INTRODUCTION TO MASM/TASM

ASSEMBLY LANGUAGE PROGRAMMING USING MASM SOFTWARE:

This software used to write a program (8086, Pentium processors etc.)

The programs are written using assembly language in editor then compile it. The complier converts assembly language statements into machine language statements/checks for errors. Then execute the compiled program.

There are different softwares developed by different companies for assembly language programming .They are

- **MASM** - Microsoft Company.
- **TASM** - Bore Land Company.

MERITS OF MASM:

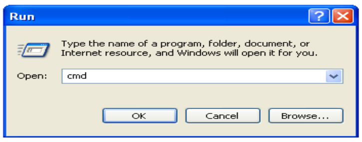
1. produces binary code
2. Referring data items by their names rather than by their address.

HOW TO ENTER INTO MASM EDITOR:

→ Click "**Start**" on the desktop.

→ Then select **Run**

→ Then it Shows inbox



→ Then type Command (CMD) which enters you into **DOS prompt**

→ Path setting

Suppose it display path as **C:\ DOCUME-\ADMIN>**

Then type **CD**

i.e.; **C:\DOCUME\\ADMIN>CD**

Then the path is **C :\>**

Then type **CD MASM**
Then the path is **C: MASM>**

Then type edit i.e.; **C: MASM>edit**

Then you enter into **MASM** text editor.

Then enter to **FILE** and select **NEW**.

And name it and then write the **ALP** (Assembly Language Program) in this editor.

After that save it as **filename's**

Then exit from the editor and go to prompt.

Then type MASM filename.ASM

I.e. C: MASM>MASM filename.ASM **or C: MASM filename.ASM, , ;**

Then link this file using **C: MASM>LINK filename.OBJ**

or **C: MASM>LINK filename.OBJ , , ;**

i.e link the program in assembly with **DOS**

then debug to create exe file

C:MASM>debug filename. EXE

Then it display **"--"** on the screen

After that type **'R'** displays the registers contents and starting step of the program.

'T' Tracing at contents of program step by step.

Suppose you need to go for break point debugging. Then type that instruction no where you need to check your register. For example T_{10} it will display the contents of register after executing 10 instructions.

DEBUG:

This command utility enables to write and modify simple assembly language programs in an easy fashion. It provides away to run and test any program in a controlled environment.

We can change any part of the program and immediately execute the program with an having to resemble it. We can also run machine language(Object files) directly by using DEBUG

DEBUG COMMANDS:

ASSEMBLE A [address] ; Assembly the instructions at a particular address

COMPARE C range address ; Compare two memory ranges

DUMP D [range] ; Display contents of memory

ENTER E address [list] ; Enter new or modifies memory contents beginning at specific Location

FILL F range list ; Fill in a range of memory

GO G [=address] [addresses] ; Execute a program in memory

HEX H value1 value2 ; Add and subtract two Hex values

INPUT I port

LOAD L [address] [drive] [first sector] [number]

MOVE M range address

NAME N [pathname] [arg list]

OUTPUT O port byte

PROCEED P [=address] [number]

QUIT Q

REGISTER R [register]

SEARCH S range list

TRACE T [=address] [value]

UNASSEMBLE U [range]

WRITE W [address] [drive] [first sector] [number]

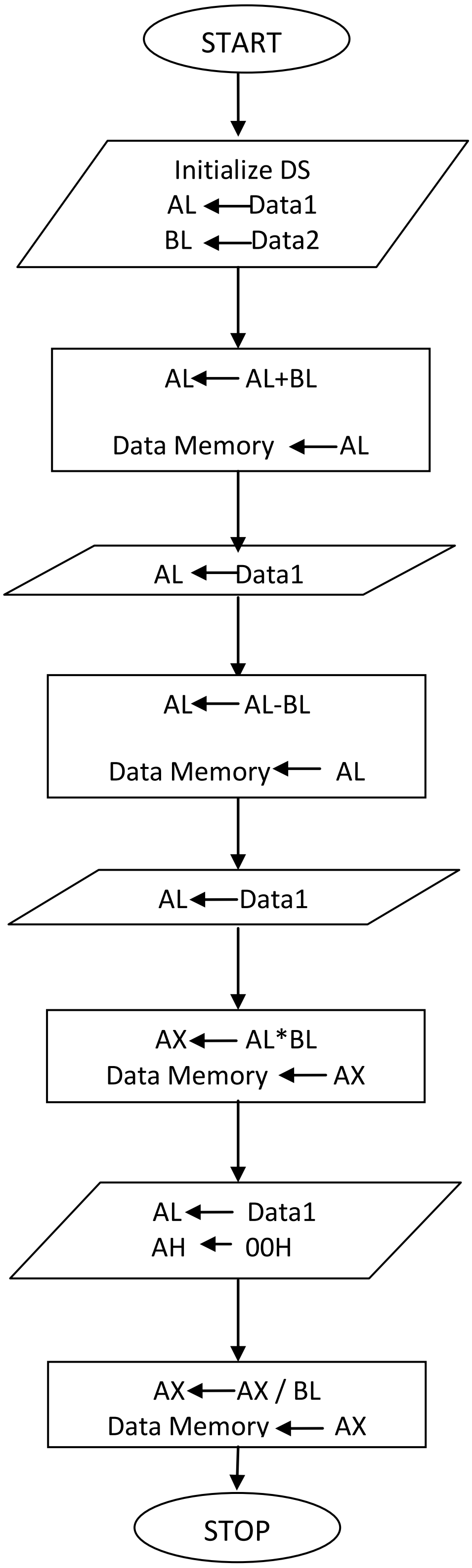
ALLOCATE expanded memory XA [#pages]

DEALLOCATE expanded memory XD [handle]

MAP expanded memory pages XM [Lpage] [Ppage] [handle]

DISPLAY expanded memory status XS

FLOW CHART:


START
Initialize DS
AL ← Data1
BL ← Data2
AL ← AL+BL
Data Memory ← AL
AL ← Data1
AL ← AL-BL
Data Memory ← AL
AL ← Data1
AX ← AL*BL
Data Memory ← AX
AL ← Data1
AH ← 00H
AX ← AX / BL
Data Memory ← AX
STOP

Exp No:
Date:

ARITHMETIC OPERATIONS ON 8-BIT DATA

ABSTRACT: Assembly language program to perform all arithmetic operations on 8-bit data
PORTS USED: None
REGISTERS USED: AX, BL
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load the given data to registers AL& BL
Step4: Perform addition and Store the result
Step 5: Repeat step 3
Step6: Perform subtraction and Store the result
Step7: Repeat step 3
Step8: Perform multiplication and Store the result
Step9: Repeat step 3
Step10: Perform division and Store the result
Step11: stop.

MANUAL CALCULATIONS:

PROGRAM:

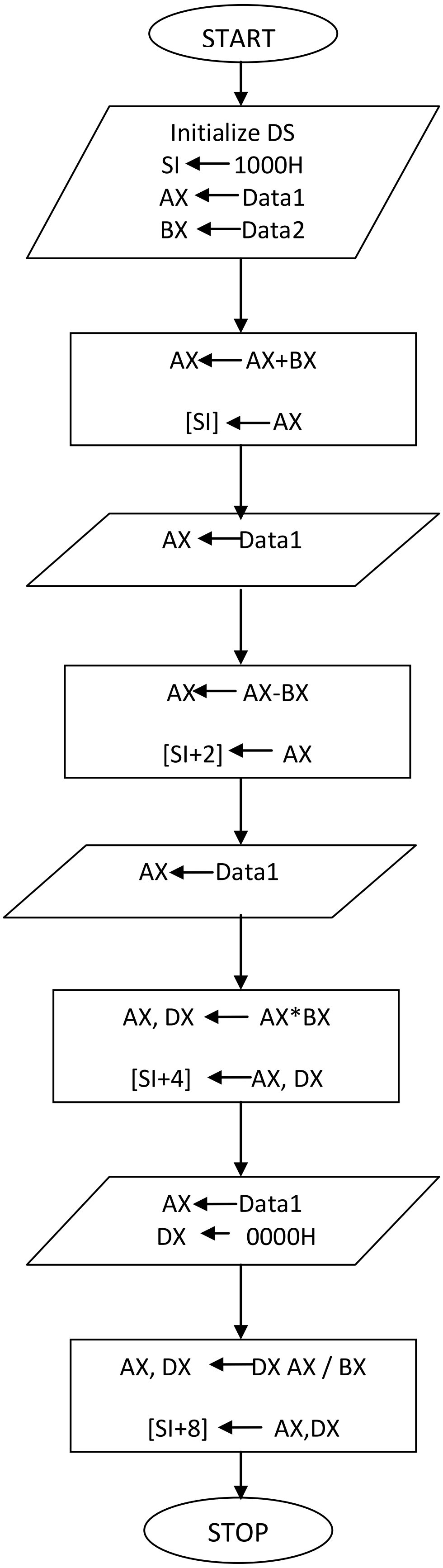
```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
N1 EQU  04H
N2 EQU  06H
RESULT DB  06H    DUP (00)
DATA ENDS
CODE SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV  AL, N1
        MOV  BL, N2
        ADD  AL, BL
        MOV  [RESULT], AL
        MOV  AL, N1
        SUB  AL, BL
        MOV  [RESULT+1], AL
        MOV  AL, N1
        MUL          BL
        MOV  [RESULT+2], AL
        MOV  [RESULT+3], AH
        MOV  AL, N1
        MOV  AH, 00H
        DIV   BL
        MOV  [RESULT+4], AL
        MOV  [RESULT+5], AH
        MOV  AH, 4CH
        INT  21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
SI ← 1000H
AX ← Data1
BX ← Data2
AX ← AX+BX
[SI] ← AX
AX ← Data1
AX ← AX-BX
[SI+2] ← AX
AX ← Data1
AX, DX ← AX*BX
[SI+4] ← AX, DX
AX ← Data1
DX ← 0000H
AX, DX ← DX AX / BX
[SI+8] ← AX,DX
STOP

Exp No:
Date:

ARITHMETIC OPERATIONS ON 16-BIT DATA

ABSTRACT: Assembly language program to perform all arithmetic operations on 16bit data
PORTS USED: None
REGISTERS USED: AX, BX, SI
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Initialize SI with some memory location
Step4: Load the given data to registers AX & BX
Step5: Perform addition and Store the result
Step6: Repeat step 4
Step7: Perform subtraction and Store the result
Step8: Repeat step 4
Step9: Perform multiplication and Store the result
Step10: Repeat step 4
Step11: Perform division and Store the result
Step12: Stop

MANUAL CALCULATIONS:

PROGRAM:

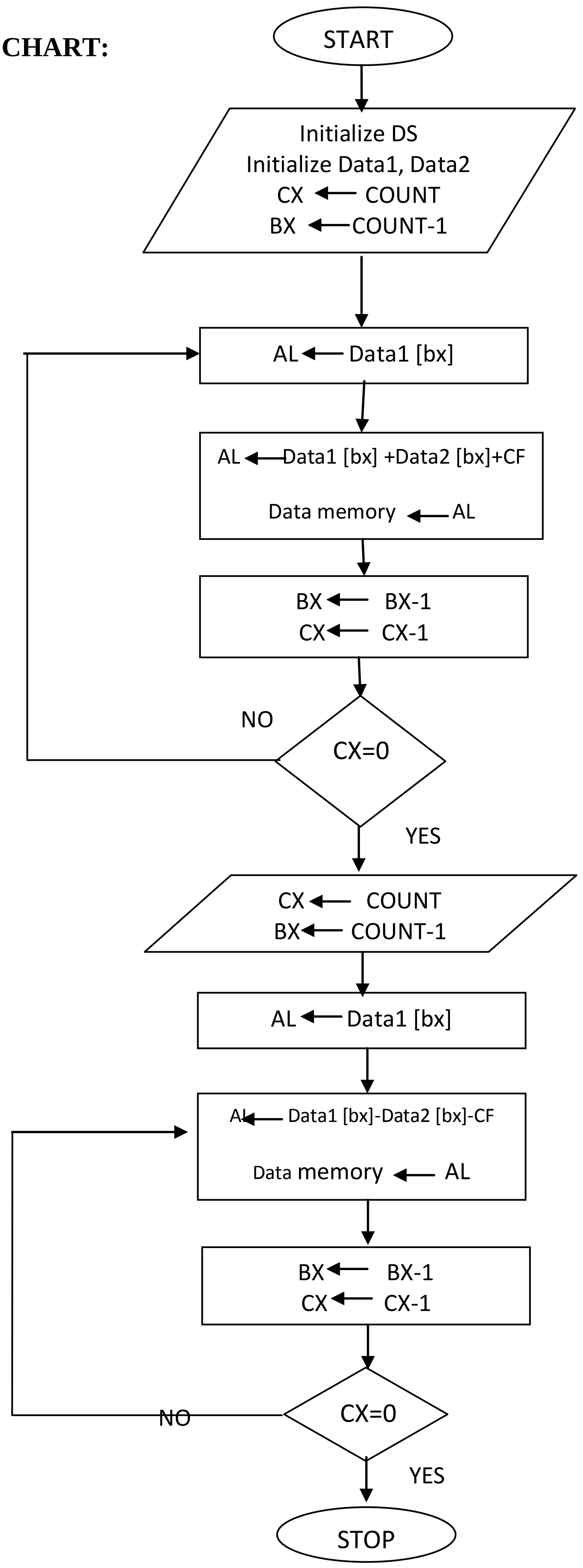
```
ASSUME      CS: CODE,    DS: DATA
DATA  SEGMENT
N1 EQU   8888H
N2 EQU   4444H
DATA ENDS
CODE SEGMENT
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  SI, 5000H
       MOV  AX, N1
       MOV  BX, N2
       ADD   AX, BX
       MOV  [SI], AX
       MOV  AX, N1
       SUB  AX, BX
       MOV  [SI+2], AX
       MOV  AX, N1
       MUL  BX
       MOV  [SI+4], AX
       MOV  [SI+6], DX
       MOV  AX, N1
       MOV  DX, 0000
       DIV  BX
       MOV  [SI+8], AX
       MOV  [SI+0AH], DX
       MOV  AH, 4CH
       INT  21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
Initialize Data1, Data2
CX ← COUNT
BX ← COUNT-1
AL ← Data1 [bx]
AL ← Data1 [bx] +Data2 [bx]+CF
Data memory ← AL
BX ← BX-1
CX ← CX-1
NO
CX=0
YES
CX ← COUNT
BX ← COUNT-1
AL ← Data1 [bx]
AL ← Data1 [bx]-Data2 [bx]-CF
Data memory ← AL
BX ← BX-1
CX ← CX-1
CX=0
NO
YES
STOP

Exp No:

Date:

MULTIBYTE ADDITIONS AND SUBTRACTION

ABSTRACT: Assembly language program to perform multibyte addition and subtraction
PORT USED: None
REGISTERS USED: AL, BX, CX
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load CX register with count
Step4: Load BX register with No. of bytes
Step5: Copy the contents from the memory location n1 [BX] to AL
Step6: Perform addition with second number n2 [BX]
Step7: Store the result to the memory location sum [BX]
Step8: Decrement BX
Step9: Decrement CX, if CX not equal to Zero jump to step5
Step10: Load CX register with count
Step11: Load BX register with no: of bytes
Step12: Store the contents from memory location n1 [BX] to AL
Step13: Perform subtraction with second number n2 [BX]
Step14: Store the result to the memory location sum [BX]
Step15: Decrement BX
Step16: Decrement CX, if CX not equal to Zero jump to step12
Step17: Stop

MANUAL CALCULATIONS:

PROGRAM:

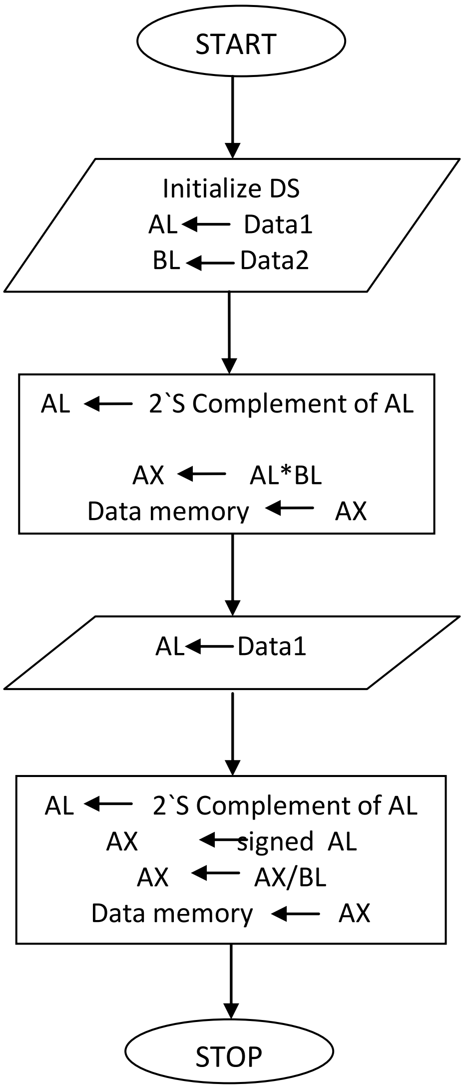
```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
N1  DB          33H, 33H, 33H
N2  DB          11H, 11H, 11H
COUNT           EQU  0003H
SUM   DB  03H  DUP (00)
DIFF   DB  03H  DUP (00)
DATA     ENDS
CODE SEGMENT
ORG 1000H
START:          MOV  AX, DATA
                MOV  DS, AX
                MOV  CX, COUNT
                MOV  BX, 0002H
                CLC
BACK:           MOV  AL, N1 [BX]
                ADC  AL, N2 [BX]
                MOV  SUM [BX], AL
                DEC  BX
                LOOP  BACK
                MOV  CX, COUNT
                MOV  BX, 0002H
                CLC
BACK1:          MOV  AL, N1 [BX]
                SBB   AL, N2 [BX]
                MOV   DIFF [BX], AL
                DEC   BX
                LOOP  BACK1
                MOV  AH, 4CH
                INT  21H
   CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
AL ← Data1
BL ← Data2
AL ← 2`S Complement of AL
AX ← AL*BL
Data memory ← AX
AL ← Data1
AL ← 2`S Complement of AL
AX ← signed AL
AX ← AX/BL
Data memory ← AX
STOP

Exp No:
Date:

SIGNED OPERATIONS ON 8 -BIT DATA

ABSTRACT: Assembly language program to perform signed operations
PORT USED: None
REGISTERS USED: AL, BL
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load AL with first number
Step4: Do 2's compliment of AL
Step5: Load BL with second number
Step6: Perform signed Multiplication
Step7: Store the result in data memory
Step8: Load AL with first number
Step9: Repeat step 4
Step10: Convert AL to AX
Step11: Perform signed division
Step12: Store the result in data memory
Step13: Stop

MANUAL CALCULATIONS:

PROGRAM:

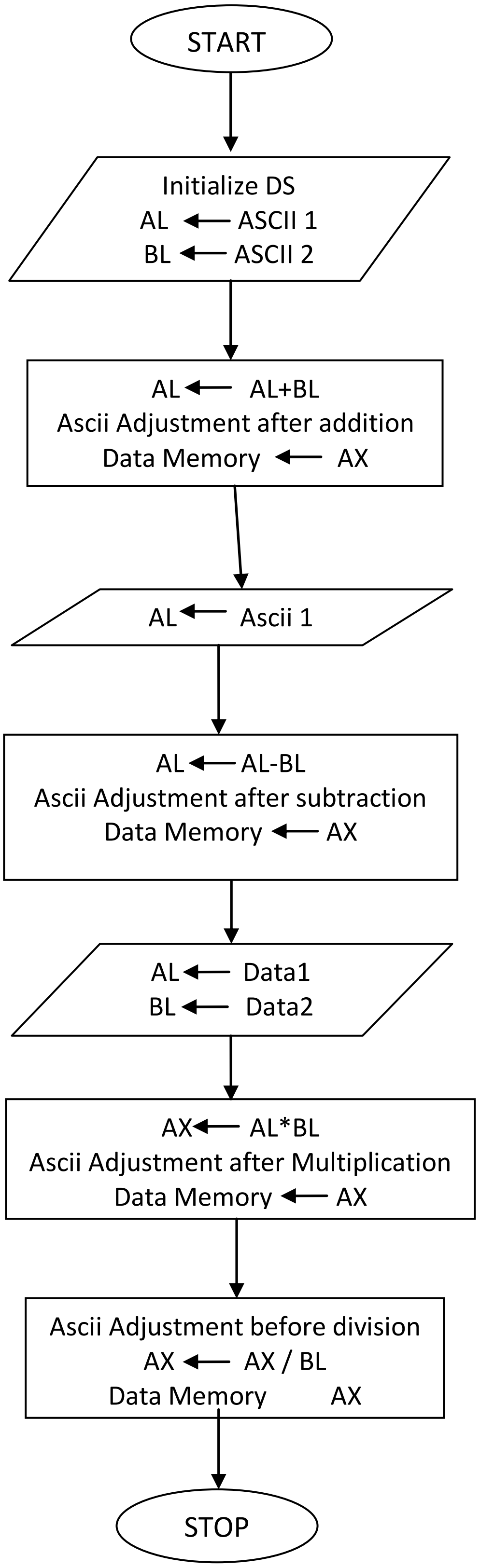
```
ASSUME CS: CODE , DS: DATA
DATA SEGMENT
N1  DB  08H
N2  DB  04H
RESULT  DW  02 DUP (00)
DATA ENDS
CODE SEGMENT
START:      MOV  AX, DATA
            MOV   DS, AX
            MOV   AL, N1
            NEG    AL
            MOV   BL, N2
            IMUL  BL
            MOV   [RESULT], AX
            MOV   AL, N1
            NEG    AL
            CBW
            IDIV   BL
            MOV  [RESULT+2], AX
            MOV   AH, 4CH
            INT  21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
AL ← ASCII 1
BL ← ASCII 2
AL ← AL+BL
Ascii Adjustment after addition
Data Memory ← AX
AL ← Ascii 1
AL ← AL-BL
Ascii Adjustment after subtraction
Data Memory ← AX
AL ← Data1
BL ← Data2
AX ← AL*BL
Ascii Adjustment after Multiplication
Data Memory ← AX
Ascii Adjustment before division
AX ← AX / BL
Data Memory AX
STOP

Exp No:
Date:

ASCII ARITHMETIC OPERATIONS

ABSTRACT: Assembly language program to perform ASCII arithmetic operations
PORT USED: None
REGISTERS USED: AL, BL, SI
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load SI with Memory location
Step4: Load AL with first number in ASCII form
Step5: Load BL with Second number in ASCII form
Step6: Perform addition
Step7: Perform ASCII adjustment after addition
Step8: Store the result to the data memory
Step9: Load AL with first number in ASCII form
Step10: Perform subtraction
Step11: Perform ASCII adjustment after subtraction
Step12: Store the result to the data memory
Step13: Load AL with first number
Step14: Perform multiplication
Step15: Perform ASCII adjustment after multiplication
Step16: Store the result to the data memory
Step17: Load AL with first number
Step18: Perform ASCII adjustment before division
Step19: Perform division
Step20: Store the result to the data memory
Step21: Stop

MANUAL CALCULATIONS:

PROGRAM:

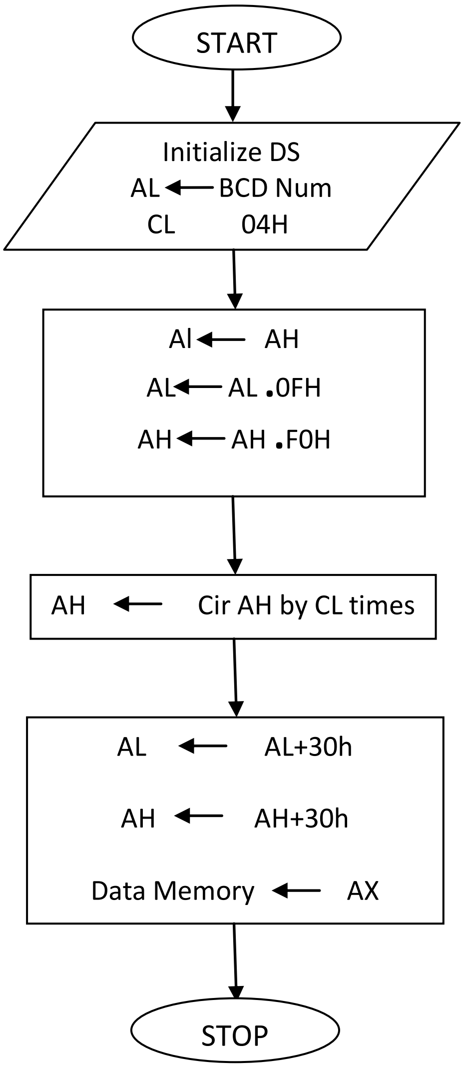
```
ASSUME        CS: CODE, DS: DATA
DATA SEGMENT
N1 DB '8'
N2 DB '4'
DATA ENDS
CODE SEGMENT
ORG 1000H
START:
        MOV AX, DATA
        MOV DS, AX
        MOV SI, 5000H
        XOR AX, AX
        MOV AL,N1
        MOV BL,N2
        ADD AL ,BL
        AAA
        MOV[ SI], AX
        MOV AL, N1
        SUB AL, BL
        AAS
        MOV [SI+2], AX
        MOV AL, 08H
        MOV BL, 04H
        MUL BL
        AAM
        MOV [SI+4], AX
        AAD
        DIV BL
        MOV [SI+6], AX
        MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
AL ← BCD Num
CL 04H
Al ← AH
AL ← AL .0FH
AH ← AH .F0H
AH ← Cir AH by CL times
AL ← AL+30h
AH ← AH+30h
Data Memory ← AX
STOP

Exp No:

Date:

BCD TO ASCII CONVERSION

ABSTRACT: Assembly language program to convert BCD number to ASCII number
PORT USED: None
REGISTERS USED: AL, AH, CX
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load AL with BCD number
Step4: Copy the contents from AL to AH
Step5: Perform AND operation on AL with 0Fh
Step6: Perform AND operation on AL with F0h
Step7: Rotate the AH contents by four times
Step8: Perform OR operation on AL with 30h
Step10: Perform OR operation on AH with 30h
Step11: Store the result to the memory location
Step12: Stop

MANUAL CALCULATIONS:

PROGRAM:

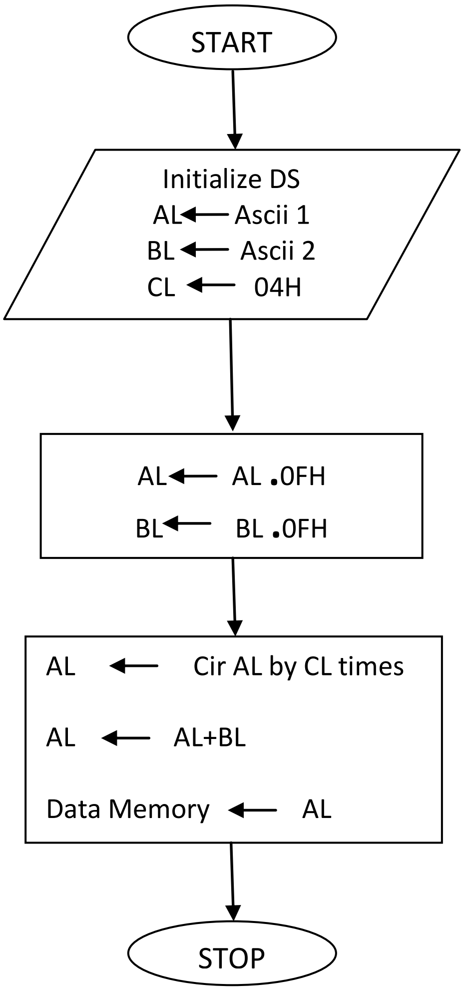
```
ASSUME  CS: CODE,DS: DATA
DATA SEGMENT
BCD   DB 17H
ASCII   DW  ?
DATA ENDS
CODE SEGMENT
ORG 1000H
START: MOV AX, DATA
       MOV DS, AX
       MOV AL, BCD
       MOV CL, 04
       MOV AH, AL
       AND AL, 0FH
       AND AH, 0F0H
       ROR AH, CL
       OR AL, 30H
       OR AH, 30H
       MOV ASCII, AX
       MOV AH, 4CH
       INT 21H
  CODE ENDS
  END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
AL ← Ascii 1
BL ← Ascii 2
CL ← 04H
AL ← AL . 0FH
BL ← BL . 0FH
AL ← Cir AL by CL times
AL ← AL+BL
Data Memory ← AL
STOP

Exp No:
Date:

