GOVERNMENT POLYTECHNIC, AMRAVATI

(An Autonomous Institute of Government of Maharashtra)

NBA Accredited Institute

Certificate

Name of Department: Computer Science and Engineering.

This is to certify that Mr. Ayush Shashikant Bulbule Identity Code 19CM007 has completed the practical work of the course CM3409 Microprocessors during the Academic year 2020-21.

Signature of the Teacher

Date: *who taught the examinee* *Head of Department*

Index

Name: Ayush Shashikant Bulbule

Roll No: 19CM007

Practical no 1.

Aim: Perform the execution & implementation of arithmetic instructions.

Software Requirement: TASM version 1.4.

Theory :-

Instructions to perform addition.

1. Add destination source: This is an instruction used to perform addition of binary values from source content with destination contents and share result in the specified destination register.

   Syntax:
   Destination = Source + Destination +CF

Program Code :

- Model small
- Stack
- Data

```
Code
mov Ax, 0001h
mov Bx, 002h
Add Ax, Bx
hlt
code ends
end
```

Instructions to perform Subtraction:

1. SUB destination, source:

This is an instruction used to perform subtraction of binary values of source to from destination, and store the result in the specified loc. / register.
Syntax:
Destination = Destination – Source – CF

Program:

- model small
- stock
- data

code
mov Ax, 001h
mov Bx, 002h
Sub Ax, Bx
hlt
code ends
end

<u>To create a assembly file:</u>
edit add.asm 'asm' is an extension for assembly file.

'C: \TASM \ ADD.ASM'

Directory   Assembly file

- modelsmall } indicates that the memory available is 64 kbs for both data segment as well as code segment.
- Data } indicates description of data segment starts

a db 08h   addition of two nos / a is a variable which will occupy 1 byte of memory out of 64 Kbs /
b db 02h   / 08h is its value /y for b db 02h

MOV DS, AX – contain add which is moved to DS
MOV Ah, a – move calue of a from memory to Ah register

Add Ah, b –> [Ah] + b
mov sum, Ah –> [Ah] -> sum
mov, 4ch
int 21h
end start
ends

@ data means address of data -> address of memory location from where its data ‘a, b, sum’ are stored in the memory that address is to be moved in register Ax.

file -> save
exit -> console window

Assemble the program / compile the tasm add.asm -> enter -> edit add.asm

tlink add -> enter -> td add.asm for execution.

Instructions to perform Multiplication:

1. Mul: -This is an instruction used to multiply unsigned byte by byte / word by word.
   Syntax:
   MUL multiplier

2. IMUL: This is an instruction used to multiply signed byte by byte / word by word
   Syntax:
   IMUL multiplier.

Program:

- model small
- stack
- data

code

mov Ax, 0001h
mov Bx, 0002h
Mul Ax
hlt
code ends
end

Instructions to perform Division:

1. DIV:

This instruction is used to divide the unsigned word by byte or unsigned double word by word.
Syntax:
DIV divisor

2. IDIV: It is an instruction used to divide the signed word by byte or signed double word by word.

   Syntax:
   IDIV divisor

Program:

- model small
- stack
- data

code
mov Ax, 001h
mov Bx, 0002h
DIV Ax
hlt
code ends
end

Conclusion: Thus, we have performed the execution and implementation of arithmetic instructions.

Name: Ayush Shashikant Bulbule

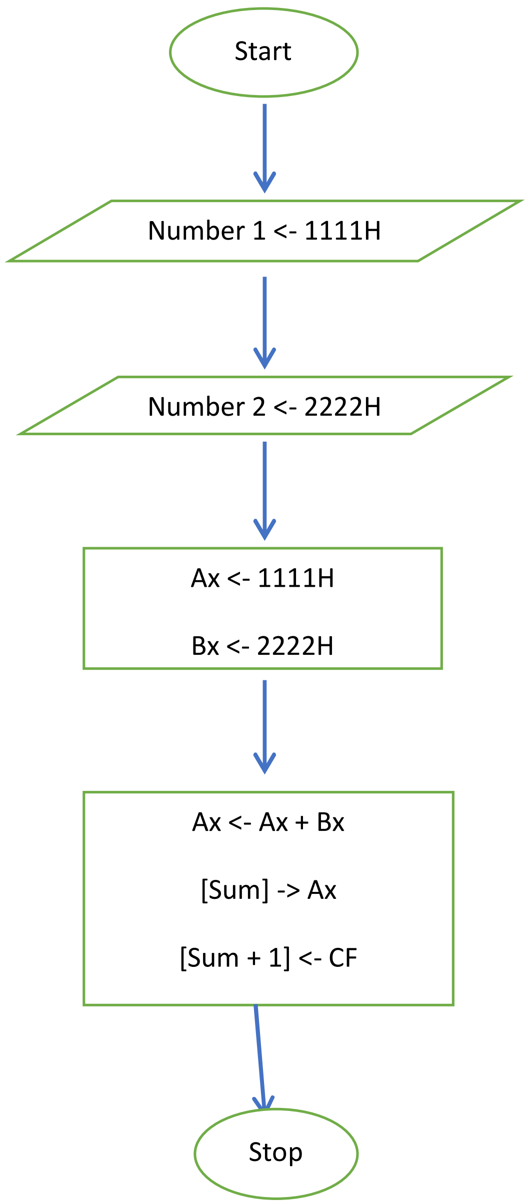
Roll No: 19CM007

Practical no 2.

Aim: Perform an assembly language program for adding 2 numbers.

Software Requirement: TASM version 1.4.

Flow Chart:



Theory:

<u>Algorithm –</u>

Step 1: Initialize the data and code segment.

Step 2: Read the no. 1 i.e. data word from Data Segment.

Step 3: Read the no. 2 i.e. data word from Data Segment.

Step 4: Assume the arithmetic / logical instructions in the code segment procedure.

Step 5: Get Number 1 in Ax and Number 2 in Bx.

Step 6: Add two nos.

Step 7: Store the 16-bit sum and exit the program with return code.

Step 8: End of Code Segment.

Step 9: End of Data Segment.

<u>Program:</u>

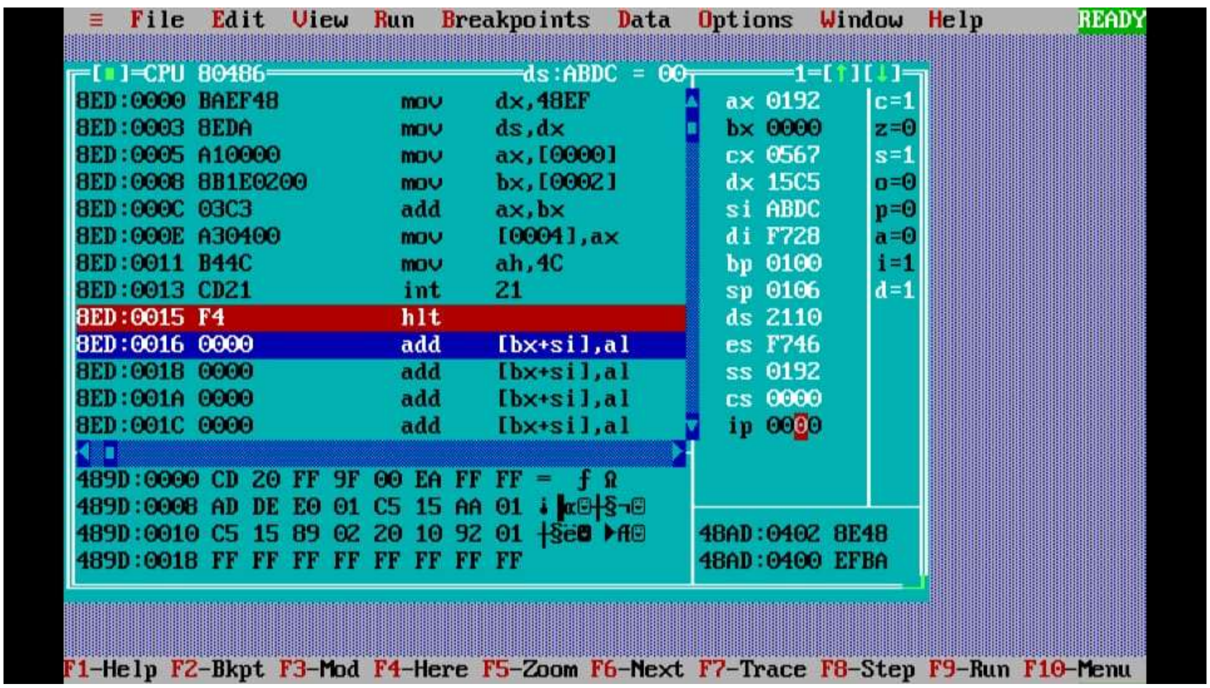
- model small
- stack

data segment
number 1 Dw 1111H
number 2 Dw 2222H
Sum Dw?
data ends

Code Segment
Assume cs: code, ds: data

Start:
Mov Dx, data
Mov Ds, dx
Mov Ax, number 1
Mov, Bx, number 2
Add Ax, Bx
Mov Sum, Ax
INC Exit
Mov Sum +1, 01

Output:



Conclusion: Thus, we performed an assembly language program for adding 2 numbers.

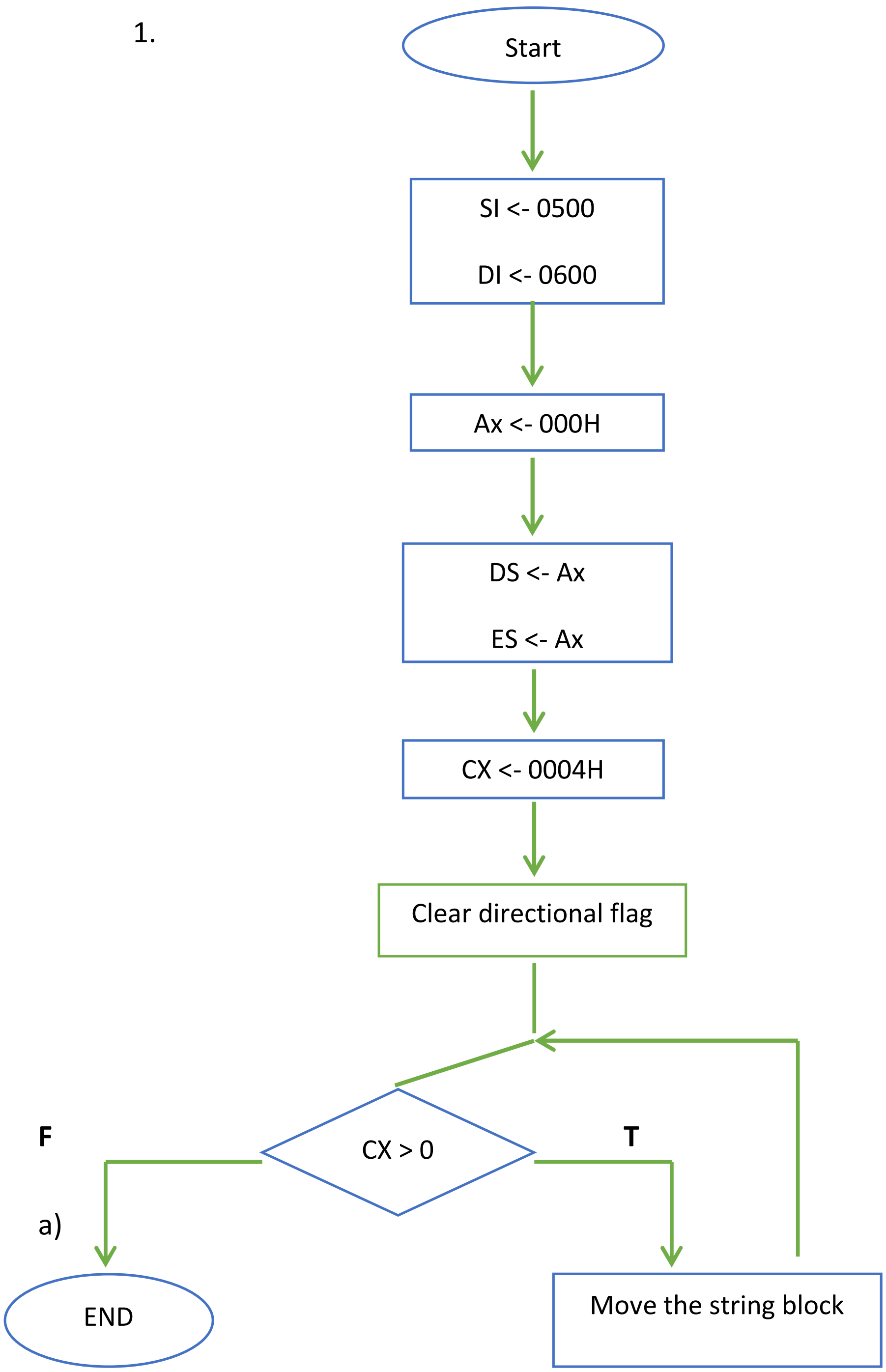
Practical no 3.

