



Practical. NO. 1.

Aim - Program Develop and execute ALP to 16 bit number

- Program of addition of 16 bit number, result is 16-bit
- Program of addition of 16bit number, result is more than 16-bit.

Theory -

Instructions -

i DW - It is used where we need to take to take a 16 bit word on a logical 8086 16 bit system. So, an operator 'num1' is declared with 16 bit storage.

ii DUP - It allows a sequence of storage location to be defined or reserved is only used as an operand of a defined directive

iii MOV - The mov instruction copies the data item referred to by it's second operand
Ex. MOV DX, DATA
initialize data segment.

iv ADD - It is used in addition it is used to add operand ex- ADD AX, BX



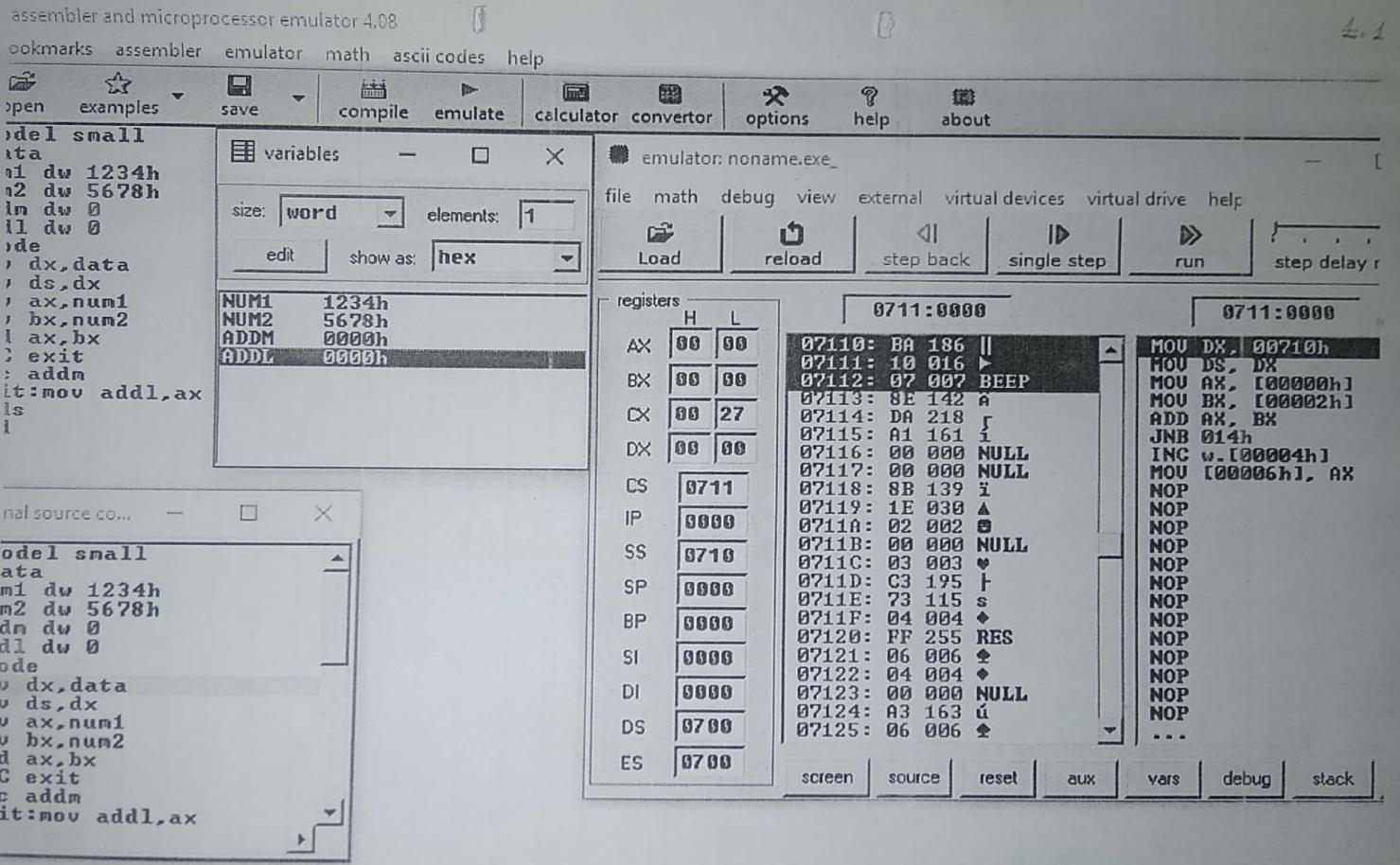
v ENDS - The code segment ends

vi JNC - This command stands for Jump if no carry and is used if to jump to a specific segment if there is no carry.

- a) Algorithm -
- i) Initialize data segment
 - ii) Load first number from memory in register
 - iii) Add second number with first no.
 - iv) Store result
 - v) Step.

program -

```
model small
    .data
        num1 dw 1234H
        num2 dw 5678H
        addm dw 0
        addl dw 0
    .code
        mov dx, datar
        mov ds, dse
        mov ax, num1
        mov bx, num2
        add ax, bx
        jnc exit
        inc addm
        exit : mov addl, ax
        ends
end
```





B)

Algorithm - i) Initialize segment

ii) load first number in register

iii) Add second number with first number.

iv) Check if result is greater than 16bit,
then go to step 5 else step 6

v) Increment M9B counter

vi) Store result

vii) Stop.

Program -

.model small

exit: mov addl, ax

.data

ends

num1 dw 0FFFFh

end.

num2 dw 0FFFh

addm dw 0

addl dw 0

.code

mov ah, data

mov ds, ah

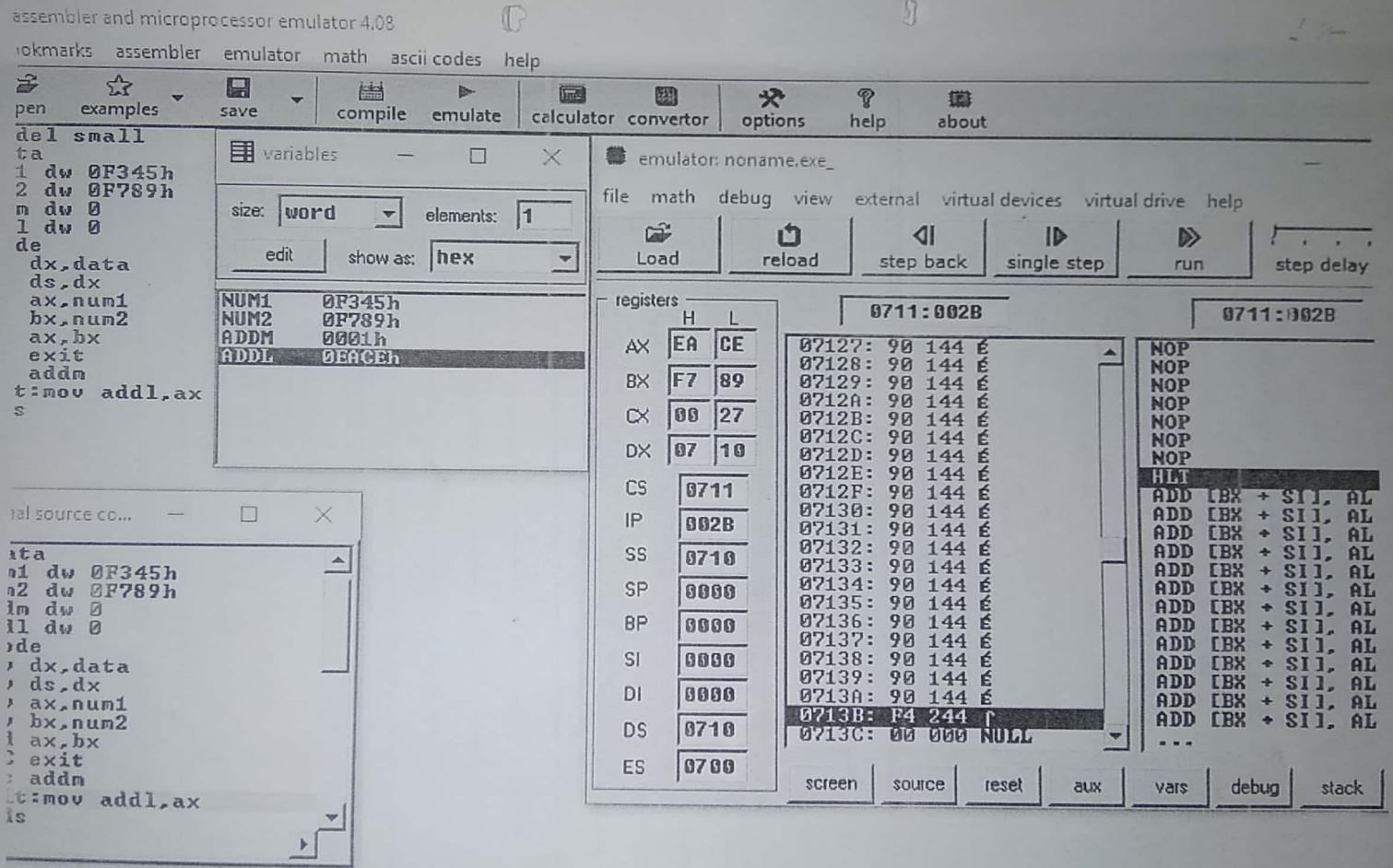
mov ax, num1

mov bx, num2

add ax, bx

JNC exit

inc addm





(8)

function addition () - multiplexer

and gate to receive both input ()

and output from both buses ()

total code required is sum of both ()

both side B gate is of result

minimum FET's requirement ()

minimum width ()

gate ()

→ minimum

minimum latency

min. value term : max

also

bus

minimum width

maximum width

→ min. width

→ max. width

also

width, width, width

Result:- Hence, we have successfully developed & executed ALP for addition of two 16 bit no.

PRACTICAL NO.2

Aim:- Develop & execute ALP to perform subtraction of two 16 bit number



Practical No 2 Program 1

Page No.

Date

Aim - Develop and Execute ALP to perform subtraction of two 16 bit numbers.

Theory -

Instructions -

i sub - It is used in subtraction,
it is used to subtract the operands

Ex. SUB AX to BX

Algorithm for subtraction -

- i) Initialize data segment
- ii) load first number from memory
- iii) subtract second no. from first
- iv) store the result
- v) stop.

program -

```
model small
data
num1 dw 1234h
num2 dw 5678h
sub1 dw 0
code
```



Page No.

Date

mov clc, @date

mov ds, dsc

mov dsc, num1

mov b, sc, num2

sub qsc, bsc

mov sub1, qsc

ends

end

36 - assembler and microprocessor emulator 4.08

bookmarks assembler emulator math ascii codes help

open examples save | compile emulate calculator convertor options help about

```
.model small
.data
num1 dw 1234h
num2 dw 5678h
subl dw 0
.code
mov dx,@data
mov ds,dx
mov ax,num1
mov bx,num2
sub ax,bx
mov subl,ax
ends
end
```

original source code

```
.model small
.data
num1 dw 1234h
num2 dw 5678h
subl dw 0
.code
mov dx,@data
mov ds,dx
mov ax,num1
mov bx,num2
sub ax,bx
mov subl,ax
ends
end
```

flags

CF	1
ZF	0
SF	1
OF	0
PF	0
AF	1
IF	1
DF	0

analyse

emulator: noname.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L	0710:0004	0711:0025
AX	BB	BC	07104: BC 188 11	NOP
BX	56	78	07105: BB 187 11	NOP
CX	00	21	07106: 00 000 NULL	NOP
DX	07	10	07107: 00 000 NULL	NOP
CS	0711		07108: 00 000 NULL	NOP
IP	0025		07109: 00 000 NULL	NOP
SS	0710		0710A: 00 000 NULL	NOP
SP	0000		0710B: 00 000 NULL	NOP
BP	0000		0710C: 00 000 NULL	NOP
SI	0000		0710D: 00 000 NULL	NOP
DI	0000		0710E: 00 000 NULL	NOP
DS	0710		0710F: 00 000 NULL	NOP
ES	0700		07110: BA 186 11	HLT

screen source reset aux vars debug stack flag

variables

size: word elements: 1

edit show as: signed

NUM1	1234h
NUM2	5678h
SUBL	-17476

practical. NO. 3

Aim :- Develop and execute an ALP to find sum of series of numbers.