ASCII TO BCD CONVERSION

ABSTRACT: Assembly language program convert ASCII number to BCD number
PORT USED: None
REGISTERS USED: AL, BL, CX
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load AL with ASCII number
Step4: Copy the contents from AL to BL
Step5: Perform AND operation on AL with 0Fh
Step6: Perform AND operation on BL with 0Fh
Step7: Rotate the AL contents by four times
Step8: Perform OR operation on AL with BL
Step9: Stop

MANUAL CALCULATIONS:

PROGRAM:

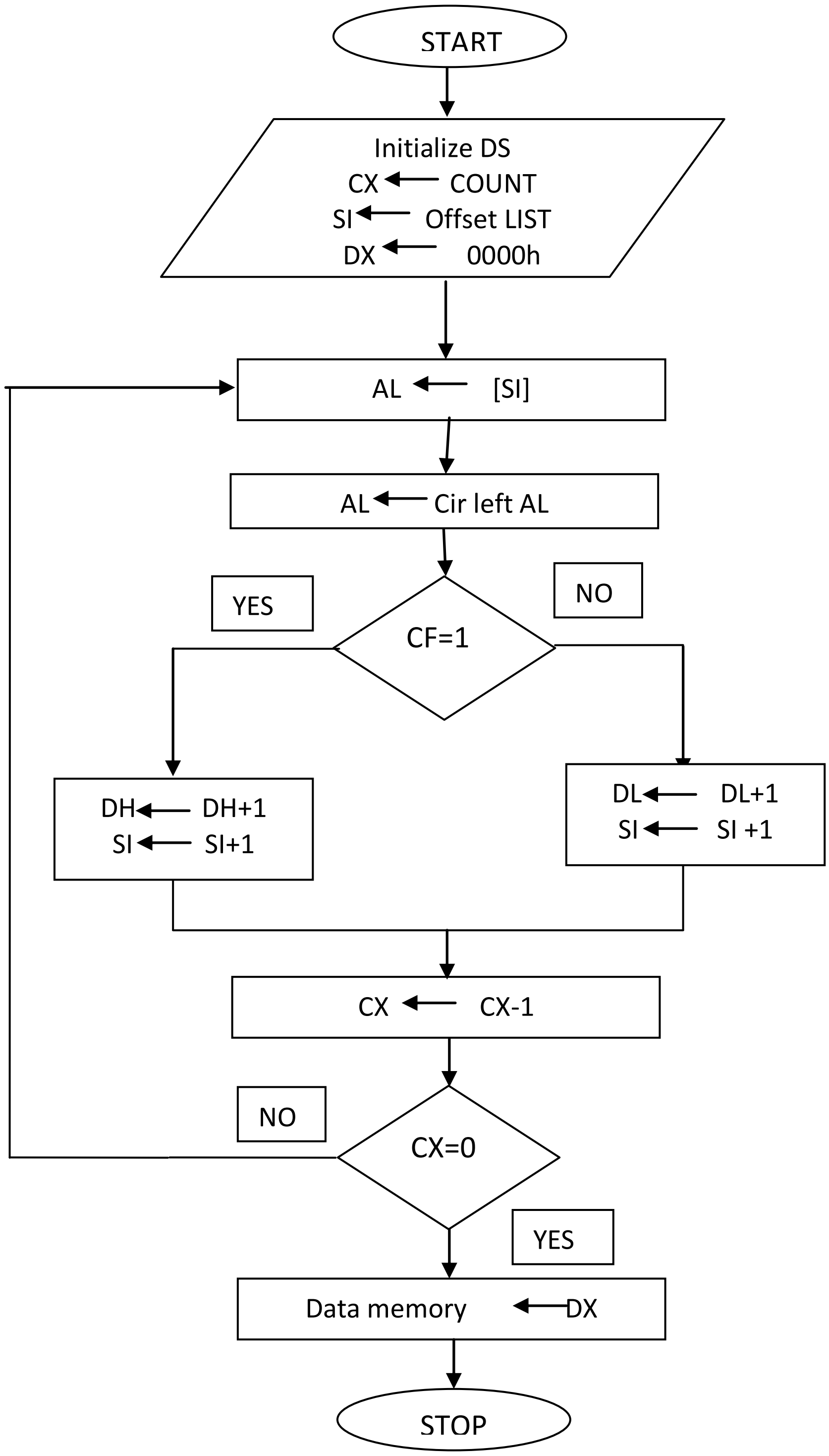
```
ASSUME   CS: CODE, DS: DATA
DATA  SEGMENT
ASCII1    DB    '1'
ASCII2   DB    '7'
BCD DB  ?
DATA ENDS
CODE SEGMENT
ORG 1000H
START:MOV AX, DATA
       MOV DS, AX
       MOV CL, 04H
       MOV AL, ASCII1
       MOV BL, ASCII2
      AND AL, 0FH
      AND BL, 0FH
      ROR AL, CL
      OR AL, BL
      MOV BCD, AL
      MOV AH, 4CH
      INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:



START
Initialize DS
CX ← COUNT
SI ← Offset LIST
DX ← 0000h
AL ← [SI]
AL ← Cir left AL
YES
CF=1
NO
DH ← DH+1
SI ← SI+1
DL ← DL+1
SI ← SI +1
CX ← CX-1
NO
CX=0
YES
Data memory ← DX
STOP

Exp No:
Date:

POSITIVE AND NEGATIVE COUNT IN AN ARRAY NUMBERS

ABSTRACT: Assembly language program to count number of positive and negative numbers
PORT USED: None
REGISTERS USED: SI, DX, CX, AL
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load CX register with count value
Step4: Initialize DX with 0000h
Step5: Load SI with offset list
Step6: Copy the contents from memory location SI to AL
Step7: Rotate left the content of AL
Step8: Jump to step13 if carry
Step9: Increment DL
Step10: Increment SI
Step11: Decrement CX and jump to step6 if no zero
Step12: Jump to step16
Step13: Increment DH
Step14: Increment SI
Step15: Decrement CX and jump to step6 if no zero
Step16: Store the result to the data memory
Step17: Stop

MANUAL CALCULATIONS:

PROGRAM:

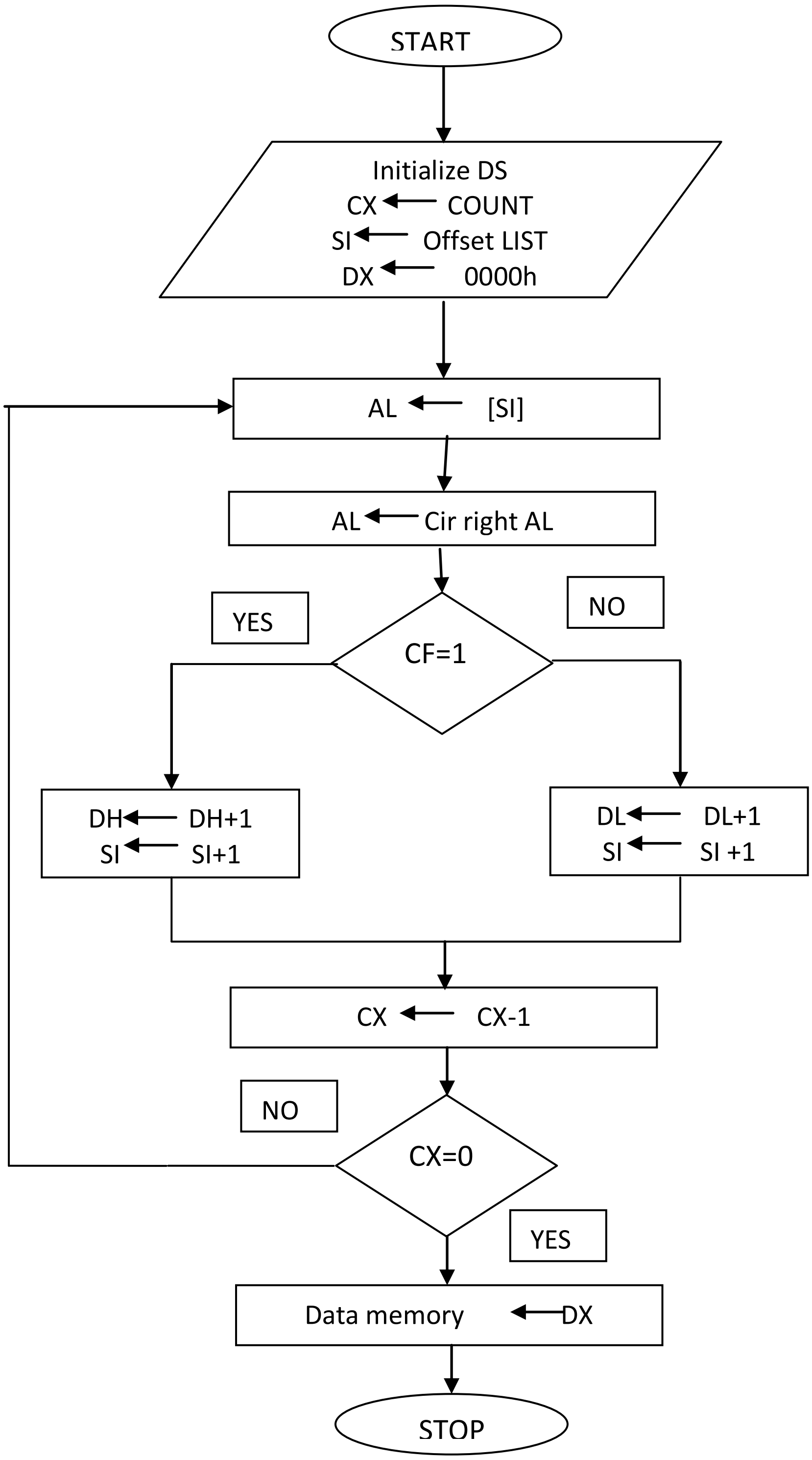
```
ASSUME  CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 0FFH, 0DDH, 04H, 05H, 98H
RESULT DW  ?
DATA ENDS
CODE SEGMENT
ORG 1000H
START:MOV AX, DATA
        MOV DS, AX
        LEA SI, LIST
        MOV CX, 0005H
        MOV DX, 0000H
BACK:MOV AL, [SI]
        ROL AL, 01H
        JC NEGATIVE
        INC DL
        INC SI
        LOOP BACK
        JMP EXIT
NEGATIVE:  INC DH
        INC SI
        LOOP BACK
EXIT: MOV [RESULT], DX
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
CX ← COUNT
SI ← Offset LIST
DX ← 0000h
AL ← [SI]
AL ← Cir right AL
YES
NO
CF=1
DH ← DH+1
SI ← SI+1
DL ← DL+1
SI ← SI +1
CX ← CX-1
NO
CX=0
YES
Data memory ← DX
STOP

Exp No:

Date:

ODD AND EVEN COUNT IN AN ARRAY NUMBERS

ABSTRACT: Assembly language program to count number of odd and even numbers
PORT USED: None
REGISTERS USED: AL, CX, DL, DH, SI
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load CX register with count
Step4: Initialize DX with 0000
Step5: Load SI with offset list
Step6: Copy the contents from memory location SI to AL
Step7: Rotate right the content of AL
Step8: Jump to step13 if carry
Step9: Increment DL
Step10: Increment SI
Step11: Decrement CX and jump to step6 if no zero
Step12: Jump to step16
Step13: Increment DH
Step14: Increment SI
Step15: Decrement CX and jump to step6 if no zero
Step16: Store the result to the data memory
Step17: Stop

MANUAL CALCULATIONS:

PROGRAM:

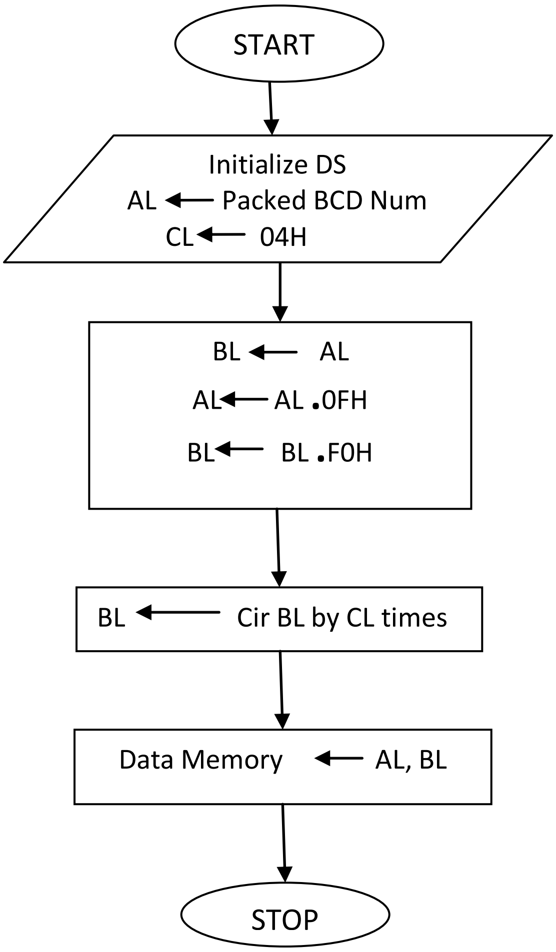
```
        ASSUME  CS: CODE, DS: DATA
        DATA SEGMENT
        LIST DB 05H,01H,03H,04H,08H,02H
        COUNT DW 0006H
        RESULT DW  ?
        DATA ENDS
        CODE SEGMENT
        ORG 1000H
        START:MOV AX, DATA
              MOV DS, AX
              MOV CX, COUNT
              MOV DX, 0000H
              MOV SI, OFFSET LIST
        BACK: MOV AL, [SI]
              ROR AL, 01H
              JC ODD
              INC DL
              INC SI
              LOOP BACK
              JMP EXIT
        ODD:  INC DH
              INC SI
              LOOP BACK
        EXIT: MOV [RESULT], DX
               MOV AH, 4CH
               INT 21H
       CODE ENDS
       END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
AL ← Packed BCD Num
CL ← 04H
BL ← AL
AL ← AL . 0FH
BL ← BL . F0H
BL ← Cir BL by CL times
Data Memory ← AL, BL
STOP

Exp No:
Date:

PACKED BCD TO UNPACKED BCD CONVERSION

ABSTRACT: Write a program to convert packed BCD number into Unpacked BCD number.
REGISTERS USED: AL, BL
PORTS USED: None.
ALOGARITHM:
Step1: Start

Step2: Initialize the data segment

Step3: Copy packed number into AL register

Step4: Copy packed number into BL register

Step5: Initialize the count CX with 04h

Step6: Perform AND operation on AL with 0Fh

Step7: Perform AND operation on BL with 0F0h

Step8: Rotate right without carry operation on BL by CL times

Step9: Move the result data memory

Step10: Stop

MANUAL CALCULATIONS:

PROGRAM:

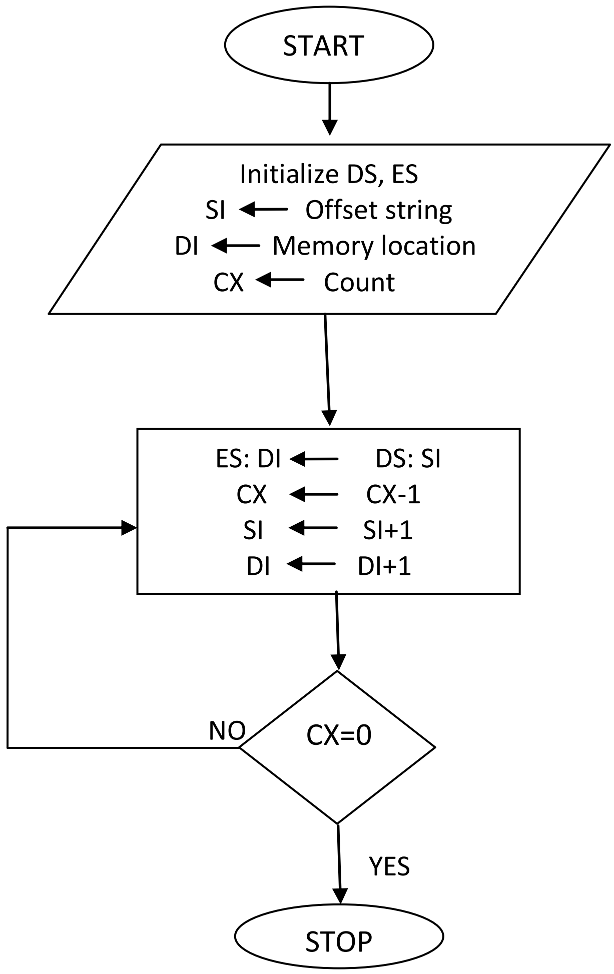
```
ASSUME    CS: CODE,    DS: DATA
DATA SEGMENT
N EQU 29H
RESULT DB 02H DUP (0)
DATA ENDS
CODE SEGMENT
ORG 2000h
START:MOV AX, DATA
        MOV DS, AX
        MOV AL, N
        MOV BL, N
        MOV CL, 04H
        AND AL, 0Fh
        AND BL, 0F0h
        ROR BL, CL
        MOV [RESULT], BL
        MOV [RESULT+1], AL
        MOV AH, 4Ch
        INT 21h
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS, ES
SI ← Offset string
DI ← Memory location
CX ← Count
ES: DI ← DS: SI
CX ← CX-1
SI ← SI+1
DI ← DI+1
NO
CX=0
YES
STOP

Exp No:
Date:

MOVE BLOCK OF DATA

ABSTRACT: Assembly language program to transfer a block of data.
PORT USED: None.
REGISTERS USED: AX, BL.
ALGORITHM:
Step1: Start

Step2: Initialize data segment & extra segment

Step3: Define the String

Step4: Load CX register with length of the String

Step5: Initialize DI with memory location

Step6: Load SI with offset list

Step7: Repeat the process of moving string byte from SI to DI until CX equals to zero

Step8: Stop

MANUAL CALCULATIONS:

PROGRAM:

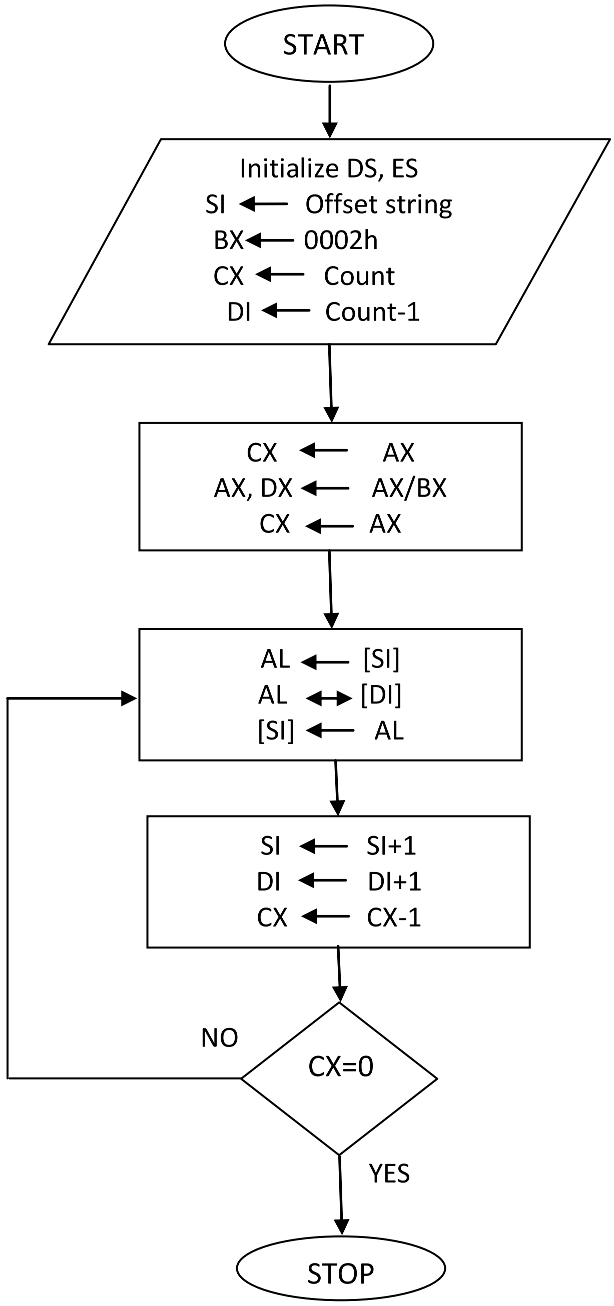
```
    ASSUME      CS: CODE, DS: DATA, ES: EXTRA
    DATA SEGMENT
    LIST DB 'ADITYA'
    COUNT EQU 06H
    DATA ENDS
    EXTRA SEGMENT
    EXTRA ENDS
    CODE SEGMENT
    ORG 1000H
    START: MOV AX, DATA
           MOV DS, AX
           MOV AX,EXTRA
           MOV ES, AX
           MOV CX, COUNT
           MOV DI, 5000H
           LEA  SI, LIST
           CLD
           REP   MOVSB
           MOV AH, 4CH
           INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOWCHART:



START
Initialize DS, ES
SI ← Offset string
BX← 0002h
CX ← Count
DI ← Count-1
CX ← AX
AX, DX ← AX/BX
CX ← AX
AL ← [SI]
AL ↔ [DI]
[SI] ← AL
SI ← SI+1
DI ← DI+1
CX ← CX-1
NO
CX=0
YES
STOP

Exp No:
Date:

REVERSAL OF GIVEN STRING

ABSTRACT: Assembly language program to reverse a given string
PORT USED: None
REGISTERS USED: AX, BL
ALGORITHM:
Step1: Start
Step2: Initialize data segment & extra segment
Step3: Load CX register with count
Step4: Copy the contents from CX to AX
Step5: Load SI with offset list
Step6: Initialize DI with (count-1)
Step7: Initialize BX with 02
Step8: Perform division with BX
Step9: Copy the contents from AX to CX
Step10: Move the contents from memory location SI to AL
Step11: Exchange the contents of AL with [DI]
Step12: Move the contents from memory location AL to SI
Step13: Increment SI
Step14: Decrement DI
Step15: Decrement CX and jump to step10 if no zero
Step16: Stop

MANUAL CALCULATIONS:

PROGRAM:

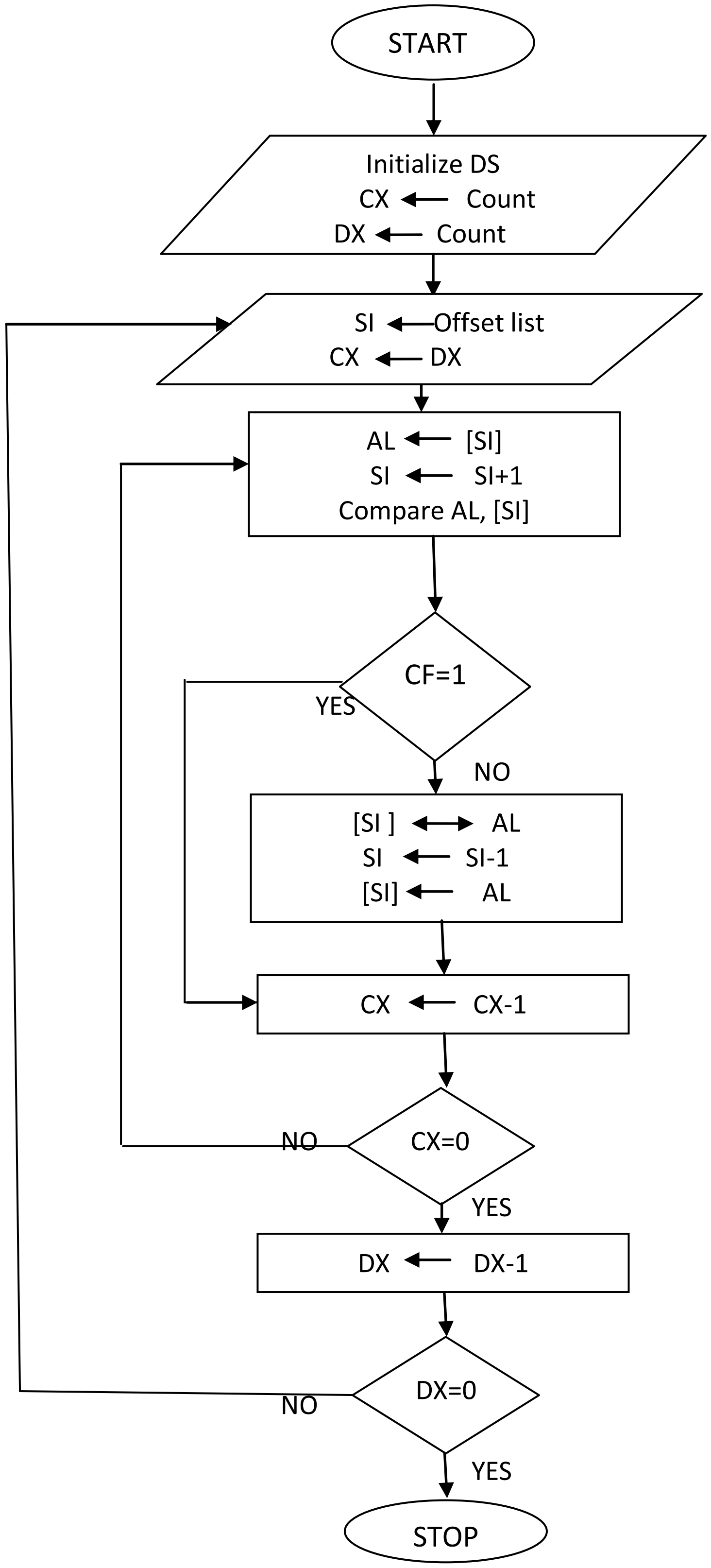
```
    ASSUME CS: CODE, DS: DATA
    DATA SEGMENT
    LIST DB 'MICRO PROCESSOR'
    COUNT EQU ($-LIST)
    DATA ENDS
    CODE SEGMENT
    ORG 1000H
    START: MOV AX, DATA
        MOV DS, AX
        MOV CX, COUNT
        MOV AX, CX
        MOV SI, OFFSET LIST
        MOV DI, (COUNT-1)
        MOV BX, 02
        DIV BX
        MOV CX, AX
    BACK:    MOV AL,[SI]
        XCHG AL,[DI]
        MOV [SI], AL
        INC SI
        DEC DI
        LOOP BACK
        MOV AH, 4CH
          INT 21H
    CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:



START
Initialize DS
CX ← Count
DX ← Count
SI ← Offset list
CX ← DX
AL ← [SI]
SI ← SI+1
Compare AL, [SI]
CF=1
YES
NO
[SI] ↔ AL
SI ← SI-1
[SI] ← AL
CX ← CX-1
NO
CX=0
YES
DX ← DX-1
NO
DX=0
YES
STOP

Exp No:
Date:

SORTING OF ‘N’ NUMBERS

ABSTRACT: Assembly language program to do sorting of numbers in a given series
PORT USED: None
REGISTERS USED: CX, DX, AL, SI
ALGORITHM:
Step1: Start
Step2: Initialize data segment
Step3: Load CX register with count
Step4: Copy the contents from CX to DX
Step5: Load SI with offset list
Step6: Copy the contents from DX to CX
Step7: Move the contents from memory location SI to AL
Step8: Increment SI
Step9: Compare AL contents with [SI]
Step10: Jump to step15 if carry
Step11: Exchange the contents of AL with [SI]
Step12: Decrement SI
Step13: Move the contents from AL to memory location SI
Step14: Increment SI
Step15: Decrement CX and jump to step7 if no zero
Step16: Decrement DX and jump to step5 if no zero
Step17: Stop

MANUAL CALCULATIONS:

PROGRAM:

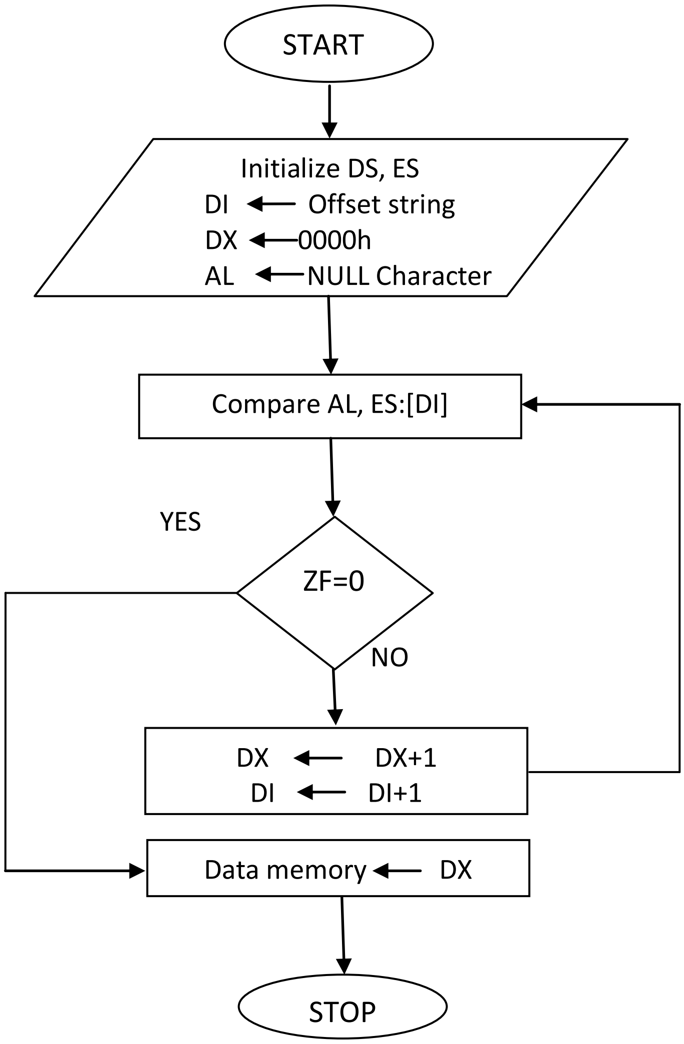
```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 56H, 12H, 72H,32H
COUNT EQU 0003H
DATA ENDS
CODE SEGMENT
ORG 1000H
START:  MOV AX, DATA
        MOV DS, AX
        MOV CX, COUNT
        MOV DX, CX
AGAIN:  MOV SI, OFFSET LIST
        MOV CX, DX
BACK:   MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC NEXT
        XCHG [SI], AL
        DEC SI
        MOV [SI], AL
        INC SI
NEXT:   LOOP BACK
        DEC DX
        JNZ  AGAIN
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS, ES
DI ← Offset string
DX ←0000h
AL ←NULL Character
Compare AL, ES:[DI]
YES
ZF=0
NO
DX ← DX+1
DI ← DI+1
Data memory ← DX
STOP

Exp No:
Date:

LENGTH OF THE GIVEN STRING

ABSTRACT: Assembly language program to find the Length of a string

PORT USED: None

REGISTERS USED: AX, BL

ALGORITHM:

Step1: Start

Step2: Initialize data segment & extra segment

Step3: Load AL with '$'

Step4: Load SI with offset list

Step5: Initialize DX with 0000

Step6: Scan string byte from DI memory location until AL =ES: DI

Step7: Jump to step10 if equal

Step8: Increment DX

Step9: Jump to step6

Step10: Store the result to the memory location

Step11: Stop

MANUAL CALCULATIONS:

PROGRAM:

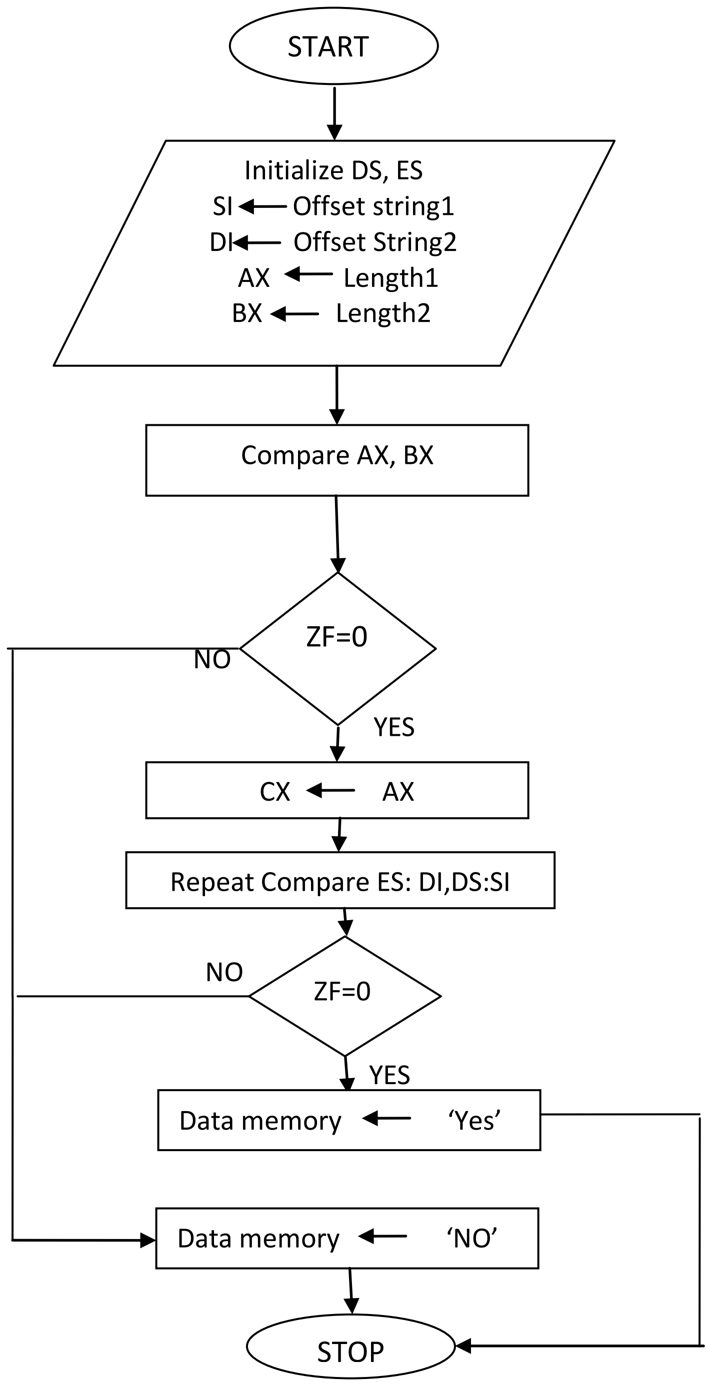
```
ASSUME   CS: CODE, ES: EXTRA
EXTRA  SEGMENT
LIST  DB  'ADITYA$'
LEN  DW  ?
EXTRA  ENDS
CODE SEGMENT
ORG 1000H
START: MOV AX, EXTRA
       MOV ES, AX
       MOV AL,'$'
       LEA DI, LIST
       MOV DX, 0000H
       CLD
BACK:  SCASB
       JE EXIT
       INC DX
       JMP BACK
EXIT:  MOV LEN, DX
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS, ES
SI ← Offset string1
DI ← Offset String2
AX ← Length1
BX ← Length2
Compare AX, BX
ZF=0
NO
YES
CX ← AX
Repeat Compare ES: DI,DS:SI
NO
ZF=0
YES
Data memory ← 'Yes'
Data memory ← 'NO'
STOP

Exp No:
Date:

COMPARISON OF TWO STRINGS

ABSTRACT: Assembly language program to compare two strings.
PORT USED: None
REGISTERS USED: AX, BL
ALGORITHM:
Step1: Start
Step2: Initialize data segment & extra segment
Step3: Load AX with length of String 1
Step4: Load BX with length of String 2
Step5: Compare AX with BX
Step6: Jump step14 if not equal
Step7: Copy the contents from AX to CX
Step8: Load SI with first location of string 1
Step9: Load DI with first location of string 2
Step10: Repeat comparing string byte until count equals to zero
Step11: jump to step 14 if not equal
Step12: Store the result to the data memory
Step13: Jump to step 15
Step14: Store another result to the data memory
Step15: Stop

MANUAL CALCULATIONS:

PROGRAM:

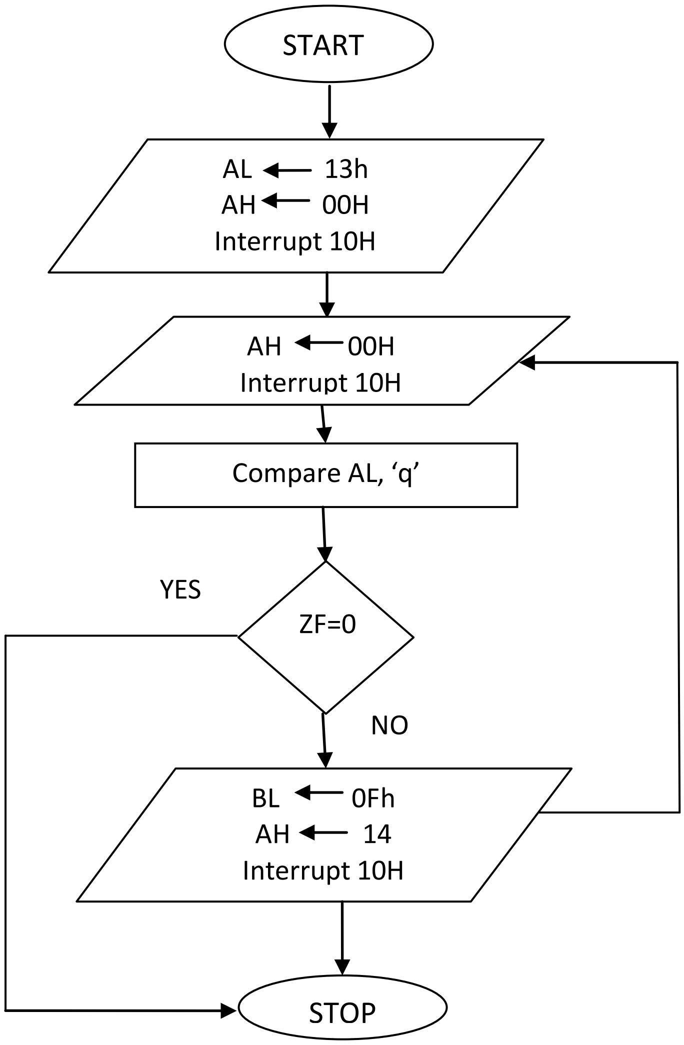
```
ASSUME CS: CODE, DS: DATA, ES: EXTRA
DATA SEGMENT
ORG 1000H
LIST1    DB  'ADITYA'
LEN1   EQU  ($-LIST1)
RESULT DW  ?
DATA ENDS
EXTRA SEGMENT
ORG 5000H
LIST2           DB  'ADITYA'
LEN2  EQU  ($-LIST2)
EXTRA ENDS
CODE SEGMENT
ORG 1000H
START: MOV AX, DATA
        MOV  DS, AX
        MOV    AX,EXTRA
        MOV  ES, AX
        MOV  AX.LEN1
        MOV  BX, LEN2
        CMP   AX, BX
        JNE    EXIT
        MOV  CX, AX
        MOV  SI, OFFSET LIST1
        MOV  DI, OFFSET LIST2
        CLD
        REP    CMPSB
        JNE    EXIT
        MOV  RESULT, 5555H
        JMP   NEXT
EXIT: MOV  RESULT, 0FFFFH
NEXT:          MOV  AH, 4CH
        INT     21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
AL ← 13h
AH ← 00H
Interrupt 10H
AH ← 00H
Interrupt 10H
Compare AL, ‘q’
YES
ZF=0
NO
BL ← 0Fh
AH ← 14
Interrupt 10H
STOP

Exp No:
Date:

READING KEYBOARD BUFFERED WITH ECHO

ABSTRACT: To Read the Keyboard Buffered with Echo.
REGISTERS USED: AH, AL, SI.
PORTS USED: None.
ALGORITHM:
Step1: Start.

Step2: Load the number 13h into AL register.

Step3: Initialize the AH register with 00h

Step4: Display interrupt

Step5: Initialize the AH register with 00h

Step6: Key board Interrupt

Step7: Compare the data in AL register with character ‘q’.

Step8: If equal to zero go to step 12.

Step9: Move the number 0Fh into BL register.

Step10: Move the number 14 into AH register.

Step11: Keyboard Interrupt.

Step12: Load the number 4C in AH register.

Step13: Stop.

PROGRAM:

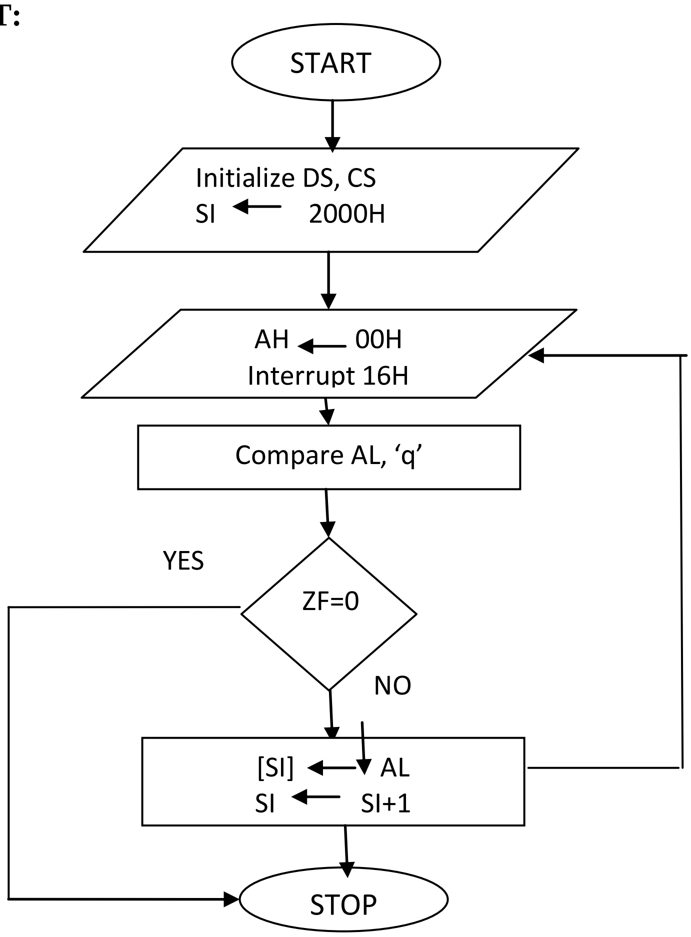
```
ASSUME CS: CODE
CODE SEGMENT
ORG 1000h
START: MOV AH, 00H
       MOV Al, 13H
       INT 10H
BACK:  MOV AH, 00h
       INT 16H
       CMP AL, ‘q’
       JE EXIT
       MOV BL, 0FH
       MOV AH, 14
       INT 10H
       JMP BACK
EXIT:  MOV AH, 4CH
       INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS, CS
SI ← 2000H
AH ← 00H
Interrupt 16H
Compare AL, 'q'
YES
ZF=0
NO
[SI] ← AL
SI ← SI+1
STOP

Exp No:

Date:

READING KEYBOARD BUFFERED WITHOUT ECHO

ABSTRACT: To read the string or character from keyboard without ECHO by using BIOS Commands.

REGISTERS USED: AH, AL, SI

PORTS USED: None.

ALGORITHM:

Step1: Start

Step2: Initialize SI with Offset Result.

Step3: Initialize AH with 00h

Step4: Keyboard Interrupt

Step5: Compare AL with character q.

Step6: Copy the contents AL into SI register.

Step7: If equal to zero go to step 10

Step8: Increment SI.

Step9: Go to step 3 without condition.

Step10: Terminate the program.

Step11: Stop.

PROGRAM:

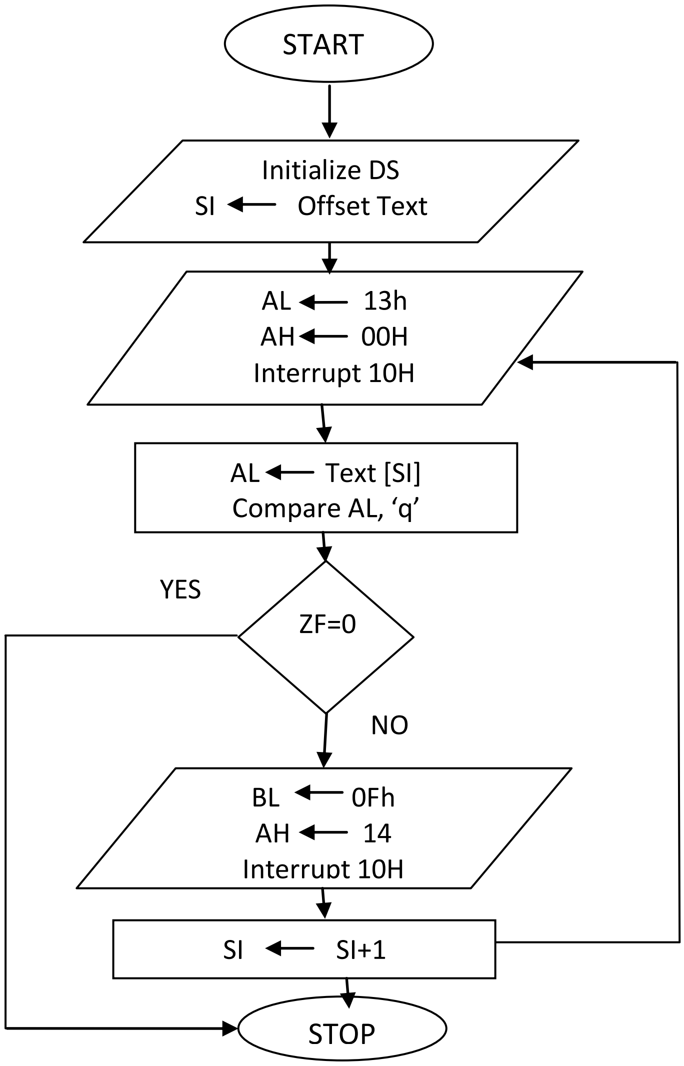
```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
ORG 3000h
RESULT DB 50h DUP (0)
DATA ENDS
CODE SEGMENT
ORG 1000h
START: MOV SI, OFFSET RESULT
BACK:  MOV AH, 00h
       INT 16h
       CMP AL, ‘q’
       MOV [SI], AL
       JE EXIT
       INC SI
       JMP BACK
EXIT:  MOV AH, 4Ch
       INT 21h
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
Initialize DS
SI ← Offset Text
AL ← 13h
AH ← 00H
Interrupt 10H
AL ← Text [SI]
Compare AL, 'q'
YES
ZF=0
NO
BL ← 0Fh
AH ← 14
Interrupt 10H
SI ← SI+1
STOP

Exp No:
Date:

STRING DISPLAY

ABSTRACT: To display the string character by using BIOS commands.
REGISTER USED: AL, AH, SI.
PORTS USED: None
ALGORITHM:
Step1: Start
Step2: Set the screen in Graphic mode
Step3: Initialize AH with 00h
Step4: Set the keyboard display mode.
Step5: Initialize SI with 0000h.
Step6: Copy the contents SI into AL register.
Step7: Compare AL register with null character '**!**'
Step8: If equal go to step 11.
Step9: Move the number 14 into AH register.
Step10: Move the number 05h into BL register.
Step11: Set keyboard display mode.
Step12: Go to step 6.
Step 13: Terminate the program.
Step14: Stop.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
TEXT DB ‘ADITYA MICROPROCESSORS LAB!'
DATA ENDS
CODE SEGMENT
ORG 1000H
START: MOV AX, DATA
        MOV DS, AX
        MOV AH, 00H
        MOV AL, 13H
        INT 10H
        MOV SI, 00H
BACK: MOV AL, TEXT [SI]
        CMP AL,'!'
        JE EXIT
        MOV AH, 14
        MOV BL, 05H
        INT 10H
        INC SI
        JMP BACK
EXIT: MOV AH, 4CH
       INT 21H
CODE ENDS
END START
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

Exp No:
Date:

DIGITAL TO ANALOG CONVERTER

GENERATION OF WAVE FORMS:
AIM: program to generate the following wave forms:

- Triangular wave forms
- Saw tooth wave forms
- Square wave

REGISTERS USED: general purpose registers: AL, DX, and CX
PORTS USED: Out (port-B)
CONNECTION: J_4 of ESA 86/88 to J_1 DAC interface.
DESCRIPTIONS: As can be from the circuit only 17 lines from the connector are used totally. The port A and port B of 8255 programmable peripheral interface are used as output ports. The digital inputs to the DAC's are provided through the port A and port B of 8255.the analog outputs of the DAC's are connected to the inverting inputs of op-amps µA741 which acts as current to voltage converters. The out puts from the op- amps are connected to points marked X_{out} and Y_{out} at which the wave forms are observed on a CRO. (port A is used to control X_{out} port B is used to control Y_{out}).the difference voltage for the DAC's is derived from an on-board voltage regulator µA 723 .it generates a voltage of about 8V.the offset balancing of the op-amps is done by making use of the two 10k pots provided. The output wave forms are observed at X_{out} and Y_{out} on an oscillator.

THEORY:

BASIC DAC TECHNIQUE:

$V_o = K\ V_{FS}\ (d1\ .2^{-1} + d_2\ .\ 2^{-2} + \ldots\ldots\ldots + d_n\ .\ 2^{-n}\ )$

Where d_1 = MSB, d_2 = LSB

V_{FS} = Full scale reading / out put voltage

K --- Conversion factor is adjusted to 'unity'.

D/A converters consist of 'n' bit binary word 'D'and is combined with a reference voltage V_R to give an analog output. The out put can be either voltage or current

Out put voltage $V_o = K\ V_{FS}\ (d_1\ .2^{-1} + d_2.\ 2^{-2} + \ldots\ldots\ldots + d_n.\ 2^{-n})$

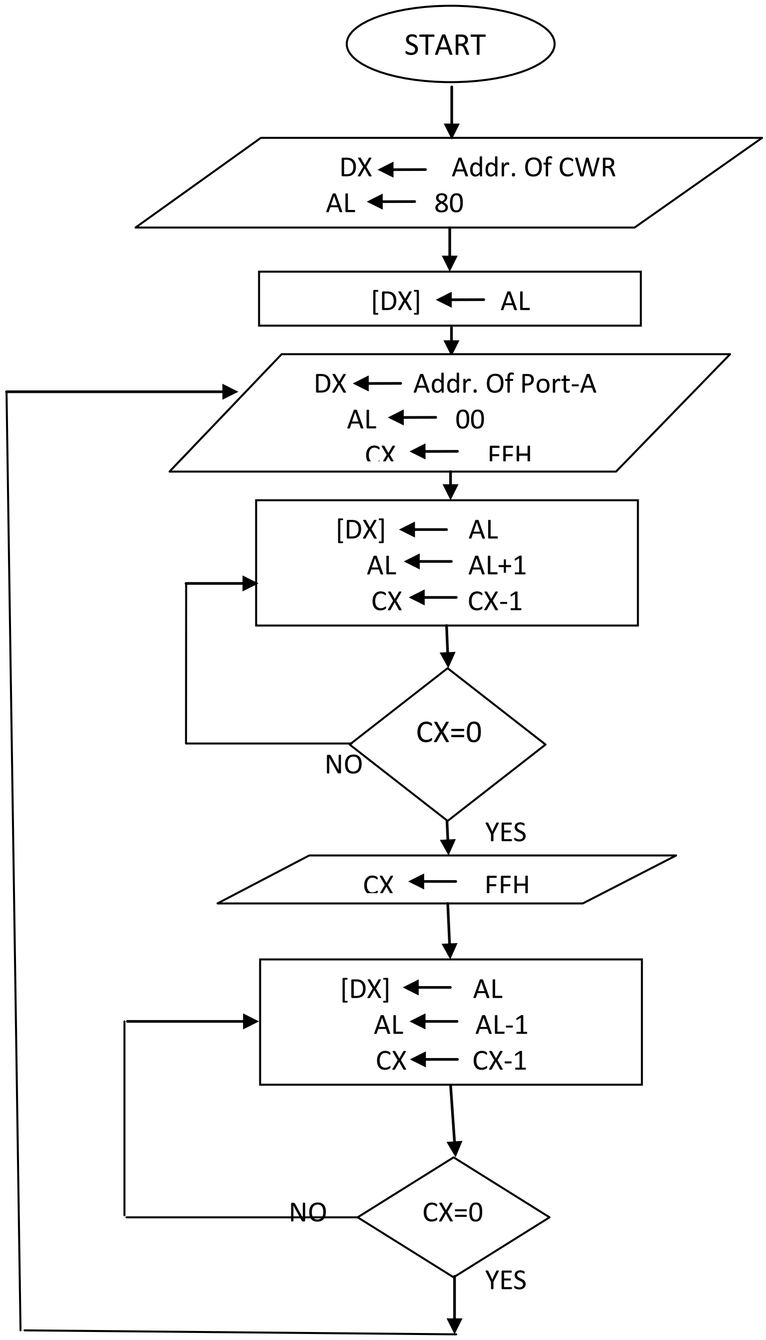
MSB weight = ½ V_{FS} if d_1 = 1 and all are zero's, K = 1.

LSB weight = $V_{FS}/2^n$ if dn = 1 and all are zero's, K = 1

DUAL DAC INTERFACE:

- This program generates a square wave or a Triangular wave at points X_{out} or Y_{out} of interface. The waveforms may be observed on an oscilloscope.

FLOW CHART:


START
DX ← Addr. Of CWR
AL ← 80
[DX] ← AL
DX ← Addr. Of Port-A
AL ← 00
CX ← FFH
[DX] ← AL
AL ← AL+1
CX ← CX-1
CX=0
NO
YES
CX ← FFH
[DX] ← AL
AL ← AL-1
CX ← CX-1
CX=0
NO
YES

- This program can be executed in STAND-ALONE MODE or SERIAL MODE of operation.
- The program starts at memory location 3000H

ALGORITHM: (FOR TRIANGULAR WAVE):

Step 1: Start
Step 2: Initialize the control word register with all ports as simple I/O mode
Step 3: Load Maximum Amplitude value to CX register.
Step 4: Load 00 to AL register.
Step 5: Initialize the port A address.
Step 6 : Locate the contents of AL to DX register.
Step 7: Increment the value in AL by one.
Step 8: Locate AL contents to DX register.
Step 9: Decrement the value of CX register by one and go to step 6 if CX not equal to zero.
Step 10: Load 00FF to CX register.
Step 11: Decrement the value of AL by one.
Step 12: Locate the contents in AL register to DX register.
Step 13: Decrement the value of CX by one and go to step 12 if CX not equal to zero.
Step 14: Otherwise move to step 3
Step 15: Stop.

PROGRAM (FOR TRIANGULAR WAVE):

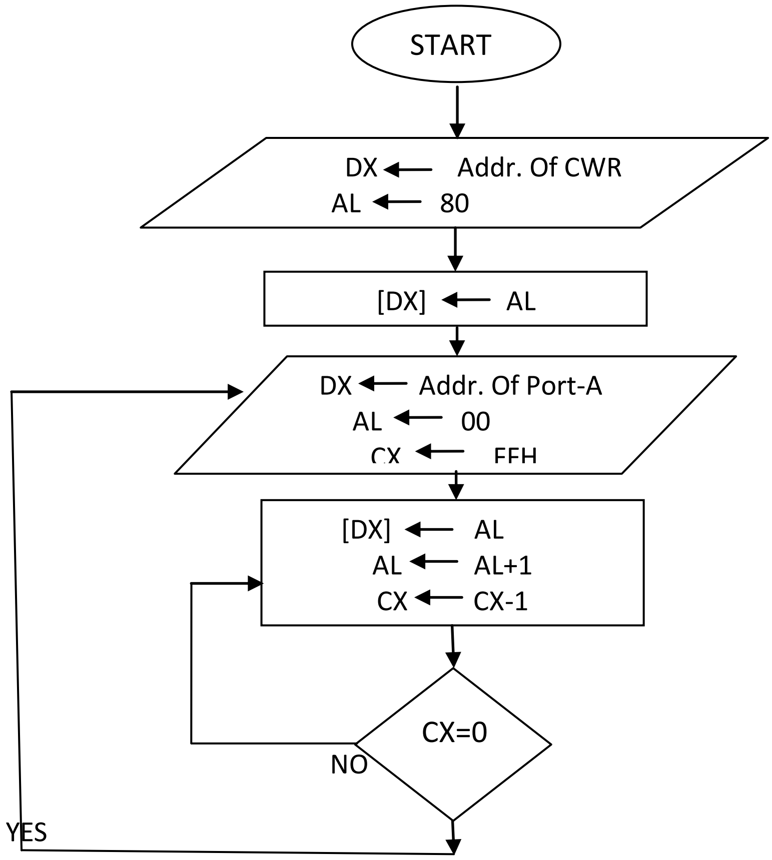
```
      MOV DX, 0FFE6
      MOV AL, 80
      OUT   DX, AL
      MOV DX, 0FFE0
      MOV   AL, 00
RPT:  MOV CX, 0FF
L1:   OUT DX, AL
      INC AL
      LOOP L1
      MOV CX, 0FF
L2 :  OUT DX,AL
      DEC AL
      LOOP L2
      JMP RPT
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
DX ← Addr. Of CWR
AL ← 80
[DX] ← AL
DX ← Addr. Of Port-A
AL ← 00
CX ← FFH
[DX] ← AL
AL ← AL+1
CX ← CX-1
CX=0
NO
YES