Aim: Write an assembly language program for data transfer group (a) Byte transfer (b) Block transfer (c) Reverse transfer.

Software Requirement: TASM version 1.4.

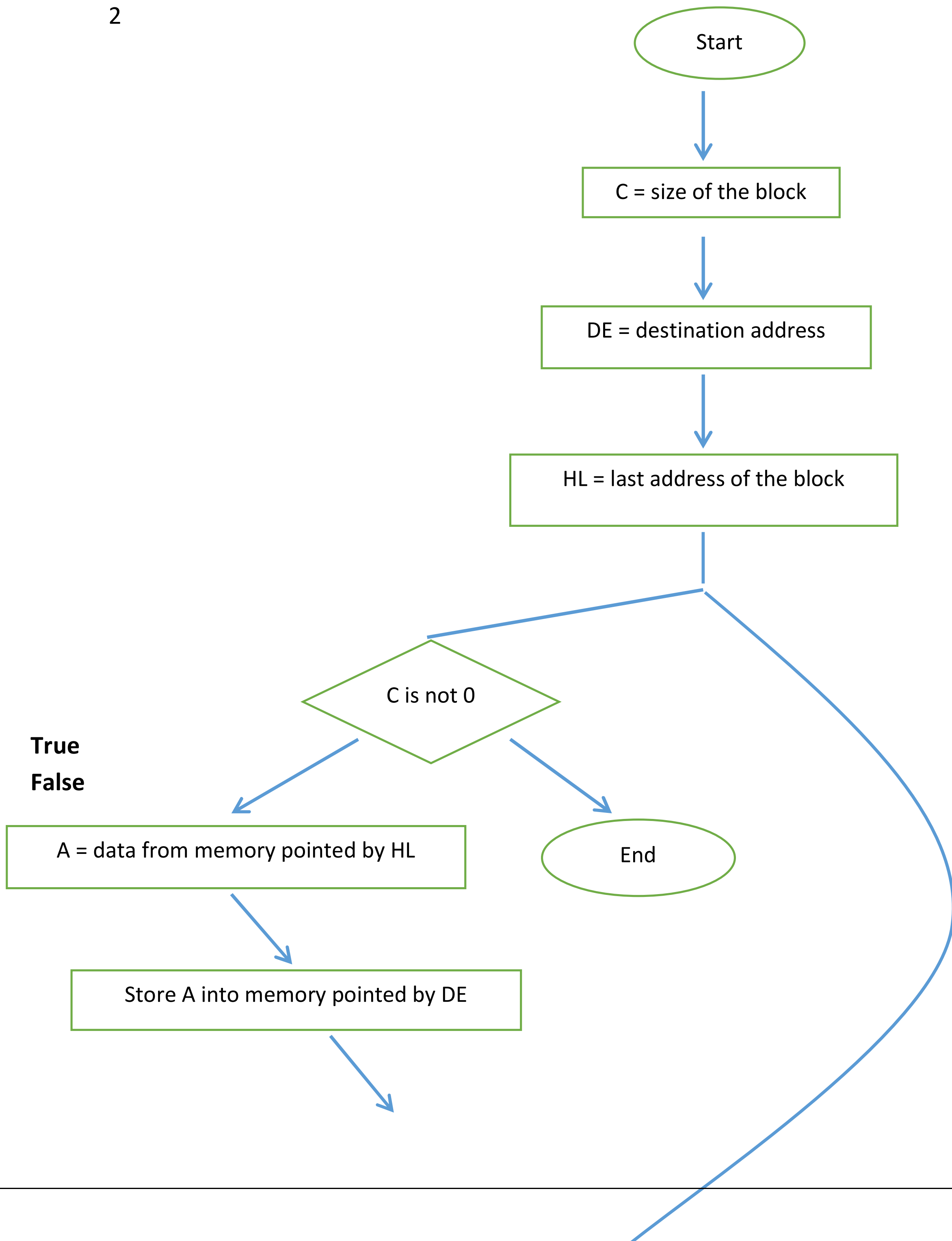
Flow Chart:

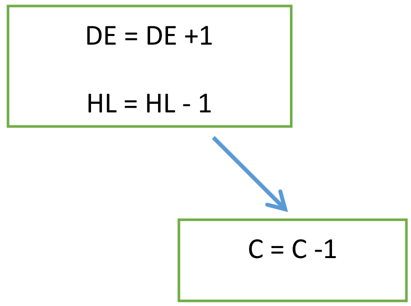
1.



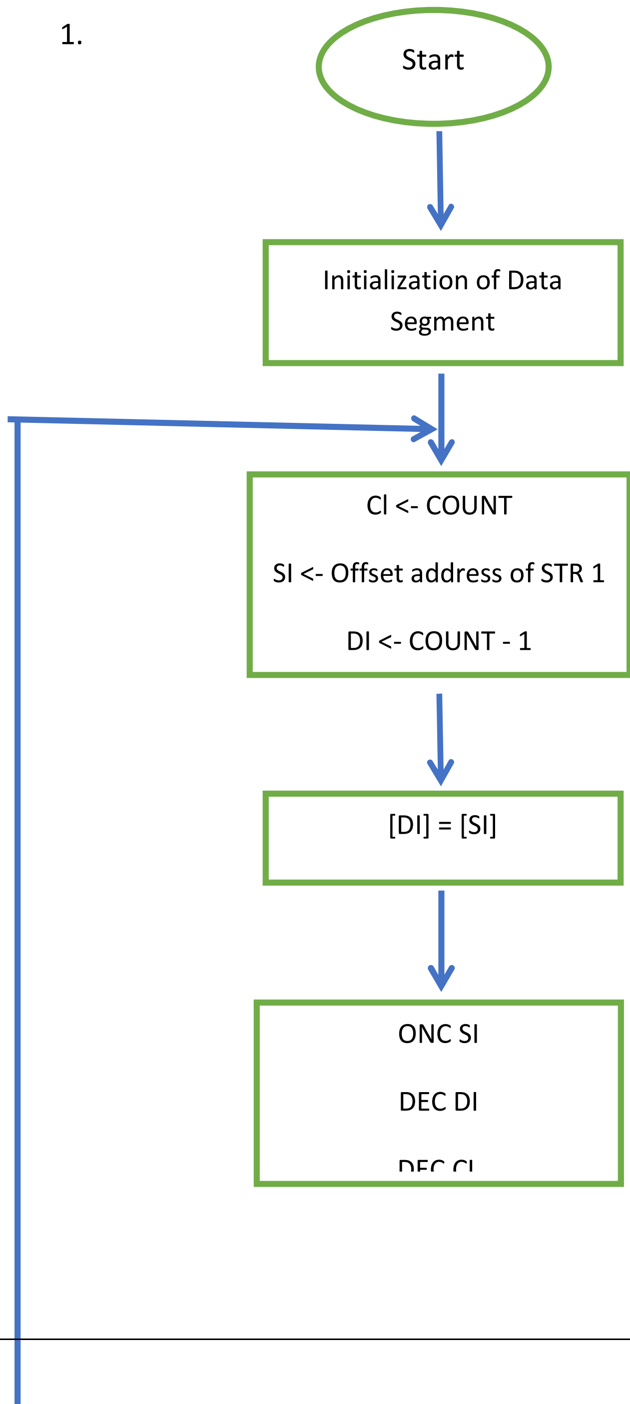
a)

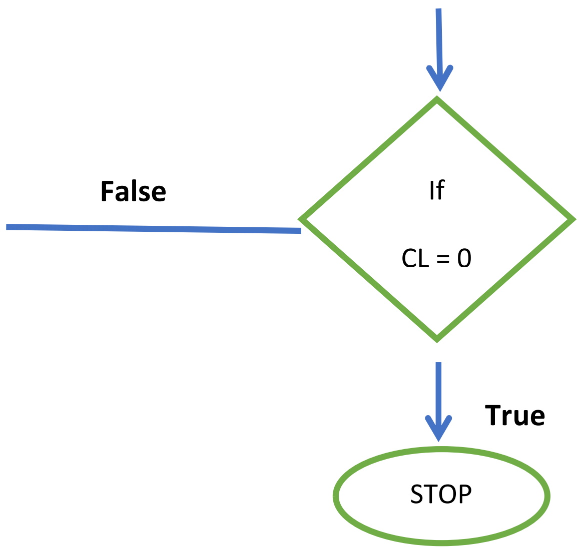
2


Start
C = size of the block
DE = destination address
HL = last address of the block
C is not 0
True
False
A = data from memory pointed by HL
End
Store A into memory pointed by DE


DE = DE +1
HL = HL - 1
C = C -1

1.


Start
Initialization of Data Segment
Cl <- COUNT
SI <- Offset address of STR 1
DI <- COUNT - 1
[DI] = [SI]
ONC SI
DEC DI



<u>Theory:</u>

Instructions to perform data transfer:

1. Mov - Used to copy the byte or word from the provided source to the provided destination.

2. MovSB - Used to move the byte 1 word from one string to another.

3. LEA - Used to load the address of operant into the provided register.

4. REP- Used to repeat the given instruction till Cx ≠ 0.

5. CLD- Used to clear the directional flag, i.e. Df = 0.

a) Byte Transfer:

<u>Algorithm</u>:

Step1: Assign value 0500 SI and 0600 DI

Step 2: Assign value 0000 H to Ax.

Step 3: Move the content of Ax in Ds.

Step 4: Move the content of Ax in ES.

Step 5: Assign the value 0004H to CX.

Step 6: Clear the directional flag.

Step 7: Repeat until CX = 0, more string block.

Step 8: End of code.

Program:

- model small
- stack
- data
- code

```
mov SI, 0500
mov DI, 0600
mov Ax, 0000
mov Ds, Ax
mov Es, Ax
mov Cx, 004
cld
rep
movSb
hlt
code ends
end
```

b) Reverse Transfer:

Algorithm:

Step 1: Create s string.

Step 2: Traverse through the string.

Step 3: Push the characters in the stack.

Step 4: Count the no. of characters.

Step 5: Load the starting address of the string.

Step 6: POP the top character of the stack until count is not equal to zero.

Step 7: Put the character and reduce the count and increase the address.

Step 8: Continue until the count is grater than zero.

Step 9: Load the effective address of the string in DX using LEA command.

Step 10: Print the string by calling the interrupt with 9H in AH.

Step 11: The string must be terminated by '$' sign.

