AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:
Theoretical Calculations:

16-bit MULTIPLICATION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 1506H
OPR2 DW 0AC05H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,0000H
MUL       BX
MOV     RES1, AX
MOV     RES2,DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

16-bit. DIVISION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 0F506H
OPR2 DW 0AC50H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,0000H
DIV       BX
MOV     RES1, AX
MOV     RES2,DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

16 BIT SIGNED ARITHMETIC OPERATIONS

1. **ADDITION:**

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 0BCDEH
OPR2 DW 0ABCDH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
STC
ADD    AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 0BCDEH
OPR2 DW 0ABCDH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
CLC
ADC    AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

2. **SUBTRACTION:**

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 0BCDEH
OPR2 DW 0ABCDH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
STC
SUB    AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:

Theoretical Calculations:

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 0BCDEH
OPR2 DW 0ABCDH
RES DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
CLC
SBB    AX, BX
MOV     RES, AX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:
Theoretical Calculations:

3. **MULTIPLICATION:**

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 1111H
OPR2 DW 1111H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,0000H
MUL       BX
MOV     RES1, AX
MOV     RES2, DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 1111H
OPR2 DW 8888H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,0000H
IMUL       BX
MOV     RES1, AX
MOV     RES2, DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

AX: BX:

OUTPUT:

AX: DX:

FLAG STATUS:

Theoretical Calculations:

4. DIVISION:

```
 ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 2224H
OPR2 DW 1111H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,00H
DIV       BX
MOV     RES1, AX
MOV     RES2, DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1 DW 2224H
OPR2 DW 1111H
RES1 DW 1 DUP (0H)
RES2 DW 1 DUP (0H)
 DATA ENDS
CODE   SEGMENT
START:
MOV    AX, DATA
MOV    DS, AX
MOV    AX, OPR1
MOV     BX, OPR2
MOV      DX,00H
IDIV BX
MOV     RES1, AX
MOV     RES2, DX
INT         03H
CODE   ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX: DX:
FLAG STATUS:
Theoretical Calculations:

ASCII OPERATIONS

1. AAA:

```
ASSUME CS: CODE
CODE   SEGMENT
START:
MOV    AL, 35H
MOV    BL,39H
MOV    AH,00H
ADD      AL,BL
AAA
INT     03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:
Theoretical Calculations:

2. AAS:

```
ASSUME CS: CODE
CODE   SEGMENT
START:
MOV    AL, 39H
MOV    BL,35H
MOV    AH,00H
SUB       AL,BL
AAS
INT     03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AL: BL:

OUTPUT:

AL: AH:

FLAG STATUS:

Theoretical Calculations:

3. AAM:

```
ASSUME CS: CODE
CODE   SEGMENT
START:
MOV    AL, 05H
MOV    BL,09H
MOV    AH,00H
MUL BL
AAM
INT     03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL: BL:
OUTPUT:
AL: AH:
FLAG STATUS:

4. AAD:

```
ASSUME CS: CODE
CODE   SEGMENT
START:
MOV    AL, 05H
MOV    BL,06H
MOV    AH,03H
AAD
DIV BL
INT     03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

AL: AH: BL:

OUTPUT:

AL: AH:

FLAG STATUS:

Result: Arithmetic operation –Signed and unsigned Arithmetic operation, ASCII – arithmetic operations were performed.

EXPERIMENT – 02

Arithmetic operations – Multi byte Addition and Subtraction,

AIM : To perform multibyte addition, subtraction, sum of squares and sum of cubes using TASM.

Experimental Requirements : PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd... or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

1.MULTI BYTE ADDITION

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1   DB   12H, 34H, 56H, 78H
OPR2   DB   23H, 34H, 66H, 86H
RES   DW   1   DUP (0H)
DATA    ENDS
CODE   SEGMENT
START: MOV    AX, DATA
MOV    DS, AX
MOV    SI, OFFSET   OPR1
MOV    DI, OFFSET   OPR2
MOV    BX, OFFSET   RES
MOV    CX, 0004H
MOV      AH, 00H
BACK: MOV     AL, [SI]
MOV     DL, [DI]
ADC        AL, DL
MOV      [BX], AL
INC         SI
INC         DI
INC         BX
LOOP     BACK
INT     03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
SI: DI:
OUTPUT:
RES:
FLAG STATUS:
Theoretical Calculations:

2. MULTI BYTE SUBTRACTION