ALGORITHM (FOR SAW TOOTH WAVE):

Step 1: Start
Step 2: Initialize the control word register with all ports as simple I/O mode
Step 3: Load Maximum Amplitude value to CX register.
Step 4: Load 00 to AL register.
Step 5: Initialize the port A address.
Step 6 : Locate the contents of AL to DX register.
Step 7: Increment the value in AL by one.
Step 8: Locate AL contents to DX register.
Step 9: Decrement the value of CX register by one and go to step 6 if CX not equal to zero.
Step 10: Go to step 4 and Repeat

PROGRAM (FOR SAW TOOTH WAVE):

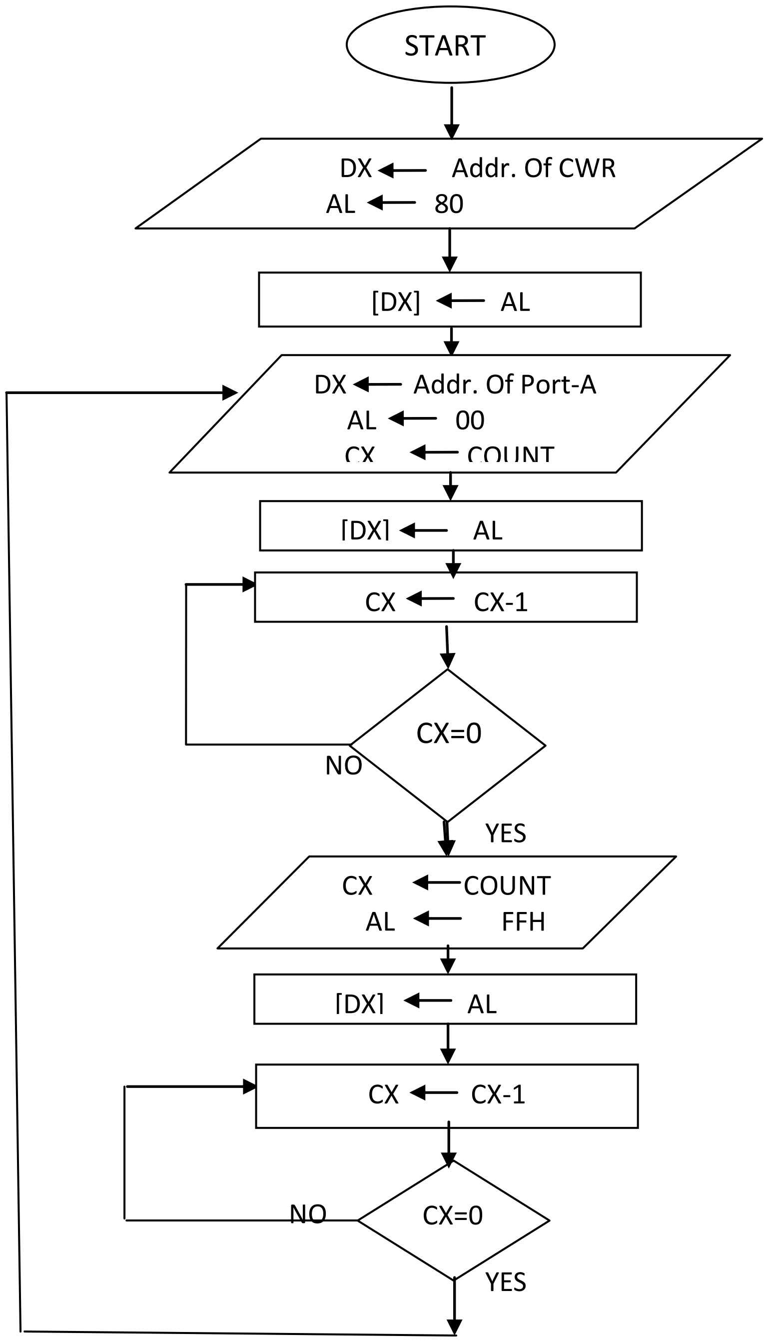
```
        MOV DX, 0FFE6
        MOV AL, 80
        OUT DX,AL
        MOV DX, 0FFE0
        MOV AX, 00
RPT:    MOV CX, 0FF
L1:     OUT DX, AX
        INC AX
        LOOP L1
        MOV AX, 00
        OUT DX, AX
        JMP RPT
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

FLOW CHART:


START
DX ← Addr. Of CWR
AL ← 80
[DX] ← AL
DX ← Addr. Of Port-A
AL ← 00
CX ← COUNT
[DX] ← AL
CX ← CX-1
CX=0
NO
YES
CX ← COUNT
AL ← FFH
[DX] ← AL
CX ← CX-1
NO
CX=0
YES

ALGORITHM (FOR SQUARE WAVE):

Step 1: Start
Step 2: Initialize the control word register with all ports as simple I/O mode
Step 3: Load Maximum Amplitude value to AX register.
Step4: Initialize the port A address.
Step 5 : Transmit the contents of AX to port A
Step 6: Create Delay
Step 7: Load Minimum Amplitude value to AX register.
Step 8: Transmit the contents of AX to port A
Step 9: Create Delay
Step 10: Go to Step 3 and Repeat

ALGORITHM FOR DELAY

Step 1: Load the CX register with 93h
Step 2: Decrement CX
Step 3: Repeat Step 2 till CX is zero
Step4: Return to main program

PROGRAM (FOR SQUARE WAVE):

```
        MOV DX, 0FFE6
        MOV AL, 80
        OUT DX, AL
        MOV DX, 0FFE0
RPT:    MOV AX, 0FF
        OUT DX, AX
        CALL DELAY
        MOV AX, 00
        OUT DX, AX
        CALL DELAY
        JMP RPT
DELAY PROGRAM:
        MOV CX, 1E
   L1:  NOP
        NOP
        LOOP L1
        RET
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

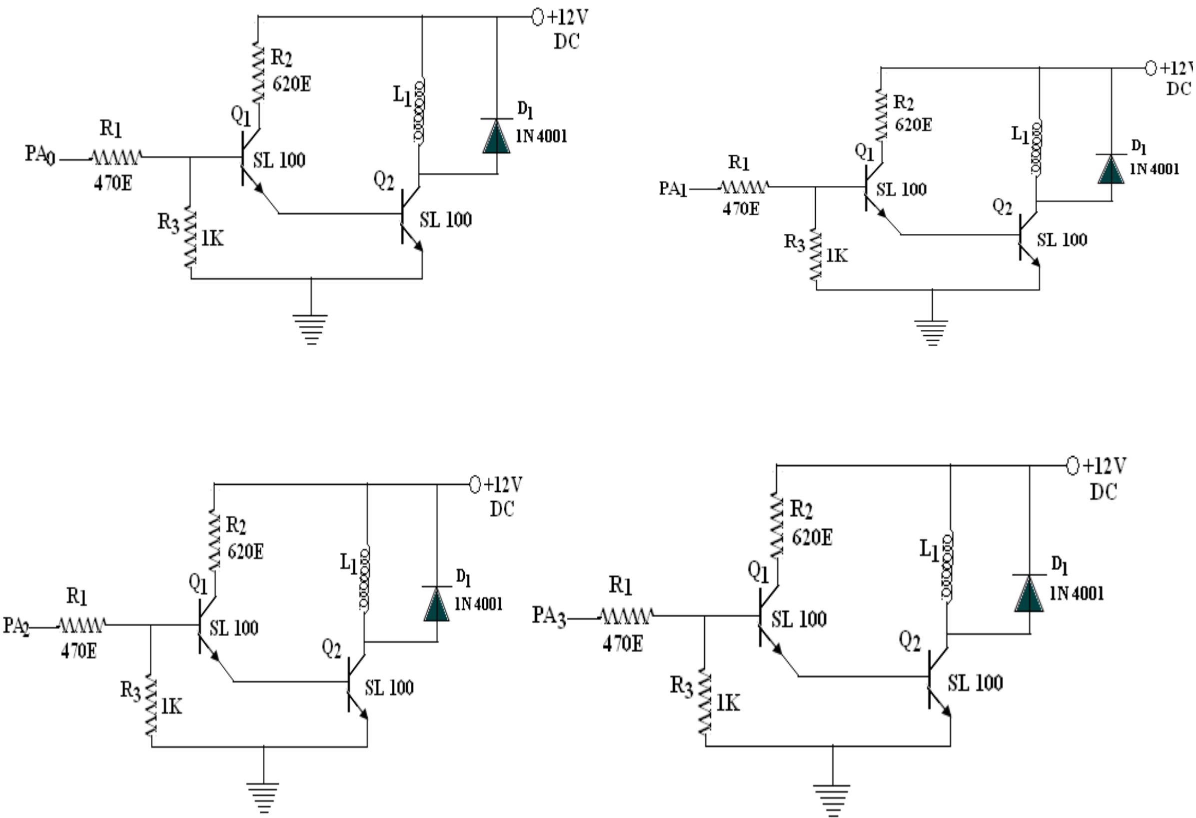
RESULT:

Exp No:

Date:

STEPPER MOTOR INTERFACING

AIM: program to design a stepper motor to rotate shaft of a 4 phase stepper motor in clockwise 15 rotations.
REGISTERS USED: General purpose registers: AL , DX , CX
PORTS USED: Port B, port C (out)
CONNECTIONS: J4 of ESA 86/88E to J_1 of stepper motor.
OPERATING PRINCIPLE OF PERMANENT MAGNET STEPPER MOTOR:



It consists of two stator windings A,B and a motor having two magnetic poles N and S. when a voltage +v is applied to stator winding A, a magnetic field F_a is generated. The rotor positions itself such that its poles lock with corresponding stator poles.

With the winding ‘A’ excited as before ,winding ‘b’ is now to F_a. the resulting magnetic field F makes an angle of 45^0. the rotor consequently moves through 45^0 in anti clockwise direction, again to cause locking of rotor poles with corresponding stator poles.
While winding ‘B’ has voltage +V applied to it, winding ‘A’ is switched off. The rotor then moves through a further 45^0 in anti-clockwise direction to aligne itself with stator field F_b. with voltage +V on winding B, a voltage –V is applied to winding A. then the stator magnetic field has two components F_a , F_b and their resultant F makes an angle of 135^0 position.

In this way it can be seen that ,as the pattern of excitation of the state of winding is changed, the rotor moves successively through45^0 steps. And completes one full revolution in anti clock-wise direction. A practical PM stepper motor will have 1.8^0 step angle and 50 tooth on it's rotor;there are 8 main poles on the stator, each having five tooth in the pole face. The step angle is given by

A = 360 / (N * K) degrees

Where N = number of rotor tooth.

K = execution sequence factor.

PM stepper motors have three modes of excitation i,e..

- Single phase mode
- Two phase mode
- Hybrid mode

Single phase mode: in this mode only one of the motor winding is excited at a time. There are four steps in the sequence, the excitation sequence factor K=2 ,so that step angle is 90^0.

Two phase mode: Here both stators phase are excited at a time. There are four steps in the excitation sequence, K = 2 and the step angle is 90^0. However, the rotor positions in the two phase mode are 45^0 way from those in single phase mode.

Hybrid mode: this is a combination of single and two phase modes. There are 8 steps in excitation sequence=2 and step angle = 45^0. a voltage +V is applied to a stator winding during some steps, which voltage V is applied during certain other steps. This requires a bipolar regulated power supply capable of yielding +V,-V and zero outputs and a air of SPDT switches, which is quite cumbersome. Consequently each of the two stator windings is split into two sections A1-A2 , B1-B2. these sections are wound differentially. These winding sections can now be excited from a univocal regulated power supply through switcher S1 to S4. this type of construction is called bipolar winding construction. Bipolar windingesults in reduced winding inductance and consequently improved torque stepping rate.
Description: the stepper motor interfaces uses four transistor pairs (SL 100 and 2N 3055) in a Darlington pair configuration. Each Darlington pair is used to excite the particular winding of the motor connected to 4 pin connector on the interface. The inputs to these transistors are from the 8255 PPI I/O lines of the microprocessor kit or from digital I/O card plugged in the PC. "port A" lower nibble PA_0 , PA_1, PA_2 , PA_3 are the four lines brought out to the 26 pin FRC male connector(J_1) on the interface module. The freewheeling diodes across each winding protect transistors from switching transients.
Theory:
A motor used for moving things in small increments is known as stepper motor. Stepper motor rotate from one fixed position to next position rather than continuous rotation as in case of other ac or dc motor stepper motors are used in printers to advance the paper from one position to advance the paper from one position to another in steps. They are also used to position the read/write head on the desired track of a floppy disk. To rotate the shaft the stepper motor a sequence of pulses are applied to the windings in a predefined sequence. The number of pulses required for one complete rotation per pulse is given by $360^0/N_T$. where

"N_T" is the number of teeth on rotor. Generally the stepper motor is available with 10 to 30^0 rotation. They are available with two phases and four phase common field connections.

Instead of rotating smoothly around and around as most motors, stepper motors rotate or step one fixed position to next. Common step size range from 0.9^0 to 30^0. it is stepped from one position to next by changing the currents through the fields in the motor.

The two common field connections are referred to as two phase and four phase. The drive circuitry is simpler in 4 phase stepper. The figure shows a circuitry that can interface a small 4 stepper motor to four microcomputer port lines.

The 7406 buffers are inverting, so. A high on ah output port pin turns on current to a winding. The purpose of clamp diodes across each winging is to save transistors from inductive kick. Resistors R_1 and R_2 are current limiting resistors.

Typical parameters of stepper motor:

1. Operating voltage - 12 volts

2. Current rating - 1.2 Amp

3. Step angle - 1.8^0

4. Step for revolution - 200(No. of teeth on rotor)

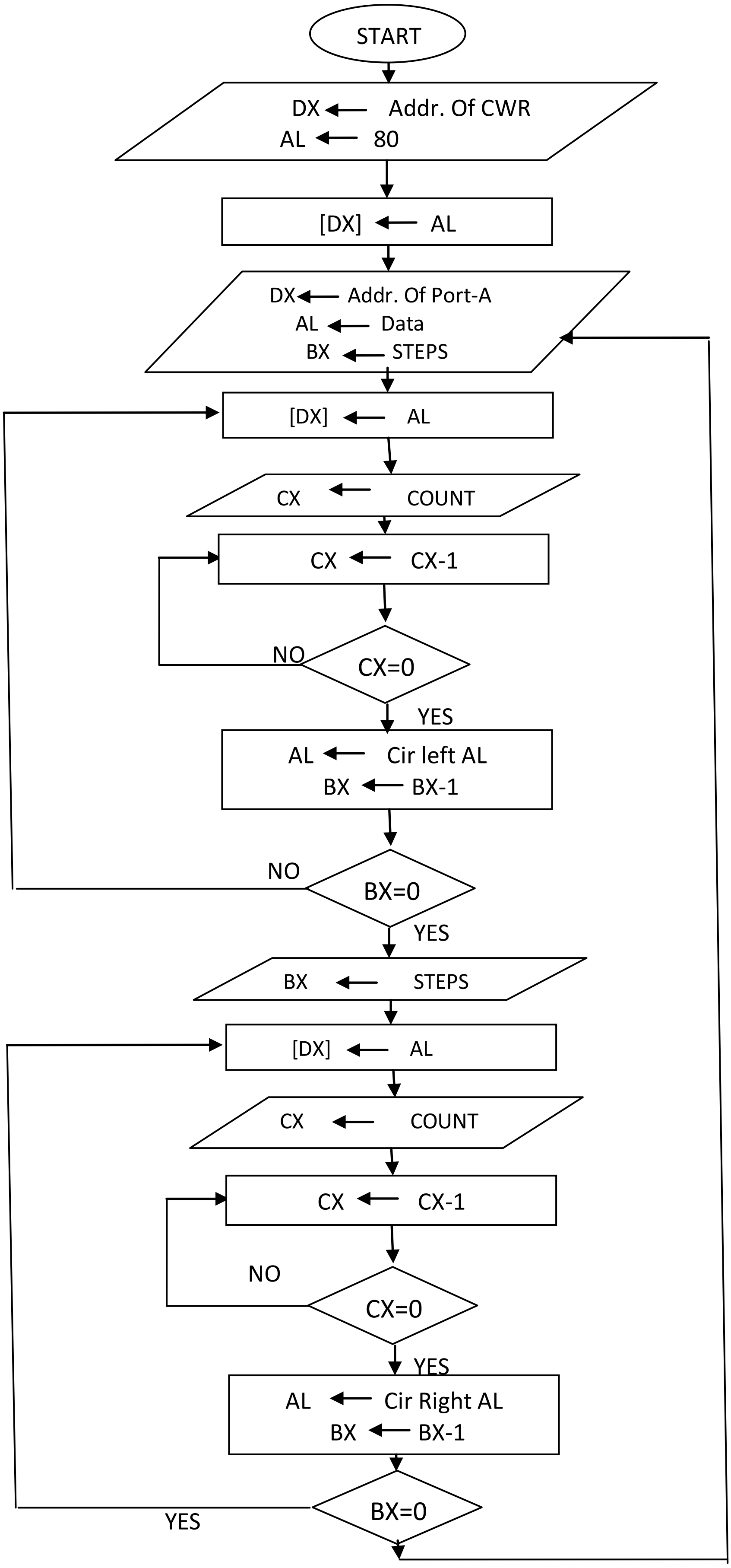
5. Torque - 3 kg/cm

Working of stepper motor:

Suppose that SW_1 and SW_2 are turned ON. Turning OFF SW_2 and turning ON SW_4 cause the motor to rotate one step of 1.8^0 clockwise. Changing to SW_4 and SW_3 ON will cause the motor to rotate 1.8^0 clockwise another. Changing SW_3 and SW_2 ON will cause another step. To step the motor in counter clock wise direction simply work through the switch sequence in the reverse direction.

The switch pattern for changing from one step to another step in clockwise direction is simply rotated right one position. For counter clockwise direction rotated left one position

FLOW CHART



NO YES

ALGORITHM:

Step 1: Start
Step 2: move the control word address 0FFE6 to register DX
Step 3: move 80 to AL register.
Step 4: locate the contents in AL register to DX register using port out.
Step 5: Intialize BX with 07d0
Step 6: move port A address ie.,,0FFE0 to DX register.
Step 7: move 11 to AL register.
Step 8: locate the contents in AL register to DX register using port out.
Step 9: move 300 to CX register.
Step 10: repeat step 8 until the content in CX register becomes equal to zero.
Step 11: Rotate carry left through bit.
Step 12: Decrement BX by one
Step 13: repeat steps from 8 until the content in BX register becomes equal to zero.
Step 14: Intialize BX with 07d0
Step 15: locate the contents in AL register to DX register using port out.
Step 16: move 300 to CX register.
Step 17: repeat step 15 until the content in CX register becomes equal to zero.
Step 18: Rotate carry left through bit.
Step 19: Decrement BX by one
Step 20: repeat steps from 15 until the content in BX register becomes equal to zero.
Step 21: jump to location / step 8.

PROGRAM:

```
        MOV     DX, 0FFE6
        MOV     AL, 80
        OUT     DX, AL
RPT:    MOV     BX,07D0
        MOV     DX, 0FFE0
        MOV     AL, 11
BACK:   OUT     DX, AL
        MOV     CX, 0300
L1:     LOOP    L1
        ROL     AL, 1
        DEC     BX
        JNZ     BACK
        MOV     BX,07D0
        MOV     AL, 11
BACK1:OUT       DX, AL
        MOV     CX, 0300
L2:     LOOP    L2
        ROR     AL, 1
        DEC     BX
        JNZ     BACK1
        JMP     RPT
```

CODE TABLE:

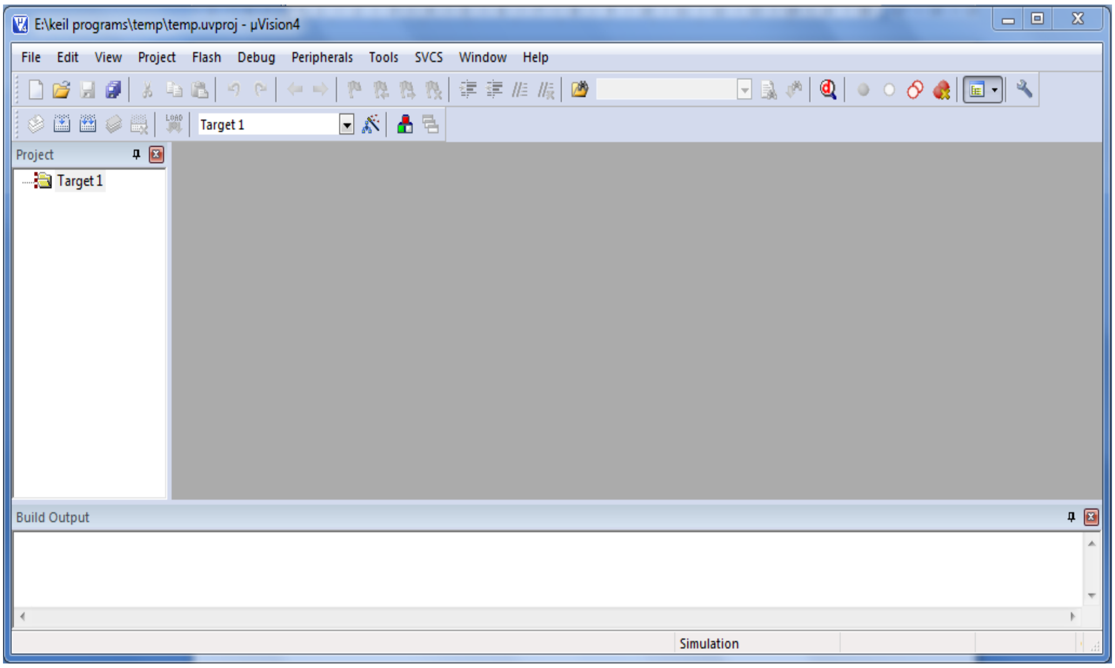
Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

RESULT:

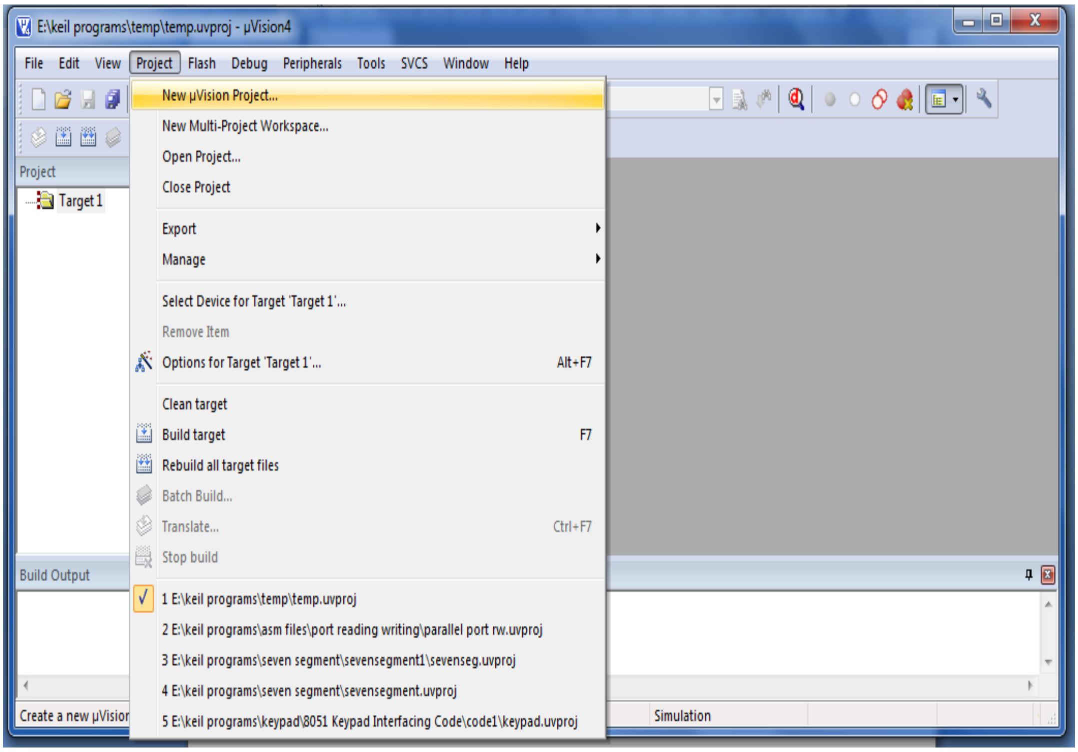
BASIC TUTORIAL FOR KEIL SOFTWARE

This tutorial will assist you in writing your first 8051 Assembly language program using the popular Keil
Compiler. Keil offers an evaluation package that will allow the assembly and debugging of files 2K or less.