Program:

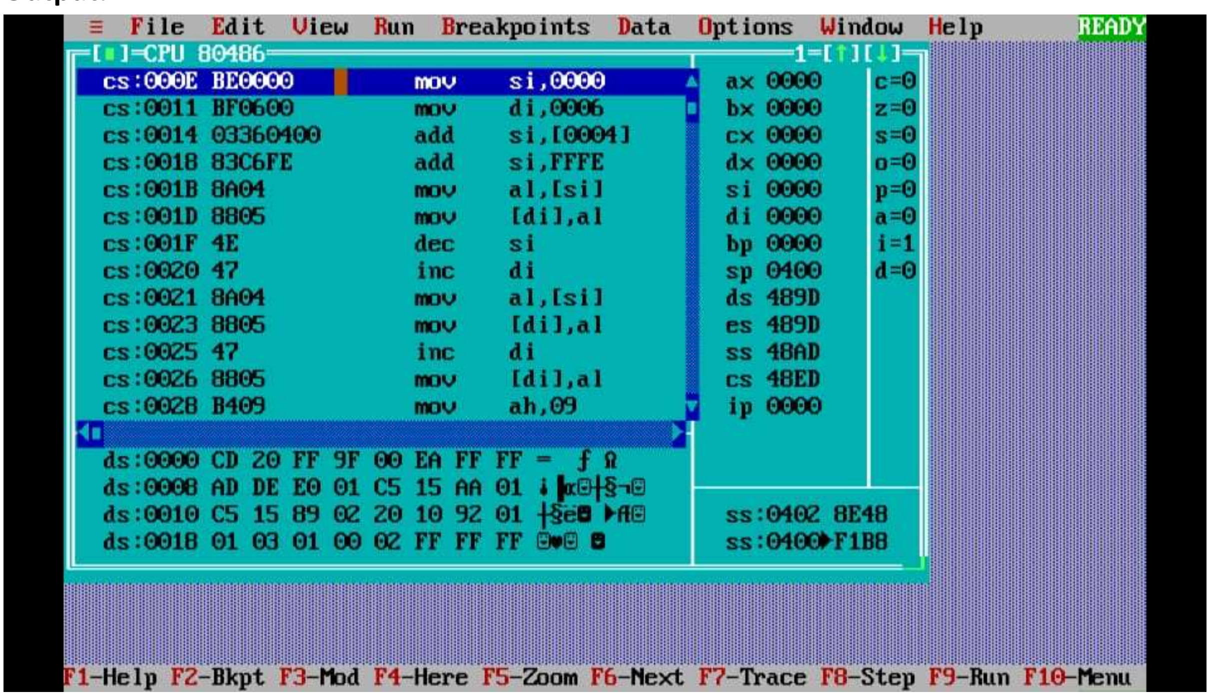
- model small
- stack

```
data segment
str1 db 'abc $'
strlen1 dw $-str1
strrev db?
data ends

Code Segment
assume CS : code, ds : data
begin:
mov ax, data
mov ds, ax
mov es, ax
mov cx, strlen1
Add cx, -2
Lea Si, str1
Lea di, strrev
Add Si, strlen1
Add Si, -2
Loop 1:
mov al, [si]
```

```
mov [di], al
dec si
inc di
mov al, [si]
mov [di], al
inc di
mov al, '$'
print:
mov ah, 09h
Lea dx, strrev
int 21h
Exit:
mov ax, 4ch
int 21h
code ends
end begin
end
```

Output:



Conclusion: Thus, we wrote an assembly language program for data transfer group (a) Byte transfer (b) Block transfer (c) Reveres transfer.

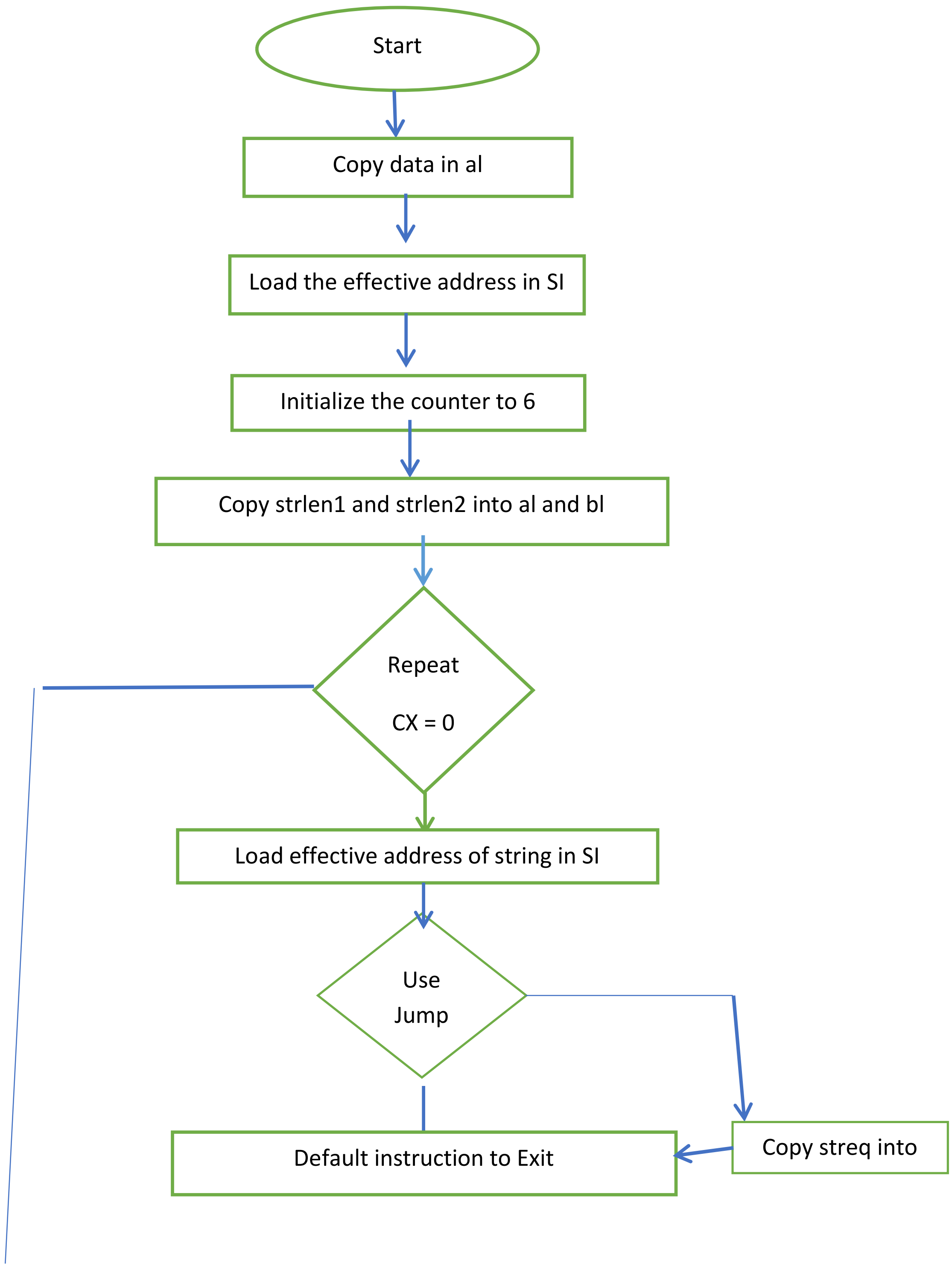
Practical no 4.

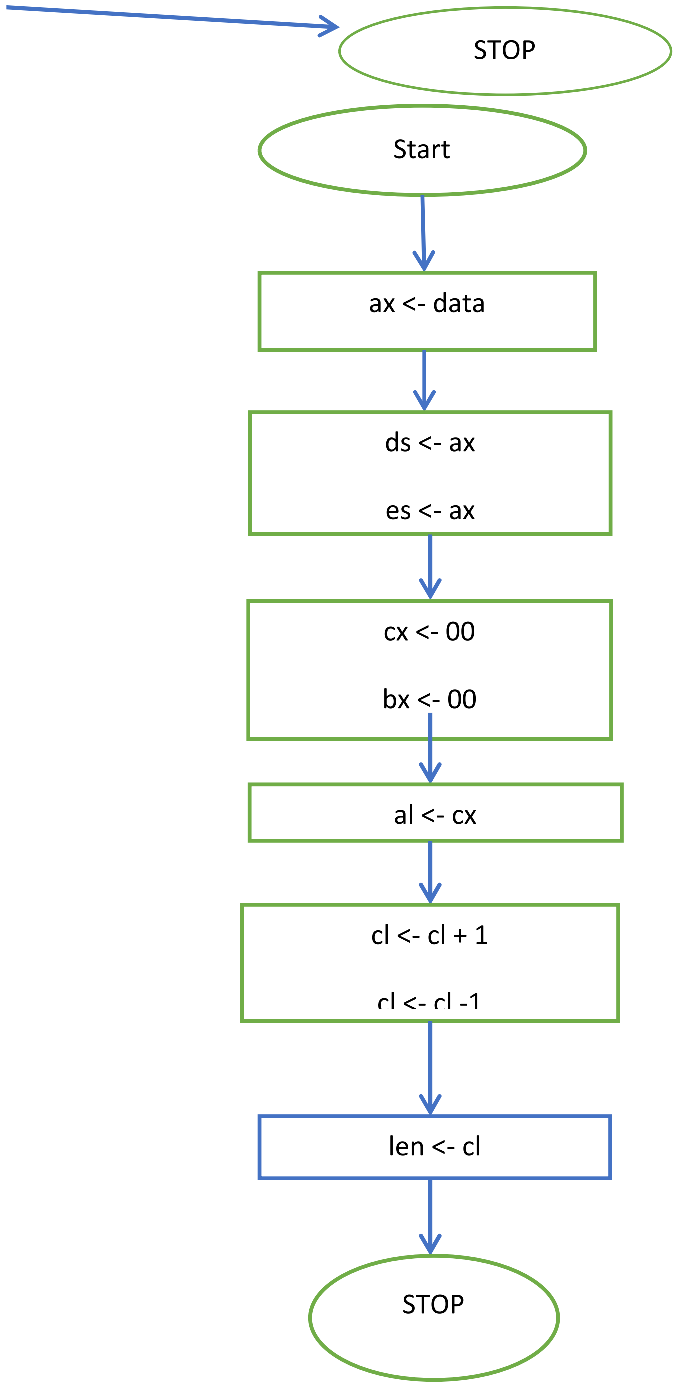
Aim: Perform an assembly language program for adding 2 numbers.

Software Requirement: TASM version 1.4.