```
ASSUME CS: CODE, DS: DATA
DATA   SEGMENT
OPR1   DB   23H, 34H, 66H, 86H
OPR2   DB   12H, 34H, 56H, 78H
RES   DW   1   DUP (0H)
DATA    ENDS
CODE   SEGMENT
START:MOV    AX, DATA
MOV    DS, AX
MOV    SI, OFFSET   OPR1
MOV    DI, OFFSET   OPR2
MOV    BX, OFFSET   RES
MOV    CX, 0004H
MOV      AH, 00H
BACK: MOV     AL, [SI]
MOV     DL, [DI]
SBB        AL, DL
MOV      [BX], AL
INC         SI
INC         DI
INC         BX
LOOP     BACK
INT     03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
SI: DI:
OUTPUT:
RES:
FLAG STATUS:
Theoretical Calculations:

Result: Multibyte addition, subtraction has been performed using TASM.

EXPERIMENT-03

Logic operations – Shift and rotate – Sum of Squares, Sum of Cubes

AIM: To perform logical operations on 16-bit using TASM.

Experimental Requirements: PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd...   or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with   tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs**.**

Logical Instructions:

1.AND:

```
ASSUME CS: CODE
CODE    SEGMENT
START:
MOV    AX, 3355H
MOV    BX, 5355H
AND    AX, BX
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

2. **OR:**

```
ASSUME CS: CODE
CODE    SEGMENT
START:
MOV    AX, 3355H
MOV    BX, 5355H
OR        AX, BX
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: BX:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

3. **NOT:**

```
ASSUME CS: CODE
CODE    SEGMENT
START:
MOV    AX, 3355H
NOT    AX
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

4. XOR:

```
ASSUME CS: CODE
CODE    SEGMENT
START:
MOV    AX, 3355H
MOV    BX, 5355H
XOR    AX, BX
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AX: BX:

OUTPUT:

AX:

FLAG STATUS:

Theoretical Calculations:

5.TEST:

```
 ASSUME CS: CODE

CODE    SEGMENT
START:
MOV    AX, 3355H
MOV    BX, 5355H
TEST    AX, BX
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AX: BX:

OUTPUT:

AX:

FLAG STATUS:

Theoretical Calculations:

Shift and Rotate Instructions

1. **SHR:**

```
ASSUME CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
SHR     AX, CL
 INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

2. SHL:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
SHL    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
SAL    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AX: CL:

OUTPUT:

AX:

FLAG STATUS:

Theoretical Calculations:

3. ROTATE RIGHT:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
STC
ROR    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
CLC
ROR    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

4. ROTATE LEFT:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
STC
ROL    AX, CL
INT   03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
CLC
ROL    AX, CL
INT   03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

5. ROTATE RIGHT THROUGH CARRY:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
STC
RCR    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
CLC
RCR    AX, CL
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

6. **ROTATE LEFT THROUGH CARRY:**

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
STC
RCL    AX, CL
INT   03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

```
ASSUME    CS: CODE
CODE    SEGMENT
START:
MOV    AX, 0ABCDH
MOV    CL, 04H
CLC
RCL    AX, CL
INT   03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AX: CL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations:

SUM OF SQUARES ($1^2 + 2^2 + 3^2 + \ldots + n^2$)