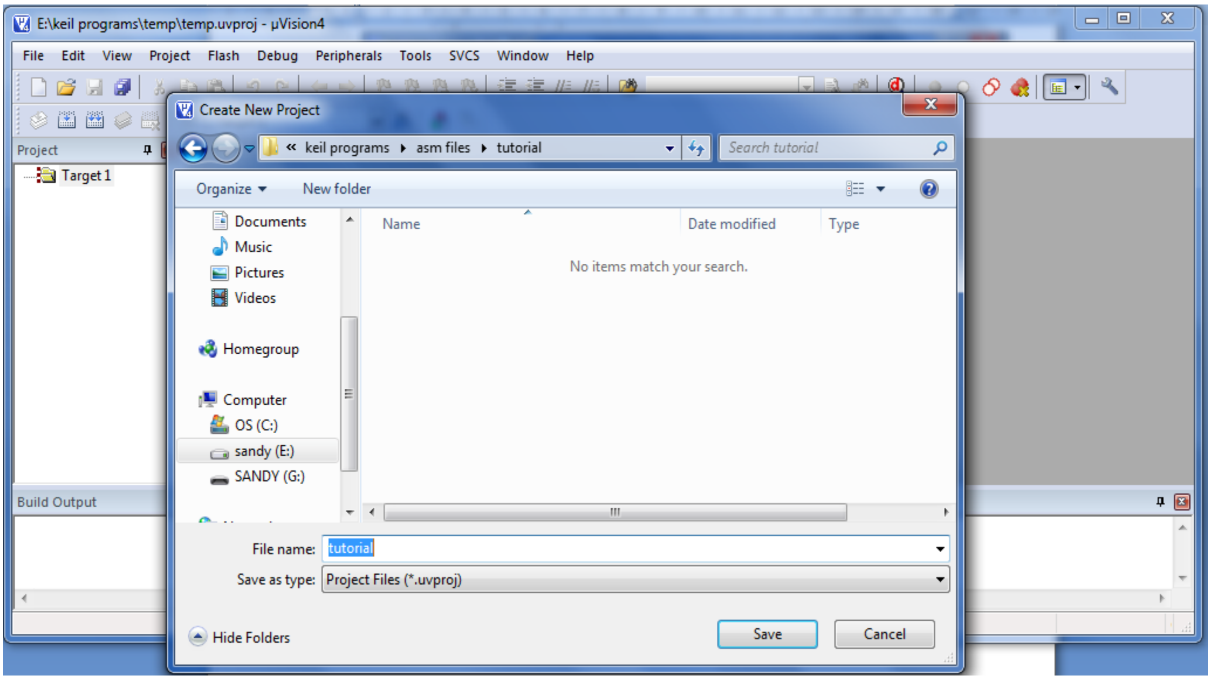
1. Open Keil from the Start menu



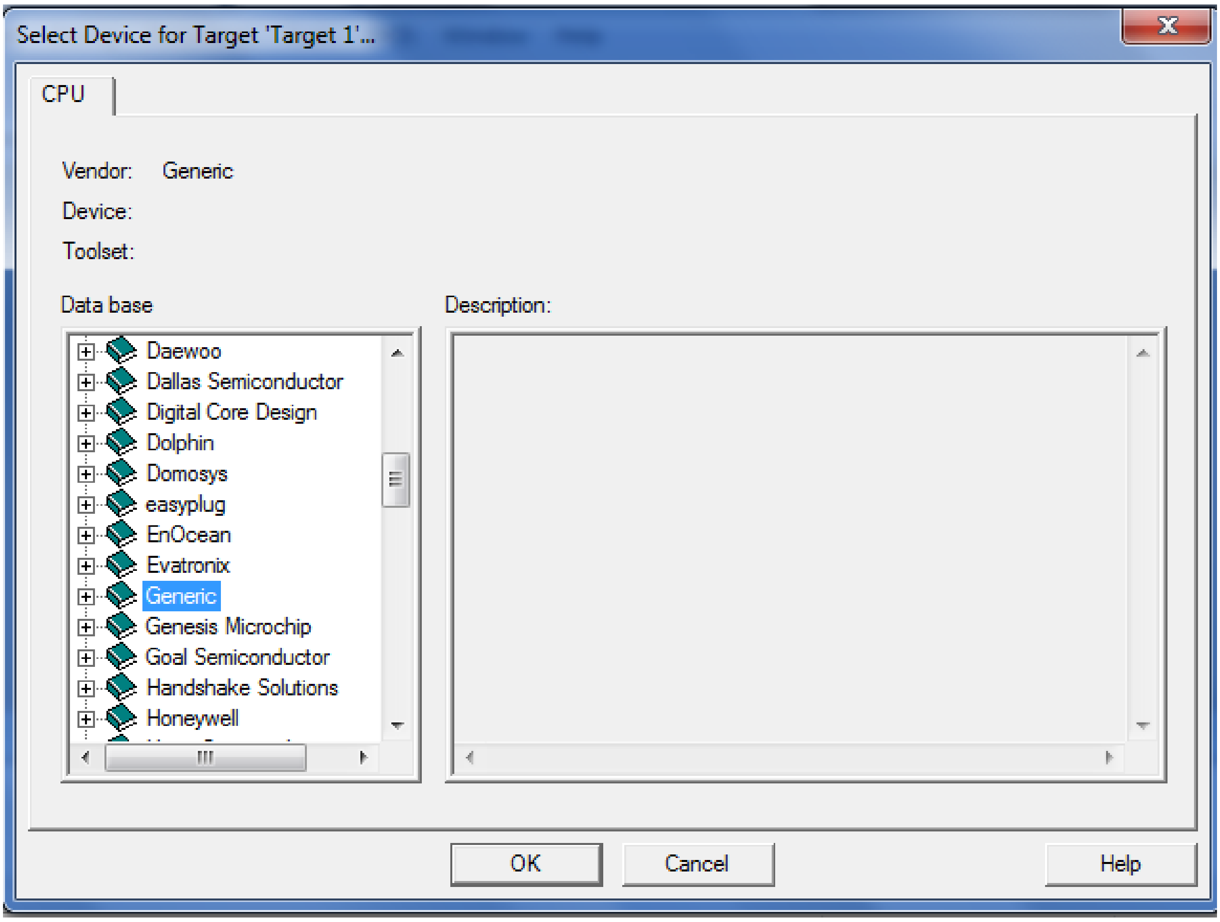
2. Select New project from the project menu



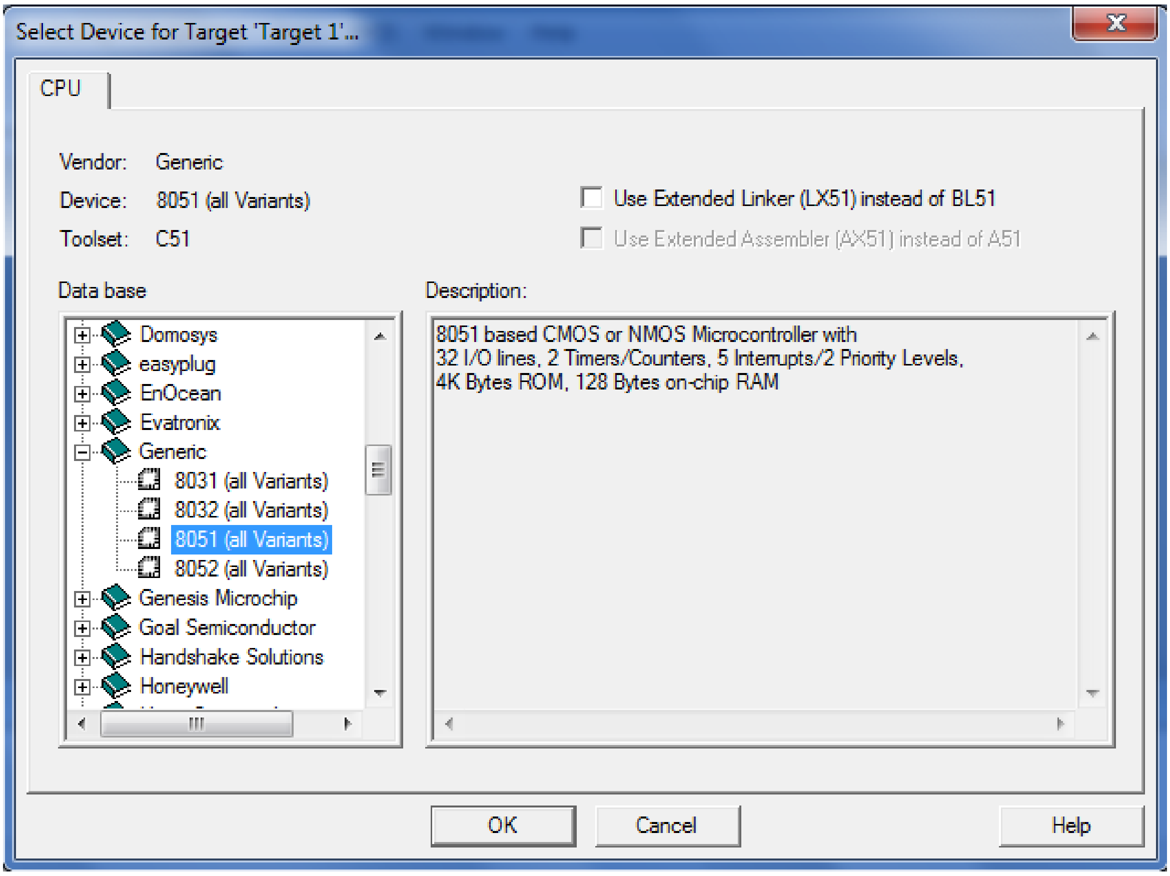
3. Give the Project name



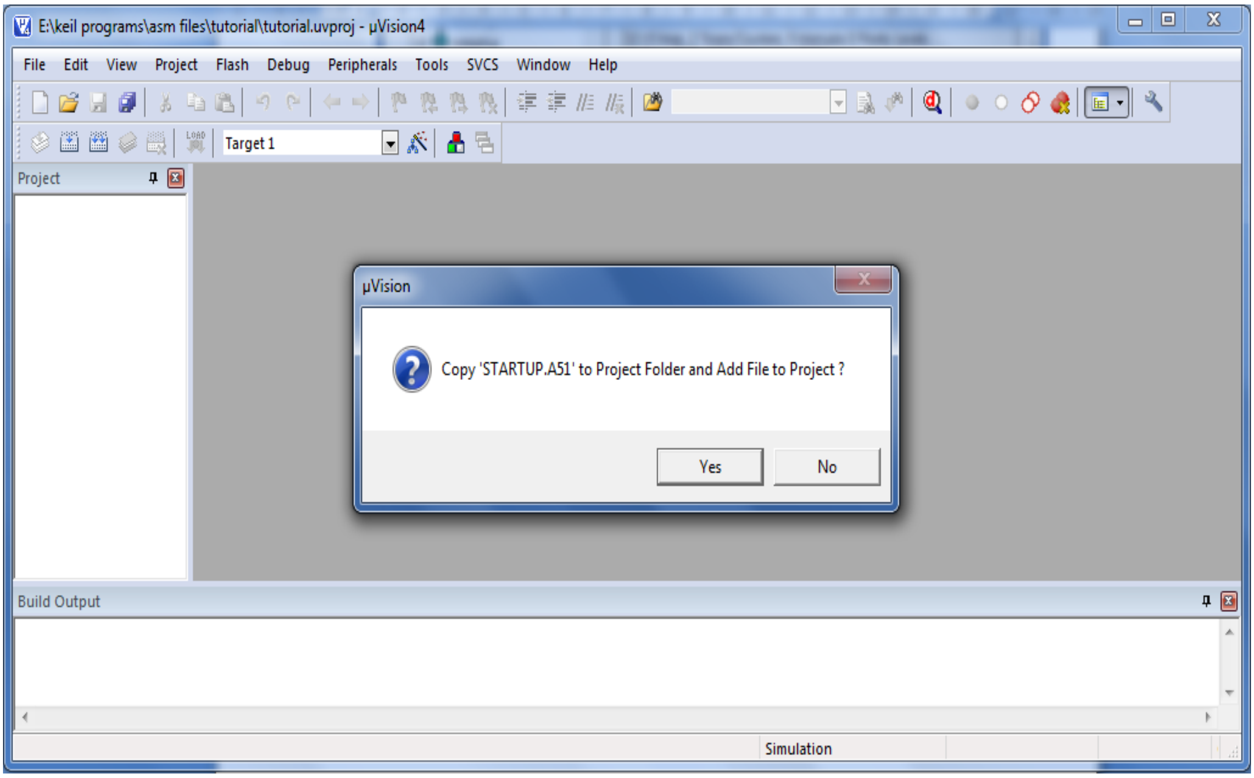
4. Click on the save button
5. The device window will be displayed. Select the part you will be using to test with. For now we will use Generic. Double Click on the Generic



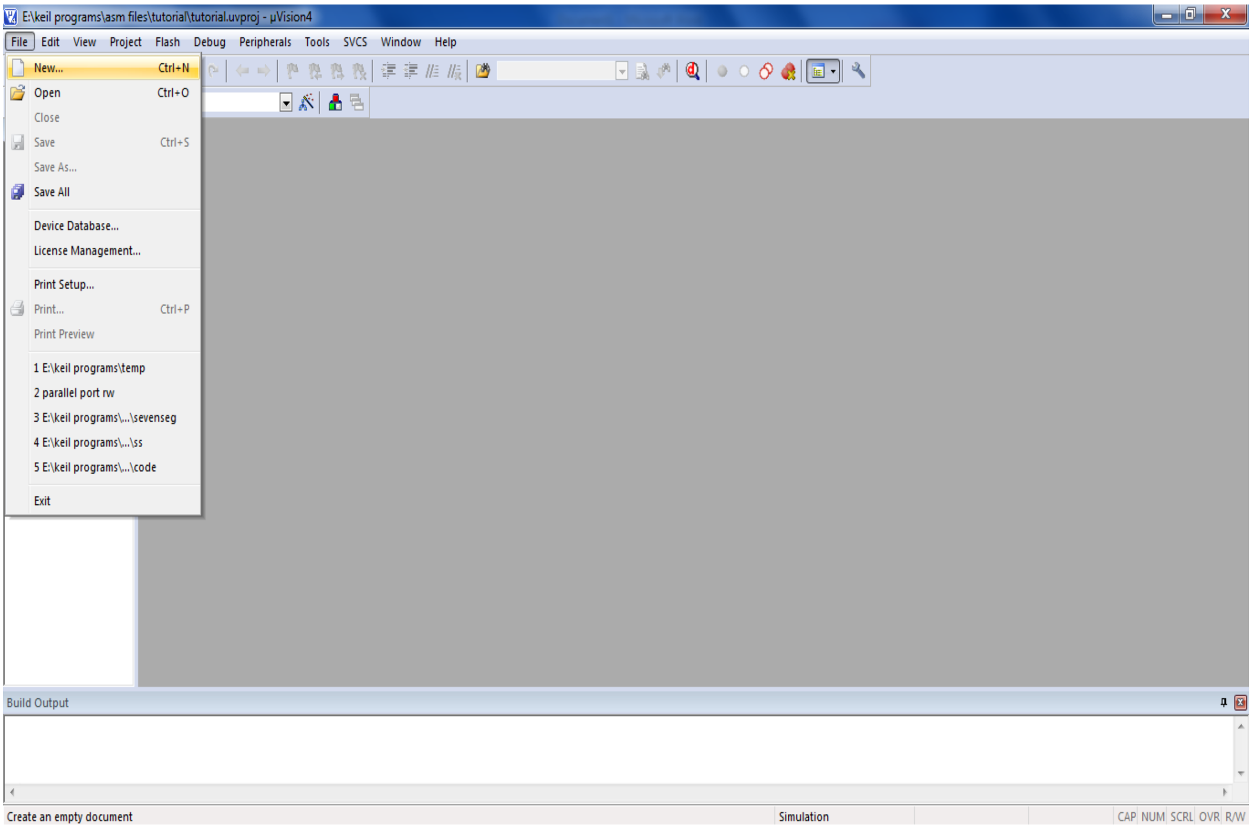
6. Scroll down and select the 8051(all Variants)part
7. Click OK



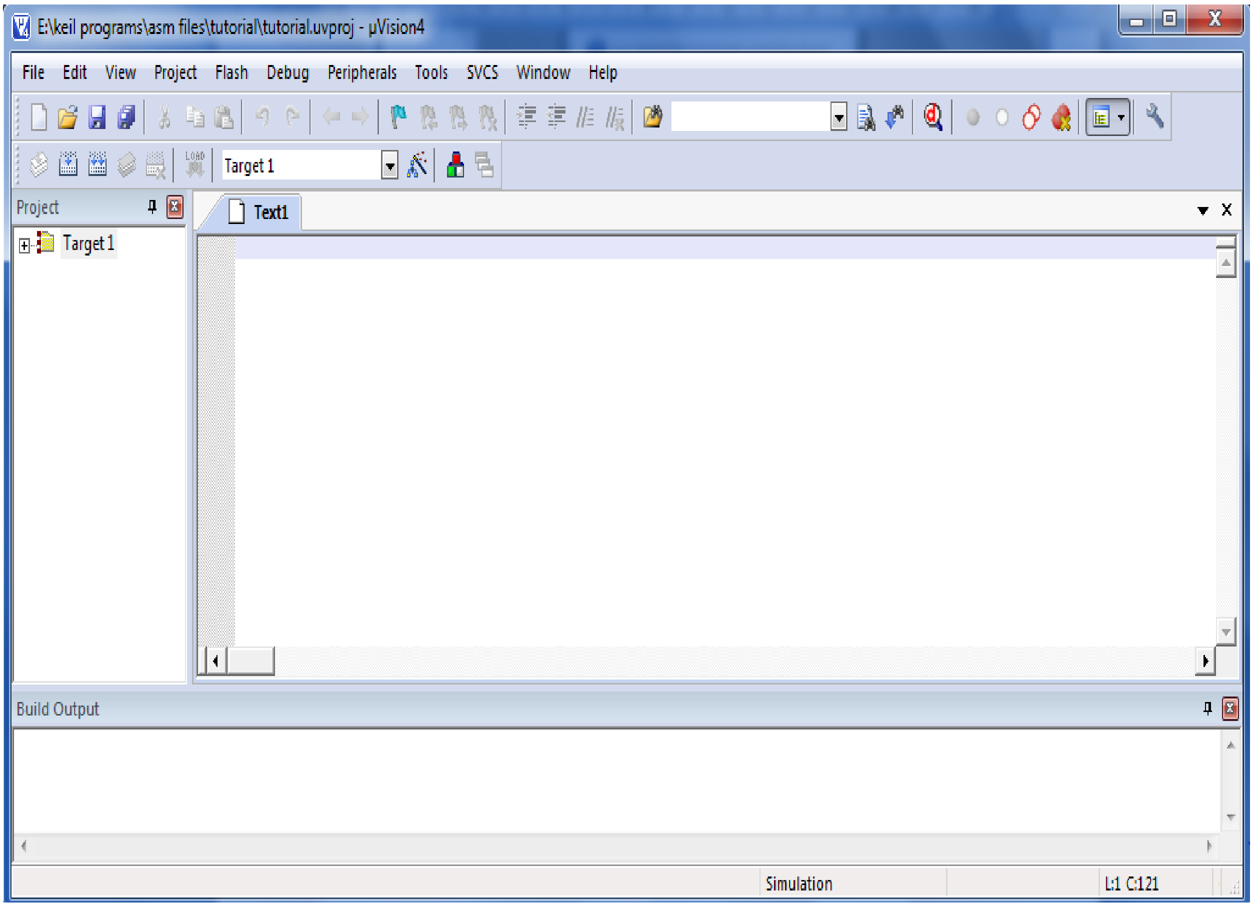
8. The dialog box will appear as below press YES



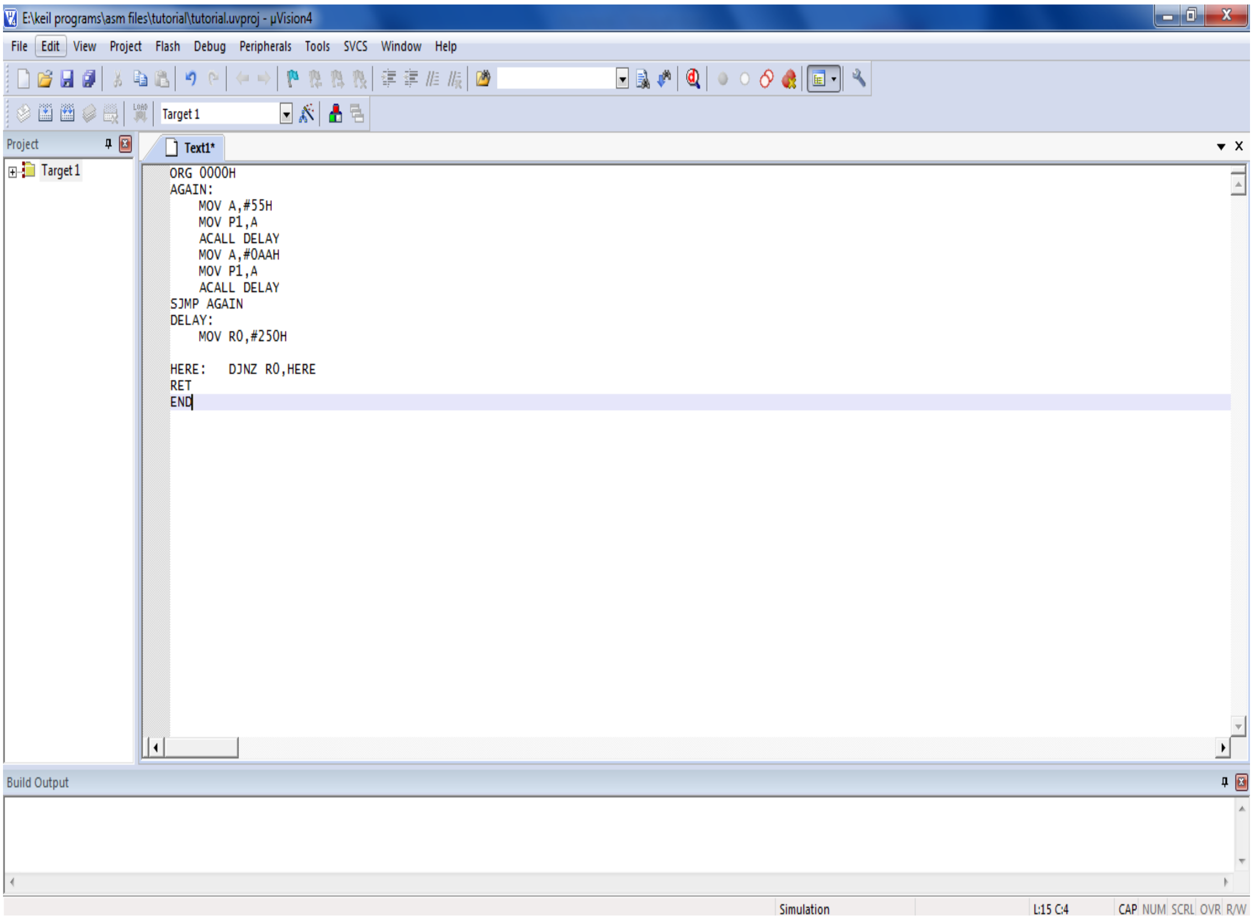
9. Click File menu and select NEW



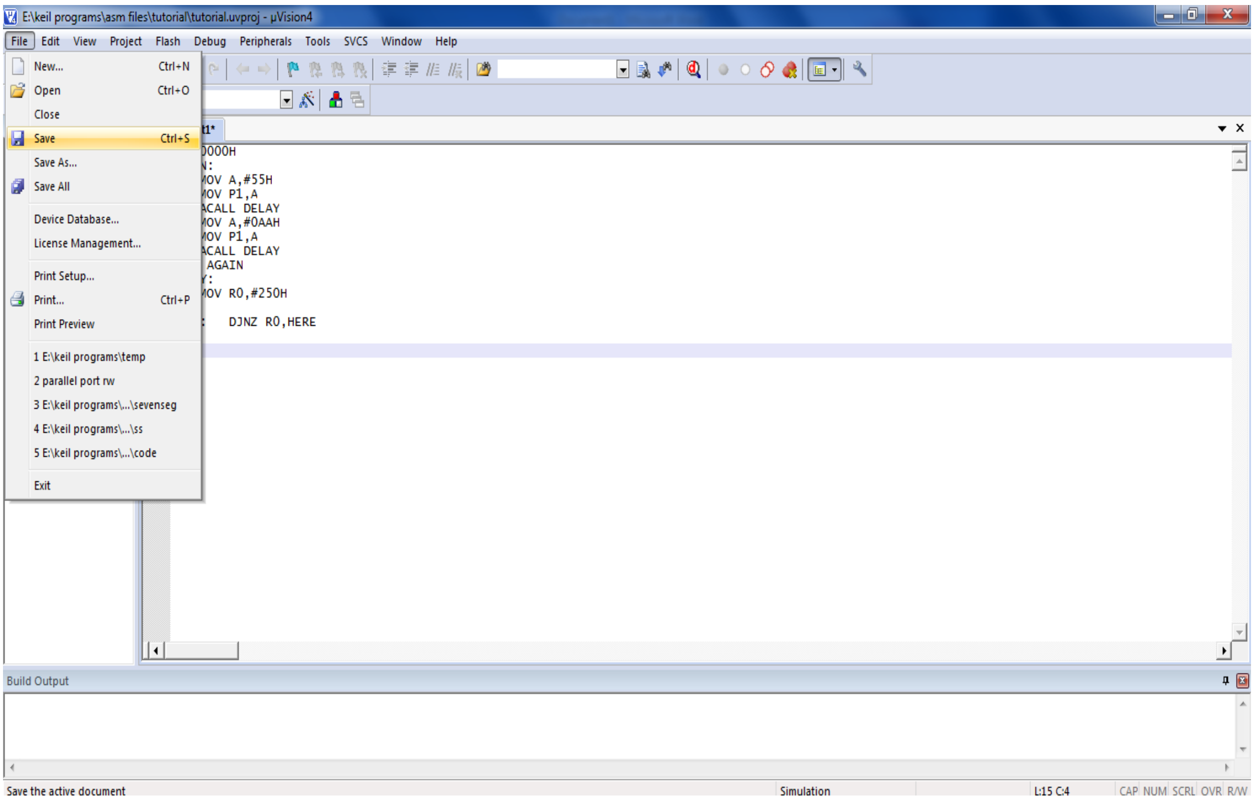
10. A new window will open up in the Keil IDE



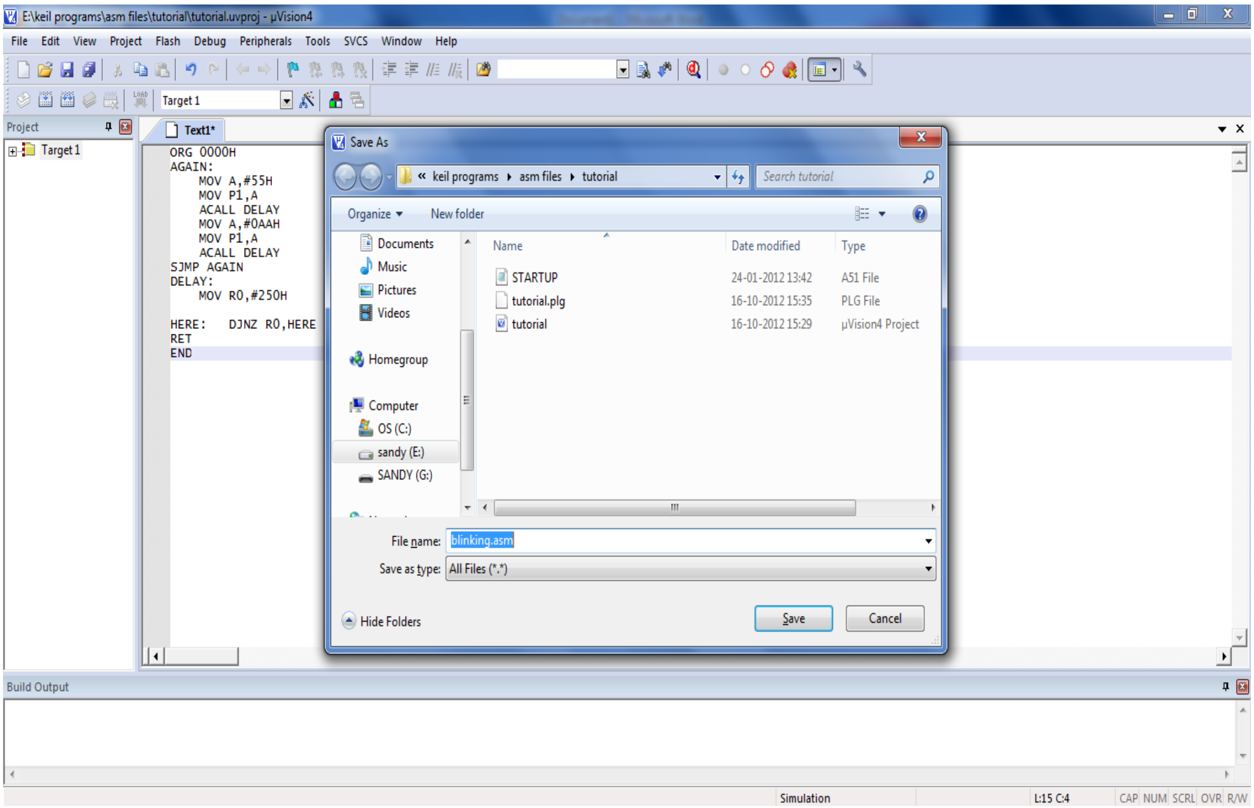
11. asm program on the editor



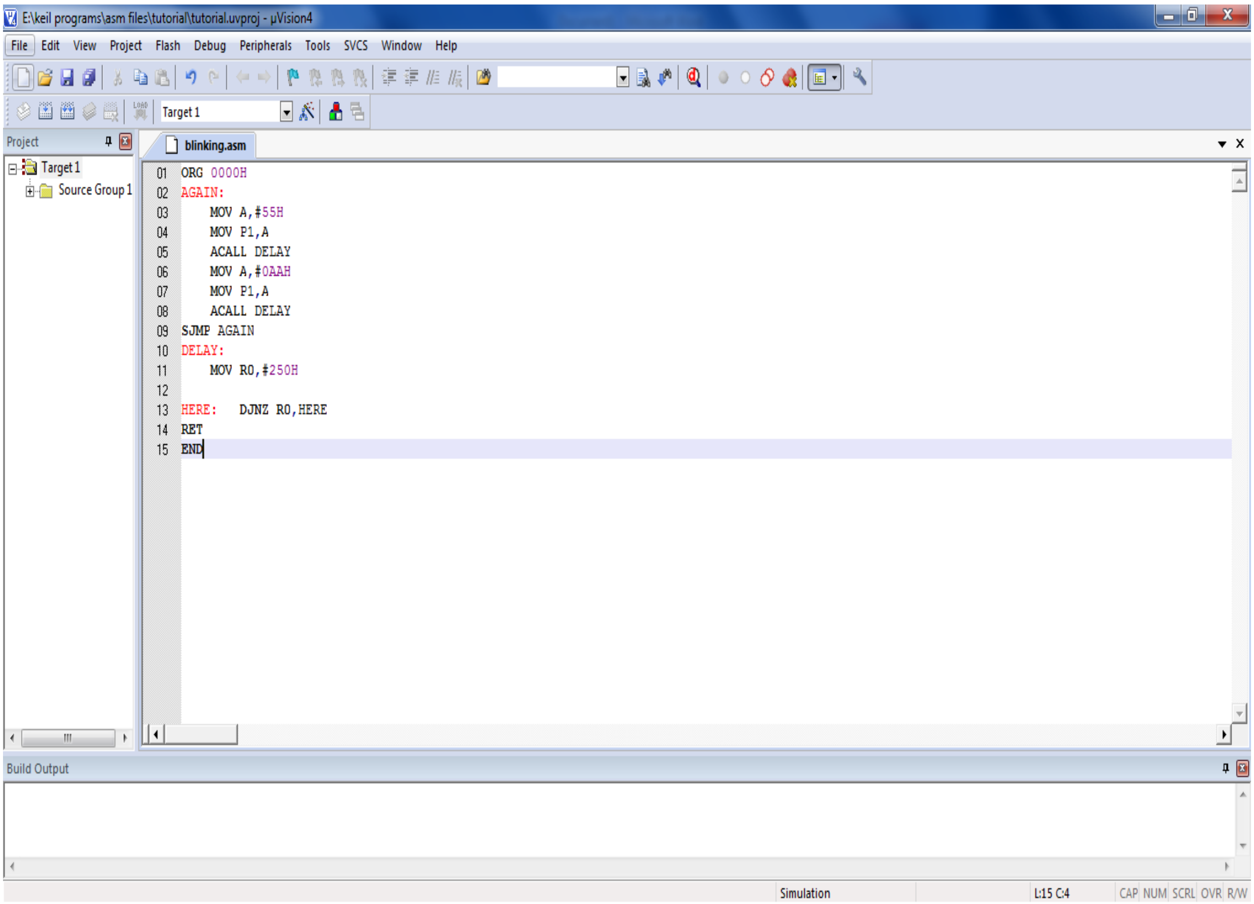
12. Click file menu and save



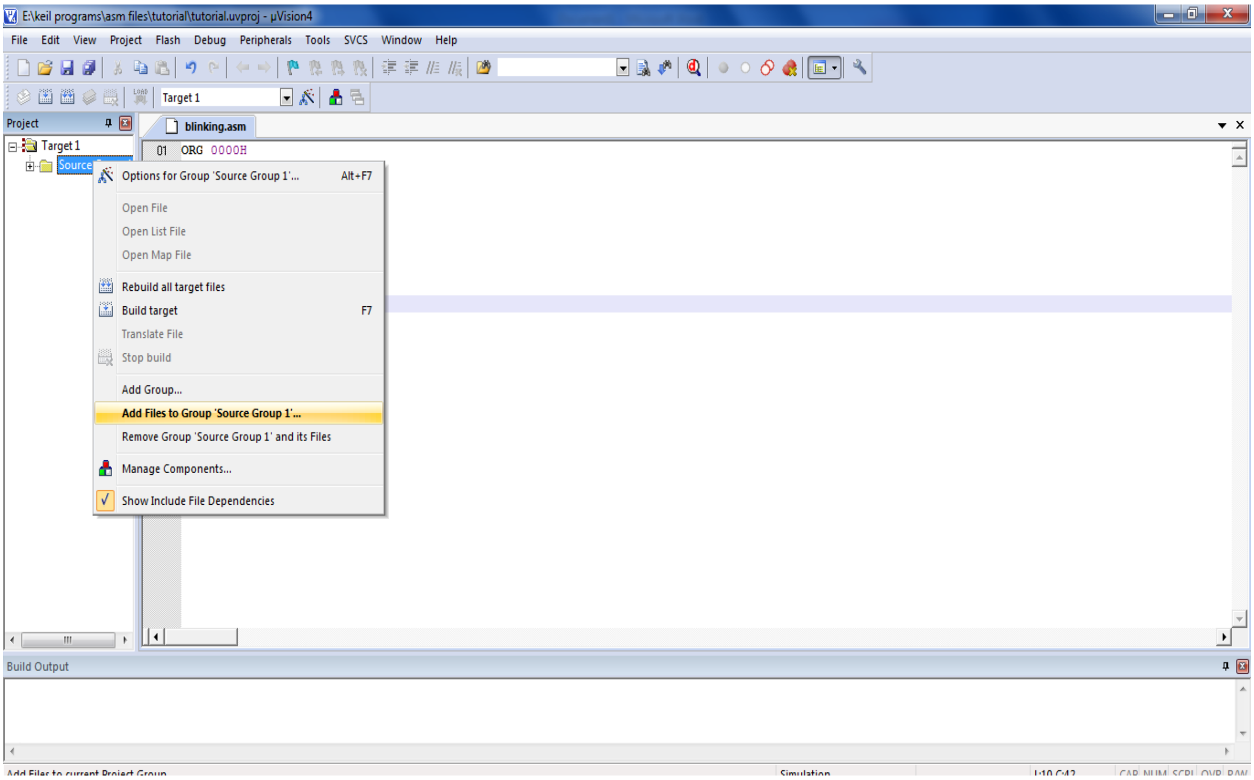
13. Name the file blinking.asm and click the save button