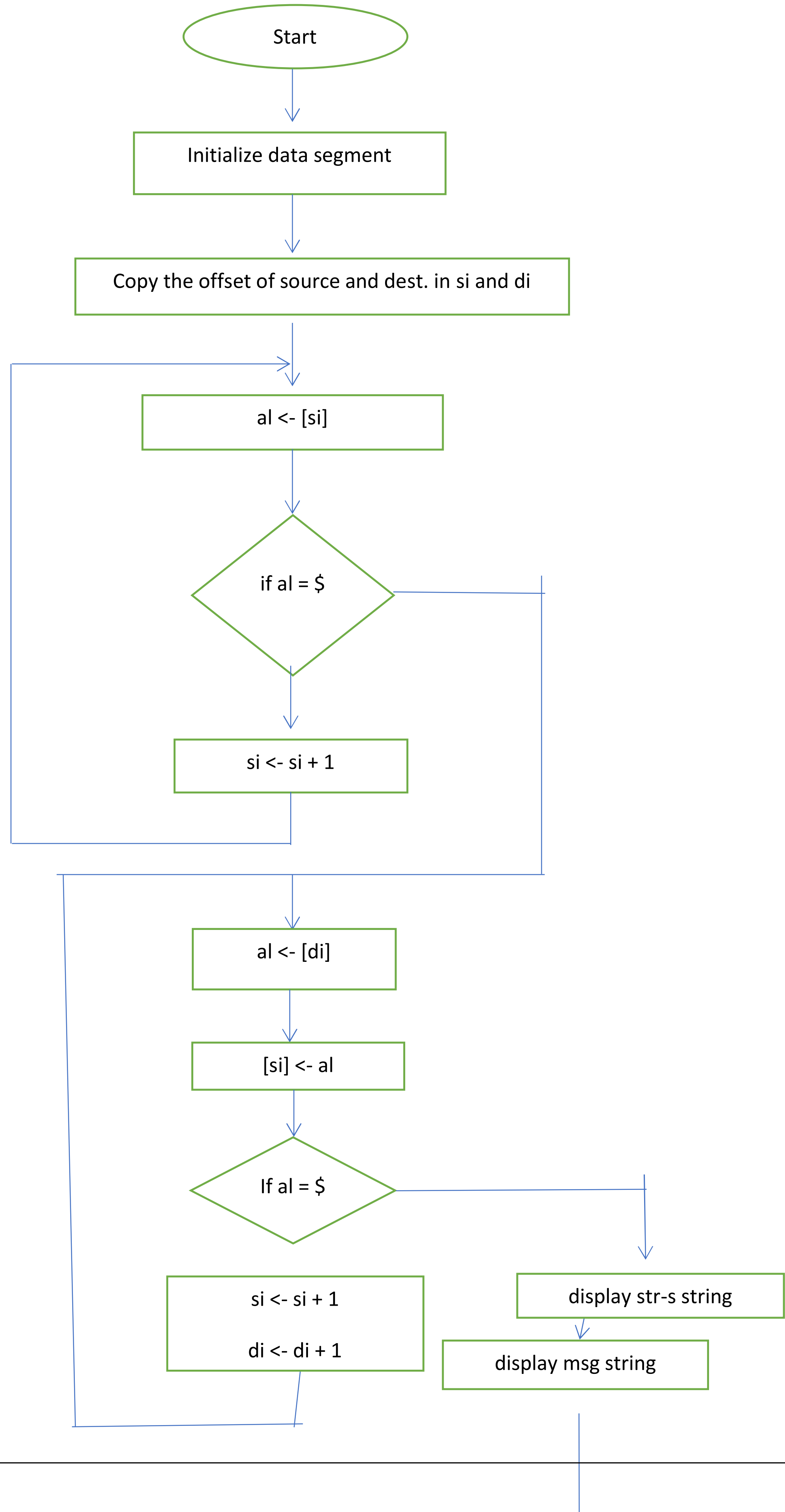
Flow Chart:

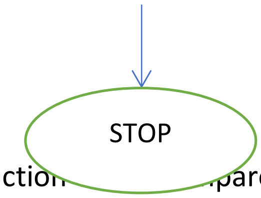
1.




STOP
Start
ax <- data
ds <- ax
es <- ax
cx <- 00
bx <- 00
al <- cx
cl <- cl + 1
cl <- cl -1
len <- cl
STOP

2.


Start
Initialize data segment
Copy the offset of source and dest. in si and di
al <- [si]
if al = $
si <- si + 1
al <- [di]
[si] <- al
If al = $
si <- si + 1
di <- di + 1
display str-s string
display msg string



Theory:

1. cmp – The cmp instructor is one of the type of instructio pare two operands. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not. It doesn't disturb the destination operands.

2. jmp – This instruction is called as unconditionally jump. The program sequence is transferred to the memory location specified by the is bit address given in the operand.

3. cmpsb – This instruction is used to compare a byte in one string. The comparison is executed by subtracting byte (word) in DI from the byte word in SI.

4. jne – This instruction is a conditional jump that follows a test. It jumps to the specified location if XE is 0.jne is commonly found after a cmp instruction.

a) Compare two strings:

Program:

- model small
- stack

data segment

str1 db 'student $'
strlen db $-str1
str2 db?
strlen2 db $-str2
streq db 'strings are equal $'
struneq db 'not equal $'
data ends

Code Segment
assume CS : code, DS : data

Start:
mov ax, data
mov ds, ax

```
move s, ax
lea si, str1
lea di, str2
mov cx, 06
mov al, strlen1
mov bl, strlen2
cmp al, bl
REP
cmpsb
jne Not_Equal
jmp Equal

Not-Equal
mov ah, 09h
lea dx, strune
int 21h
jmp Exit

Equal:
mov ah, 09h
lea dx, streq
int 21h

Exit:
mov ah, 4ch
int 21h
code ends
end start
```

<u>Algorithm:</u>

Step 1: Start.

Step 2: Copy the data in Ax register.

Step 3: Copy the Ax register value in data and extra segment register.

Step 4: Load the effective address of str1 in SI register.

Step 5: Similarly, load the effective address of str1 in DI register.

Step 6: Initialize counter to 6.

Step 7: Copy strlen 1 and strlen2 into al and bl register.

Step 8: Repeat until CX = 0.

Step 9: Use jne instructor if they are not equal.

Step 10: Load the effective address of string if not equal in AX register.

Step 11: Use JMP instruction if they are equal.

Step 12: If they are equal then stored their address into dx register.

Step 13: Use default instructions to exit the program.

Step 14: Stop.

b) To find length of strings:

Algorithm:

Step 1: Start.

Step 2: Copy the default value of data into Ax register.

Step 3: Copy value of Ax into data and extra segment register.

Step 4: Move any value of your own into bx and cx register.

Step 5: More the content of bx register into al.

Step 6: Increase al by 1 by initializing L1.

Step 7: In L1, decrease CL by 1

Step 8: More or copy the value of CL into len.

Step 9: Use the default instruction to exit the program.

Step 10: Stop.

Program:

- model small
- stack

```
data segment

str1 db ‘student $’
lend b?
data ends

code segment
assume CS : code, DS : data

Start:
mov ax, data
mov ds, ax
mov es, ax
mov bx, 00
mov cx, 00


Loop:
mov al, [bx]
cmp al, 06
je L1
inc CL
jmp LOP

L1:
dec CL
mov len, CL
```

```
Exit:
mov ah, 4ch
int 21h
code ends
end start
end
```

To concatenate two strings:

<u>Algorithm:</u>

Step 1: Start.

Step 2: Initialize data segment and extra segment.

Step 3: Copy the offset address of source and destination strings in ‘si’ and ‘di’ resp.

Step 4: Copy data pointed by ‘si’ in ‘al’.

Step 5: Compare ‘al’ with ‘$’ symbol.

Step 6: Jump to step 9 if ‘al’ and ‘$” are equal.

Step 7: Increment the address in ‘si’.

Step 8: Jump to step 4.

Step 9: Copy data pointed by ‘di’ in ‘al’.

Step 10: Copy data of ‘al’ with ‘$’ symbol.

Step 11: Compare ‘al’ with ‘$’ symbol.

Step 12: Jump to step 15 if ‘al’ and ‘$’ are equal.

Step 13: Increment ‘si’ and ‘di’.

Step 14: Jump to step 9.

Step 15: Display ‘msg’ string.
Step 16: Display ‘str_s’ string (concatenated string).

Step 17: Terminate the program.

Step 18: Stop.

Program:

- model small
- data

str_s dw ‘Amravati $’.
Str_d dw ‘Maharashtra $’.
msg1 dw ‘After concatenation..$’

- code

```
mov ax, @ data
mov ds, ax
mov si, offset str_s
Next:   mov al, [si]
cmp al, $
je exit
inc si
        jmp Next

Exit: mov di, offset str_d

up:     mov al, [di]
        cmp al, $
        je exit1
        mov [si], al
        inc si
        inc di
        jmp up

Exit1:  mov al, $
        mov si, al
```

```
mov ah, 09h
lea dx, msg1
int 21h
mov ah, 09h
lea dx, str_d
int 21h
mov ah, 4ch
int 21h
ends
end
```

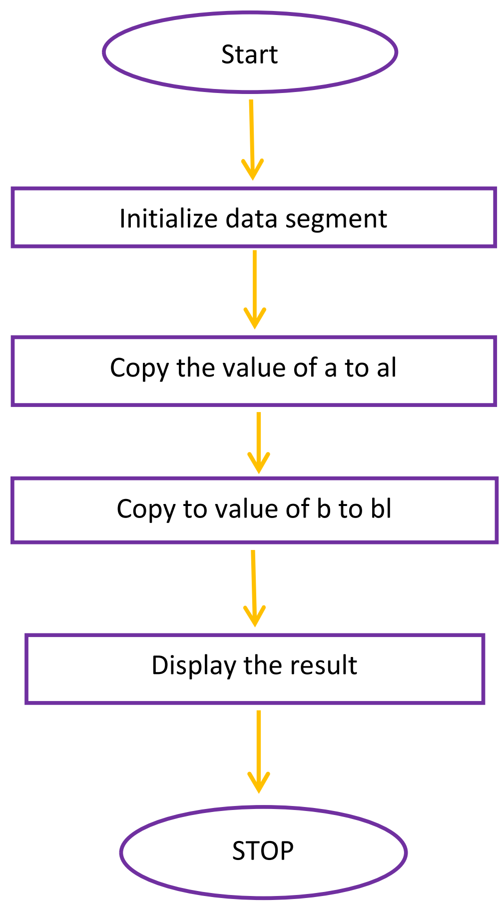
Conclusion: Thus, we wrote an ALP to : a) compare two strings b) find length of string c) concatenate two string.

Practical no 5.

Aim: Write an assembly language program to swap the contents of 2 registers.

Software Required: Tasm version 1.4

Flowchart:



Theory:

Program:

```
.MODEL SMALL
.STACK 100H
.DATA
  NUM1 DB 05H
  NUM2 DB 02H
.CODE
  MOV AX , @DATA
  MOV DS , A
  MOV BL , NUM1
  MOV CL , NUM2

  MOV NUM2 , BL
  MOV NUM1 , CL

  MOV AH , 4CH  ; Intrupt to exit
  INT 21H
END
```

Algorithm:

Step 1: Start.

Step 2: Initialize data segment.

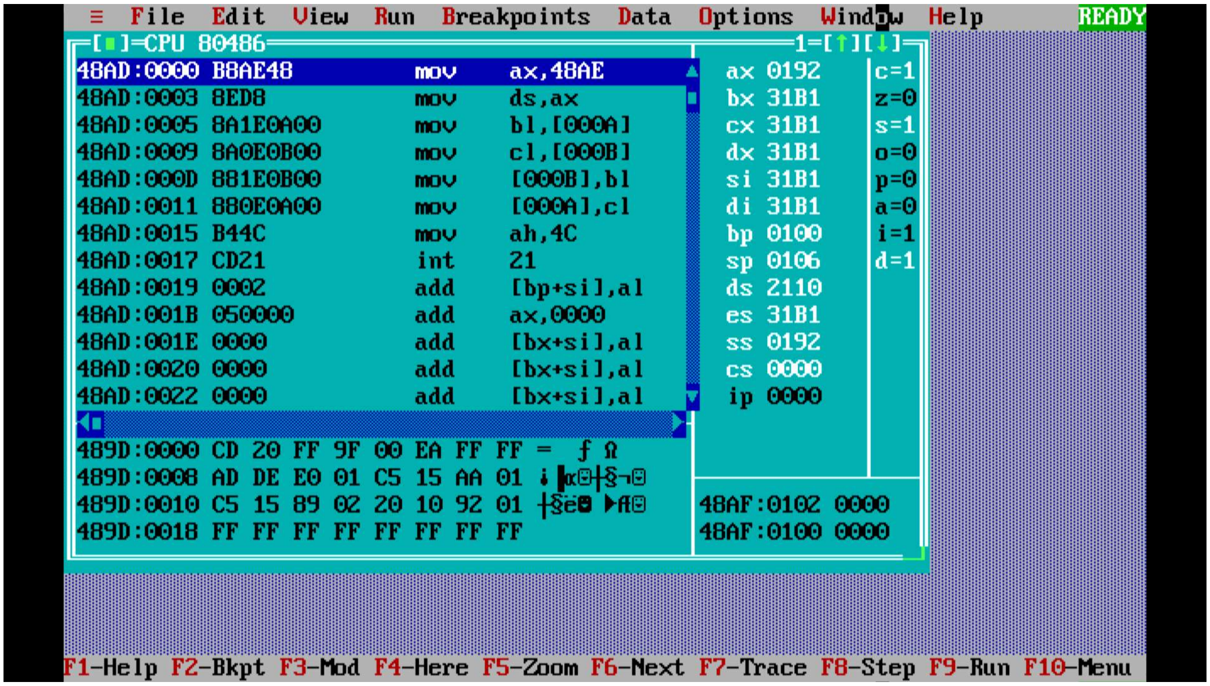
Step 3: Copy the value of a to al.