```
ASSUME CS:CODE
CODE SEGMENT
START: MOV CL,07H
MOV DX,0000H
MOV AH,00H
L1: MOV AL,CL
MUL CL
ADD DX,AX
LOOP L1
INT 03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
CL:
OUTPUT:
DX:
FLAG STATUS:
Theoretical Calculations:

SUM OF SQUARES IN AN ARRAY

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
ARR1 DB 05H,07H,06H,04H
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
MOV DS,AX
MOV SI,OFFSET ARR1
MOV CX,0004H
MOV DX,0000H
MOV AH,00H
L1: MOV BL,[SI]
    MOV AL,BL
   MUL BL
   ADD DX,AX
   INC SI
   LOOP  L1
   INT 03H
   CODE ENDS
   END START
   END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:

INPUT:

ARR1:

OUTPUT:

DX:

FLAG STATUS:

Theoretical Calculations:

SUM OF CUBES ($1^3 + 2^3 + 3^3 + \ldots + n^3$)

```
ASSUME CS:CODE
CODE SEGMENT
START: MOV CL,07H
MOV DX,0000H
MOV AH,00H
L1: MOV AL,CL
MUL CL
MUL CL
ADD DX,AX
LOOP L1
INT 03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
CL:
OUTPUT:
DX:
FLAG STATUS:
Theoretical Calculations:

SUM OF CUBES IN AN ARRAY

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
ARR1 DB 05H,07H,06H,04H
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
MOV DS,AX
MOV SI,OFFSET ARR1
MOV CX,0004H
MOV DX,0000H
MOV AH,00H
L1: MOV BL,[SI]
   MOV AL,BL
   MUL BL
   MUL BL
   ADD DX,AX
   INC SI
   LOOP  L1
   INT 03H
   CODE ENDS
   END START
   END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
ARR1:
OUTPUT:
DX:
FLAG STATUS:
Theoretical Calculations

Result: Logic operations – Shift and rotate – sum of squares and sum of cubes using TASM were performed.

EXPERIMENT-4

Smallest, largest number, arrange numbers in Ascending order, Descending order

AIM: To find smallest, largest number, arrange numbers in ascending order, descending order in a given series.

Experimental Requirements: PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd... or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

Smallest number

```
ASSUME    CS: CODE, DS: DATA
DATA    SEGMENT
LIST    DB   35H, 26H, 19H, 56H, 44H
DATA    ENDS
CODE    SEGMENT
START:
MOV    AX, DATA
MOV     DS, AX
MOV     CX, 0004H
MOV    SI, OFFSET    LIST
MOV    BL, [SI]
L2:   MOV    AL, [SI+1]
CMP     BL, AL
JB    L1
MOV    BL, AL
L1:   INC     SI
LOOP     L2
INT        03H
CODE    ENDS
END     START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
LIST:
OUTPUT:
BL:
FLAG STATUS:
Theoretical Calculations

Largest number

```
ASSUME    CS: CODE, DS: DATA
DATA    SEGMENT
LIST    DB   35H, 26H, 19H, 56H, 44H
DATA    ENDS
CODE    SEGMENT
START:
MOV    AX, DATA
MOV     DS, AX
MOV     CX, 0004H
MOV    SI, OFFSET    LIST
MOV    BL, [SI]
L2:   MOV    AL, [SI+1]
CMP     BL, AL
JA    L1
MOV    BL, AL
L1:   INC     SI
LOOP     L2
INT        03H
CODE    ENDS
END     START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
LIST:
OUTPUT:
BL:
FLAG STATUS:
Theoretical Calculations:

Ascending order

```
ASSUME    CS: CODE, DS: DATA
DATA    SEGMENT
STR1    DB    'BINDHU$'
DATA    ENDS
CODE    SEGMENT
START: MOV    AX, DATA
MOV     DS, AX
MOV     DX, 0005H
L3: MOV    CX, DX
MOV    SI, OFFSET STR1
L2: MOV   AL, [SI]
CMP    AL, [SI+1]
JB    L1
XCHG    AL, [SI+1]
XCHG   AL, [SI]
L1: INC    SI
LOOP    L2
DEC    DX
JNZ    L3
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
OUTPUT:
STR1:
FLAG STATUS:
Theoretical Calculations