14. Expand Target 1 in the tree menu

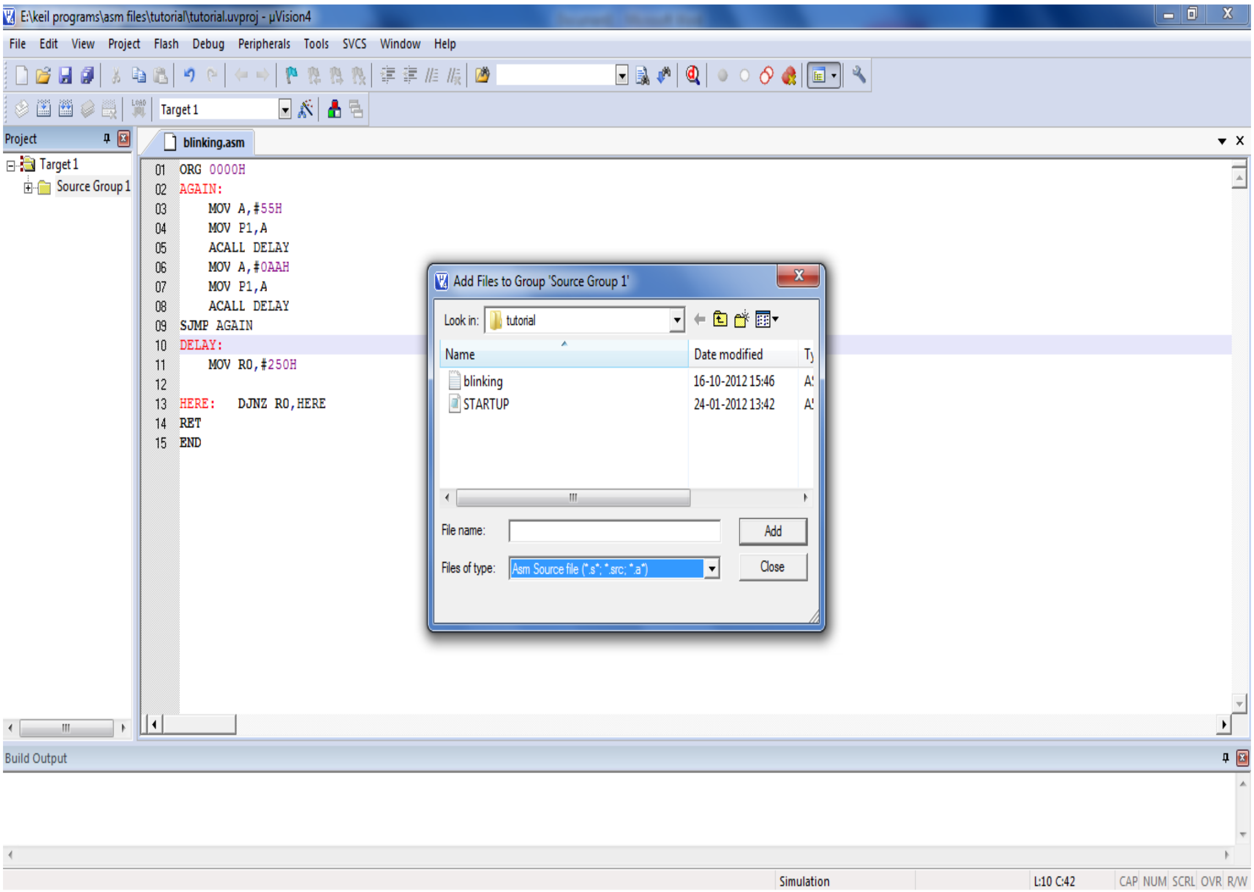


15. Right click on source group and click on add files to group 'Source Group 1'…

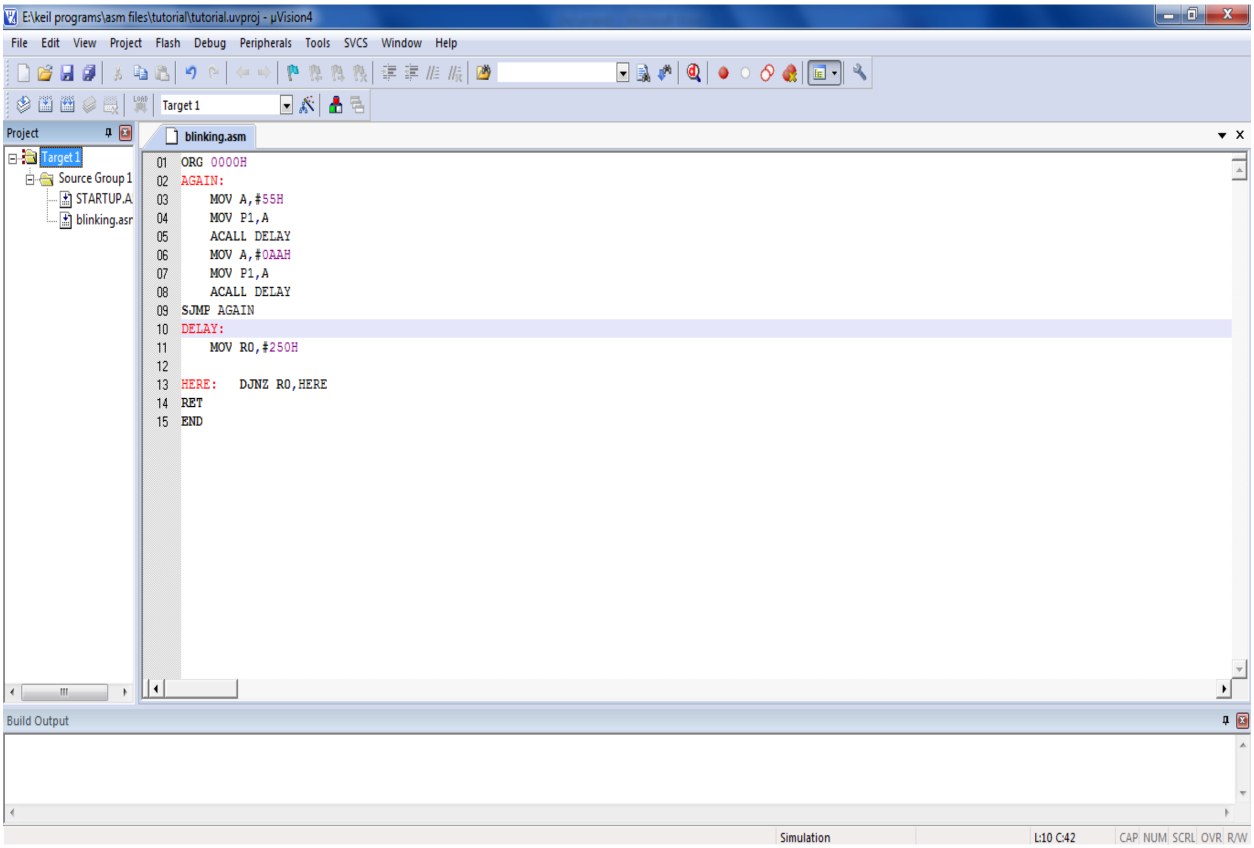


16. Change file type to asm source file(*.a*;*.src) and click on blinking.asm

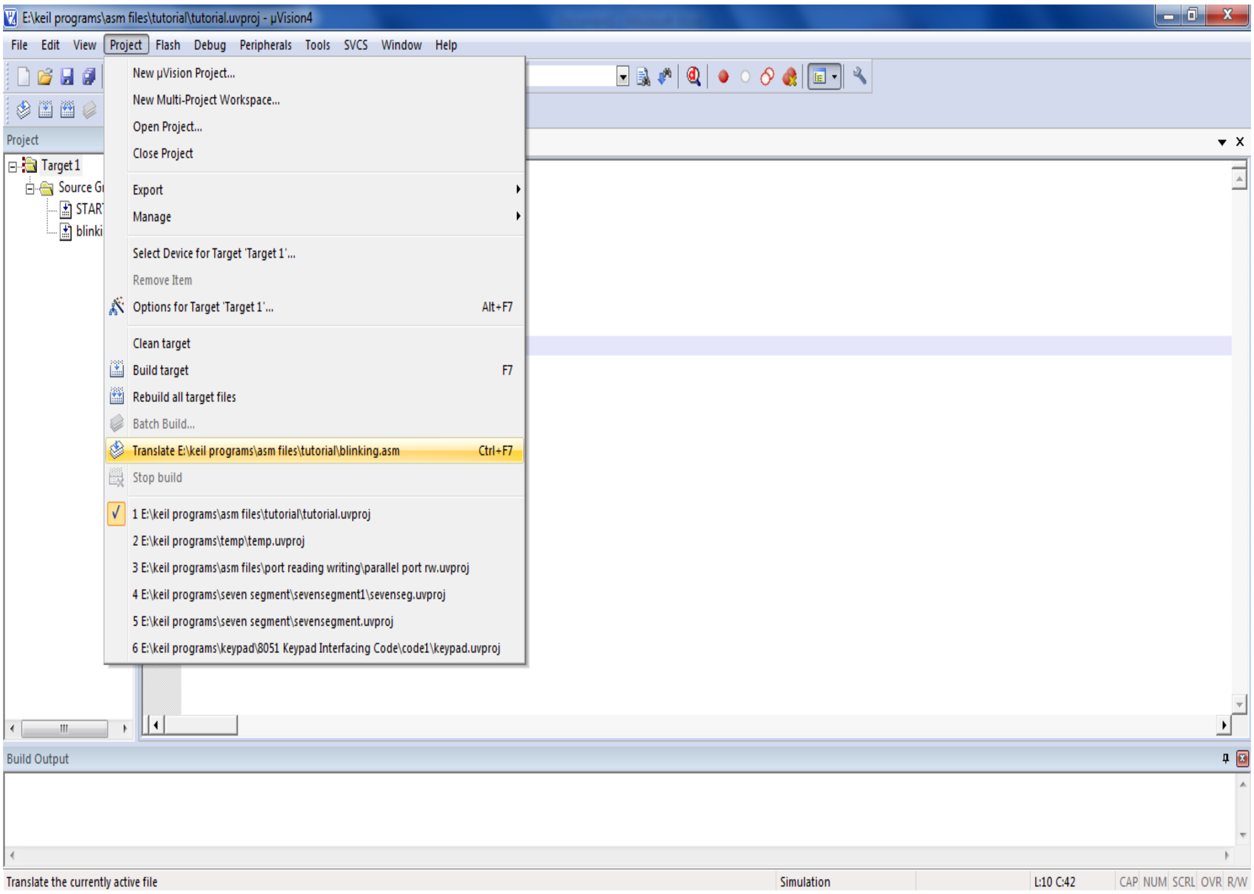
17. Click add button and click close button



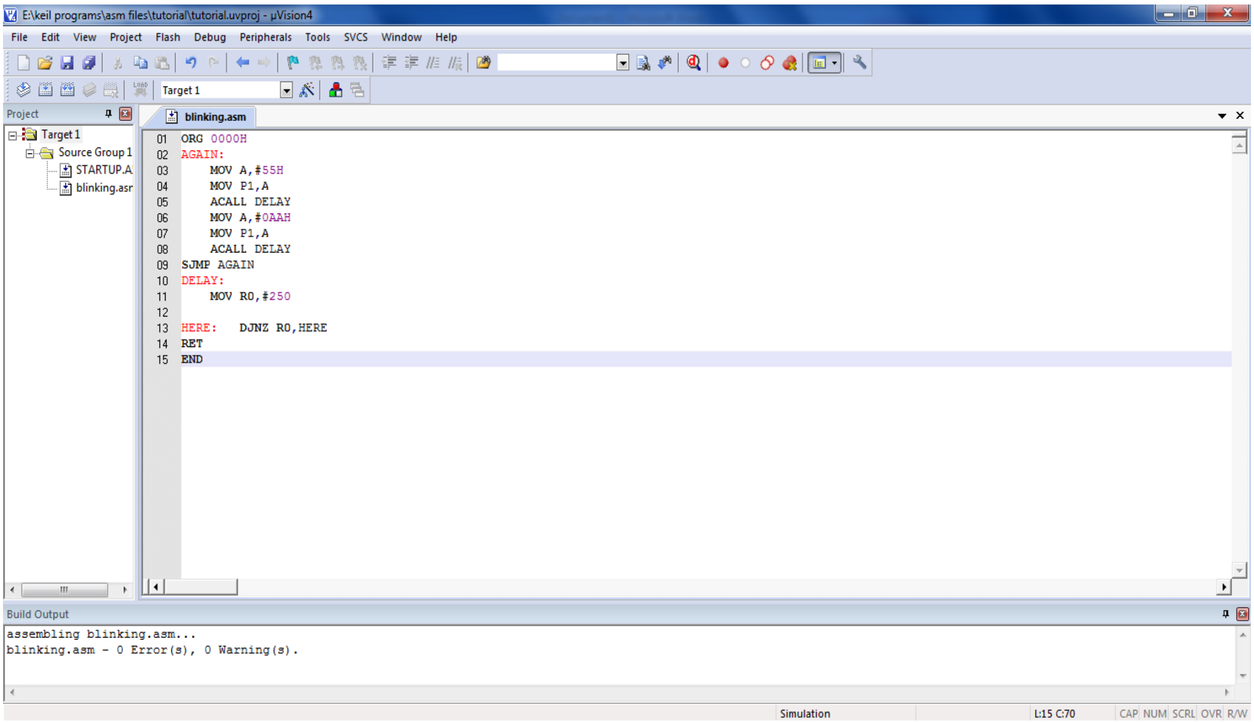
18. the source group 1 in the tree menu to ensure that the file was added t project