Step 4: Copy the value of b to bl.
Step 5: Swap contents of al and bl.

Step 6: Display the result.

Step 7: Stop.

Output:



Conclusion: Thus, we have performed an ALP to swap the contents of 2 registers.

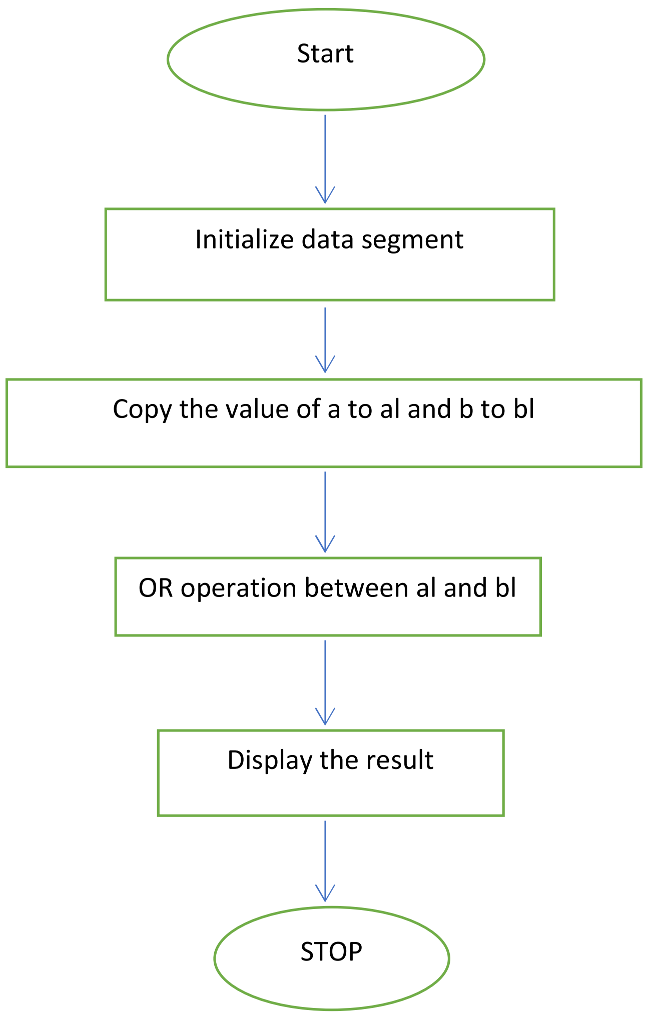
Practical no 6.

Aim: Write an assembly language program to perform OR, XOR AND Operation.

Software Required: Tasm version 1.4

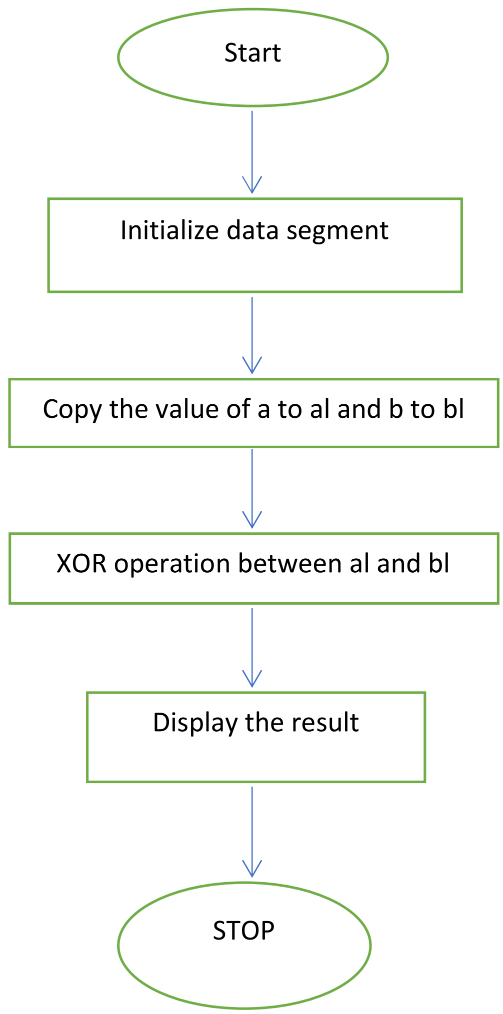
Flowchart:

1.

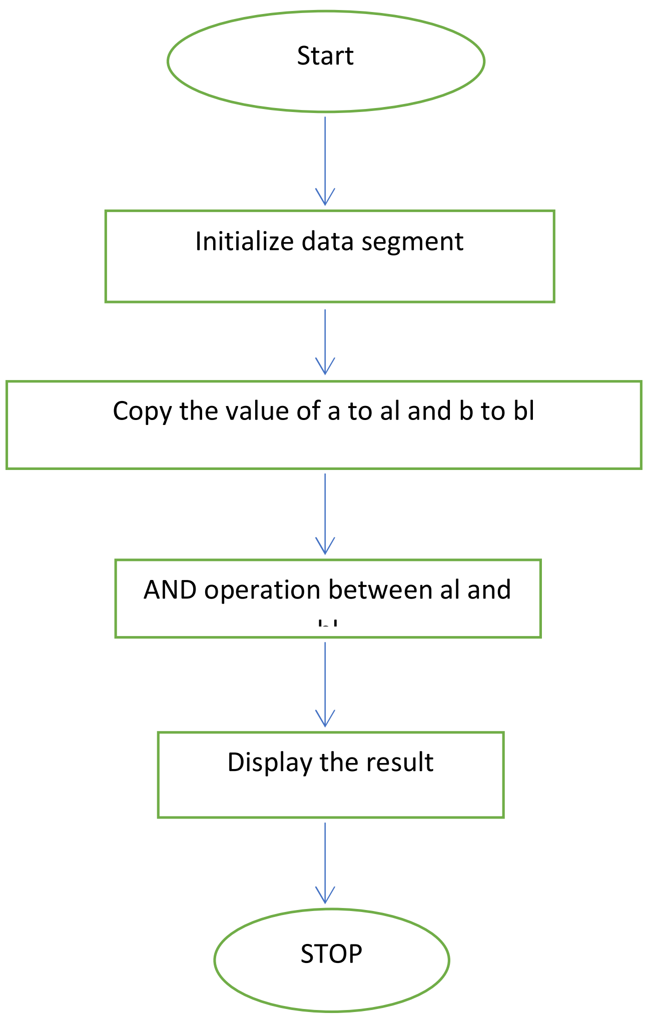


2.

3.


Start
Initialize data segment
Copy the value of a to al and b to bl
XOR operation between al and bl
Display the result
STOP

4.


Start
Initialize data segment
Copy the value of a to al and b to bl
AND operation between al and
Display the result
STOP

Theory:

Program:

(OR Operation):

```
.model small
.data
a db 02h
b db 08h

.code
Start:  mov ax, @data
mov ds, ax
mov al, a
mov bl, b
OR al, bl
mov ah, 4ch
int 21h
end start
end
```

Algorithm:

Step 1: Start.

Step 2: Initialize data segment.

Step 3: Copy the value of a to al.

Step 4: Copy the value of b to bl.

Step 5: OR operation between al and bl.

Step 6: Display the result.

Step 7: Stop.

Program: (XOR operation)

```
.model small
```

```
.data
a db 02h
b db 08h
.code
start:
mov ax, @data
mov ds, ax
mov al, a
mov bl, b
XOR al, bl
mov ah, 4ch
int 21h
end start
end
```

Algorithm:

Step 1: Start.

Step 2: Initialize data segment.

Step 3: Copy the value of a to al.

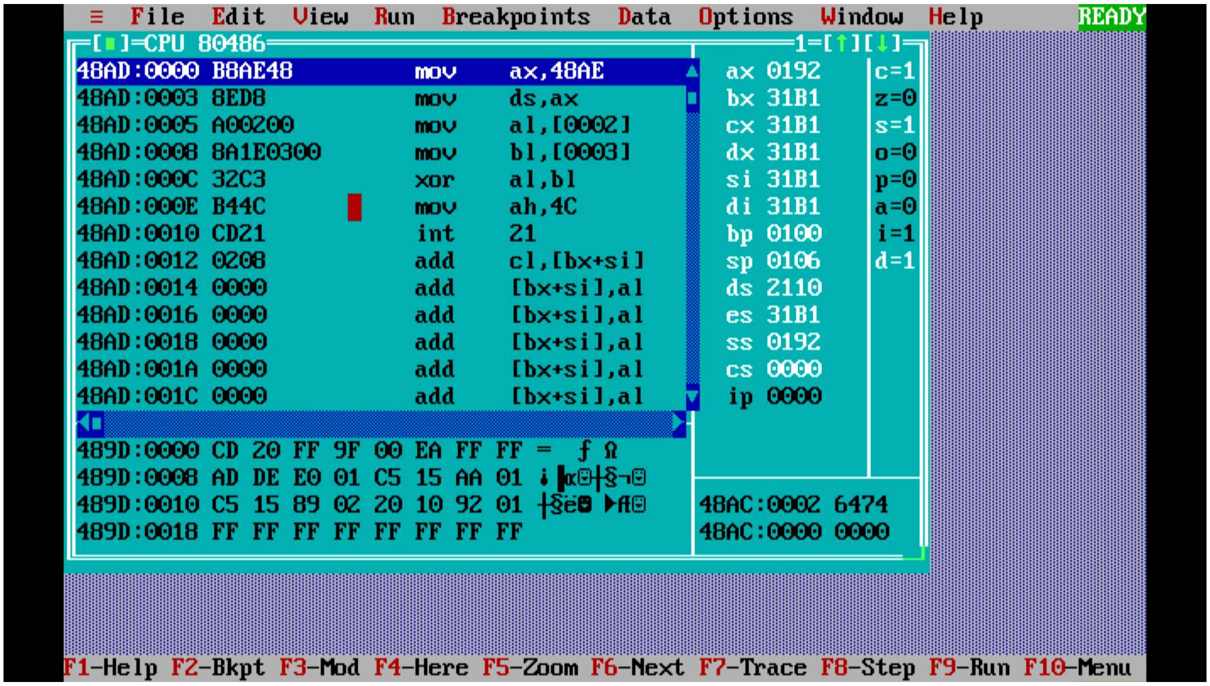
Step 4: Copy the value of b to bl.

Step 5: AND operation between al and bl.

Step 6: Display the result.

Step 7: Stop.

Output 2: XOR



Conclusion: Thus, we have performed an ALP to perform OR, XOR and AND operation.

Practical no 7.

Aim: To write an assembly language program for multiplication of two numbers using multiple addition method.