Practical No. 3

Aim :- Develop and Execute an ALP to find sum of series of numbers.

• Program :-

data segment

array dw 01, 02, 03, 04, 05, 06, 07, 08, 09, 10

sum_low dw 0000H

sum_up dw 0000H

ends

code segment

start:

assume cs:code, ds: data

mov dx, data

mov cl, 0AH

mov ax, 0000H

mov si, offset array

up : mov ax, [si]

add sum_low, dx

inc next

next inc sum-up

next : inc si

inc si

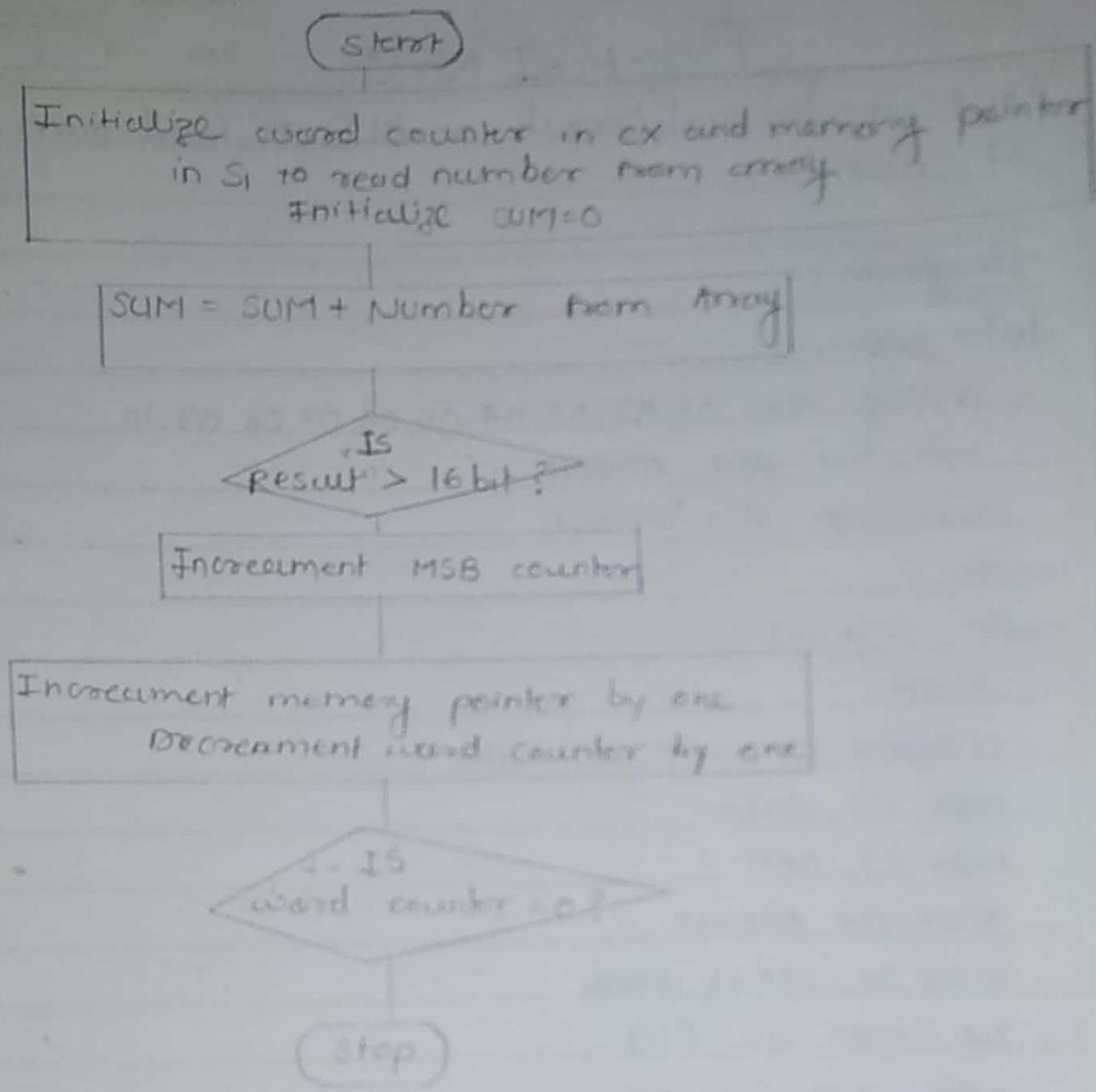
loop up

mov ax, 4C00H

ends

end start

- flowchart of sum of series



Conclusion: Hence, we successfully developed the required sum program.

bookmarks assembler emulator math ascii codes help

open examples save compile emulate calculator converter options help about

```
.model small
.data
array dw 1111h,2222h,3333h,4444h,5555h,6666h,7777h,8888h,9999h,0000h
sum_lsb dw 0
sum_msb dw 0
.code
mov ax,@data
mov ds,ax
mov cx,0ah
mov si,offset array
up:
mov ax,[si]
add sum_lsb,ax
jnc next
inc sum_msb
next:
inc si
inc si
loop up
end
```

original source code

```
.data
array dw 1111h,2222h,3333h,4444h,5555h,6666h,7777h,8888h,9999h,0000h
sum_lsb dw 0
sum_msb dw 0
.code
mov ax,@data
mov ds,ax
mov cx,0ah
mov si,offset array
up:
mov ax,[si]
add sum_lsb,ax
jnc next
inc sum_msb
next:
inc si
inc si
loop up
end
```

flags

	H	L
OF	0	0
ZF	0	0
SF	0	0
DF	0	0
PF	1	0
CS	0712	
IP	002F	
SS	0710	
SP	0000	
IF	1	0
AF	0	0
DF	0	0

analyse

registers

	0712:002F	0712:002F
AX	00 00	07140: 90 144 E
BX	00 00	07141: 90 144 E
CX	00 00	07142: 90 144 E
DX	00 00	07143: 90 144 E
CS	0712	07144: 90 144 E
IP	002F	07145: 90 144 E
SS	0710	07146: 90 144 E
SP	0000	07147: 90 144 E
BP	0000	07148: 90 144 E
SI	0014	07149: 90 144 E
DI	0000	0714A: 90 144 E
DS	0710	0714B: 90 144 E
ES	0700	0714C: 90 144 E

emulator noname.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

calculator - expression evaluator

show result as:

decimal hex oct bin

treat hex,oct,bin as:

signed word byte clear help

1111h+2222h+3333h+4444h+5555h+6666h+7777h+8888h+9999h+0000h
0FFF0h



* Algorithm :-

- i Initialize data segment

- ii Initialize byte counter and memory pointer to read number from array.

- iii Initialize sum variable with zero.

- iv sum = sum + Number from array

- v If sum > 8 bit, then goto step 6 else step 7

- vi Increment MSB result counter

- vii Increment memory pointer

- viii Decrement byte counter

- ix If byte counter = 0, then goto step 10, else 1.

- x Step

Theory :- i Loop - A loop is a sequence of instruction that is continuously repeated until a certain condition is reached. Ex. loop up.

* Viva Questions :-

Q. what is MOV?

→ It is a instruction used to copies the data item referred to by its second operand.

Result :- Hence, we develop and execute sum of series program successfully.

Result:- Hence, we have successfully developed & executed sum of series.



Page No.

Date

Practical NO. 4

Aim :- Develop and Execute an ALP to multiply two 16 bit numbers.

Theory :- a. Multiplication of unsigned numbers

- Algorithm -
 1. Initialize data segment
 2. Load first number
 3. Multiply first number with 2nd number.
 4. Store result.
 5. Stop.

- Program -

- .model small

- .data

- num1 dw 1234H

- num2 dw 5678H

- result_lsw dw 0

- result_msw dw 0

- .code

- mov ax, @data

- mov ds, ax

- mov ax, num1

- mov dx, num2

- mov result_lsw, ax

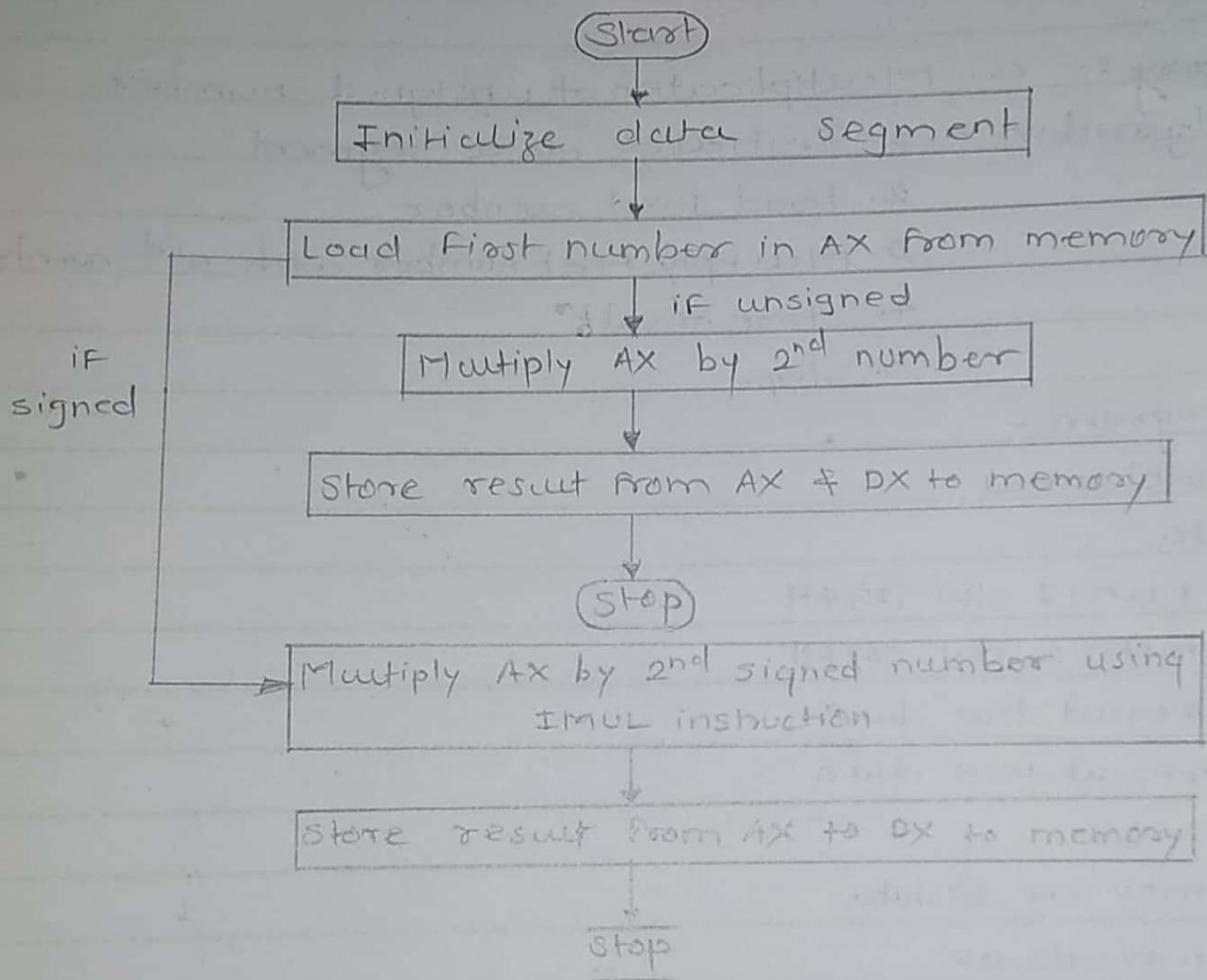
- mov result_msw, dx

- ends

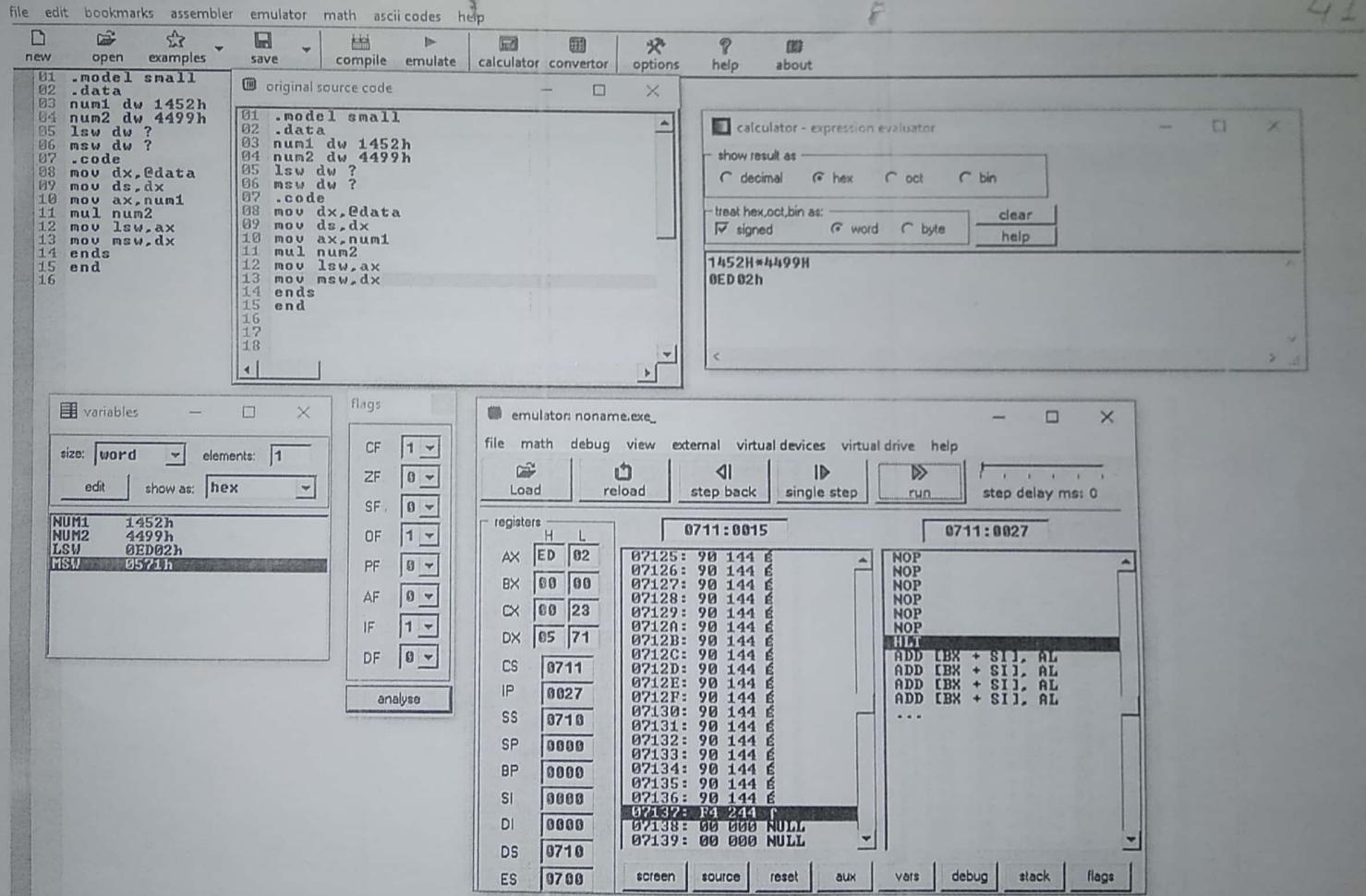
- end

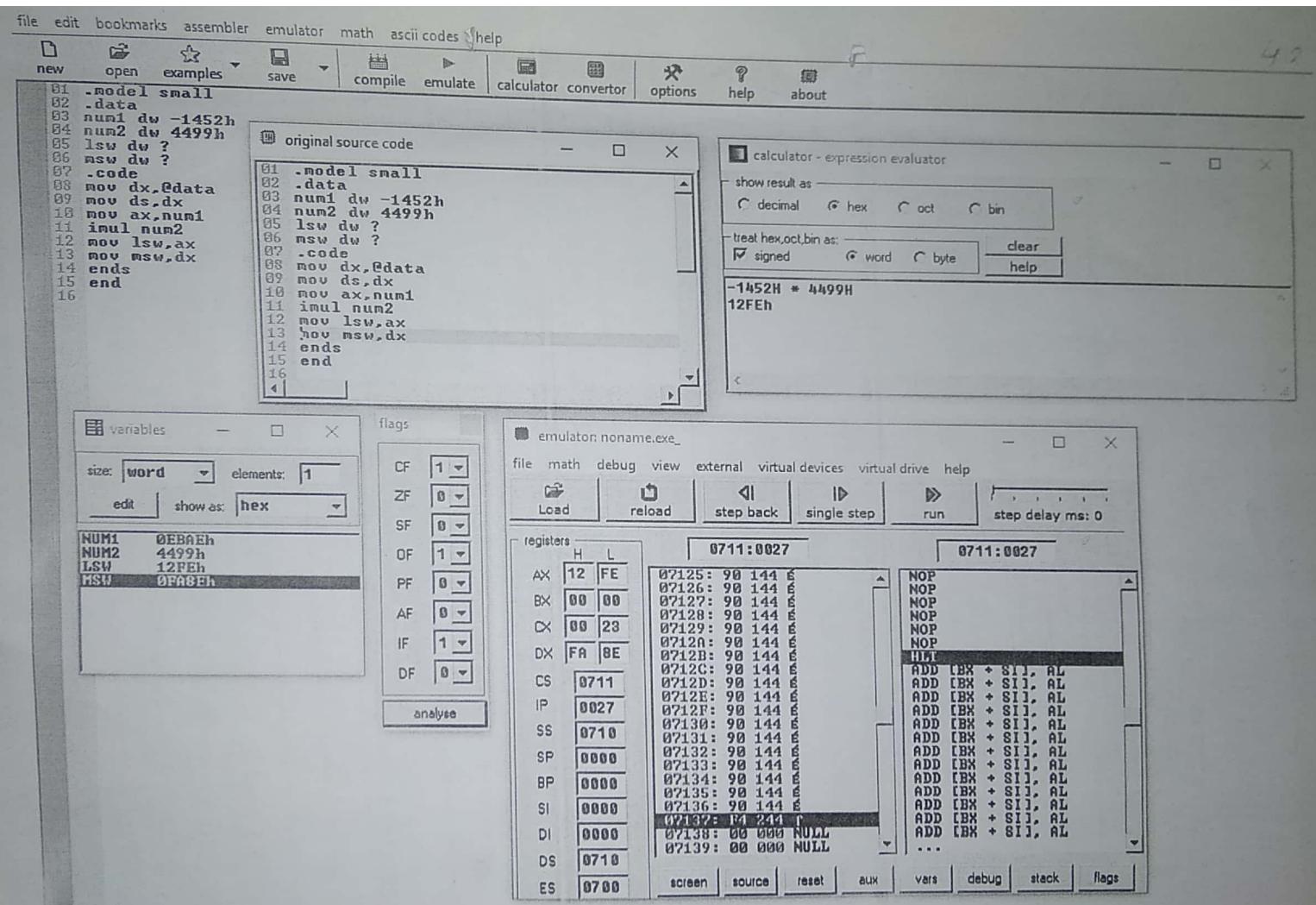
- Instructions - MUL :- The MUL multiplies first unsigned number with the second one ex:- mul num2.

- Flowchart to multiply two unsigned numbers



Conclusion :- Hence, we have successfully developed & executed two multiplication ALP programs, of 16 bit numbers.







B. Multiplication of signed with unsigned number

• Algorithm

1. Initialize data segment
2. Load first number
3. Multiply first No. with 2nd
4. Store result + stop.

• Program

.model small

.data

num1 dw -1234H

num2 dw +5678H

result_lsw dw 0

result_msw dw 0

.code

mov ax @data

mov ds, ax

mov ax, num1

imul num2

mov result_lsw, dx

mov result_msw, dx

ends

end

- Instructions - IMUL :- The instruction imul is used to multiply signed operands Ex- imul num2.

• viva questions :-

i) What is AX?

→ AX is an accumulator register

ii) What is maximum clock frequency of 8086 micro processor?

→ 5MHz

Result :- 1 Input → num1 : 1234H 2 Input → num1 : -1234H

num2 : 5678H

num2 : +5678H

Output → LSH : 0060H

Output → LSH : OFFA0H

MSW : 0626H

MSW : OF30H

Result: Hence, we have successfully developed a
sequential ALU for multiplication of
16 bit numbers.



Practical No. 5

Aim :- Develop and Execute an ALP to divide two 16 bit numbers.

Theory :- A. Division of two unsigned numbers.

• Algorithm

• program

1. Initialize data segment .model small
2. Load first number .data
3. Divide first number by second number dividend dw 6547h divisor dw 3421h
4. Store quotient & remainder quotient dw 0 remainder dw 0
5. Stop.

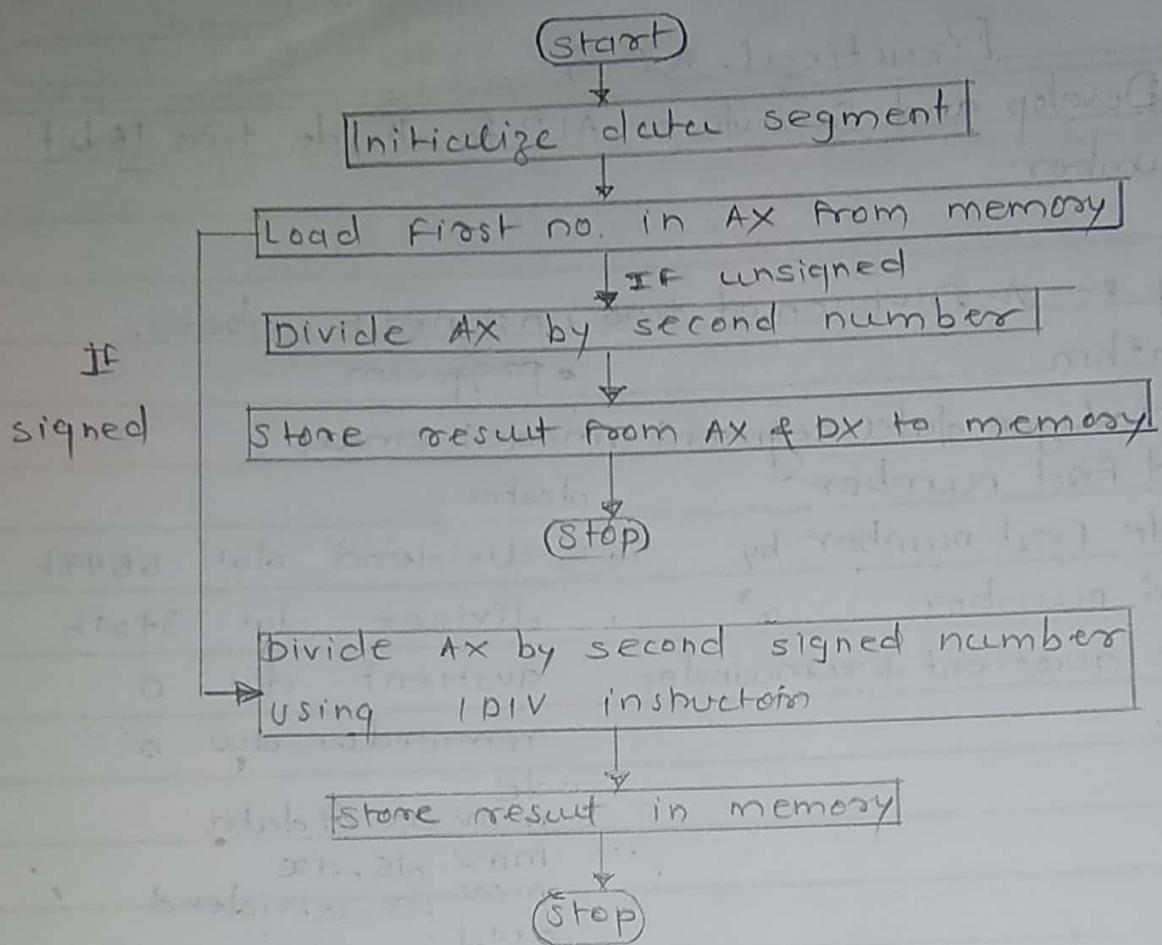
• code

```
mov ax @data  
mov ds, ax  
mov ax, dividend  
cwd;  
div divisor  
mov quotient, dx  
mov remainder, dx  
ends  
end
```

- Instructions - a. DIV : it is used to divide a number with a number next to it. Ex. div divisor

B. CWD : it is used to convert singed byte to double word

- Flowchart for unsigned & signed division



Conclusion:- Hence, we have developed & executed two 16 bit division ALP programs, successfully



The screenshot shows a Windows application window with two panes. The left pane displays assembly code:

```
.model small
.data
dividend dd 123455h
divisor dw 1234h
quotient dw 0
rem dw 0
;code
mov ax,0
mov dx,0
mov ax,word ptr dividend
mov dx,word ptr dividend+2
div divisor
mov quotient,ax
mov rem,dx
end
```

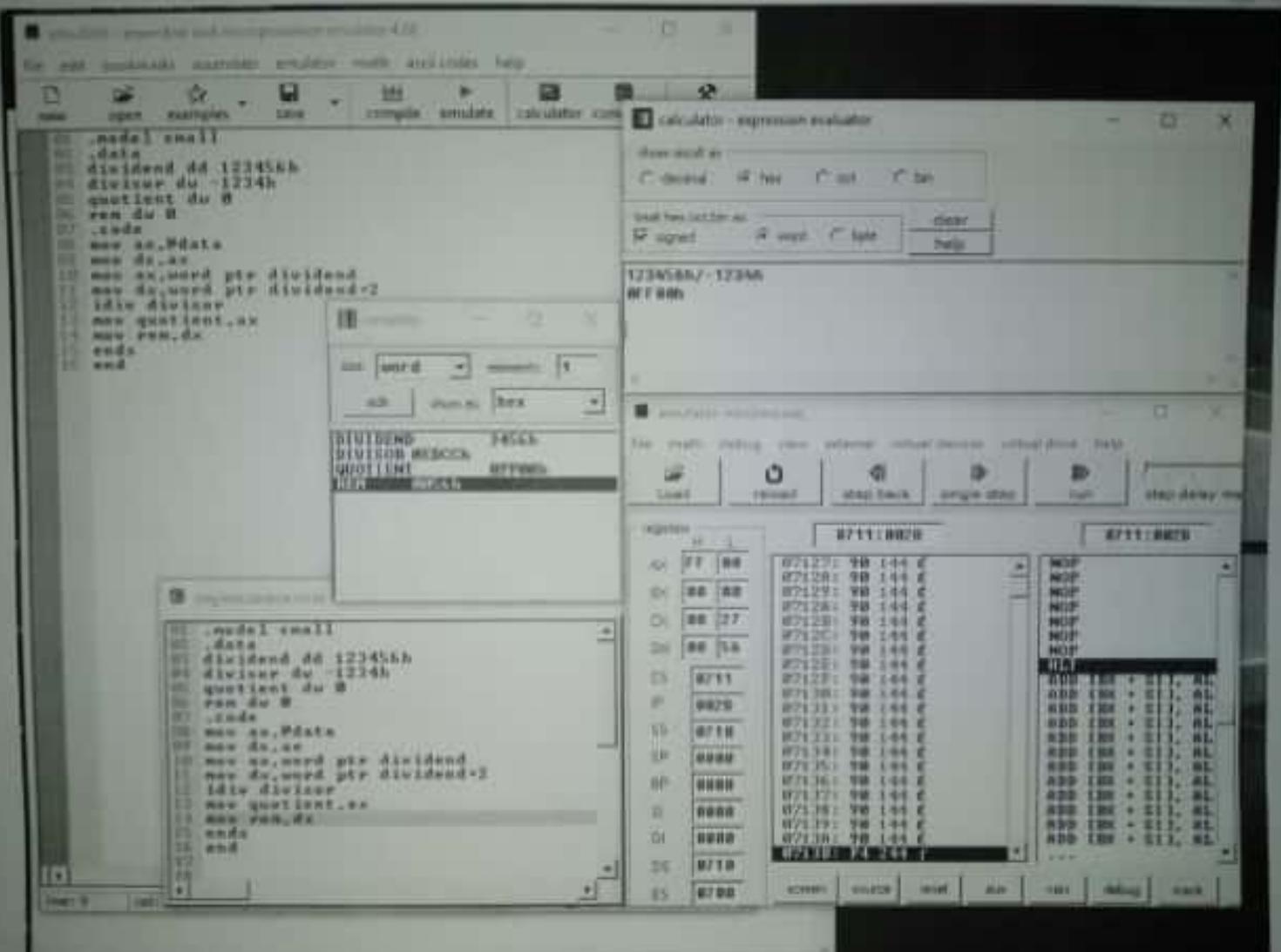
The right pane shows a memory dump window with the following data:

	word	content
00000000	dividend	123455
00000004	divisor	1234
00000008	quotient	0000
0000000C	rem	0000

123456789 / 123456789

Digitized by srujanika@gmail.com

Load **Record** **Stop Watch** **Single Step** **Run** **Help Online**





B. Division of signed number with unsigned number.

• Algorithm

+ Initialize data segment .model small

2. Load 16 bit dividend ~~as~~ .data

3. Divide fst no. with 2nd no.

4. Store quotient & remainder

5. Stop.

• program

dividend dw -6547h

divisor dw +3421h

quotient dw 0

remainder dw 0

• code

mov dx @data

mov ds, dx

mov ax, dividend

endl;

idiv divisor

mov quotient, ax

endl;

end

• Instructions - IDIV :- it divides signed number with unsigned no. Ex. idiv num2/divisor.

• Viva Questions

+ What is difference between DIV and IDIV in 8086?

→ DIV : It only used in unsigned numbers

IDIV : It only used in signed numbers

During multiplication bits are stored in which register

→ AH = High order 8 bits ; AL = Lower order 8 bits.

Result :-

1 Input → num1 : 6547h

num2 : 3421h

2 Input → num1 : -6547h

num2 : +3421h

Output → Quotient : 0001h

Output → Quotient : OFFFFh

Remainder : 3126h

Remainder : ECEDAh



Practical NO. 6

Aim:- Develop and Execute an ALP to find smallest numbers.

- Theory :-
- Algorithm -
 1. Initialize data segment
 2. Initialize byte counter + memory pointer.
 3. Read number from array.
 4. Increment memory pointer to read next number.
 5. Decrement byte counter.
 6. compare two numbers
 7. If number < next number then goto step 9.
 8. Replace number with next number which is smallest
 9. Increment memory pointer to read next number.
 10. Decrement byte counter by +
 11. If byte counter ≠ 0, then goto step 6
 12. store result.
 13. stop.

• program

.model small

.data

array dw 1324h, 2134h, 4512h, 3210h, 1112h,
4563h, 3122h, 9999h, 7654h, 3216h

small dw 0

,code

mov ax, @data

mov ds, ax

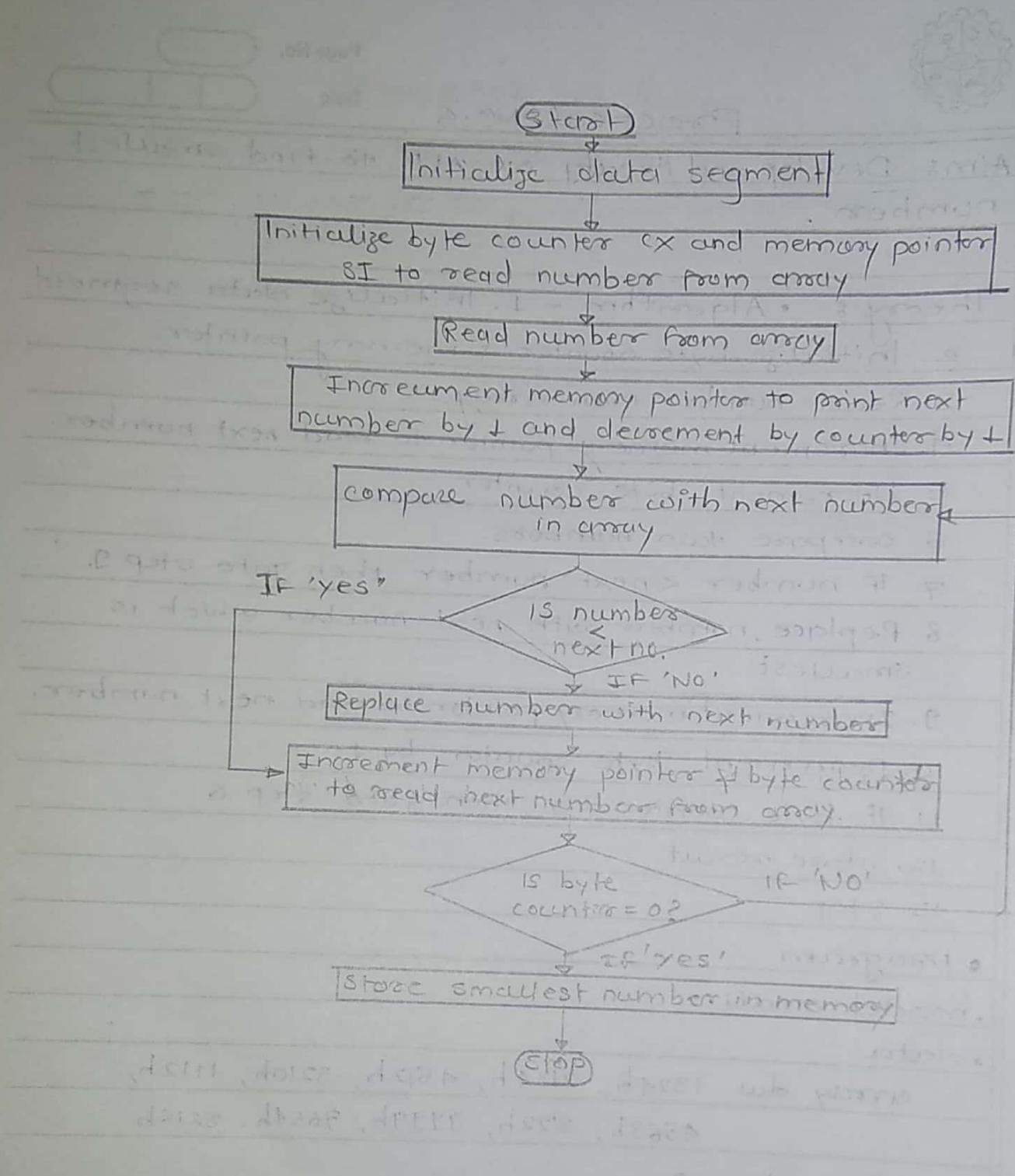
mov ax, 0ah

mov si, offset array

mov dx, [si]

dec cx

- Flowchart to find smallest number in an array.



Conclusion:- Hence, we successfully developed & executed ALP to find smallest number in array.

```
model small
data
array dw 1234h,4567h,6789h,1222h,3452h,7898h,3
small dw 0 h
.code
mov ax,data
mov ds,ax
mov es,00h
mov si,offset array
mov ax,1000h
dec cx
repz ax,si
inc si
cmp ax,1000h
je next
mov ax,1000h
next:loop up
mov small,ax
end
```

Model: small

Data:

Array: dw 1234h,4567h,6789h,1222h,3452h,7898h,3

Small: dw 0 h

Code:

MOV AX,DATA
MOV DS,AX
MOV ES,00H
MOV SI,OFFSET ARRAY
MOV AX,1000H
DEC CX
REPZ AX,SI
INC SI
CMP AX,1000H
JE NEXT
MOV AX,1000H
NEXT:LOOP UP
MOV SMALL,AX
END

The screenshot shows the Win32Dbg debugger interface. The assembly pane at the top displays assembly code for address 0712:000F, showing instructions like NOP, ADD, and MUL. The registers pane shows CPU register values. The stack pane shows the current stack contents. The memory pane shows memory dump data. Navigation buttons like Load, Reset, Step Back, Single Step, and Stop are visible. A status bar at the bottom shows the current address (0712:000F), update frequency (update), and file paths.



up:

```
inc si  
inc si  
cmp ax,[si]  
ja next  
mov ax,[si]
```

next:

```
loop up  
mov small,ax  
ends  
end.
```

- Instructions - a. cmp :- it compares numerical values by destination with source and sets flag appropriately

Ex. cmp AL,[si]

- b. dec :- The dec is used to decreament source by ± 1

Ex. dec ax.

- c. jc :- it is a address operand. It jumps to the address if carry flag is 1. Ex. jc next.

• Viva Questions

1. How many flags are there in 8086 microprocessor

→ There are nine flag in 8086 microprocessor.

Result :-

ARRAY 1324h
SHALL 1112h



Practical No. 7

Aim :- Develop and Execute an ALP to find Largest number from array.

Theory :-

- Algorithm.
 - 1. Initialize data segment.
 - 2. Initialize word counter and memory pointer
 - 3. Read number from the array.
 - 4. Increment memory pointer to read next number.
 - 5. Decrement word counter.
 - 6. Compare two numbers
 - 7. Replace first number with second number.
 - 8. Increment memory pointer to read next number.
 - 9. Decrement word counter by 1
 - 10. If word counter $\neq 0$, then goto step 6.
 - 11. Store result and stop.

• Program.

.model small

.data

array dw 1324h, 2134h, 4512h, 3210h, 1112h,
4563h, 3122h, 9999h, 7654h, 3216h

.code

mov dx, @data

mov ds, ax

mov cx, 0Ah

mov si, offset array

mov ax, [si]

dec cx

File Edit Options View Assembler Simulator Watch Registers Help

File Edit Debug View External Virtual Devices Virtual Drive Help

Load Reload Step Back Single Step Run Stop Debugger

Registers	H	L	0712:002F	0712:002F
AX	78	90	0712:002F: 90 144 E	NOP
BX	00	00	0712:0031: 90 144 E	NOP
CX	00	00	0712:0032: 90 144 E	NOP
DX	00	00	0712:0033: 90 144 E	NOP
DS	0712		0712:0034: 90 144 E	NOP
ES	002F		0712:0035: 90 144 E	NOP
SS	0710		0712:0036: 90 144 E	NOP
BP	0000		0712:0037: 00 000 NULL	ADD EBX + ECX, BL
SP	0000		0712:0038: 00 000 NULL	ADD EBX + ECX, BL
S	0012		0712:0039: 00 000 NULL	ADD EBX + ECX, BL
DI	0000		0712:003A: 00 000 NULL	ADD EBX + ECX, BL
D0	0710		0712:003B: 00 000 NULL	ADD EBX + ECX, BL
EI	0700		0712:003C: 00 000 NULL	ADD EBX + ECX, BL

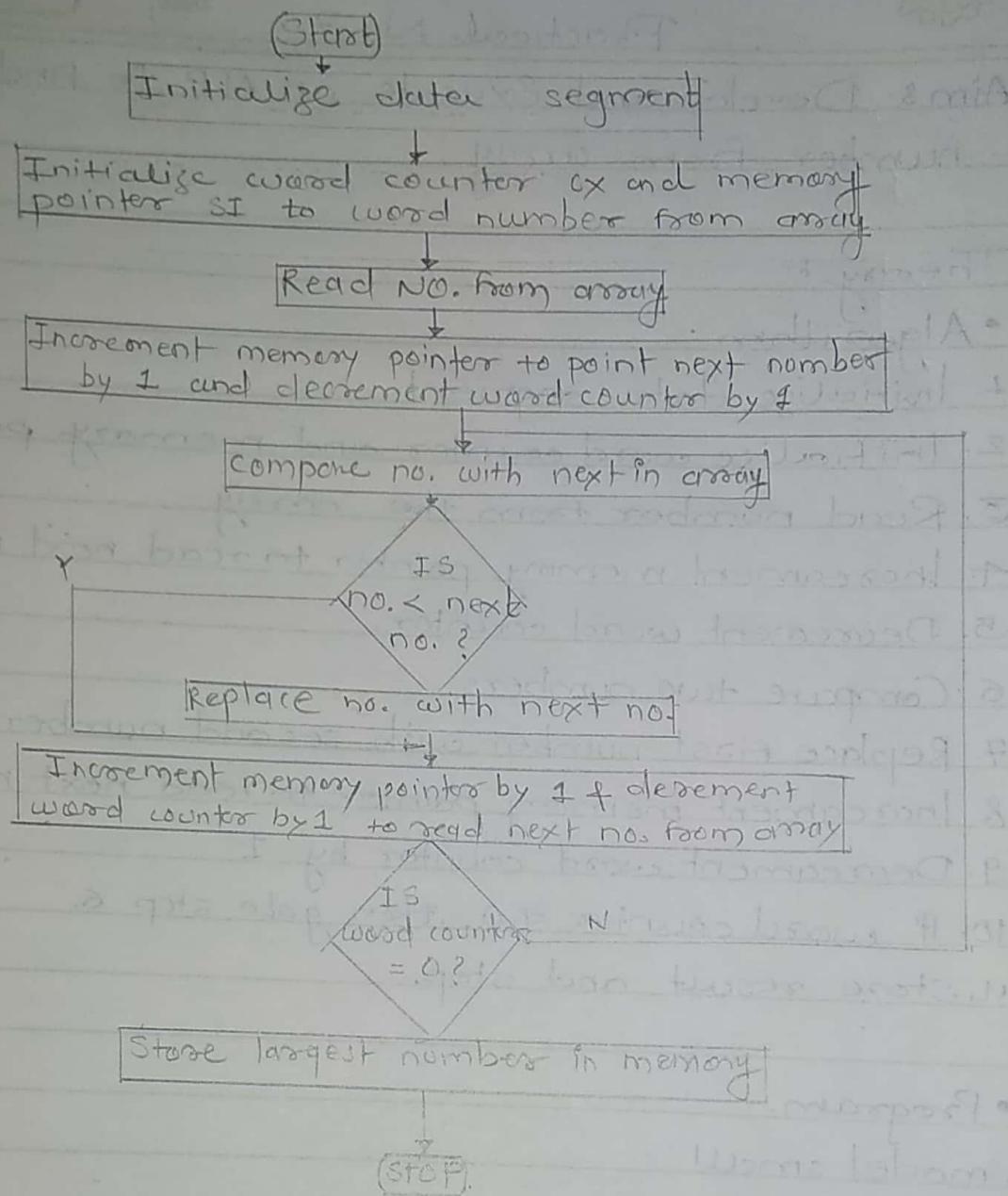
Run Word Address 1

0712:002F: 32345
0712:0030: 20000

Random Access Memory

0712:002F	update	Table	Set
0712:002F	FF		
0712:0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0031	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0032	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0033	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0034	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0035	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0036	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0037	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0038	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:0039	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:003A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0712:003B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

• Flowchart to find largest number from array.



Conclusion:- Hence, we have successfully find largest number from array by developing & executing ALP.



up:

inc si

inc si

cmp ax, [si]

jnc next

mov ax, [si]

next:dec ex

loop up

mov large, ax

ends

end.

- Instructions - JNC :- it is a address operand, it jumps to the address if carry flag is 0.
Ex:- jnc next.

• Viva Questions

Q What is the stack pointer?

→ Stack pointer is a special purpose 16 bit register in the microprocessor which holds the address of top of the stack.

Result :-

ARRAY 1324h

LARGE 9999h



Practical. NO. 8

Aim :- Develop and Execute an ALP to perform block transfer data.

- Theory :-
- Algorithm :-
 1. Initialize data and extra segments.
 2. Initialize word counters.
 3. Initialize memory pointer for source and destination array.
 4. Read number for source array.
 5. Copy it to destination array.
 6. Increment memory pointer for source & destination array for next number.
 7. Decrement word counter by 1.
 8. If word counter ≠ 0, then goto step 4. else Stop.

• Program

.model small

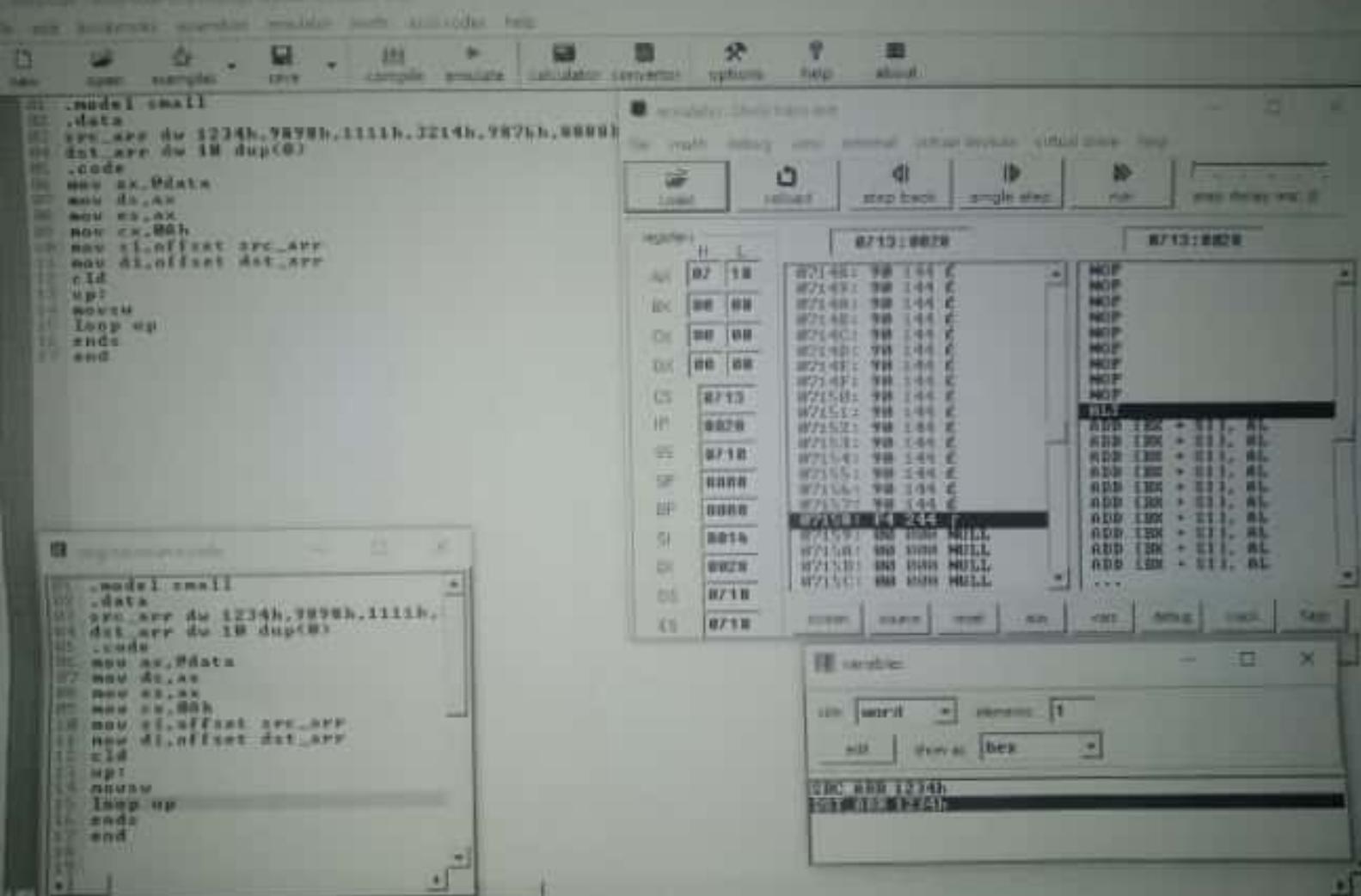
.data

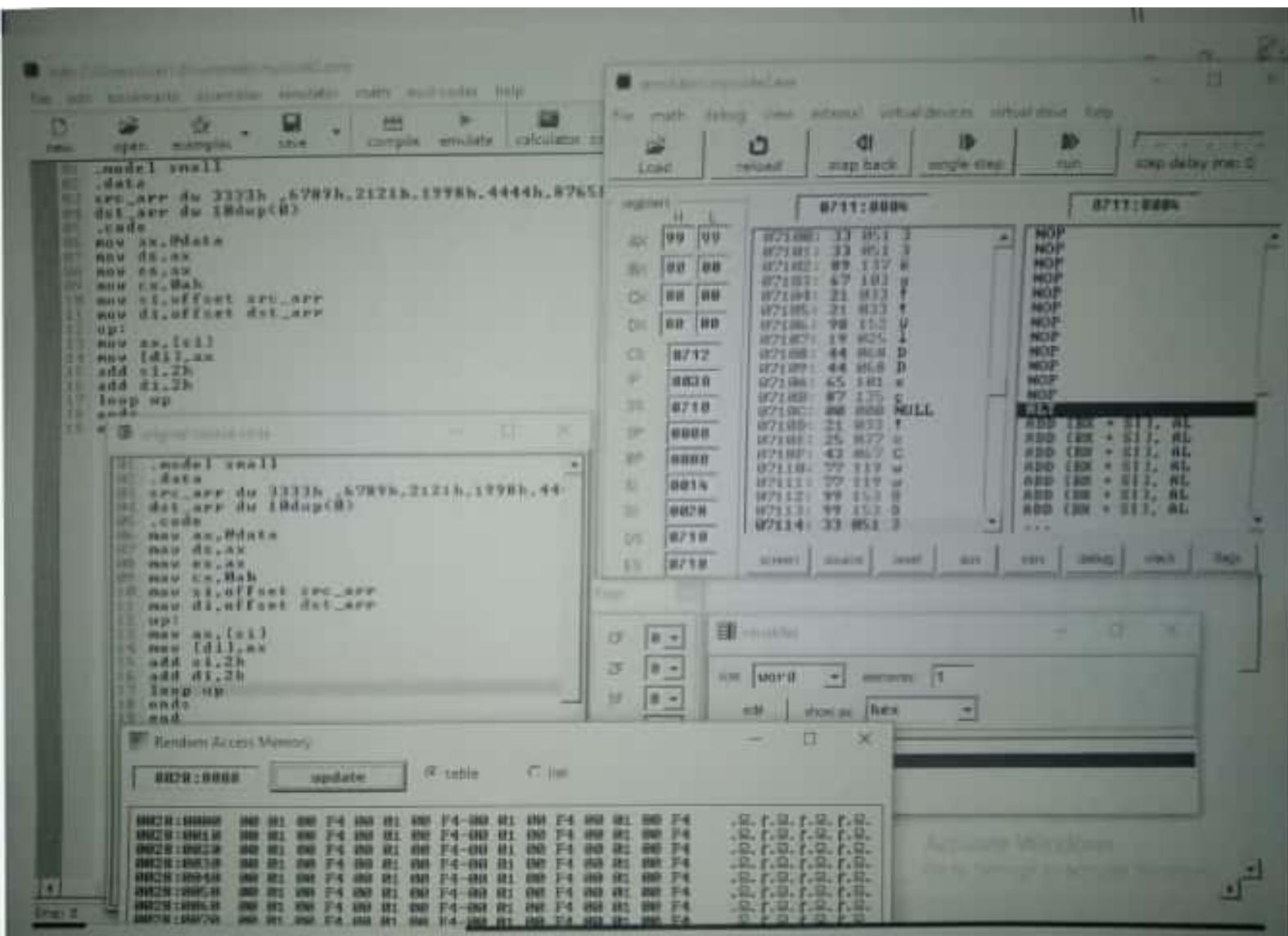
```
src_crw dw 1234h, 2345h, 3456h, 5678h, 6789h,  
        7890h, 1345h, 3254h, 7689h
```

```
dst_crw dw 10 dup (0)
```

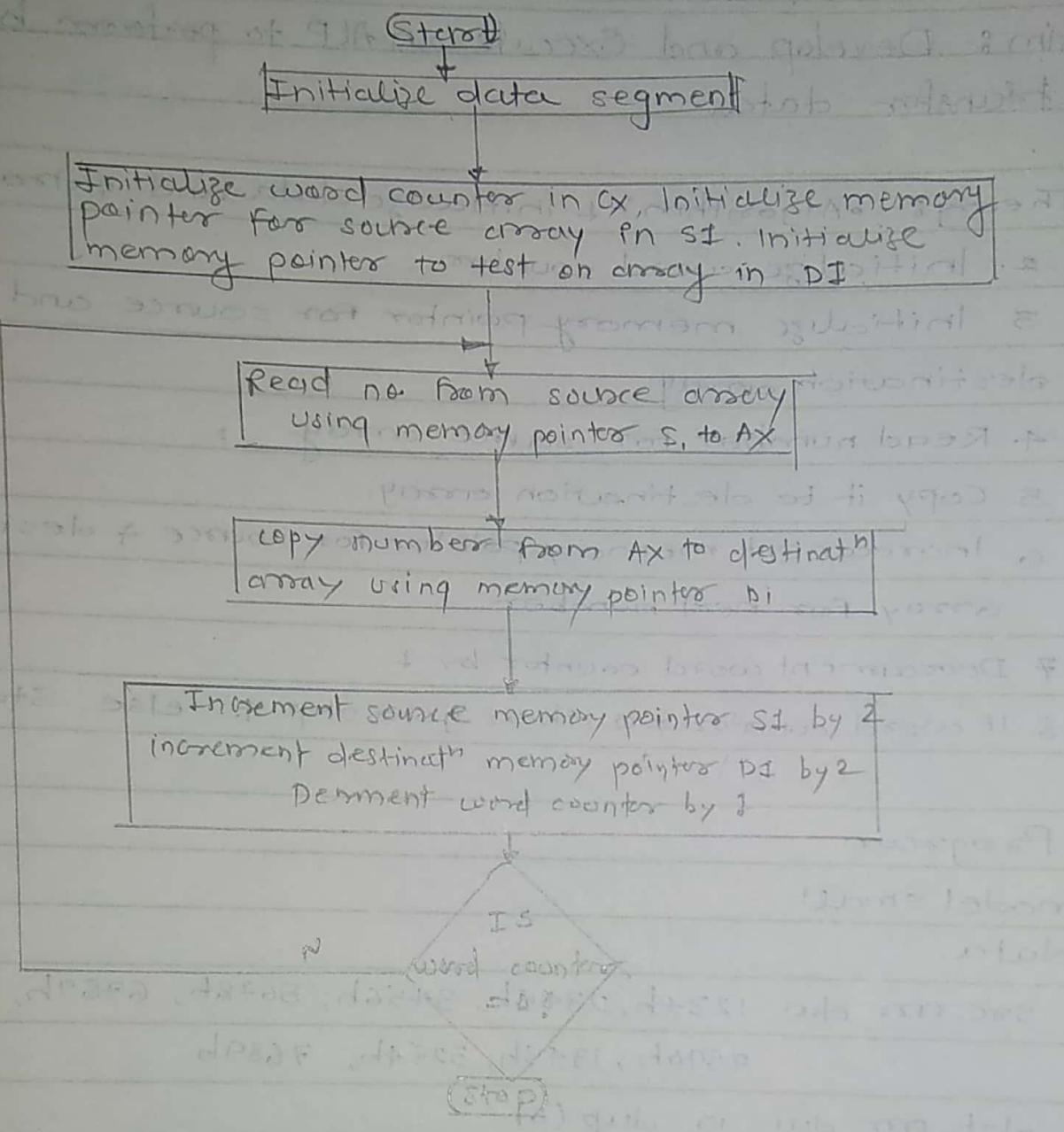
.code

```
    mov ax, @data  
    mov ds, ax  
    mov cs, ax  
    mov cx, 0ah  
    mov si, offset src_crw  
    mov di, offset dst_crw  
    eid
```





• Flowchart



Conclusion :- Hence, we successfully developed ALP to perform block transfer data.



up:

mov sw

loop up

ends

end

- Instructions - a. CLD :- It is used to clear the flag.
- b. REP :- It is a instruction used to repeat the instruction till CX = 0.

• Program

.model small

.data

.code

mov ax, 4000h

mov ds, ax

mov ax, 5000h

mov es, ax

mov si, 0000h

mov di, 0000h

mov cx, 0000h

rep mov sw

ends

end.

• Viva Questions

+ what is instruction set ?

→ It is the set of instruction that microprocessor can execute.

Result :- (i) SRC-ARR 1234h

DST-ARR 1234h



Practical NO. 9

Aim :- Develop and Execute an ALP to compare two string inst?

Theory :- • Algorithm -

1. Initialize data segment

2. Initialize extra segment

3. Take counter of ten.

4. Compare string characters by character.

5. If characters are not same, then goto step 8

7. If characters are same, then display "strings are same".

8. 'String are not same'

9. Stop.

• Program

.model small

.data

str_s db 'FearTheWalkingDead \$'

str_d db 'FearWalkingDead \$'

msg1 db 'string are same \$'

msg2 db 'string are not same \$'

.code

mov ax, @data

mov ds, ax

mov es, ax

mov cx, 0ah

mov si, offset str_s;

mov di, offset str_d;

up:

cmpsb

jne exitz

loop up

mov ah, 0ah

lea dx, msg1

The screenshot shows a debugger interface with the following components:

- Assembly View:** Displays the assembly code for the program. It includes labels like .model small, .data, and .code, and instructions like mov ax, @data, mov ds, ax, and cmpsb.
- Registers View:** Shows the current values of various CPU registers.
- Stack View:** Displays the stack contents, showing strings like "INFORMATIONS!" and "strings are same".
- Memory Dump View:** Shows memory starting at address F400:0200. The dump area is labeled "F400:0200" and contains hex values FF, 255, CD, 205, etc.
- Registers View (Bottom):** Shows additional CPU register values.
- Random Access Memory View:** Shows memory starting at address F400:0204. The dump area is labeled "F400:0204" and contains hex values CF, 00, 00, 00, 00, 00, 00.

Practical - 10

Aim - Develop and Execute an ALP to display length of string

Theory -

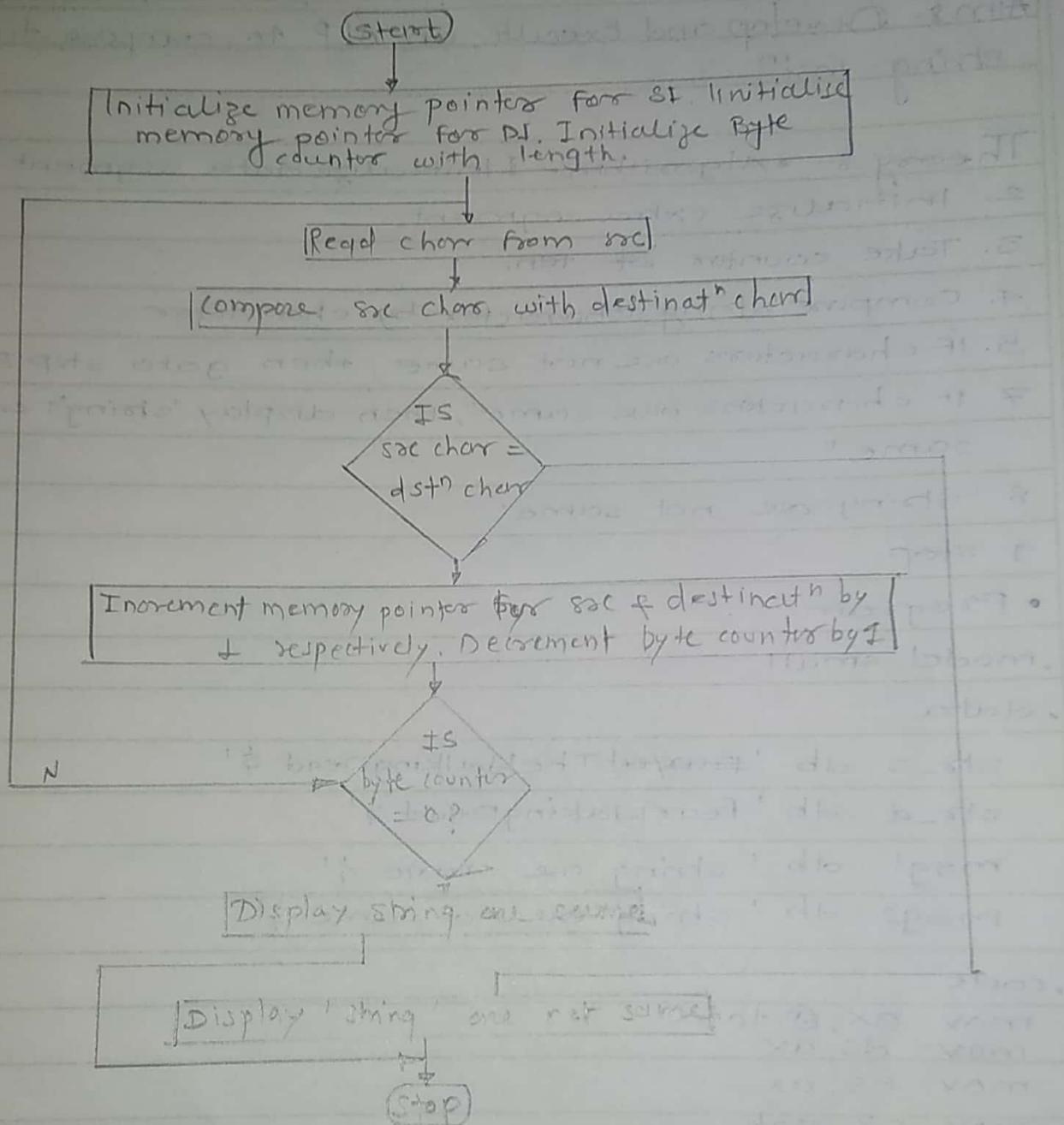
Algorithm.

1. Initialize data segment.
2. Initialize memory pointer in SI and a length counter with 0 for source string.
3. Read characters from source string.
4. Check if character = ?, if yes go to step 7
5. Increment length counter by 1, increment memory pointer by 1
6. Go to step 3
7. Display the length
8. Exit

Program

```
.model small
.data
str$ db 'INFORMATION$'
```

• Flowchart



Conclusion:- Hence, we successfully developed ALP to compose two strings.

length db dup(0)

code:

mov ax, @data

mov ds, ax

mov si, offset strg

next:

mov al, [si]

cmp al, '\$'

JNE exit

inc si

inc length

jmp next

exit:

end

End

Instructions:

1. LEA

This instruction is used to load address of operand into provided register



*Viva Questions

1 - Which character defines ~~when~~ a strings end ?

→ A ' \$ ' sign / character defines a strings end

2 - What is the use of LEA instruction ?

→ This instruction is used to load address of operand into provided register

Result :

STRS 'I', 'N', 'F', 'O', 'R', 'M', 'A', 'T', 'I', 'O', 'N'

LENGTH 0BH



```
int 21h  
jmp exit3  
exit 2:  
    mov ah, 0dh  
    lea dx, msg2  
    int 21h  
    exit3:  
exit 3:  
    mov ah, 4ch  
    int 21h  
    ends  
end.
```

- Instructions - a. CMPS :- It can be used to compare two strings of bytes or words
- b. Int (Interrupt Type) :- In interrupt structure of 8086 i.e. 256 interrupt are define corresponding to the type from 00H to FFH.

• Viva Questions

1. What is LEA ?

→ It is the load address of operand used to load into the provided registers.

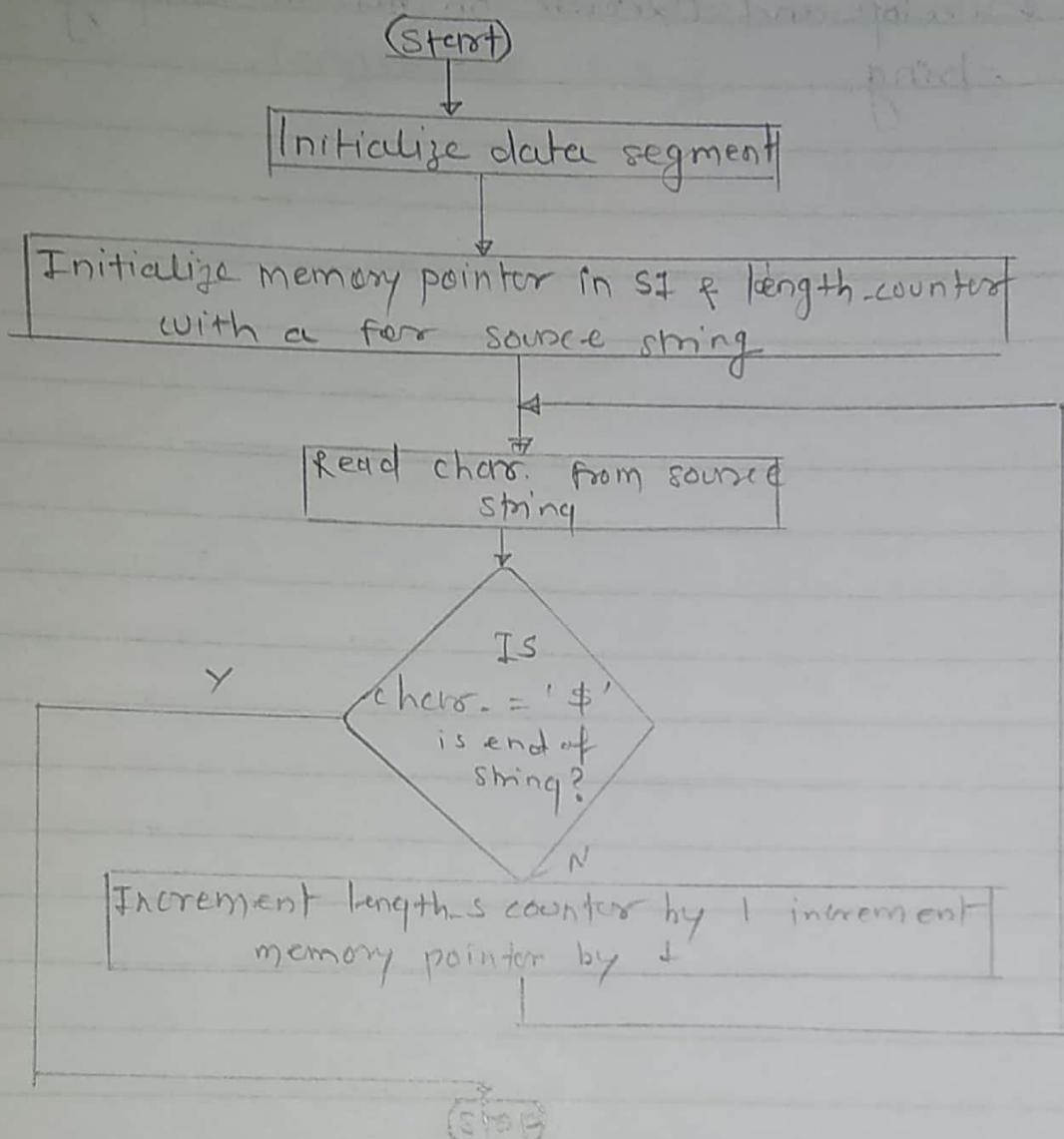
2. What is compare instruction?

→ This instruction is used to compare two string bytes

Result :-

Strings are not same.

- Flowchart





Practical No. 11

Aims :- Determine digital equivalent of a analog signal by ADC.

Apparatus :- 1. Microprocessor 8086

2. Power Supply +5V dc, +12Vdc

3. ADC Interface board - 1

Theory :- An ADC usually has two additional control lines : the SOC input to tell the ADC when to start the conversion and the EOC output to announce when the conversion is complete.

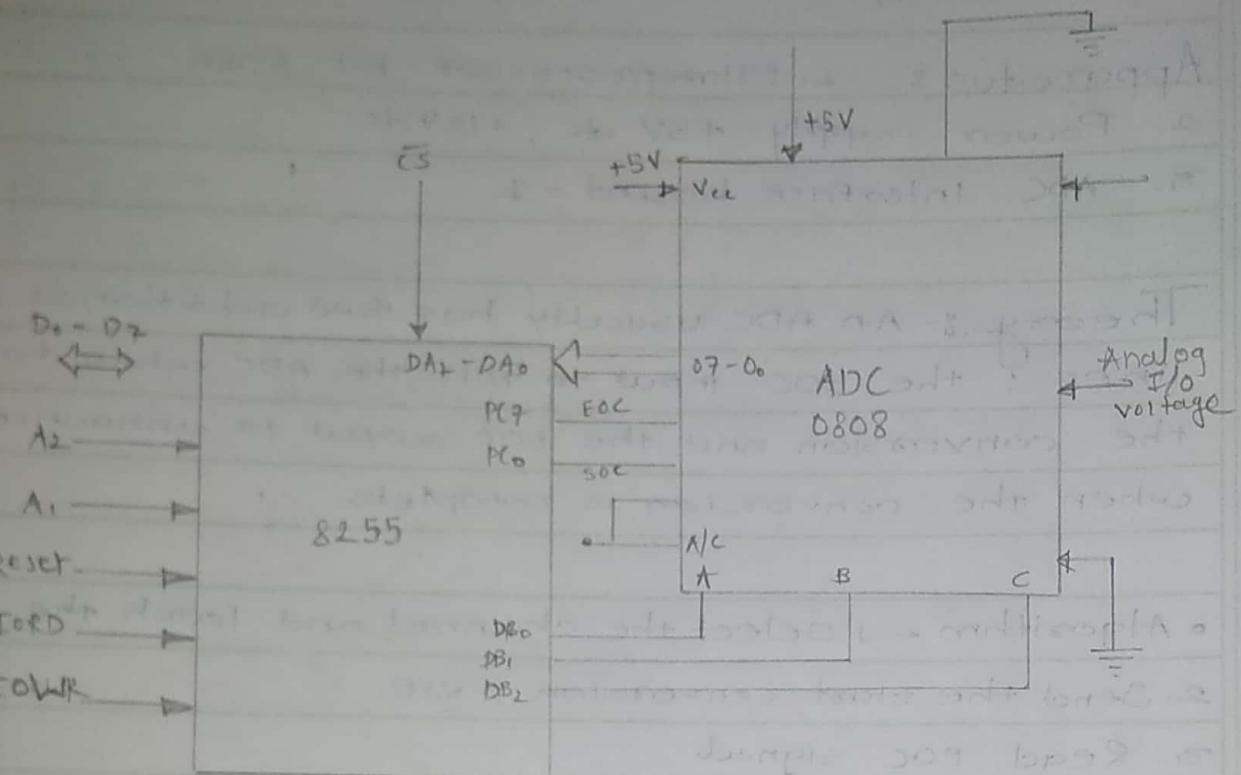
- Algorithm -
 - 1. Select the channel and lach the address
 - 2. Send the start conversion pulse.
 - 3. Read EOC signal
 - 4. If $EOC=1$ continue, else goto step 3.
 - 5. Read the digital output.
 - 6. Store it in a memory location

• Interfacing Analog to Digital data converters (cont...)

Example - Interfacing ADC 0808 with 8086 using 8255 ports

Use port A of 8255 for transferring digital data output of ADC to the CPU, and port C for control signals.

Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC



Interfacing 8080 with 8085 ports

(After) connection with 8080 at port A programmed.
Now 8080 port A has 8080 with programmed -
8080 ports 8080 programmed with 8085 to A busy 8080
programmed with 8085 has 8080 set at 8080 to
the 8080 to free 8080 program our task is
to connect alternative to 8080 ports to 8080 8080
ports and 8080 ports to 8080



Solution - The analog input I/P₂ is used and therefore address pins A, B, C should be 0, 1, 0 respectively to select I/P₂. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.

• Interfacing Analog to Digital data converters.

→ Port A acts as a 8-bit input data port to receive the digital data output from the ADC.

The 8255 control word is written as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	0	1	1	0	0	0

The required ALP is as follows:

```
MOV AL, 98h ; Initialize 8255 as  
OUT CWR, AL ; discussed above  
MOV AL, 02h ; select I/P2 as analog.  
OUT PortB, AL ; input.
```

• Interfacing Analog to Digital data converters

```
MOV AL, 00H ; Give start of conversion.  
OUT Port C, AL ; Pulse to the ADC  
MOV AL, 01H  
OUT Port C, AL  
MOV AL, 00H  
OUT Port C, AL
```



Intuit : IN AL, Port C ; Check for EOC by
RCR ; reading port C upper &
JNC Wait ; rotating through carry
IN AL, Port A ; If EOC, read digital equivalent
HLT ; in AL
; stop

Result :- Hence, we performed this practical successfully



Practical. NO. 12

Aims :- Develop and Execute an ALP to interface a stepper motor.

Theory :- • Stepper motor - A motor in which the motor is able to assume only discrete stationary angular positions is a stepper motor. The rotary motion occurs in a stepwise manner from one equilibrium position to the next two phase scheme : Any two adjacent stator windings are energized. There are two magnetic fields active in quadrature fields active in and none of the rotor pole faces can be in direct alignment with the stator poles. A partial but symmetric alignment of the rotor poles is of course possible. Stepper motor is IO device so CW should be (80)H. There are 4 poles in a typical stepper motor A1, A2, B1, B2.... energize the poles for anticlockwise rotation we have to apply the following to the poles. So we put the set of values to the IO port of 8255.

• Algorithm - For running stepper motor clockwise & anticlockwise directions :

1. Get the first data from the look up table.
2. Initialize the counter and move data into accumulator
3. Drive the stepper motor circuitry & interface delay
4. Decrement the counter is not zero repeat from 3
5. Repeat the above procedure both for backward & forward directions

Motion	Step	A	B	C	D
Clockwise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anti-clockwise	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0



• Program -

```
# Start = stepper_motor.exe #
name "Stepper"
#make_bin#
Steps_before_direction_change = 20h;
jmp start
; bin data for clockwise
dat cw db 0000_0110b
        db 0000_0100b
        db 0000_0011b
        db 0000_0010b
; bin data for counter clockwise
dat ccw db 0000_0011b
        db 0000_0001b
        db 0000_0110b
        db 0000_0010b
; bin data for clock wise
dat cw fs db 0000_0100b
        db 0000_0110b
        db 0000_0011b
        db 0000_0000b
start:
    mov bx, offset datcw
    mov si, 0
    mov cx, 0
next step:
    wait: in cl, 7
        test cl, 10000000b
        jz wait
    mov al, [bx][si]
    out 7, al
    inc si
    cmp si, 4
    jb next-step
```



```
mov si, 0  
inc cx  
cmp cx, step_before_direction_change  
jbe next_step  
mov cx, 0  
add bx, 4  
cmp bx, offset datccw-fs  
jbe next_step  
mov bx, offset datcw  
jmp next_step
```

• Instructions - & JMP :- The JMP command is used for the unconditional jump.

ii JBE :- This instruction means jump if below or equal.
It works when carry flag or zero flag is set.

iii. JE :- The JE command is used to jump when zero flag is set.

iv JB :- The JB command jumps when carry flag is set.

v Wait :- It checks pending unmasked floating point exceptions

vi Test :- It performs a bitwise AND on two operands

• Viva Question

i What is meant by stepper motor?

→ It's a DC motor that moves in discrete steps. It has multiple coils, organized in groups called 'phases'. After energizing each phase in sequence, motor rotates one step at a time.

Result :- Hence, the ALP for stepper motor in both anti-clockwise or clockwise direction is written and verified.

8

Conclusion :- Hence, we successfully developed & executed ALP to interface a stepper motor.

End of chapter

Practical - 13

Aim - Design and Prepare miniproject based on any one application of 8086 microprocessor.

Theory -

We are going to make a mini project which will be displaying a sample point 16×16 color map. It uses all possible colors.

What is a color map?

→ A color map (often called a color table or a palette) is an array of colors used to map pixel data (represented as indexes into the color table) to the actual color values.

Instructions

• org

It defines where the machine code (translated assembly program) is to place in memory. In our program we wrote - org 100h

here 100h says that machine

code starts from address (offset) in this segment.

- JE

- Is used for a conditional jump when ZF (the zero flag) is equal to 1.

- RET

- Is used to mark the end of sub-routine. It has no parameters. After execution of this instruction program control is transferred back to main program from where it had stopped.

Algorithm -

1. Define from where the machine code is to place in memory.
2. Set the video mode
3. Set the text mode (80x25). 16 colors. 8 pages
4. Disable blinking.
5. Set the current column, current row and attributes.
6. Jump to next char block (step 9)
7. In the nextrow block increment dh and compare dh to 16.
8. If zero flag is zero jump to stop point (step 12) and move al to dl
9. In next_char block set the cursor position $\langle al, dh \rangle$ and move the point to al.



10. Increment bl (i.e next attribute) and dl also.
11. Again compare dl to 16 if zero flag comes to zero jump to nextrow (step 7) or else to next-char / step 9)
12. In step-print block set cursor, column and row position. Now test the teletype output (that)
13. Now give the control to the computer using - "int 16h" code and wait for any key press.
14. the program exits as any key is pressed.

Program -

name "Colors"

org 100h

mov ax, 3

int 10h

mov ax, 1003h

mov bx, 0

int 10h

mov dl, 0

mov dh, 0

mov bl, 0

jmp next-char

next-row:

inc dh

cmp dh, 16

je stop-print

mov dl, 0

next-char:

mov ah, 02h

int 10h

mov al, 'a'

mov bh, 0

mov cx, 1

mov ah, 09h

int 10h

.

inc bh

inc dl

cmp dl, 16

je nextrow

jmp next-char

stop print:

mov al, 10

mov dh, 5
mov ah, 0ch
int 10h
mov al, 'x'
mov ah, 0ch
int 10h
mov ah, 0
int 16h
ret.

Result - Hence I have prepared a mini-project
successfully

The screenshot shows a debugger environment with the following components:

- Left Panel (Assembly View):** Displays the assembly code being debugged. The code includes instructions like `jmp next_char`, `inc dh`, `cnp dh, 16`, `je stop_print`, etc.
- Middle Panel (Registers View):** Shows the CPU register values at address F400:01C8. Registers include AX, BX, CX, DX, SI, IP, SS, SP, and BP. Most registers contain values like 00, 7B, or NULL.
- Bottom Panel (Emulator Screen):** An 80x25 character terminal window showing a grid of colored text (mostly green and yellow) on a black background.
- Right Panel (Original Source Code):** A window displaying the original C source code corresponding to the assembly code.