Descending order

```
ASSUME    CS: CODE, DS: DATA
DATA    SEGMENT
STR1    DB    'BINDHU$'
DATA    ENDS
CODE    SEGMENT
START: MOV    AX, DATA
MOV     DS, AX
MOV     DX, 0005H
L3: MOV    CX, DX
MOV    SI, OFFSET STR1
L2: MOV   AL, [SI]
CMP    AL, [SI+1]
JA    L1
XCHG    AL, [SI+1]
XCHG   AL, [SI]
L1: INC    SI
LOOP    L2
DEC    DX
JNZ    L3
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
OUTPUT:
STR1:
FLAG STATUS:
Theoretical Calculations

RESULT: Finding the smallest, largest numbers and arranging given numbers in ascending and descending orders using TASM are performed.

EXPERIMENT-5

STRING OPERATIONS

Aim : String operation and Instruction prefix: Move Block, Reverse string, Inserting, Deleting, Length of the string, String comparison.

Experimental Requirements : PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd... or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

STRING OPERATIONS

1. MOVING A BLOCK OF DATA

```
ASSUME CS: CODE, DS: DATA, ES:EXTRA
DATA SEGMENT
ORG 1000H
STR1 DB 'HI FRIEND$'
COUNT EQU $-STR1
DATA ENDS
EXTRA SEGMENT
ORG 2000H
STR2 DB 1 DUP(?)
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,COUNT-1
REP    MOVSB
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
OUTPUT:
STR2:
FLAG STATUS:
Theoretical Calculations

2. **REVERSE OF A STRING**

```
ASSUME  CS: CODE, DS: DATA
DATA SEGMENT
ORG 1000H
STR1 DB 'HI FRIEND$'
COUNT EQU $-STR1
DATA ENDS
CODE SEGMENT
START:
MOV AX, DATA
MOV DS, AX
MOV SI, OFFSET STR1
MOV DI, OFFSET STR1+COUNT-2
MOV CL, COUNT/2
BACK: MOV AL,[SI]
XCHG [DI], AL
XCHG [SI], AL
INC SI
DEC DI
LOOP BACK
INT 03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
OUTPUT:
STR1:
FLAG STATUS:
Theoretical Calculations

3. **STRING COMPARISON**

```
ASSUME CS:CODE, DS:DATA,ES:EXTRA
DATA SEGMENT
ORG 1000H
STR1 DB 'HI FRIEND$'
COUNT EQU $-STR1
DATA ENDS
EXTRA SEGMENT
ORG 2000H
STR2 DB 'HIFRIEND'
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, COUNT-1
REP    CMPSB
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
STR2:
OUTPUT:
Z=
FLAG STATUS:
Theoretical Calculations

```
ASSUME CS:CODE, DS:DATA,ES:EXTRA
DATA SEGMENT
ORG 1000H
STR1 DB 'HI FRIEND$'
COUNT EQU $-STR1
DATA ENDS
EXTRA SEGMENT
ORG 2000H
STR2 DB 'HI FRIEND$
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, COUNT-1
REP    CMPSB
INT    03H
CODE    ENDS
END    START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
STR1:
STR2:
OUTPUT:
Z=
FLAG STATUS:
Theoretical Calculations:

Result: String operation and Instruction prefix: Move Block, Reverse string and String comparison were performed.

EXPERIMENT-6

Introduction to MSP430 launch pad and Programming Environment. (Study Experiment)

Aim: To write an assembly language program to blink an LED

Experiment Requirements: PC loaded code composer studio, MSP430 LAUNCHPAD

Procedure:

1. Open code composer studio
2. Open file go to new and select CCS project
3. A CCS window opens.

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USB1 (default)

   Give a project name.

   Select empty project with main.c and press finish.

4. Write C code in main.c.
5. Select build project and build your program. It will check for errors.When it is error free go to next step, otherwise repeat until the program is error free.
6. Go to target configuration

   Select user defined

   Select new target

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.

7. Open New Target configuration

   Right click on new target configuration

   Click on Launch selected configuration

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USB1 (default)

   and Save.

8. Open the Run menu

Select connect project

Again open Run

Load Project

Browse the project

select the project.out

select,save and ok.

9. Run the program.

10. Observe the output in the console window or on the board.

Program:

```
#include <msp430.h>

void main(void) {
   WDTCTL = WDTPW | WDTHOLD;      // Stop watchdog timer

   P1DIR |= (BIT0+BIT6);        // P1.0 (Red LED), P1.1 (Green LED)

   while(1)
   {
      volatile unsigned long i;

      P1OUT &= ~BIT6;            //Green LED -> OFF
      P1OUT |= BIT0;            //Red LED -> ON

      for(i = 0; i<10000; i++);   //delay

      P1OUT &= ~BIT0;            //Red LED -> OFF
      P1OUT |= BIT6;            //Green LED -> ON

      for(i = 0; i<10000; i++);   //delay
   }
}
```

Result: Blinking of LED on the MSP430 launch pad was performed.

EXPERIMENT-7

Read input from switch and Automatic control/flash LED (soft-ware delay).

Aim: To read input from switch and Automatic control/flash LED (soft-ware delay).

Experiment Requirements: PC loaded code composer studio, MSP430 LAUNCHPAD

Procedure:

1. Open code composer studio
2. Open file go to new and select CCS project
3. A CCS window opens.

    Select MSP430G2253 in the target.

    Establish the connection by selecting the TI MSP430 USB1 (default)

    Give a project name.

    Select empty project with main.c and press finish.

4. Write C code in main.c.
5. Select build project and build your program. It will check for errors.When it is error free go to next step, otherwise repeat until the program is error free.
6. Go to target configuration

    Select user defined

    Select new target

    Select MSP430G2253 in the target.

    Establish the connection by selecting the TI MSP430 USBI

    Save.

7. Open New Target configuration

    Right click on new target configuration

    Click on Launch selected configuration

    Select MSP430G2253 in the target.

    Establish the connection by selecting the TI MSP430 USB1 (default)

    and Save.

8. Open the Run menu

    Select connect project

    Again open Run

    Load Project

    Browse the project

        select the project.out

        select,save and ok.

9. Run the program.

10. Observe the output in the console window or on the board.

Program:

```
#include <msp430.h>

/*
 * main.c
 */
void main(void)
{
        WDTCTL = WDTPW + WDTHOLD;               // Stop watchdog timer

        P1DIR |= 0x01; // Set P1.0 to output direction
        P1OUT |= BIT3;
        P1REN |= BIT3;
        while (1) // Infinite Loop
        {
        if ((BIT3 & P1IN)) // active low switch
        {
        P1OUT &= ~0x01; // if P1.3 is 1(not pressed),reset P1.0
        }else
        {
        P1OUT |= 0x01; // else set P1.0
        }
        }
        }
```

Result: Reading input from switch and Automatic control/flash LED (soft-ware delay) has been performed.

EXPERIMENT-8

Read Temperature of MSP430 with the help of ADC.

AIM: To Read Temperature of MSP430 with the help of ADC.

Experiment Requirements: PC loaded code composer studio, MSP430 LAUNCHPAD

Procedure:

1. Open code composer studio
2. Open file go to new and select CCS project
3. A CCS window opens.

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USB1 (default)

   Give a project name.

   Select empty project with main.c and press finish.
4. Write C code in main.c.
5. Select build project and build your program. It will check for errors.When it is error free go to next step, otherwise repeat until the program is error free.
6. Go to target configuration

   Select user defined

   Select new target

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.
7. Open New Target configuration

   Right click on new target configuration

   Click on Launch selected configuration

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USB1 (default)

   and Save.

   Open the Run menu

   Select connect project

   Again open Run

Load Project

Browse the project

select the project.out

select,save and ok.

8. Run the program.
9. Observe the output in the console window or on the board.

program:

```
#include <msp430g2353.h>

int temp = 0;
int main(void){
WDTCTL = WDTPW | WDTHOLD; //stop the watchdog timer


//Select 1.5 V, 64 clock cycles, enable ADC interrupt, Turn on the reference generator
ADC10CTL0 = SREF_1 + REFON + ADC10ON + ADC10SHT_3 + ADC10IE;

//Select input channel 10 and divide the clock frequency by 4
ADC10CTL1 = INCH_10 + ADC10DIV_3;

//Enable and Start conversion
ADC10CTL0 |= ENC + ADC10SC;

//Enter low power mode
__bis_SR_register(LPM0_bits + GIE);

//fetch the temperature value from ADC10MEM register
temp = ADC10MEM;

//convert it into degree celsius
temp = ((temp * 27069L - 18169625L)>>16);

return 0;
}

//ISR
#pragma vector = ADC10_VECTOR
__interrupt void adc_interrupt(void)

{
__bic_SR_register_on_exit(CPUOFF);
}
```

RESULT: Hence read Temperature of MSP430 with the help of ADC

EXPERIMENT-9

Interrupts Programming Example Using GPIO

AIM: To perform Interrupts Programming Example Using GPIO.

Experiment Requirements: PC loaded code composer studio, MSP430 LAUNCHPAD

PROCEDURE:

1. Open code composer studio
2. Open file go to new and select CCS project
3. A CCS window opens.

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Give a project name.

   Select empty project with main.c and finish.
4. Write C code in main.c.
5. Select build project and build your program.
6. Go to target configuration

   Select user define

   Select new target

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.
7. Open New Target configuration

   Right click on new target configuration

   Click on Launch selected configuration

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.
8. Open the Run menu

   Select connect project

   Again open Run

Load Project

Browse the project

select the project.out

select,save and ok.

9. Run the program.
10. Observe the output in the console window or on the board.

Program

```
#include <msp430g2353.h>

unsigned int wdtCounter = 0;
void main(void)
{
        WDTCTL = WDT_MDLY_32; // Set Watchdog Timer interval to ~32ms
        IE1 |= WDTIE; // Enable WDT interrupt
        P1DIR |= BIT0; // Set P1.0 to output direction
        P1OUT |= BIT0; // Turn on LED at 1.0
        P1IE |= BIT3; // enable P1.3 interrupt
        __enable_interrupt();

        for(;;)
        {

        }
}

// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
        if(wdtCounter == 249)
        {
                P1OUT = 0x00; // P1.0 turn off
                wdtCounter = 0;
                _BIS_SR(LPM3_bits + GIE); // Enter LPM3 w/interrupt enabled
        }
        else
        {
                wdtCounter++;
        }
}
```

RESULT: Hence performed Interrupts Programming Example Using GPIO

EXPERIMENT-10

Use of Comparator to Compare the Signal Threshold Level

AIM: Use Of Comparator To Compare The Signal Threshold Level.

Experimental Requirements : PC loaded code composer studio, MSP430 LAUNCHPAD

Procedure:

1. Open code composer studio
2. Open file go to new and select CCS project
3. A CCS window opens.

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Give a project name.

   Select empty project with main.c and finish.
4. Write C code in main.c.
5. Select build project and build your program.
6. Go to target configuration

   Select user define

   Select new target

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.
7. Open New Target configuration

   Right click on new target configuration

   Click on Launch selected configuration

   Select MSP430G2253 in the target.

   Establish the connection by selecting the TI MSP430 USBI

   Save.
8. Open the Run menu

   Select connect project

   Again open Run

Load Project

Browse the project

select the project.out

select,save and ok.

9. Run the program.

Observe the output in the console window or on the board

Program

LED ON Case:

```
#include <msp430g2353.h>
int main (void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
P1DIR |= 0x01; // P1.0 output
CACTL1 = CARSEL + CAREF0 + CAON; // 0.25 Vcc = -comp, on
CACTL2 = P2CA4; // P1.1/CA1 = +comp
while (1) // Test comparator_A output
{
if ((CAOUT & CACTL2))
P1OUT |= 0x01; // if CAOUT set, set P1.0
else P1OUT &= ~0x01; // else reset
}
}
```

LED OFF Case:

```
#include <msp430g2353.h>
int main (void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
P1DIR |= 0x01; // P1.0 output
CACTL1 = CARSEL + CAREF0 + CAON; // 0.25 Vcc = -comp, on
CACTL2 = ~P2CA4; // P1.1/CA1 = -comp
while (1) // Test comparator_A output
{
if ((CAOUT & CACTL2))
P1OUT |= 0x01; // if CAOUT set, set P1.0
else P1OUT &= ~0x01; // else reset
}
}
```

Result: Hence used Comparator To Compare The Signal Threshold Level.

EXPERIMENT-11

AVERAGE OF N NUMBERS

AIM : To perform average for a given series using TASM.

Experimental Requirements: PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd...   or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with   tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

Program:

1. AVERAGE OF N NUMBERS ((1+2+3+4+…N)/N

```
ASSUME CS:CODE
CODE SEGMENT
START:
MOV AX,0000H
MOV BL,08H
MOV CL,BL
L1: ADD AL,CL
ADC AH,00H
LOOP L1
DIV BL
INT 03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
BL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations

2. AVERAGE OF N NUMBERS IN AN ARRAY

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DB 12H,23H,45H,56H,70H
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
MOV DS,AX
MOV AX,0000H
MOV BL,05H
MOV CL,BL
MOV SI,OFFSET LIST
L1: ADD AL,[SI]
ADC AH,00H
```

```
INC SI
LOOP L1
DIV BL
INT 03H
CODE ENDS
END START
END
```

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
ARR1:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations

Result: Average for a given series was found.

EXPERIMENT-12

Conversion of Packed BCD to unpacked BCD and BCD to ASCII

AIM : To convert packed BCD to unpacked BCD and BCD to ASCII using TASM.

Experimental Requirements: PC loaded with TASM software

Procedure:

1. Switch on the PC, press windows+R then enter CMD.
2. Find the folder where TASM is located. check whether TASM.EXE, TLINK.EXE, TD.EXE are present or not
3. Enter into the directory where TASM is located by using cd... or directory name:
4. Type cd tasm in which the three files are present .Now we will be getting into c: \ or d:\ with tasm directory.
5. Type edit then a new window will be opened in which the program is entered.
6. After entering the program save the file with <filename.asm>.
7. Check for the errors or warnings by using TASM <filename> and press enter...
8. If there are no errors, then type TLINK <filename> to compile the file. If errors go back to the edit and do the necessary corrections and repeat the previous step.
9. Next type td <filename > to debug the executable file then will be getting the message program has no symbol table, press ok and then write down the instructions, registers and flags status before execution .
10. For step by step execution press F8.and for direct execution press F9 and then write down the instructions, registers and flags status after execution .Go to dump if required for noting down the required inputs and outputs.

Program:

1.PACKED BCD TO UNPACKED BCD

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

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:

AL:

OUTPUT:

AX:

FLAG STATUS:

Theoretical Calculations

2. BCD TO ASCII

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

ADDRESS	OPCODE	MNEMONIC	OPERAND	COMMENTS

Result:
INPUT:
AL:
OUTPUT:
AX:
FLAG STATUS:
Theoretical Calculations

Result: packed BCD to unpacked BCD, BCD to ASCII conversion has been Performed.