Software Required: Tasm version 1.4

Theroy:

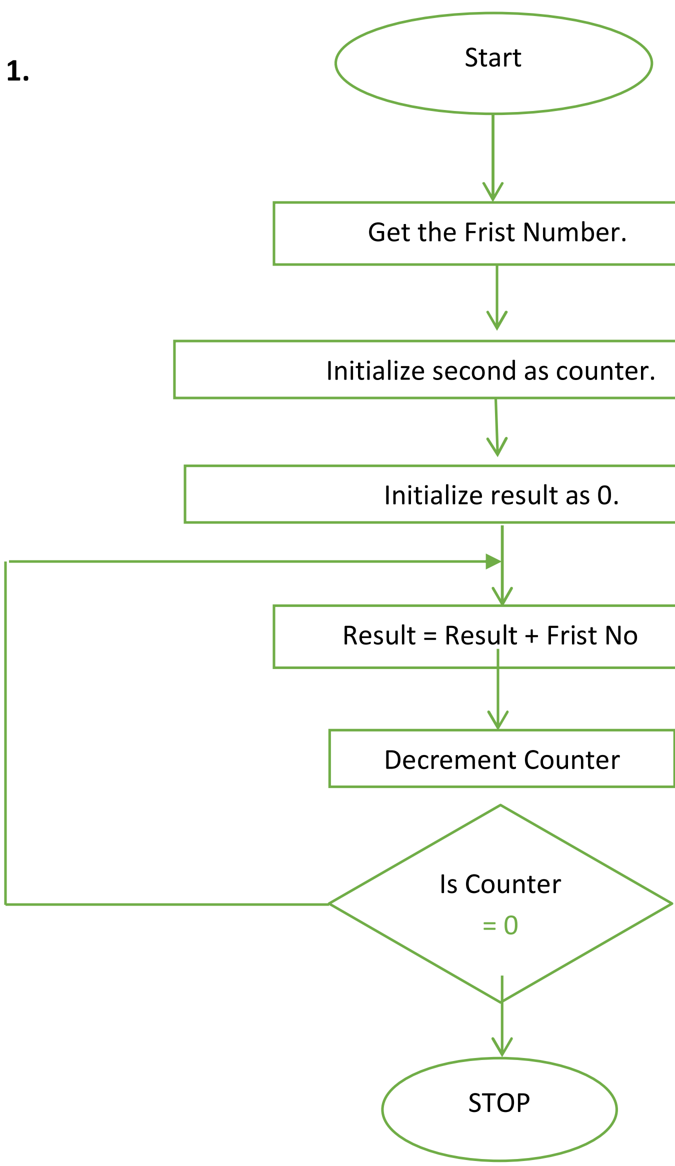
Logic AL = 12H, BL = 10H

Res = 12H+12H+12H+12H+12H+12H+12H+12H+12H+12H+12H

= 120H

Flow Chart:

1.



Assembly Program

```
.model small
.data
A db
B db
.code
mov ax, @data
mov ds, ax
mov al, a
mov bl, b
mov ah, 0h
mov dx, 0h
rep: add dx,ax
        dec bl
        cmp bl, ooh
        jnz rep
        mov ah, 4ch
        int 21h
ends
end
```

Alogorithm:

Step 1: Initialize the data segment.

Step 2: Get the frist no.

Step 3: Get the Second no as counter.

Step 4: Initialize result as 0.

Step 5: Result = result +frist no.

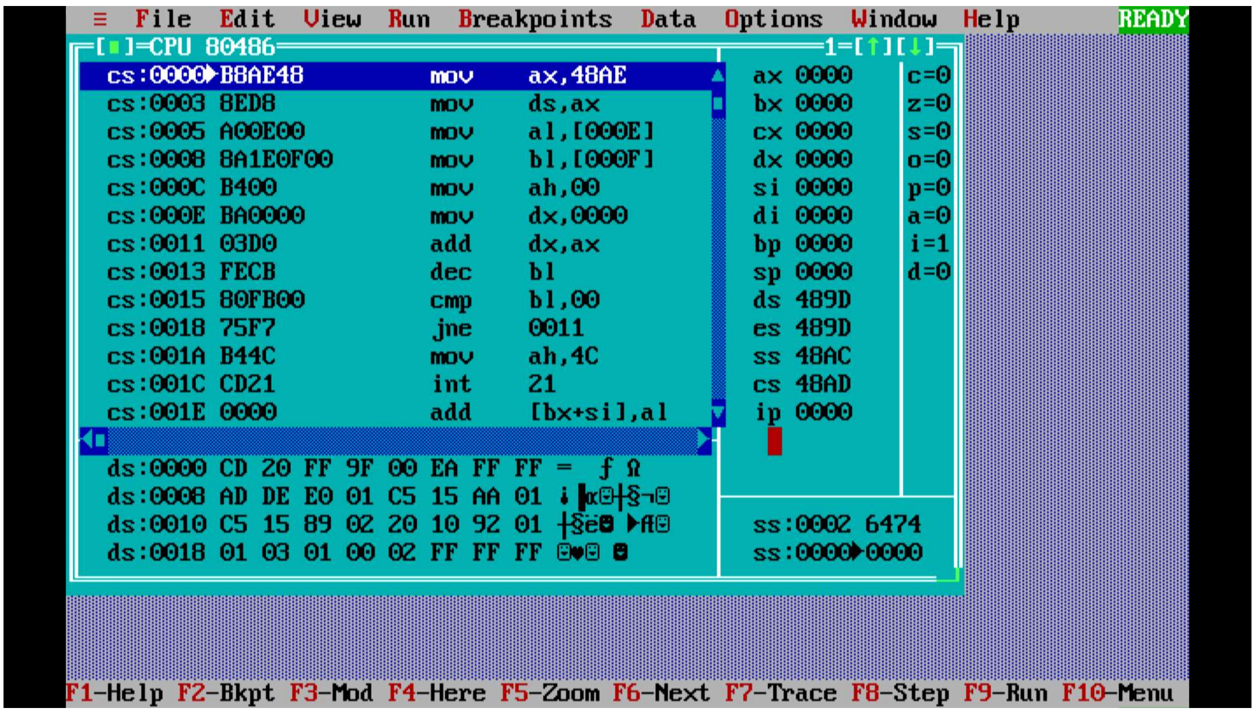
Step 6: Decrement the counter

Step 7: If count is not 0, go to step 5

Step 8: Display result.

Step 9: Stop.

Output:



Conclusion:

We successfully ASM program to find multiplication of two numbers using successive addition method.

Practical no 8.

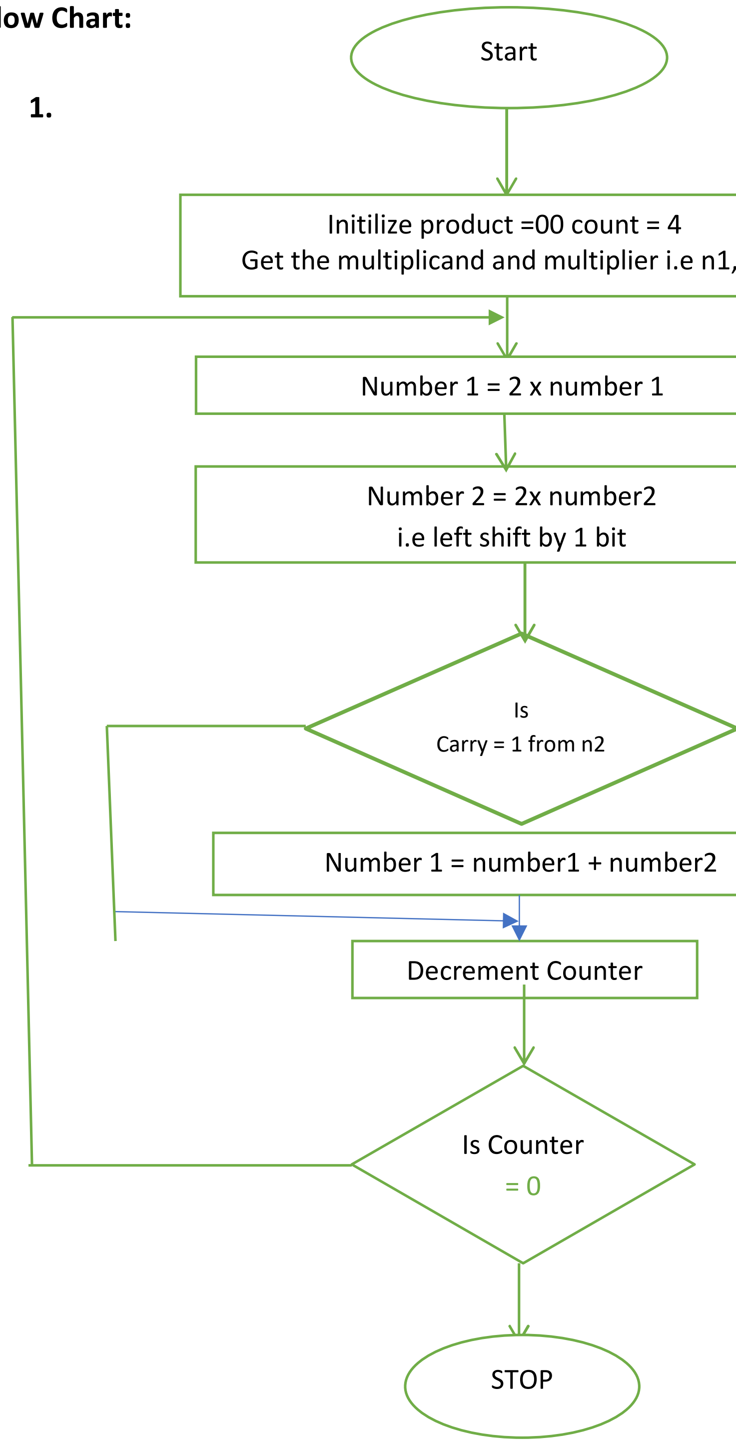
Aim: To write an assembly language program for multiplication of two numbers using Add and Shift Method

Software Required: Tasm version 1.4

Theroy:

Flow Chart:

1.



Assembly Program

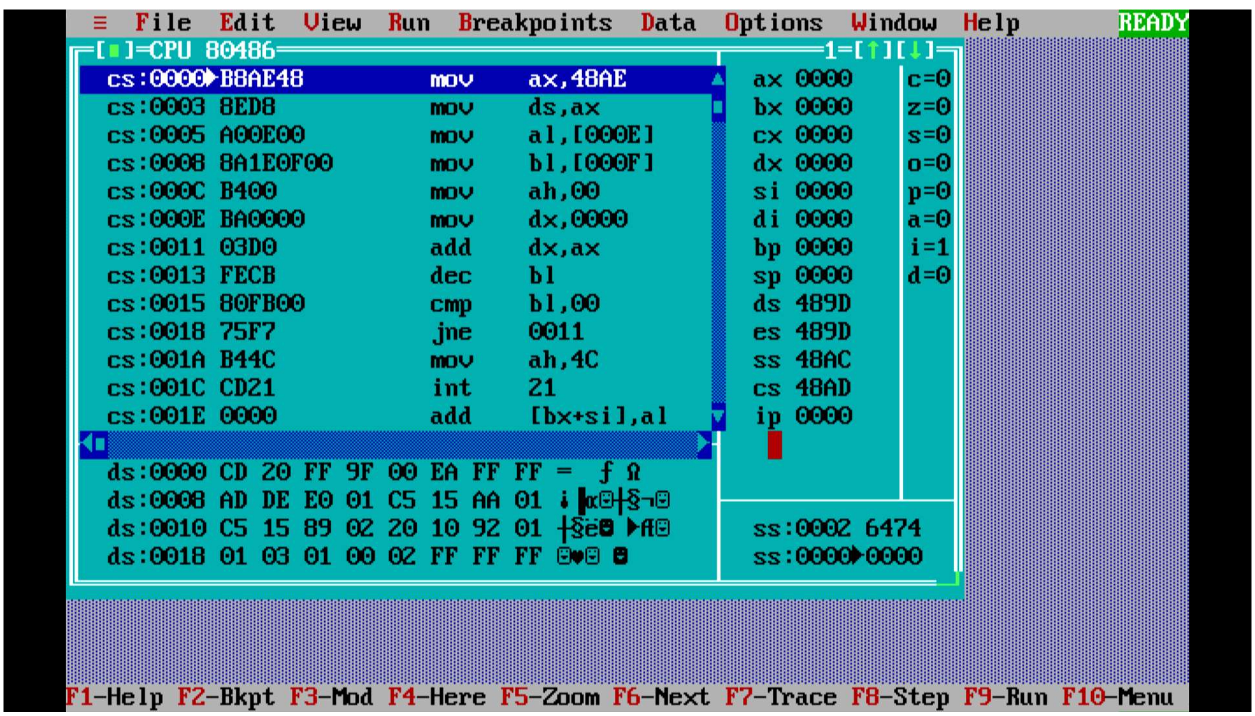
```
.model small
.data
a db 11H
b db 10H
.code
    mov    ax, @data
    mov    ds, ax
    mov    al, a
    mov    bl, b
    mov    ah, 0
    mov    dl, 04h
ad:    add    ax, ax
    rcl    bl, 01
    jnc    skip
    add    ax, bx
skip:    dec    dl
    jnz    ad
    mov    ch, 04h
    mov    cl, 04h
    mov    bx, ax
l2:    rol    bx, cl

    mov    dl, bl

    and    dl, 0fH
```

```
    cmp     dl, 09
    jbe     l4
    add     dl, 07
l4: add     dl, 30H
    mov     ah, 02
    int     21H
    dec     ch
    jnz     l2
    mov     ah, 4cH
    int     21H
end
```

Output:



Conclusion:

We successfully ASM program to find multiplication of two numbers using Add and Shift Method

Practical no 9.