19. Click the project menu and click translate current active file

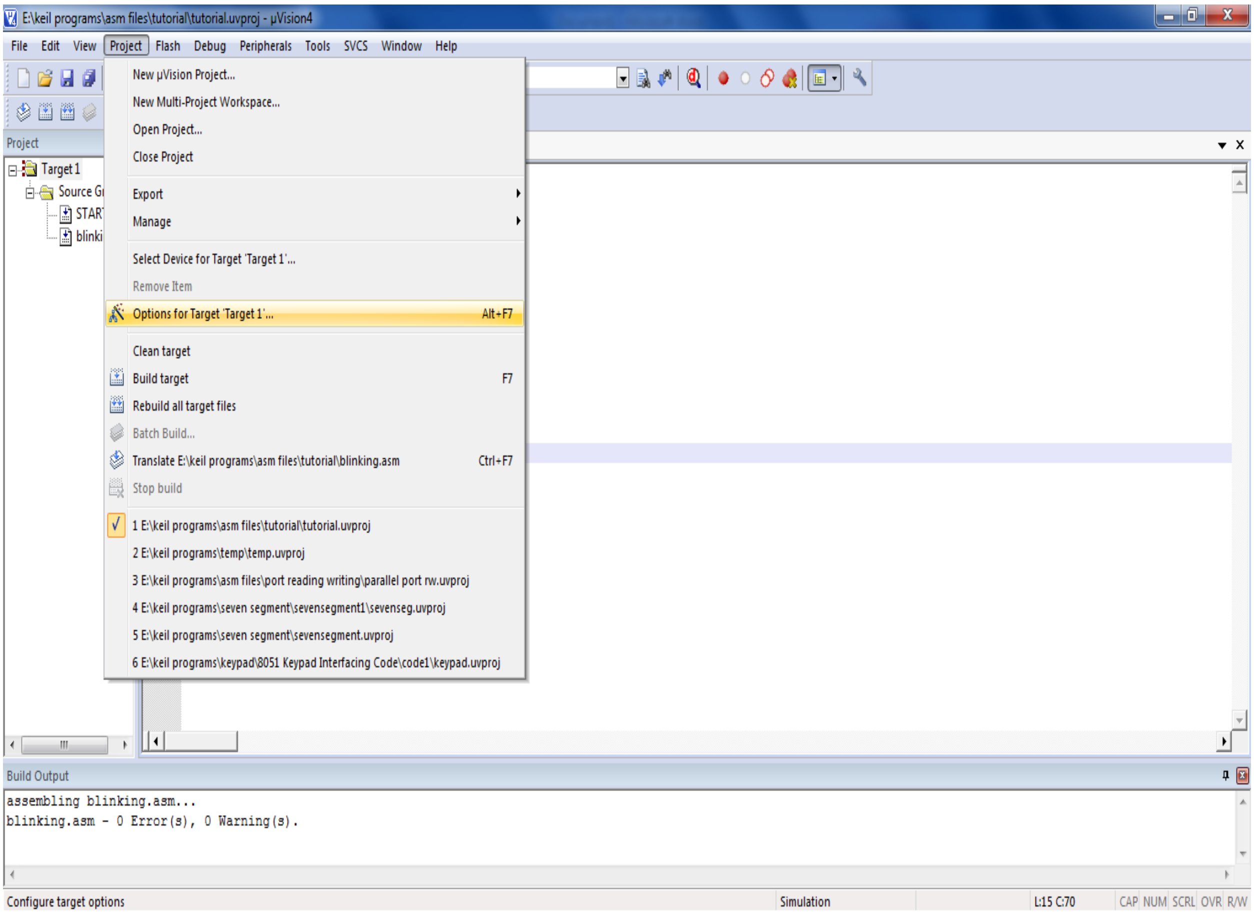


20. After click the translate in the build window shows the errors and warnings if any.

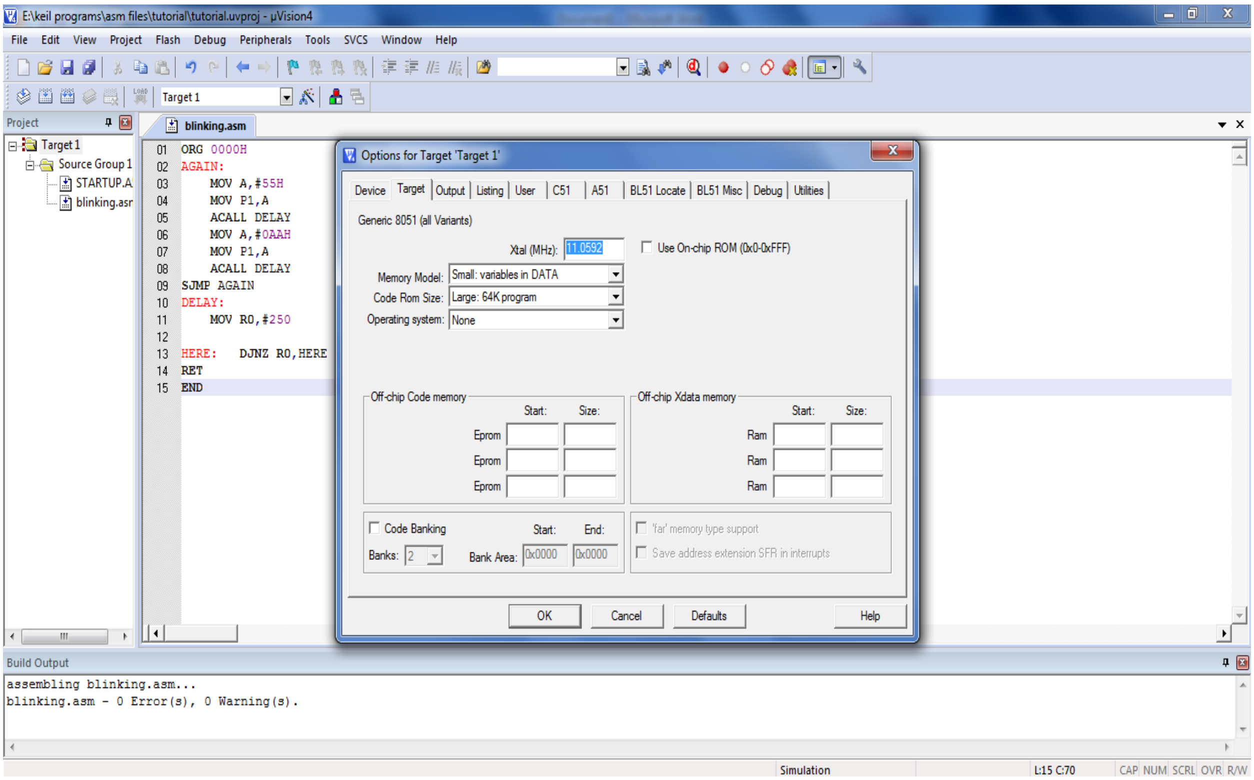


21. Click on target 1 in tree menu

22. Click on the project menu and select the options for Target1

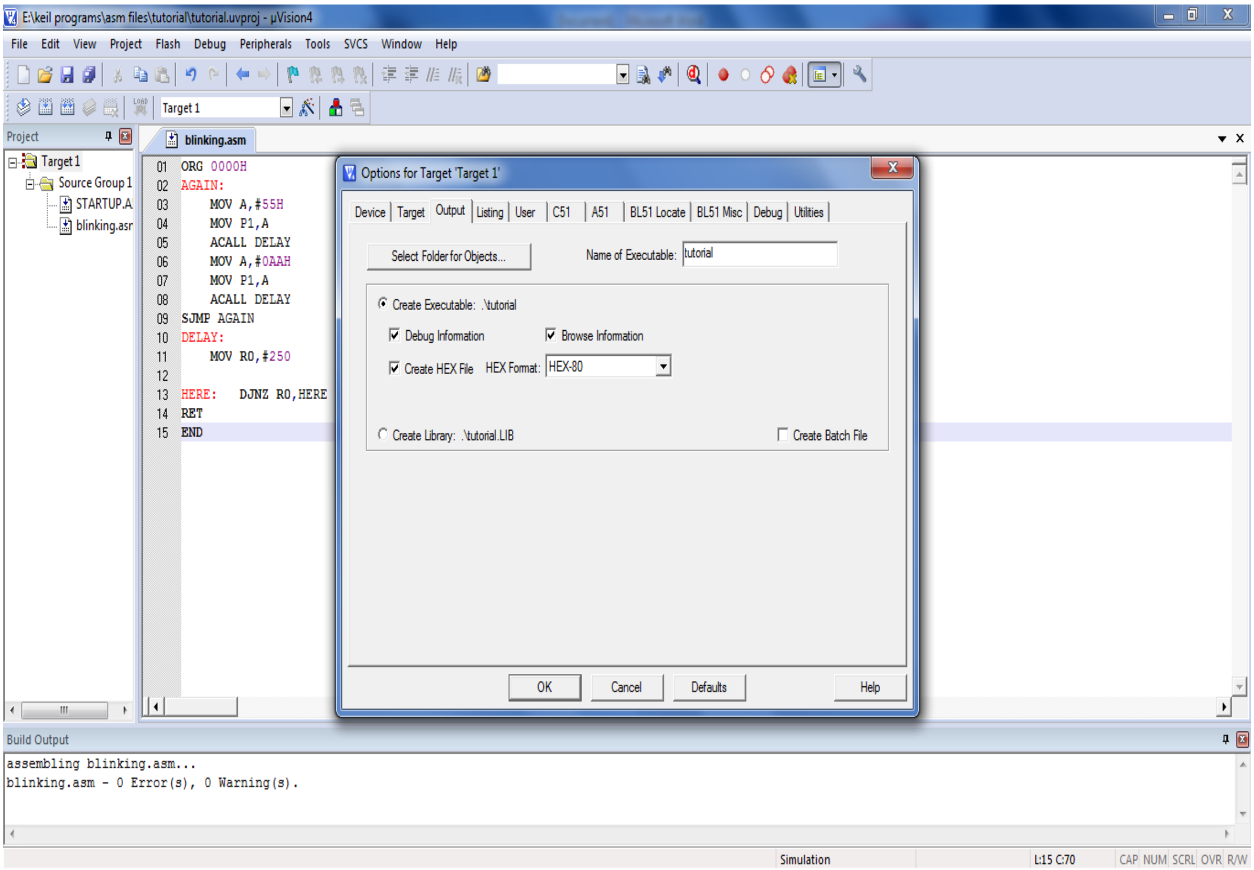


23. Select Target tab and change Xtal (Mhz) from 12.0 to 11.0592

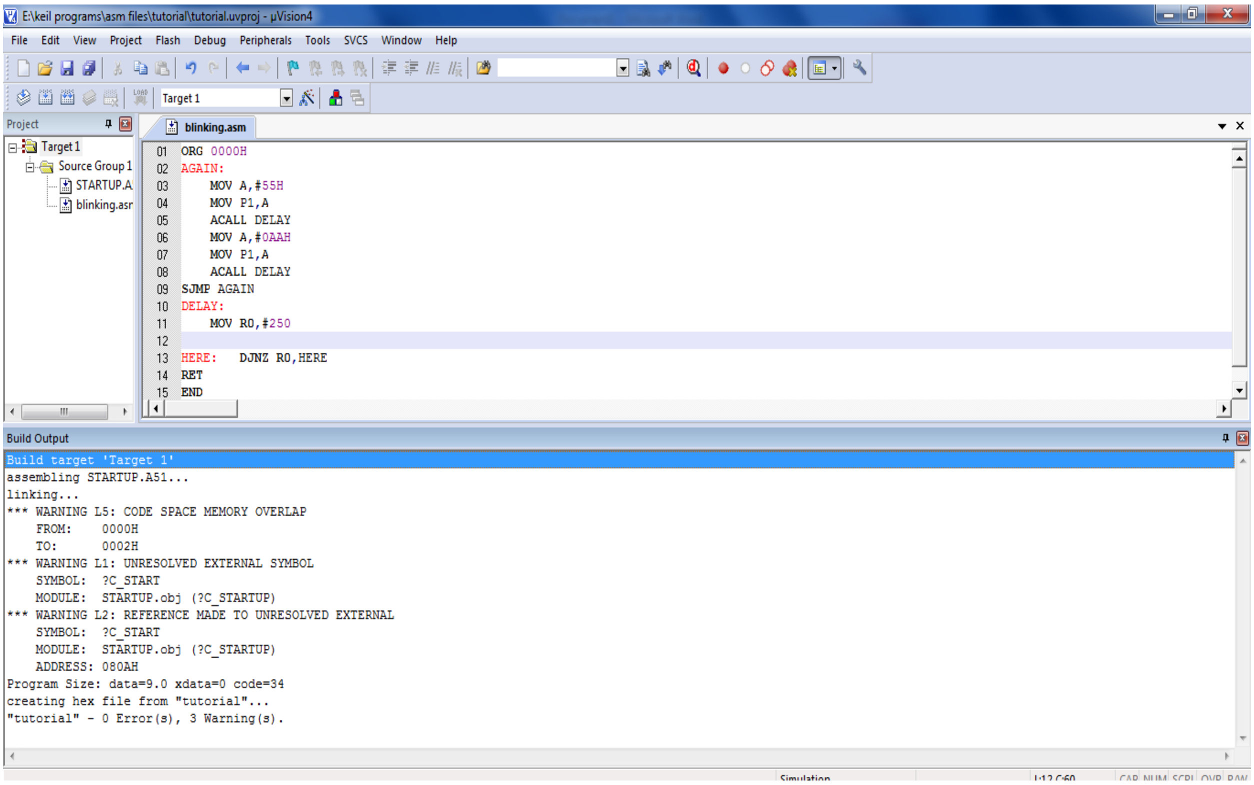


24. Select Output Tab and Click on create Hex file Check Box

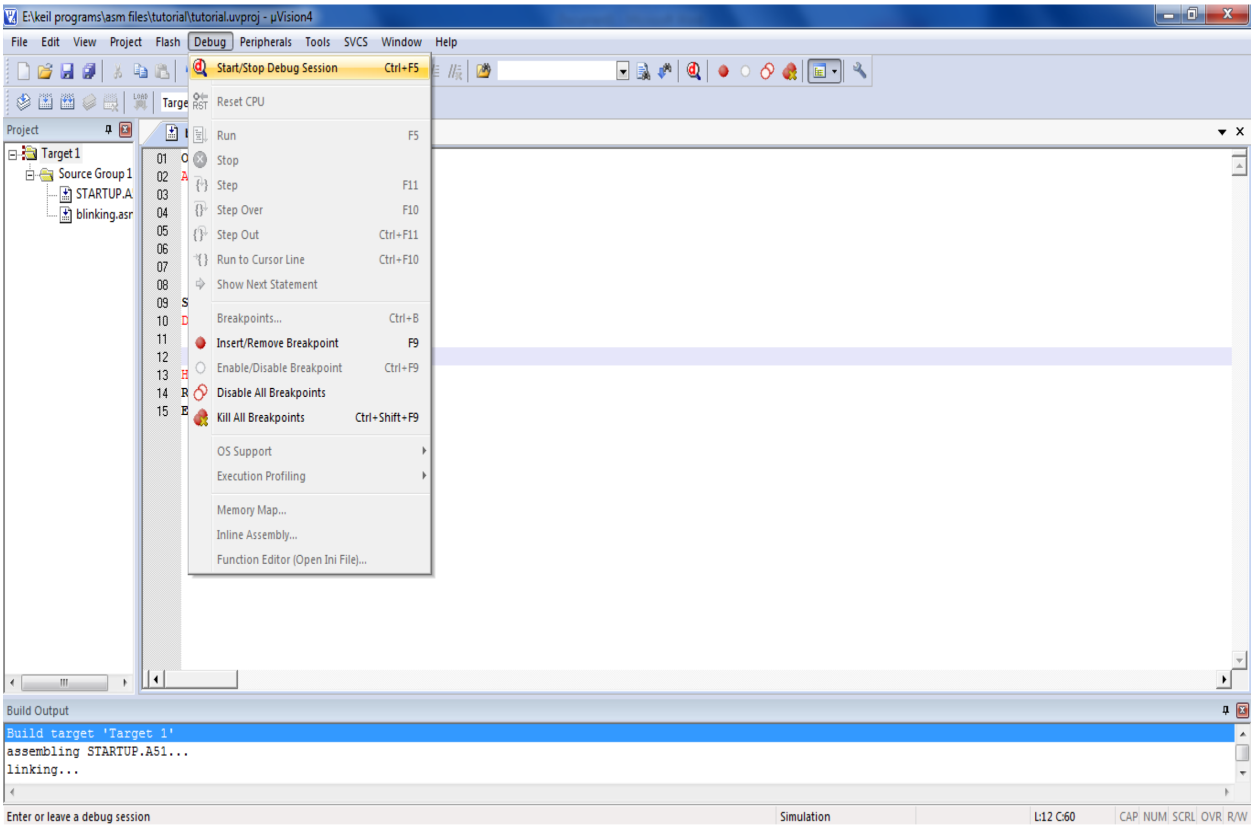
25. Click ok button



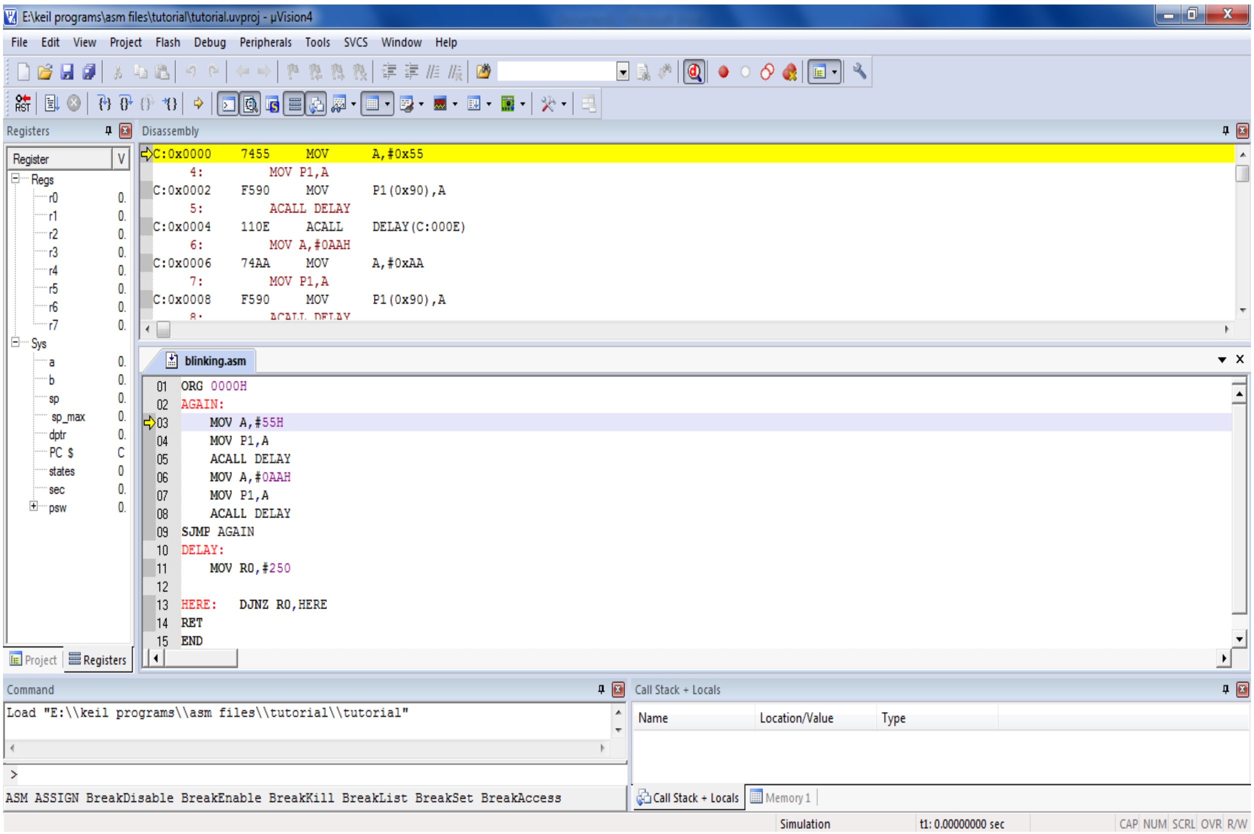
26. Click on the project menu select build target in the build window it should report errors and warnings, if any.



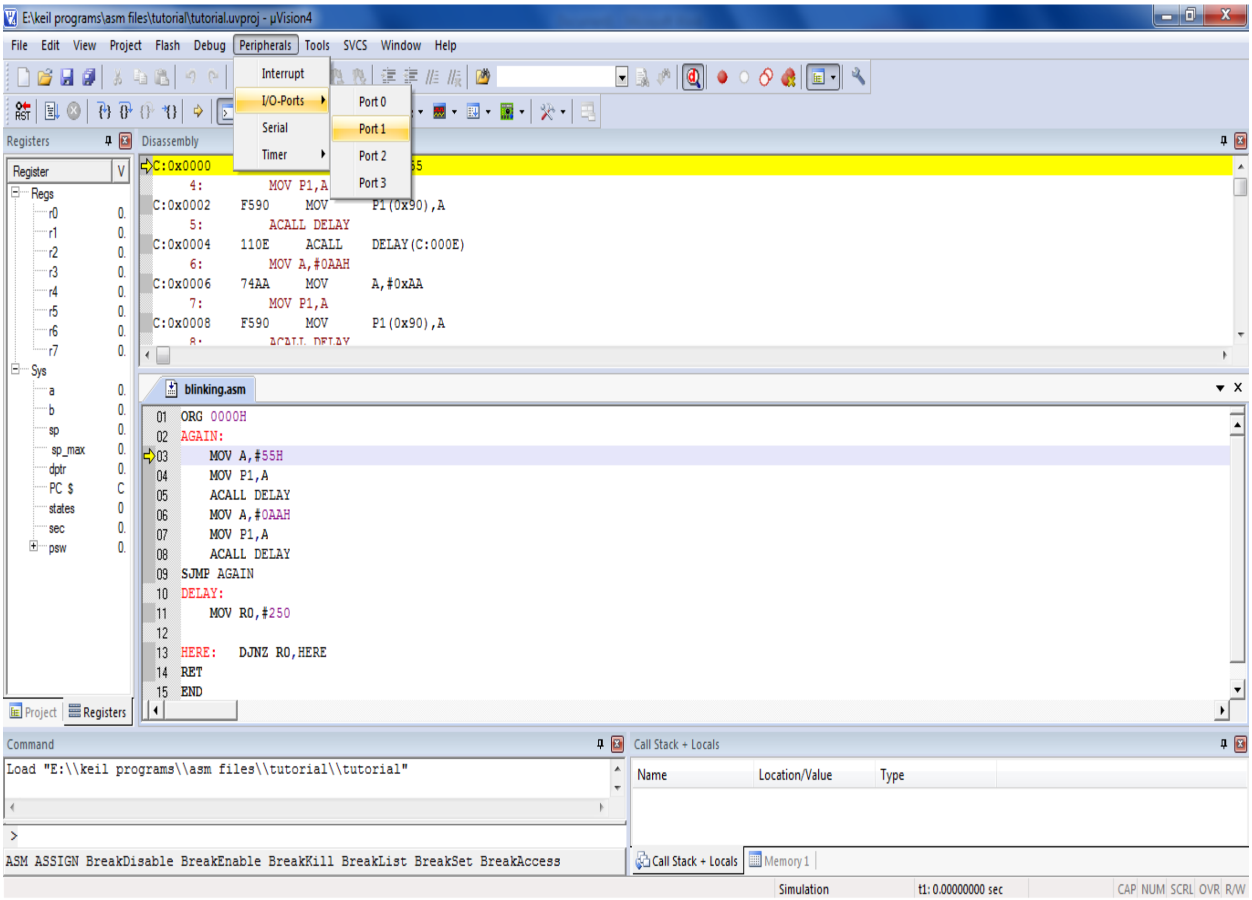
27. Click on the debug menu and select start/stop debug session



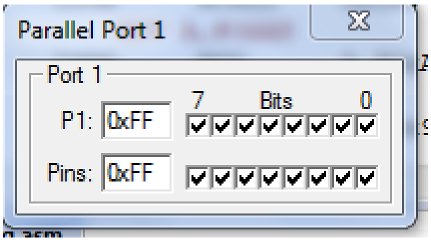
28. The keil debugger should be now be running



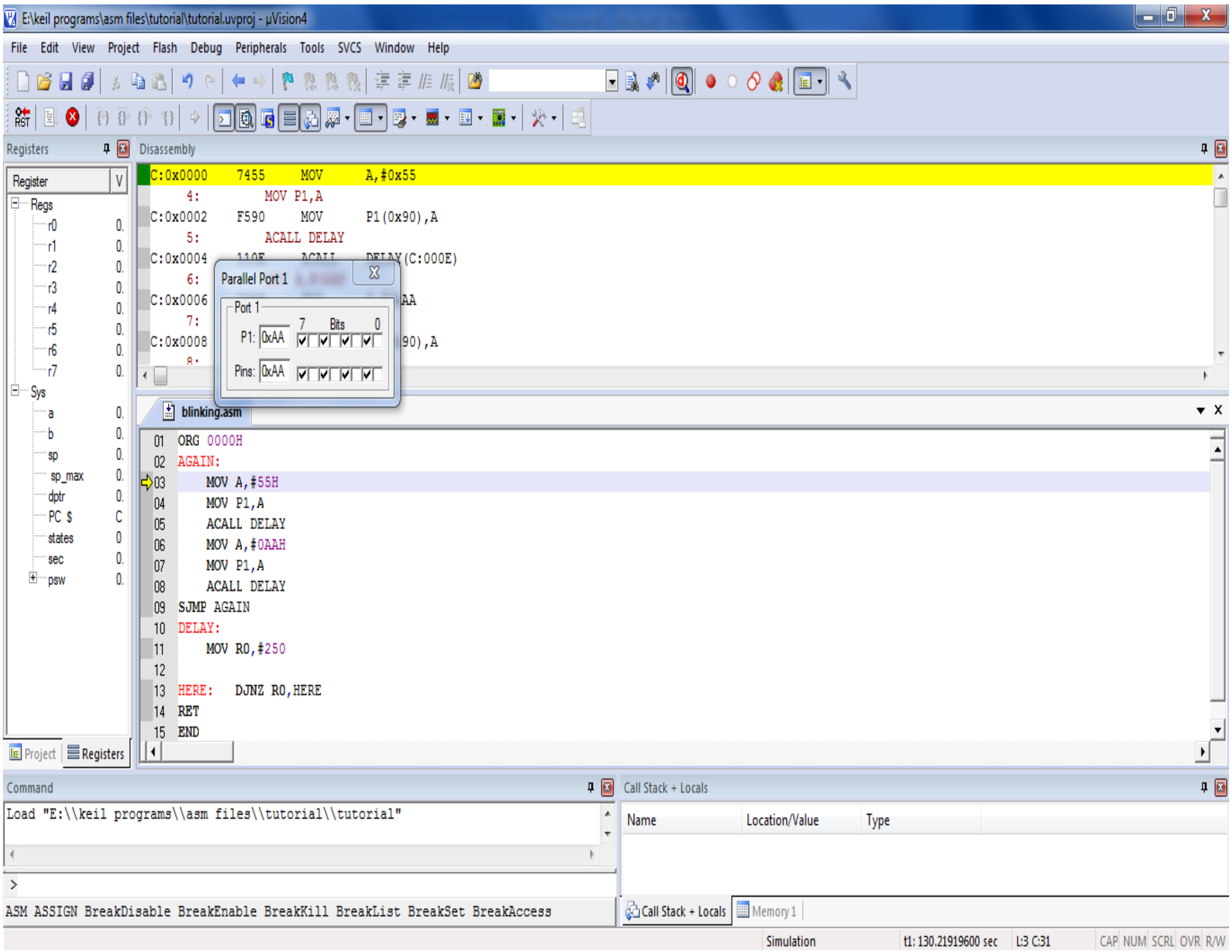
29. Click on peripherals. Select I/O ports



30. A new window should port will pop up. This represent the port and pins

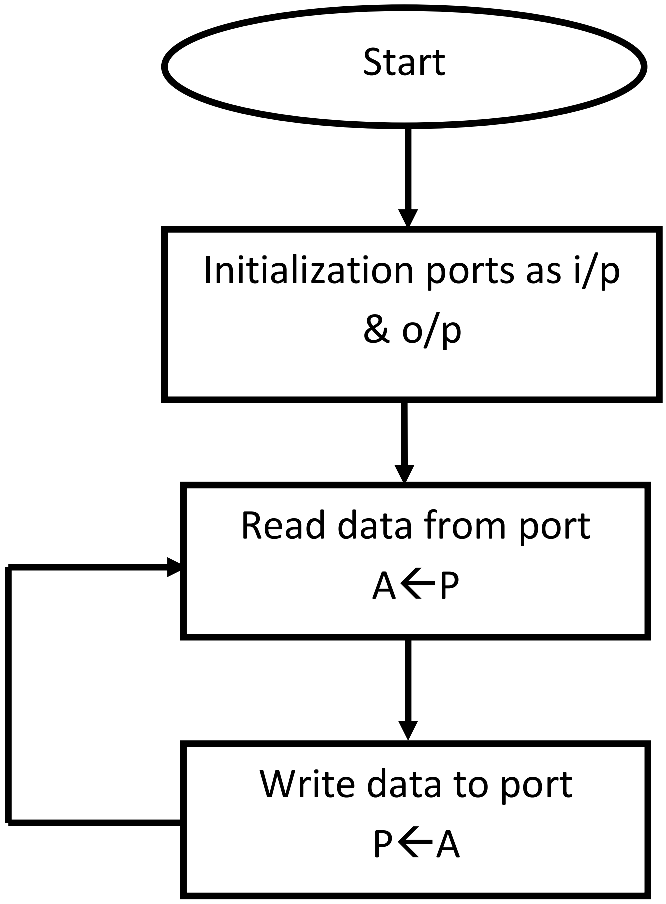


31. Press F5 on the keyboard to run the program.



32. To exit out, Click on debug menu and select start/stop debug session.

FLOW CHART:



Start
Initialization ports as i/p & o/p
Read data from port
A←P
Write data to port
P←A

Exp No:
Date:

PARALLEL PORT READ & WRITE OPERATION

ABSTRACT: Write an ALP to write and read data on a parallel port.

TOOLS USED: Keil Software

PORTS USED: P1, P2

REGISTERS USED: A (Accumulator)

ALGORITHM:

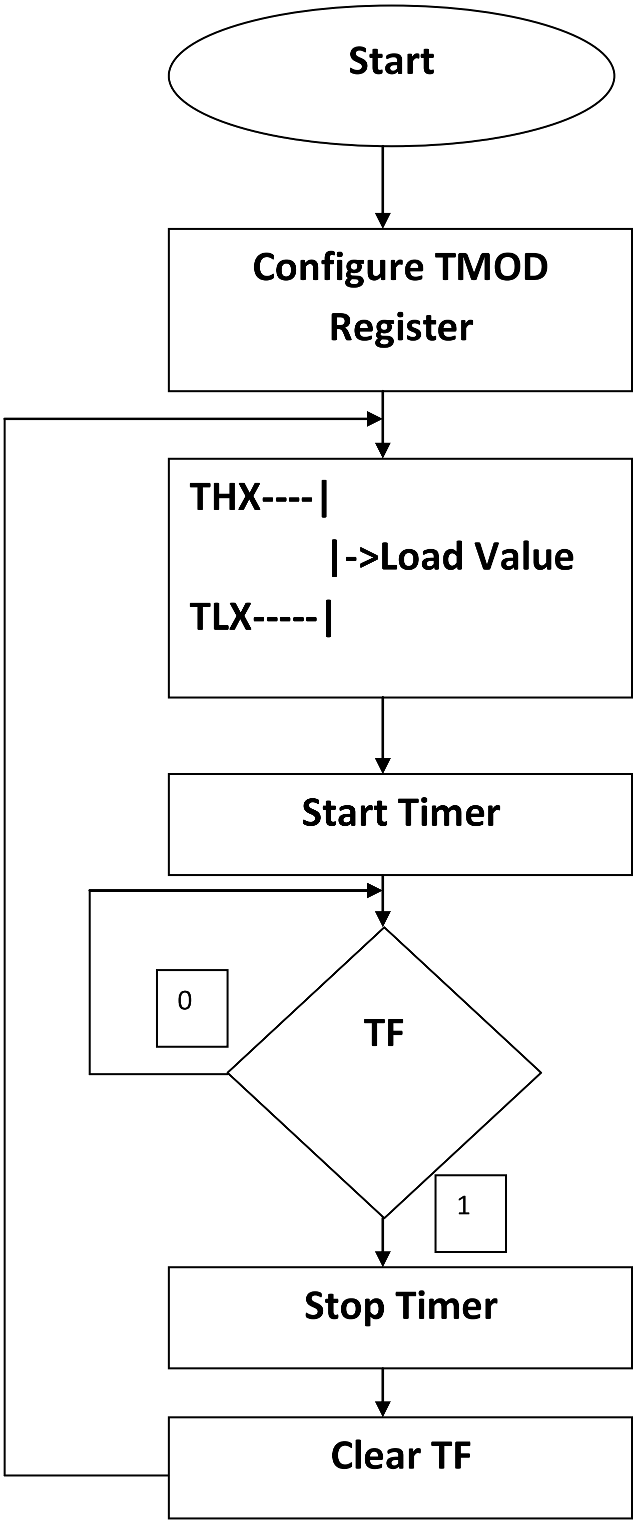
1. Write 0FFh to port selected to make it as input port.
2. Write 00h to port selected to make it as output port.
3. Read the content from port and store in to accumulator.
4. Write the content from accumulator to port.
5. Repeat Step 2

PROGRAM:

```
ORG 0000h        // orgin of program (Starting address of program)
MOV A, #0ffh         //Load ffh to Accumulator
MOV P1, A        // Content of Accumulator write to P1(Port 1) to make as a i/p port
MOV A, #00h      // Load 00h to Accumulator
MOV P2, A        //Content of Accumulator write to P2(Port 2) to make as a o/p port
BACK:
MOV A, P1        //Read the from P1 and Store into A(Accumulator)
MOV P2, A        // write from A(Accumulator) to P2
SJMP BACK        //Short jump to back
END
```

RESULT:

FLOWCHART:



Start
Configure TMOD Register
THX----|
|->Load Value
TLX-----|
Start Timer
TF
0
1
Stop Timer
Clear TF

Exp No:
Date:

TIMER MODES OPERATION

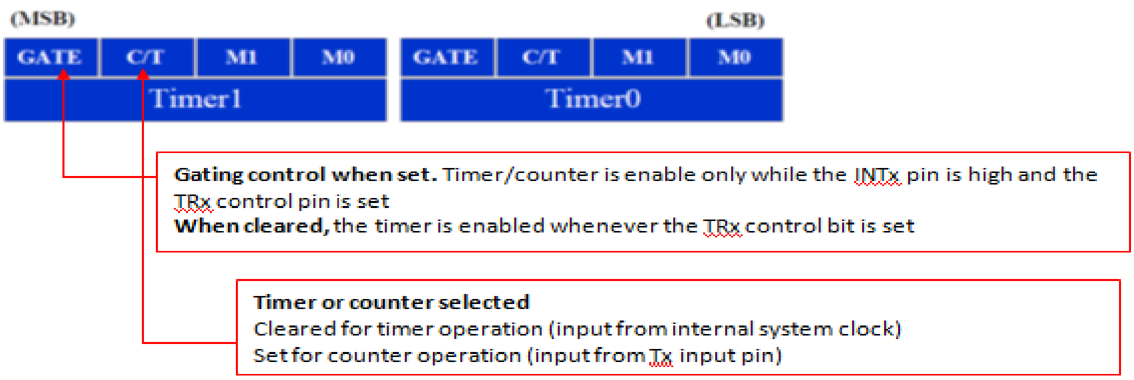
ABSTRACT: Write on ALP to create a square wave of 50% duty cycle on P1.5

TOOLS USED: Keil Software

PORTS USED: P1

REGISTERS USED: TMOD, TLX, THX

TMOD is an 8-bit register



M0	M1	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5- bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overfolws
1	1	3	**Split timer mode**

ALGORITHM:

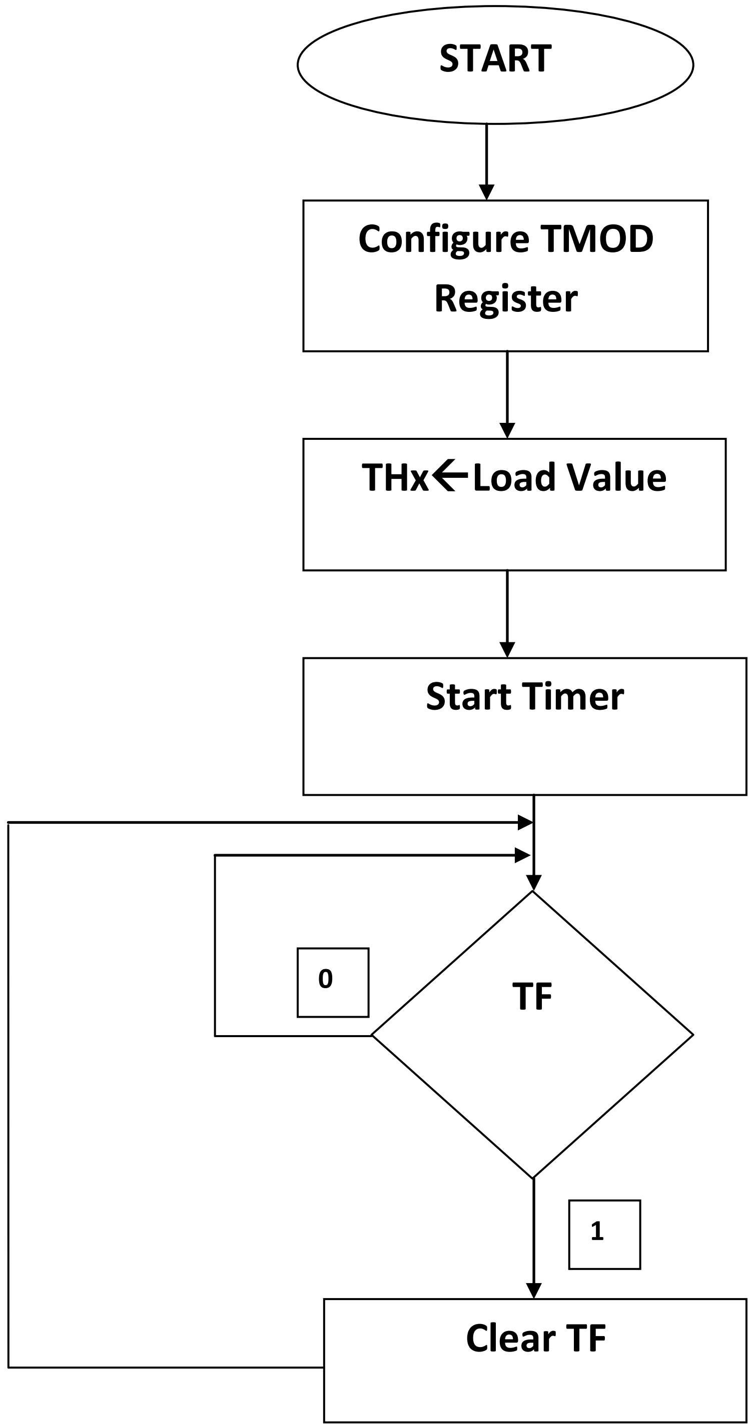
1. Configure TMOD register(Select Timer and Mode of operation)
2. Load registers THX, TLX with initial count.
3. Start the Timer
4. Monitor TF for high
5. Stop the Timer
6. Clear TF
7. Repeat Step '2'

PROGRAM:

```
MOV TMOD, #01            ;Timer 0, mode 1(16-bit mode)
HERE:
    MOV TL0, #0F2H       ;TL0=F2H, the low byte
    MOV TH0, #0FFH       ;TH0=FFH, the high byte
    CPL P1.5             ;toggle P1.5
    ACALL DELAY
SJMP HERE
DELAY:
    SETB TR0             ;start the timer 0
AGAIN:
    JNB TF0, AGAIN       ;monitor timer flag 0 -until it rolls over
    CLR TR0              ;stop timer 0
    CLR TF0              ;clear timer 0 flag
RET
END
```

RESULT:

FLOW CHART:


START
Configure TMOD Register
THx←Load Value
Start Timer
TF
0
1
Clear TF

ABSTRACT: Write an ALP to generate a square wave on P1.0.

TOOLS USED: Keil Software

PORTS USED: P1

REGISTERS USED: TMOD, THx

ALGORITHM:

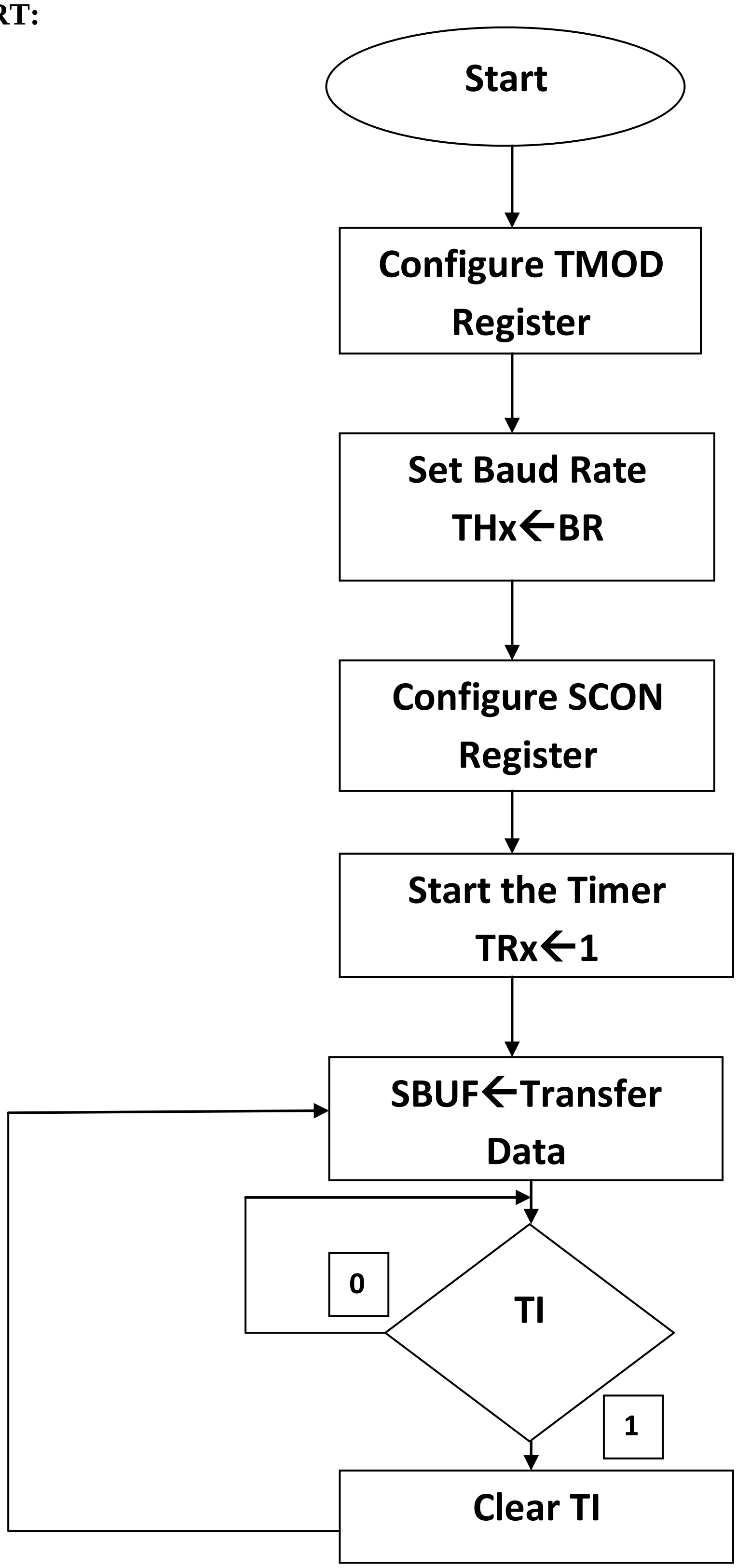
1. Configure TMOD Register
2. Load Register THx with initial count
3. Start Timer
4. Monitor TF for high
5. Clear TF
6. Repeat Step '4'

PROGRAM:

```
MOV TMOD, #20H        ;T1/8-bit/auto reload
MOV TH1, #26                  ;TH1 = 26
SETB TR1               ;start the timer 1
BACK:
        JNB TF1,BACK   ;till timer rolls over
        CPL P1.0       ;P1.0 to hi, lo
        CLR TF1        ;clear Timer 1 flag
SJMP BACK
END
```

RESULT:

FLOW CHART:


Start
Configure TMOD Register
Set Baud Rate
THx←BR
Configure SCON Register
Start the Timer
TRx←1
SBUF←Transfer Data
0
TI
1
Clear TI

Exp No:
Date:

SERIAL PORT OPERATION

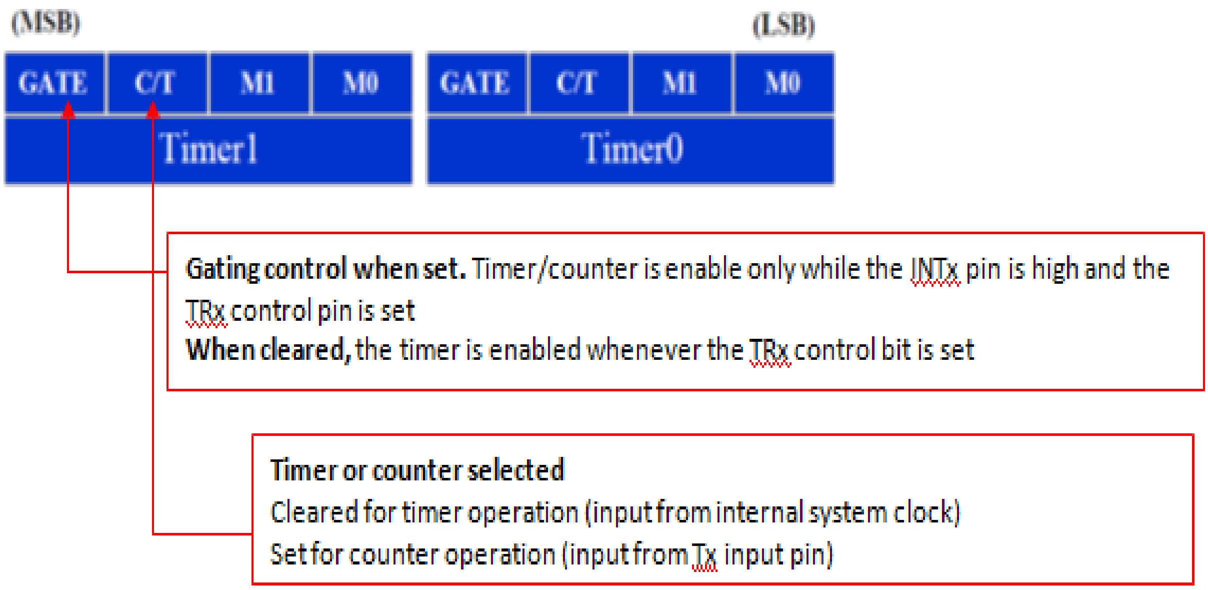
ABSTRACT: Write a ALP for the 8051 to transfer letter ‘A’ serially at 9600 baud rate, continuously

TOOLS USED: Keil Software

PORTS USED: None

REGISTERS USED: TMOD, THx, SCON, SBUF

TMOD:



M0	M1	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5- bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overfolws
1	1	3	**Split timer mode**

SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

ALGORITHM:

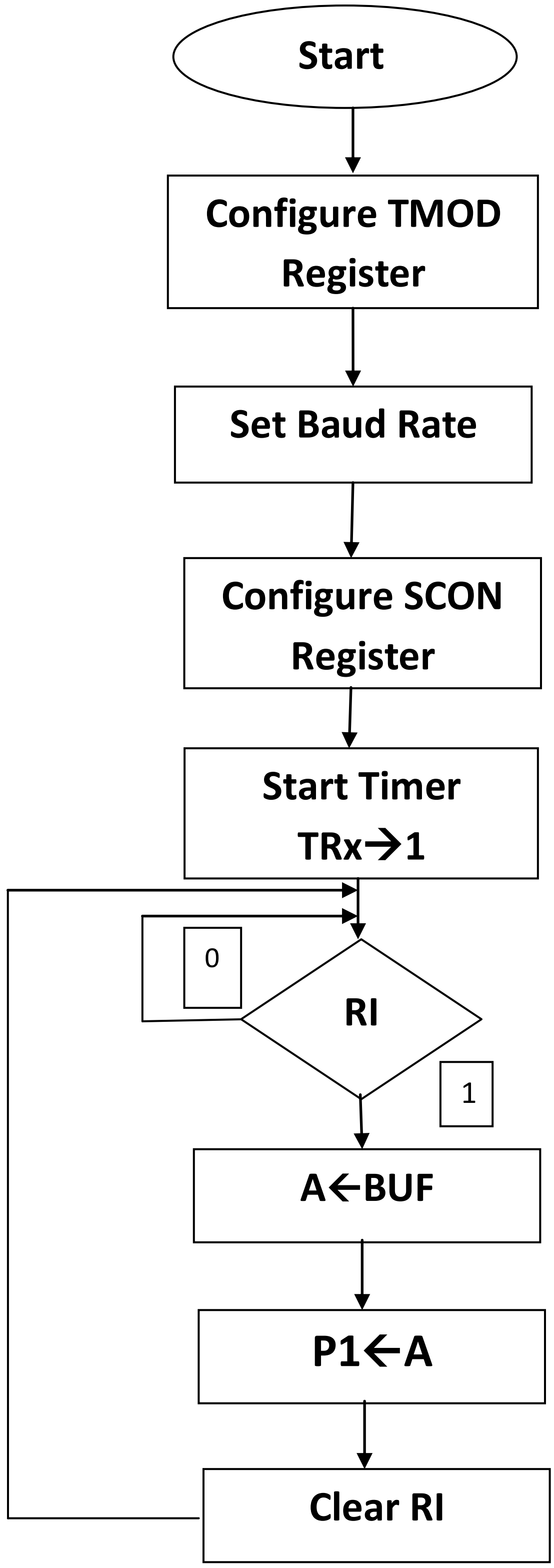
1. Configure TMOD Register (Select Timer and Mode of Operation)
2. Load THx with value to set baud rate
3. Configure SCON register according to framing of data
4. Start the timer
5. Load the transfer data in to SBUF
6. Monitor the TI bit till last bit transmitted
7. Clear the TI for next character
8. Repeat step ‘5’

PROGRAM:

```
ORG 0000H
MOV TMOD, #20H          ;timer 1,mode 2(auto reload)
MOV TH1, #-3            ;9600 baud rate
MOV SCON, #50H          ;8-bit, 1 stop, REN enabled
SETB TR1                ;start timer 1
AGAIN:
        MOV SBUF, #'A'  ;letter “A” to transfer
HERE:  JNB TI, HERE     ;wait for the last bit
CLR TI                  ;clear TI for next char
SJMP AGAIN
END
```

RESULT:

FLOW CHART:



Start
Configure TMOD Register
Set Baud Rate
Configure SCON Register
Start Timer
TRx→1
0
RI
1
A←BUF
P1←A
Clear RI

ABSTRACT: Write an ALP to receive bytes of data serially and put them in port 1, set the baud rate suitably

TOOLS USED: Keil Software

PORTS USED: P1

REGISTERS USED: TMOD, THx, SCON, SBUF, A (Accumulator)

ALGORITHM:

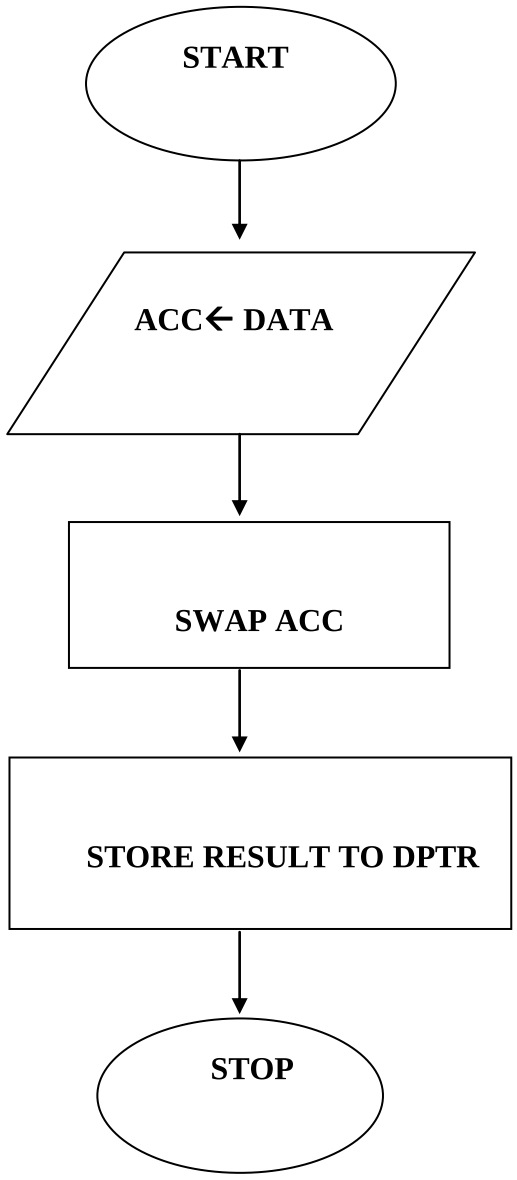
1. Configure TMOD Register (Select Timer and Mode of Operation)
2. Load THx with value to set baud rate.
3. Configure SCN register according to framing the data.
4. Start timer
5. Monitor the RI bit till last bit received
6. Load the Content present in SBUF to Accumulator.
7. Write the content of Accumulator to port1
8. Clear the RI for next character.

PROGRAM:

```
ORG 0000H
MOV TMOD, #20H   ;timer 1,mode 2(auto reload)
MOV TH1, #-6            ;4800 baud rate
MOV SCON, #50H   ;8-bit, 1 stop, REN enabled
SETB TR1          ;start timer 1
HERE:  JNB RI, HERE       ;wait for char to come in
       MOV A, SBUF        ;saving incoming byte in A
       MOV P1, A    ;send to port 1
       CLR RI             ;get ready to receive next byte
SJMP HERE          ;keep getting data
END
```

RESULT:

FLOW CHART:



Exp No:
Date:

PROGRAMS USING SPECIAL INSTRUCTIONS

ABSTRACT: Write an ALP to perform swap operation using SWAP instruction

TOOLS USED: Keil Software

PORTS USED: None

REGISTERS USED: A (Accumulator),DPTR

ALGORITHM:

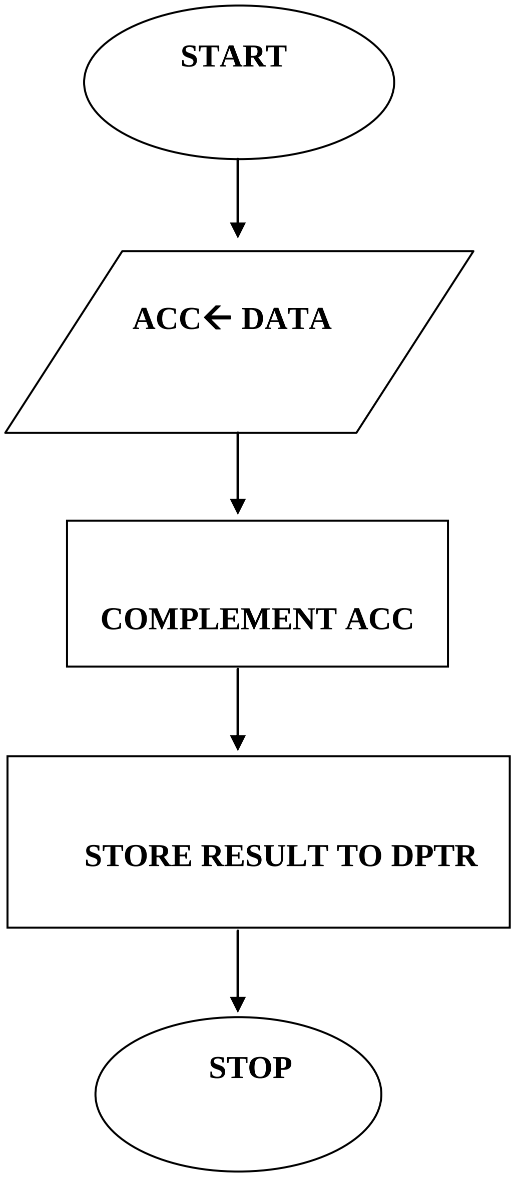
1. Load the Accumulator with an immediate value.
2. Exchange the nibbles of Accumulator using SWAP instruction
3. Store the result to memory.

PROGRAM:

```
ORG 0000H
MOV A, #35H
SWAP A
MOV DPTR,#3000H
MOVX @DPTR, A
END
```

RESULT:

FLOW CHART:



ABSTRACT: Write an ALP to perform BYTE manipulation.

TOOLS USED: Kiel Software

PORTS USED: None

REGISTERS USED: A (Accumulator), DPTR

ALGORITHM:

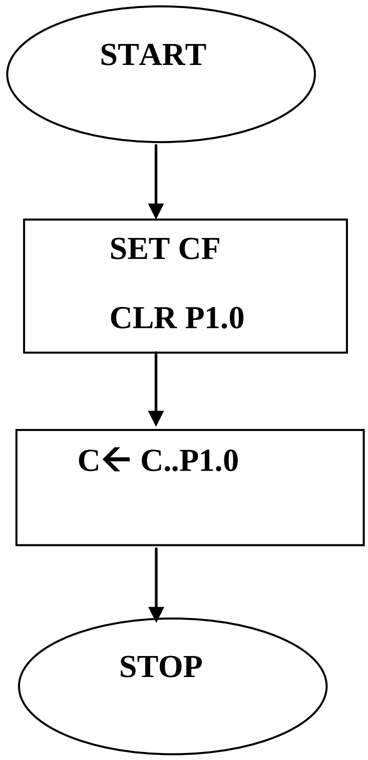
1. Load the Accumulator with an immediate value.
2. Invert the content of Accumulator using CPL instruction.
3. Store the result to memory.

PROGRAM:

```
ORG 0000H
MOV A, #55H
CPL A
MOV DPTR, #3000H
MOVX @DPTR, A
END
```

RESULT:

FLOW CHART:



ABSTRACT: Write an ALP to perform BIT manipulation, SET/RESET operations.

TOOLS USED: Kiel Software

PORTS USED: None

REGISTERS USED: None

ALGORITHM:

1**.** Set the carry flag(C)

2. Clear the bit P1.0

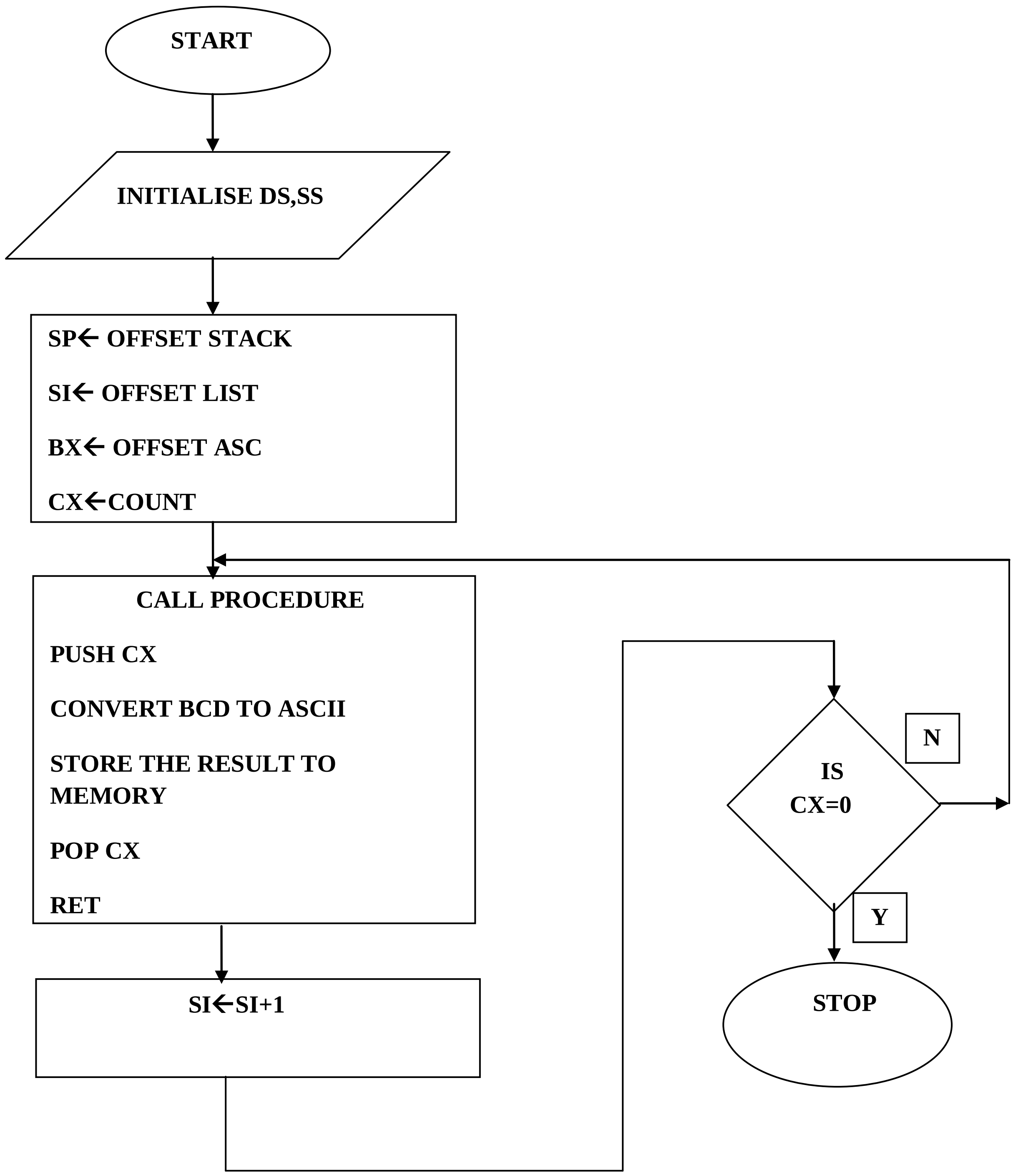
3. Perform AND operation between C and P1.0.

PROGRAM:

```
ORG 0000H
SETB C
CLR P1.0
ANL C, P1.0
END
```

RESULT:

FLOW CHART:


START
INITIALISE DS,SS
SP← OFFSET STACK
SI← OFFSET LIST
BX← OFFSET ASC
CX←COUNT
CALL PROCEDURE
PUSH CX
CONVERT BCD TO ASCII
STORE THE RESULT TO MEMORY
POP CX
RET
SI←SI+1
IS CX=0
N
Y
STOP

Exp No:
Date:

PROGRAMS USING PROCEDURES

ABSTRACT: Assembly language to perform BCD to ASCII using procedures.
PORT USED: None
REGISTERS USED: AX, BX,CX
ALGORITHM:
Step1: Start
Step2: Initialize data segment & stack segment
Step3: SP pointed to stack
Step4: SI pointed to given array in data segment
Step5: Initalize CX with number of bytes in the array
Step6: Call a procedure that converts the byte of the array to its equivalent ASCII value which is stored to memory
Step7: Increment SI
Step8: Decrement CX, jump to step 6 if non zero otherwise go to next step
Step9: Stop

MANUAL CALCULATIONS:

PROGRAM:

```
ASSUME CS:CODE,DS:DATA,SS:STACK
DATA SEGMENT
      LIST DW 5867H
      ASC  DB 04H DUP(00)
DATA ENDS
STACK SEGMENT
       DW 40H DUP(00)
      TOP_STACK LABEL WORD
STACK ENDS
CODE SEGMENT
START:   MOV AX,DATA
         MOV DS,AX
         MOV AX,STACK
         MOV SS,AX
         MOV SP,OFFSET STACK
        LEA SI,LIST
         LEA BX,ASC
       MOV CX,0002H
BACK:   CALL ASCII
         INC SI
        LOOP BACK
        JMP NEXT
      ASCII PROC NEAR
        PUSH CX
```

CODE TABLE:

Physical Address		Label	Hex Code	Mnemonic operand	Comments
Segment Address	Offset Address				

```
MOVAH,AL
MOV CL,04
AND AL,0FH
AND AH,0F0H
ROR AH,CL
OR AX,3030H
MOV [BX],AL
INC BX
MOV [BX],AH
INC BX
POP CX
RET
ASCII ENDP
NEXT: MOV AH,4CH
INT 21H
CODE ENDS
END START
```

RESULT:

BIOS INT 10H SUBPROCEDURES AND PARAMETERS

AH	FUNCTION
00H	Set display mode using value in AL AL = 0 40 x 25 BW AL = 1 40 x 25 COLOR AL = 2 80 x 25 BW AL = 3 80 x 25 COLOR AL = 4 320 x 200 COLOR AL = 5 320 x 200 BW AL = 6 640 x 200 x 2 COLOR AL = 7 80 x 25 BW AL = D 320 x 200 x 16 COLOR AL = E 640 x 200 x 16 COLOR AL = F 640 x 350 BW AL = 10 640 x 350 x 16 COLOR AL = 11 640 x 480 x 2 COLOR AL = 12 640 x 480 x 16 COLOR AL = 13 320 x 200 x 256 COLORS
01	Set cursor type CH = bottom line number for cursor CL = top line number for cursor
02	Set cursor position DH = row, DL = column, BH = page
03	Read cursor position BH = page Returns: DH = row, DL = column CH, CL = cursor mode
04	Read light pen position Returns: AH = light pen active DH = character row of pen DL = character column of pen CH = raster scan line # BX = pixel column number
05	Select active display page AL = desired page
06	Scroll active page up, blanks at bottom AL = number of lines to scroll AL = 0 blanks entire window CH = row, CL = column of upper left corner of scroll. DH = row, DL = column of lower right corner of scroll. BH = attribute to be used on blanked lines
07	Scroll active page down, blank top line AL = number of lines to scroll AL = 0 blanks entire window CH = row, CL = column of upper left corner of scroll DH = row, DL = column of lower right corner of scroll BH = attribute to be used on blanked lines

AH	FUNCTION
08	Read character and attribute at cursor BH = display page Returns: AH = attribute, AL = character
09	Write character and attribute at cursor BH = display page, CX = number of characters AL = character, BL = attribute
0AH	Write just character at cursor position BH = display page, CX = number of characters AL = character
0BH	Set CGA color palette BH = 0 - set background color BL = color BH = 1 - select color set BL = color set - 0 or 1
0CH	Write pixel at graphics cursor DX = row, CX = column, AL = color
0DH	Read pixel value DX = row, CX = column Returns AL = pixel value
0EH	Write character and advance cursor AL = character, BH = page(text mode) BL = color(graphics modes)
0FH	Get current video state Returns: AL = current video mode AH = number of character columns BH = current display page
10H	Set EGA/VGA palette registers AL = 00 - program a single palette reg AL = 01 - program border color register AL = 02 - program all palette registers AL = 03 - enable blink or intensify AL = 07 - read a single palette register AL = 08 - read the border color register AL = 09 - read all palette registers AL = 10H - program a single VGA color reg AL = 12H - program several VGA color regs AL = 13H select color subset AL = 15H - read a single VGA color reg AL = 17H - read several VGA color regs AL = 1AH - get color page state AL = 1BH - convert color register set to gray scale values
11H	Load Character generator Subfunction number determines character set. For example, if AL = 3, value in BL determines which of four EGA character sets is loaded.