Aim: To write an assembly language program for finding larger of two numbers using procedure

Software Required: Tasm version 1.4

Theroy: Instructionused
1CMPdest,
src:- Comparemaybedaterthesourceoperandwhichlaregisteroranimmediateoramemory .
Subtracts the source operand from destination &result is stored now here.
Instruction used
1CMPdest, src:- Compare may be date r the source operand which register or an immediate or a memory location with destination oper and. Substracts the source operand from destination & result is stored now here.
If src = dst+XF=1 if src dst,CF=1

PROC: The directive PRO Cindicates beginning of a procedure & follows with the name of the procedure. The team !'far' & 'near' follows the PROC directives indicating the type of a procedure. If the team is not specified then, assembler assume s near the type specifion.
Syntax:-Procedure name PROCL(NearIfax)

3)ENDP: The directive ENDP in form st he assembler the end of a procedure.
Syntax:-(Procedure_name]ENDP

4)RET:The instruction RET is used to transfer program control from the procedure back to the calling program.

5) CALL: The instruction CALL is used to transfer program control to the sub Program or a procedure by storing the return address on the stack.

Syntax: CALL Procedure-Name

6) JC: This instruction isused to jump to the specified label if car is 1 carry flag

Syntax: JC Label

Algorithm:(MainProgram)
1.Start

2.Initializethedatasegment

3.Call"COMPARE"producre

4.Terminatetheprogram

5.Stop

Algorithm(compareProgram):-

1. Capydataofnum1inalregister
2. Compareal&num1usingcmP
3. Ifcarryisgeneratedgotostep7
4. Store09Hinal
5. Storeoffsetaddressofmsg-1indx
6. call"int21H"interscupt
7. Store09Hinal

8.Storeoffsetaddressofmsg_2indx

9.call"int21H"interrupt

10.Returntothecallingfunction

Program

```
.modelsmall

.data

num1 db 10H

num2 db 20H
```

```
msg-1 db num-1 is
Greater$ msg-2 db
num-2 is Greater$'

.code

mov ax,@data

mov ds, ax

CALL  Compare

mav ah, 4ch

int 21h

Compare PROC

mov al, num1

cmp al, num2

JC great

mov ah,09H

lea dx, msg-1

int21H

great: mov
ah,09H lea
dx, msg

int21H

RET

ENDP

ENDS

END
```

Conclusion:-Thus, we have ALP to assembly language program for finding larger of two numbers using procedure

.

Practical no 10.

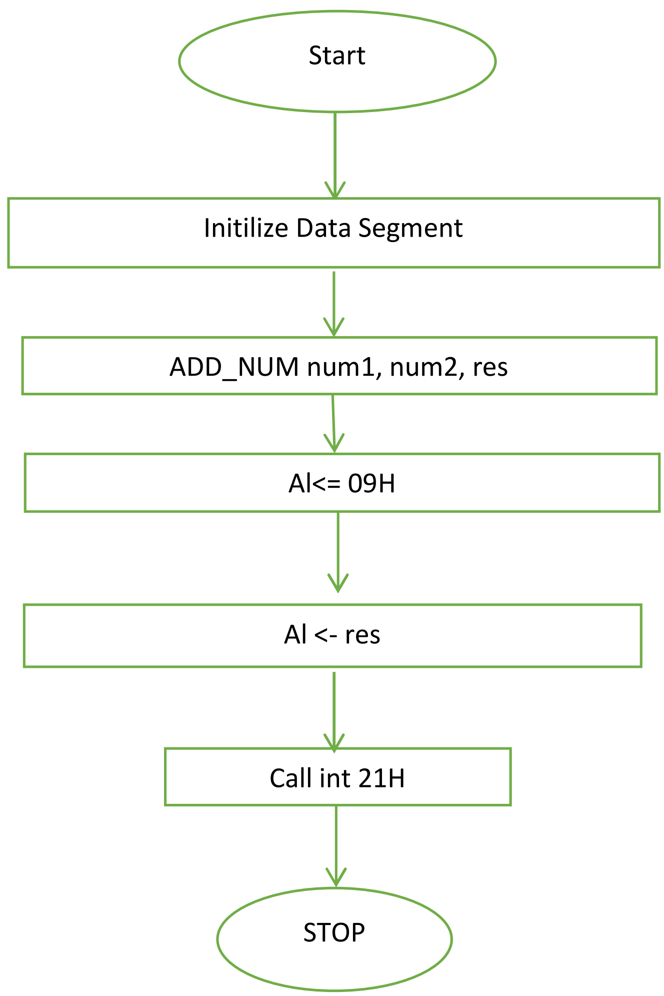
Aim: Perform addition of two numbers by using macro

Software Required: Tasm version 1.4

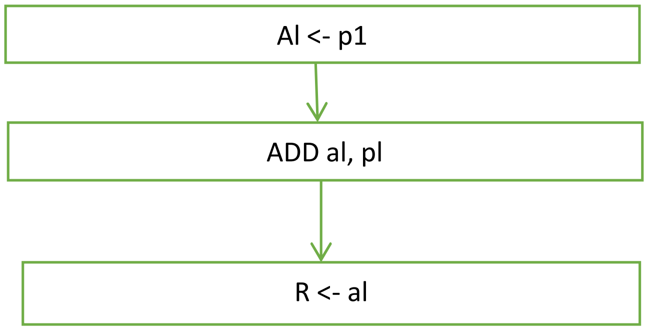
Theroy:

Flow Chart:

1.



ADD_NUM MACRO



Algorithm:

MACRO (Add_Num)

1.Store data of P1 in al register

2.Add al and P2

3.Store the result from al to R.

Algorithm(Main Program):-

1. Start

2. Initialize data segment.

3. ADD_NUM num1, num2, res

4. Store OSH in al

5. Store data of res in al

6. Call int21 interrupt

7. Terminate the program

8.Stop

Program

```
ADD_NUM, MACRO P1,P2 P
Mov al,P1
mov R, al
END M
```

```
.model small
.data
num1 db 10H
num2 db 20H
res db ?
.code
mov ax, @data
mov ds, ax
ADD_NUM num1, num2 ,res
mov ah, 02h
mov al, res
int 21h
mov ah, 4ch
ends
end
```

Conclusion :-Thus, we have ALP to assembly language program for finding Perform addition of two numbers by using macro.

Practical no 11.

Aim: Write an assembly language program for flashing of LED's at PORT A
of 8255

Software Required: Tasm version 1.4

Theroy:

The Intel 8255 general purpose programming IO device which may be used with many different microprocessors. There are 24 I/O which may be individually program into groups of 12 and used in three major modes of operation. The high performance and industry standard configuration of 8255A compatible with 8086.

Basic Modes of 8255:

Mode selection

there are three basic modes of operation that can be selected by a system software

1) mood o - basic I/O

2) mode 1 - Standard I/O

3) mode 2 - Bi-directional bus

Description:

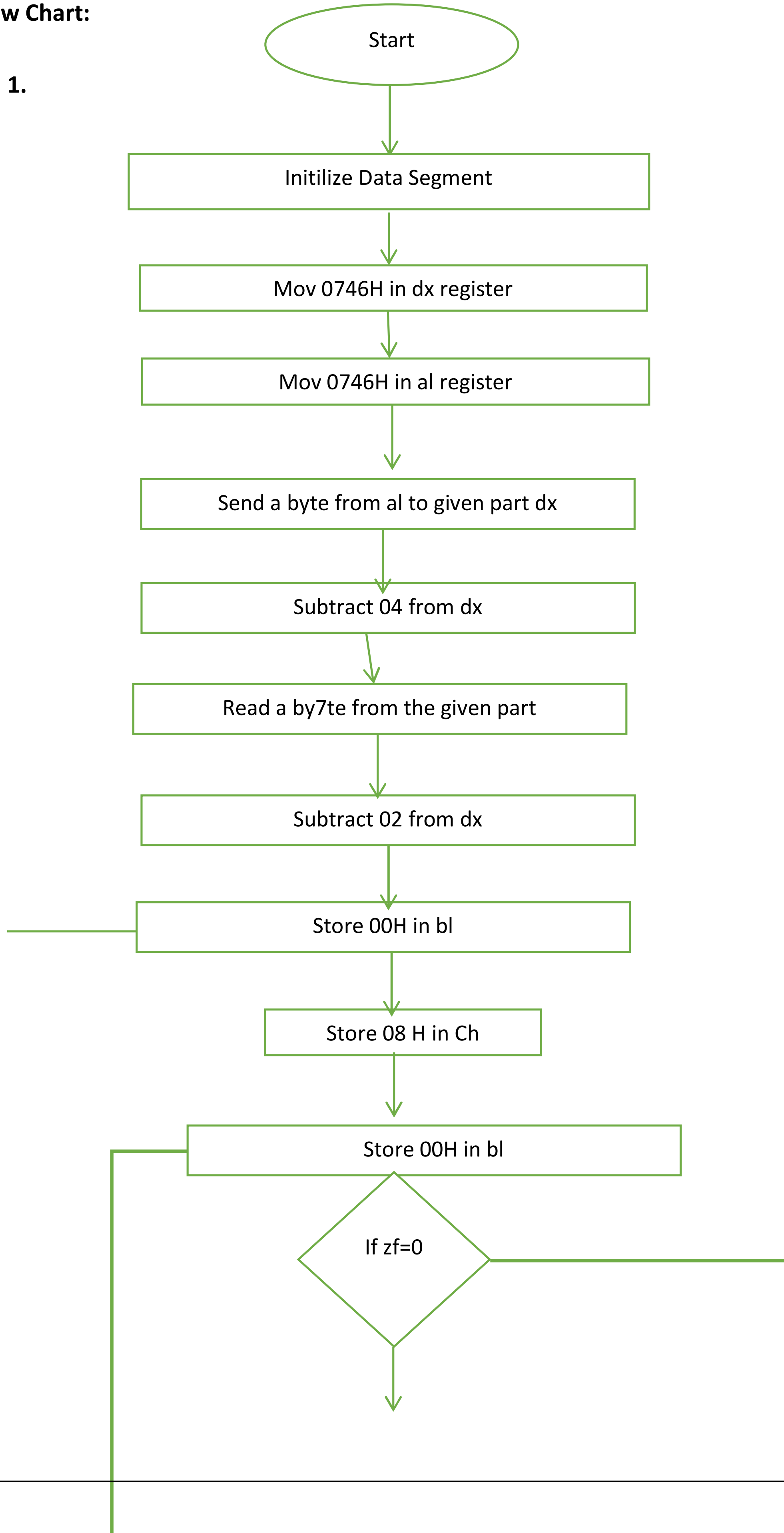
80255A is to be interfaced with lower auditor bus that is D0 - D7.

The 0 and 1 of 8255 are connected to A1 and A2 pins of microprocessor respectively. we will use the absolute decoding scheme that uses all the sixteen-address line. For deriving the device address pulse out of a 0-15 lines to address lines A2 and A1 are directly required by 8255 hence only two are used for recording addresses.

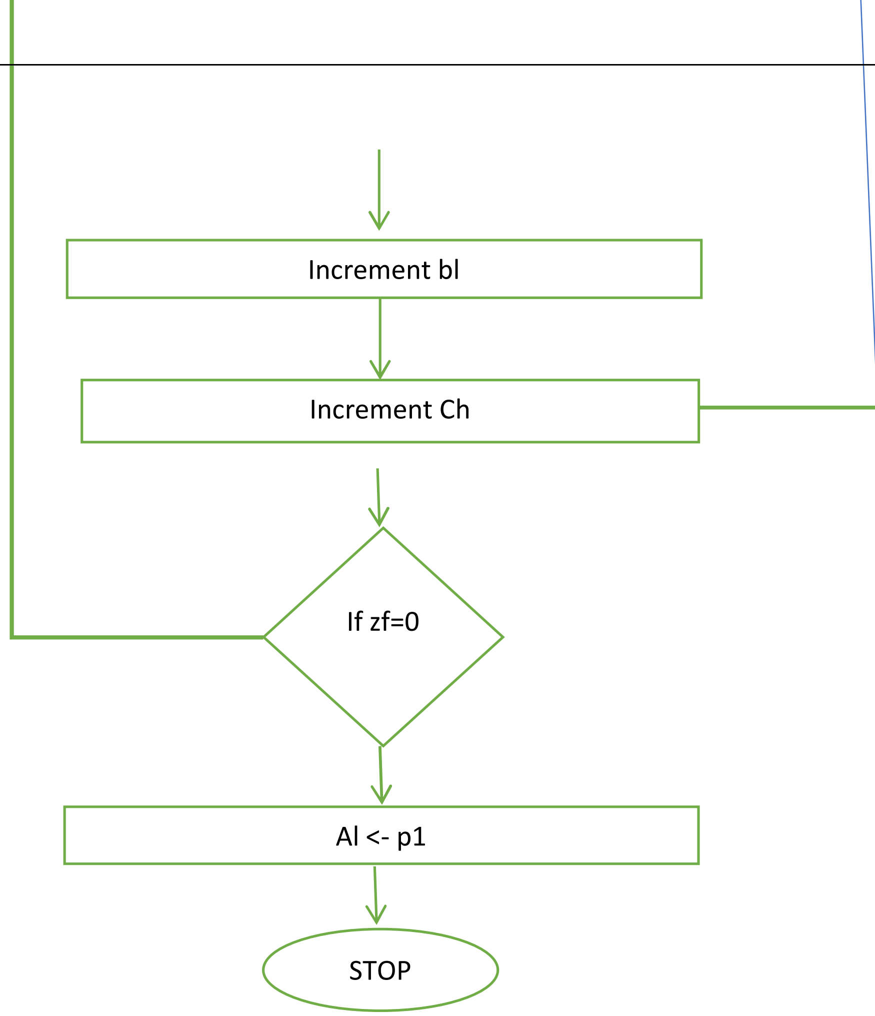
In a circuit diagram the 8085 is assumed to be in maximum mode.

Flow Chart:

1.



Start
Initilize Data Segment
Mov 0746H in dx register
Mov 0746H in al register
Send a byte from al to given part dx
Subtract 04 from dx
Read a by7te from the given part
Subtract 02 from dx
Store 00H in bl
Store 08 H in Ch
Store 00H in bl
If zf=0



Algorithm:

Step 1 : start

Step 2 : Initialise data segment

Step 3 : mov 80746 edge DX register

Step 4: Mov 82h in a and register

Step 5: Send a byte from al to given port- dx.

Step 6 : Subtract 04 from dx.

Step 7: read a byte from the given port dx to al.

Step 8: subtract 0 2 from the x

Step 9: Store 00H in bl.

Step 10: Store 08H in bl

Step 11: Rotate the data on al by cl times.

Step 12: If carry is generated, jump to step 14

Step 13 : Increment bl.

Program

```
.model small
.stack
.data
.code

Move dx, 0746H
Mov dl, 82H
Out dx, al
Sub dx, al
In al,dx
Sub dx, 02H
Mov bl, 00H
Mov ch, 08 H
Mov ch, 08H

Label 1: rol al, cl
Jnc label 2
Inc bl

Label 2:
Dec ch
Jnz label 1
Mov al,bl
Add dx, oh
out dx, al
hlt
Ends End
```

Conclusion :-Thus, Write an assembly language program for flashing of LED's at PORT A of 8255

Practical no 12

Aim: Interface 8254 times with 8086 as add addressed device & generate a square wave at its output.

Software required: Tasm. vassion 14

Theory :

8254 Counter / Timer :

It is Intel's Programmable Counter/ Timer device which is used to generate accurate time delays , Square wave and complex waves too. It is also used as counter sometimes to count no. of clock cycles. 8254 is compatible with all Intel & most other microprocessors & handles inputs From DC to 10MHZ. It is used for controlling real-time events such as real-time clock, events counter, & motor speed & direction control.

The 8254 uses HMOS technology & comes in 24-pin plastic or CERDEP package. 8254 Timer / Counter perform Binary as BCD counting. For counting it provides three independent 16-bit. counters, cach capable of handling clock inputs up to 10 MHZ. There are totall six programable counts modes where each are software programmable. It uses +5V supply for its working.

- **Intel's 8254 counter / Timer provide some Features such as :**

1. Accurate Time Delays
2. It minimum load on Microprocessor
3. Provide Real Time clock
4. Event Counter
5. Generation of Square wave.
6. Also, Generator complex waveform.

- **Contral Logic for 8054**
  - CS - Logic 0 = Enable 8254

  - RD - logic 0 = Tells microprocessor Read Count from 8254

- COR -Logic 0 = Tells microprocessor write Count / Command into 8254.
- A1 , A0 = Address I / p Pins to select modes & counters.

Pinout

8245

CS	RD	WR	Al	Ao	Operation
0	1	0	0	0	Write Counter 0
0	1	0	0	1	Write Counter 1
0	1	0	1	0	Write Counter 2
0	1	0	1	1	Write Control Word
0	1	0	0	0	Read Control 0
0	0	1	0	1	Read Control 1
0	0	1	1	0	Read Control 2
0	0	1	1	1	No Operation
0	1	1	x	x	No operation
1	x	x	x	x	8254 Not Selected

Counters in 8254 :-

There are there counters in 2254 namely c1 , c2 , & c3

- Each counture is 16-bit Identical presettablee
- Hear Down Counter operates in BCD or Hex
- Controlled by heading count to command word Register.

Modes of operation :-

- Made 0 (Interrupted or Terminal count).
- Made 1 (Programmable manashot).
- Mode 2 (Rate Generator).
- Made 3 (Square Wave Generator)
- Mode 4 (software Triggered Stroke)
- Mode 5 (Hardware Thiggered stroke)..

Square wave Generator in 8254 :-

- When count N loaded is even a :- Output romains HIGH for half the count be Low for the rest half of the count.
- When count N loaded is odds (N+1) / 2 Output remains HIGH & LOW for (N-1/2)
- Repeated operation gives square. wave.
- Generates a continuous square-wave with G set to 1
- If count is even, 50% duty cycle otherwise OUT is high 1 cycle longer.

Algorithm :-

Step 1: Start

Step 2. Initialize the data segment.

Step 3: Store TTH in 'al' register

Step 4: Send a byte from 'al' to port 86H

Step 5: Store 0 in al register

Step 6: Send a byte from 'al' to port 82 H.

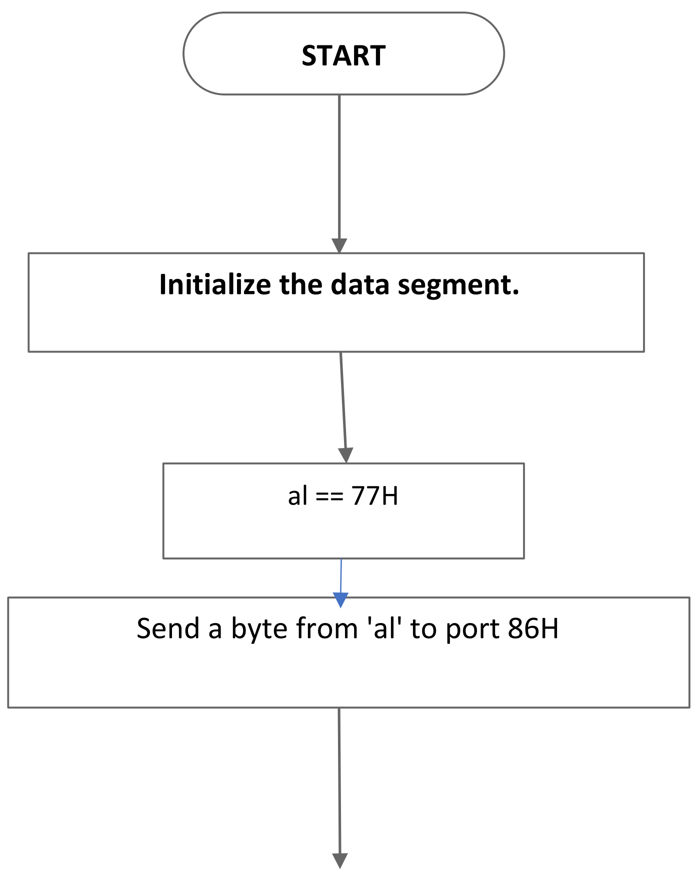
Step 7: Store 10H in al register

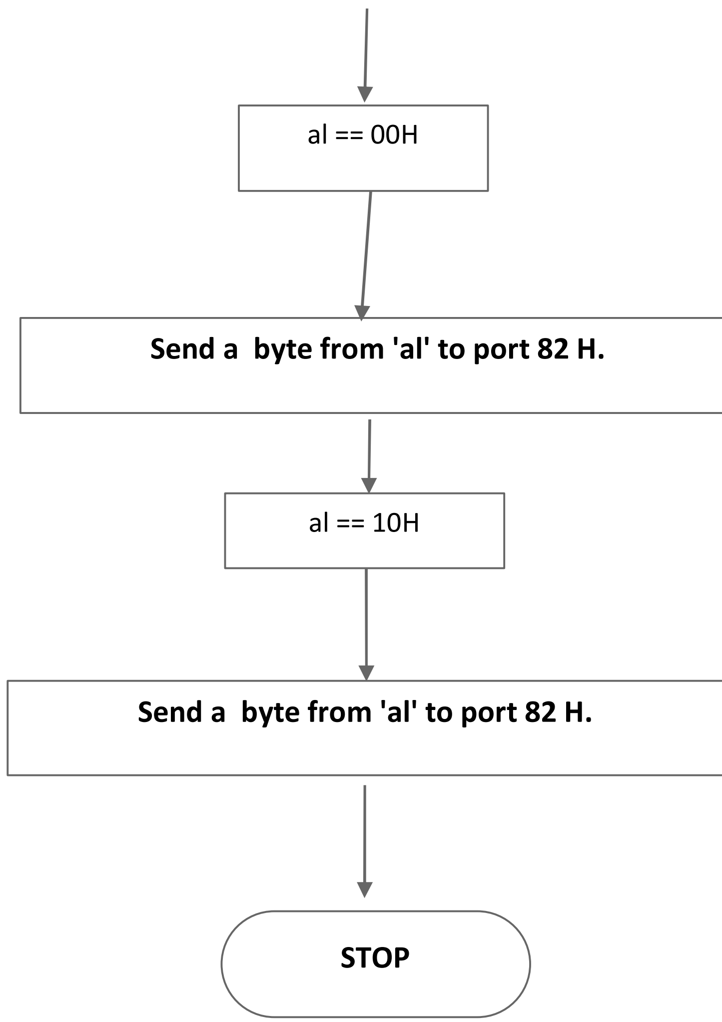
Step 8: Send a byte from 'al' to part 82H.

Step 9: Terminate the program

Step 10: Stop

Flowchart :-





Program :-

```
.model small
.data
.code

mov ax, @data
mov ds, ax
mov al, 71H
out 86H, al
```

```
Back :
      mov al, 00H
      out 82H, al
mov al, 10H
out  82H, al
Ends
End
```

Output: