Page No.

Date

Practical. NO. 1.

Aim - Program Develop and execute ALP to 16 bit number

a) Program of addition of 16 bit number, result is 16-bit

b) Program of addition of 16 bit 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

Teacher's Signature

Page No.

Date

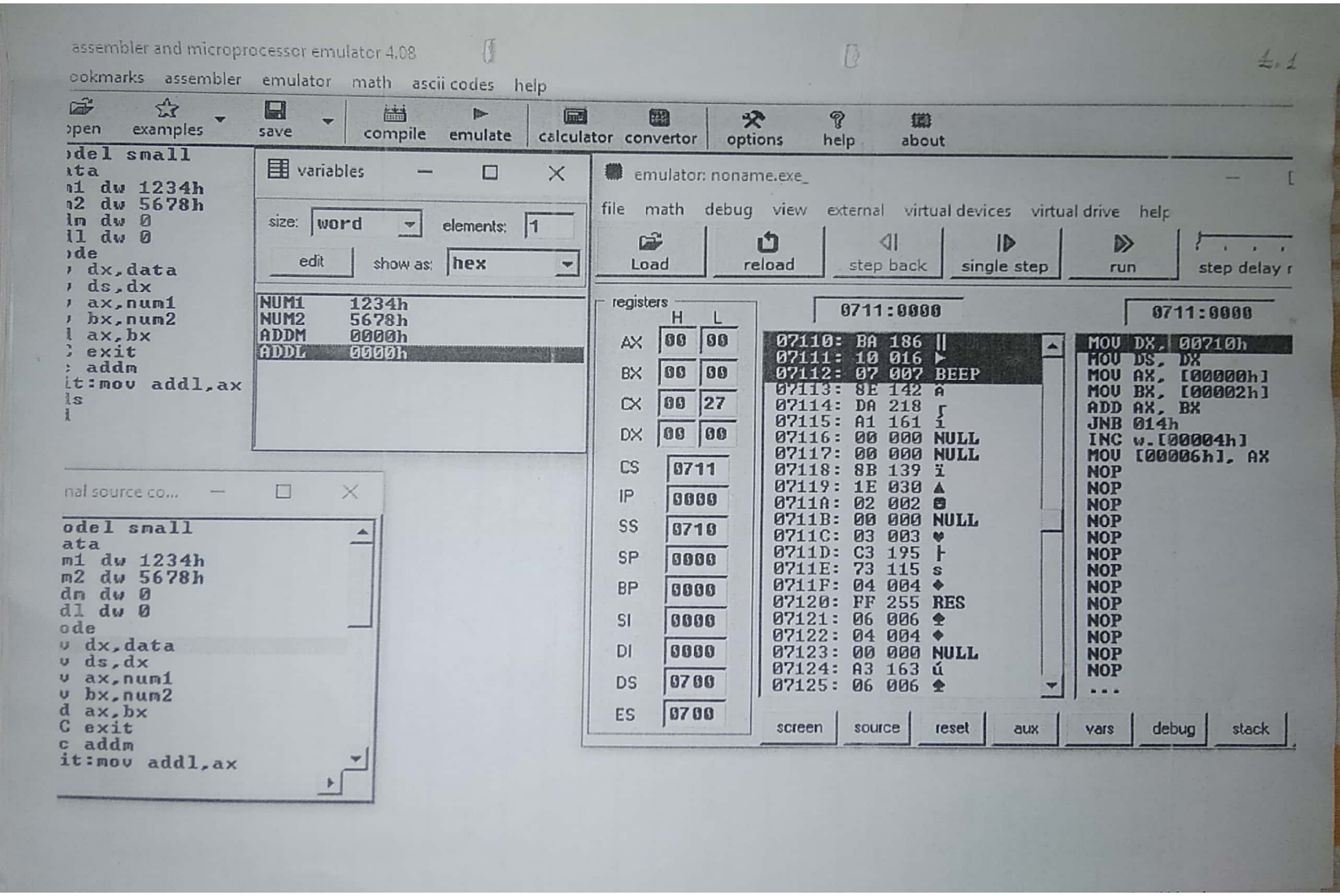
v ENDS - The code segment ends

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 , data
mov   ds , dx
mov   ax , num1
mov   bx , num2
add   ax , bx
JNC   exit
INC   addm
exit : mov addl , ax
ends
end
```



Page No.

Date

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

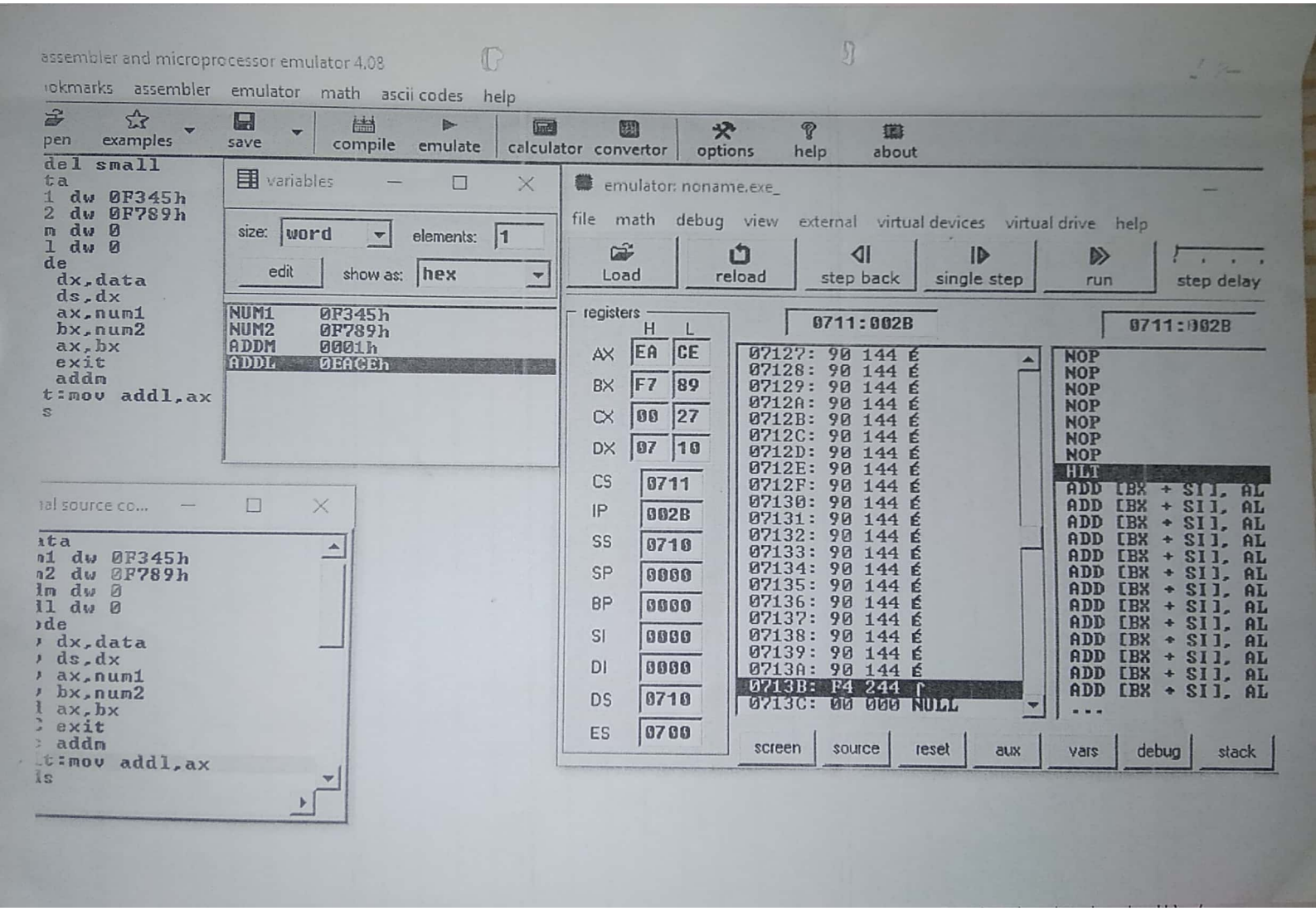
v) Increment MSB counter

vi) Store result

vii) Stop.

Program -

```
.model small
.data
num1 dw 0FFFFh
num2 dw 0FFFFh
addm dw 0
add1 dw 0
.code
mov dx, data
mov ds, dx
mov ax, num1
mov bx, num2
add ax, bx
JNC exit
inc addm
exit: mov add1, ax
ends
end.
```



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

practical. NO.2

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

Practical. No. 2

~~Program~~ 1

Page No.

Date

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

Theory -

Instructions -

i sub - It is used in subsraction, it is used to subsract the operands

Ex. SUB ax to Bx

Algorithm for subsraction -

i) Initialise data segment
ii) load first number from memory
iii) subsract second no. from first
iv) Store the result
v) Stop.

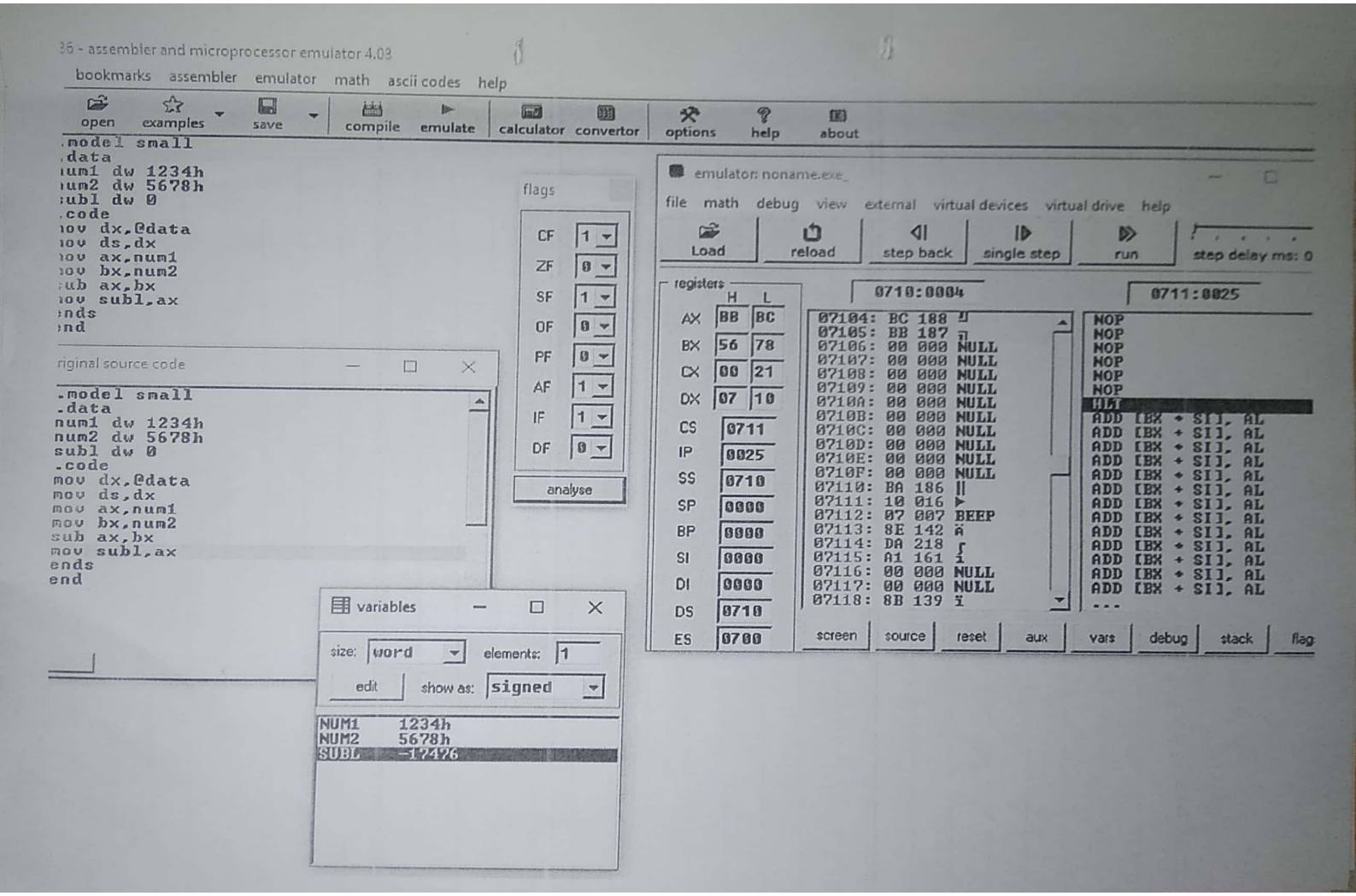
program -

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

(Students Co-op Stores G.P.N.)

Teacher's Signature

```
mov dx, @data
mov ds, dx
mov ax, num1
mov bx, num2
sub ax, bx
mov sub1, ax
ends
end
```



practical. NO. 3

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

Page No.

Date

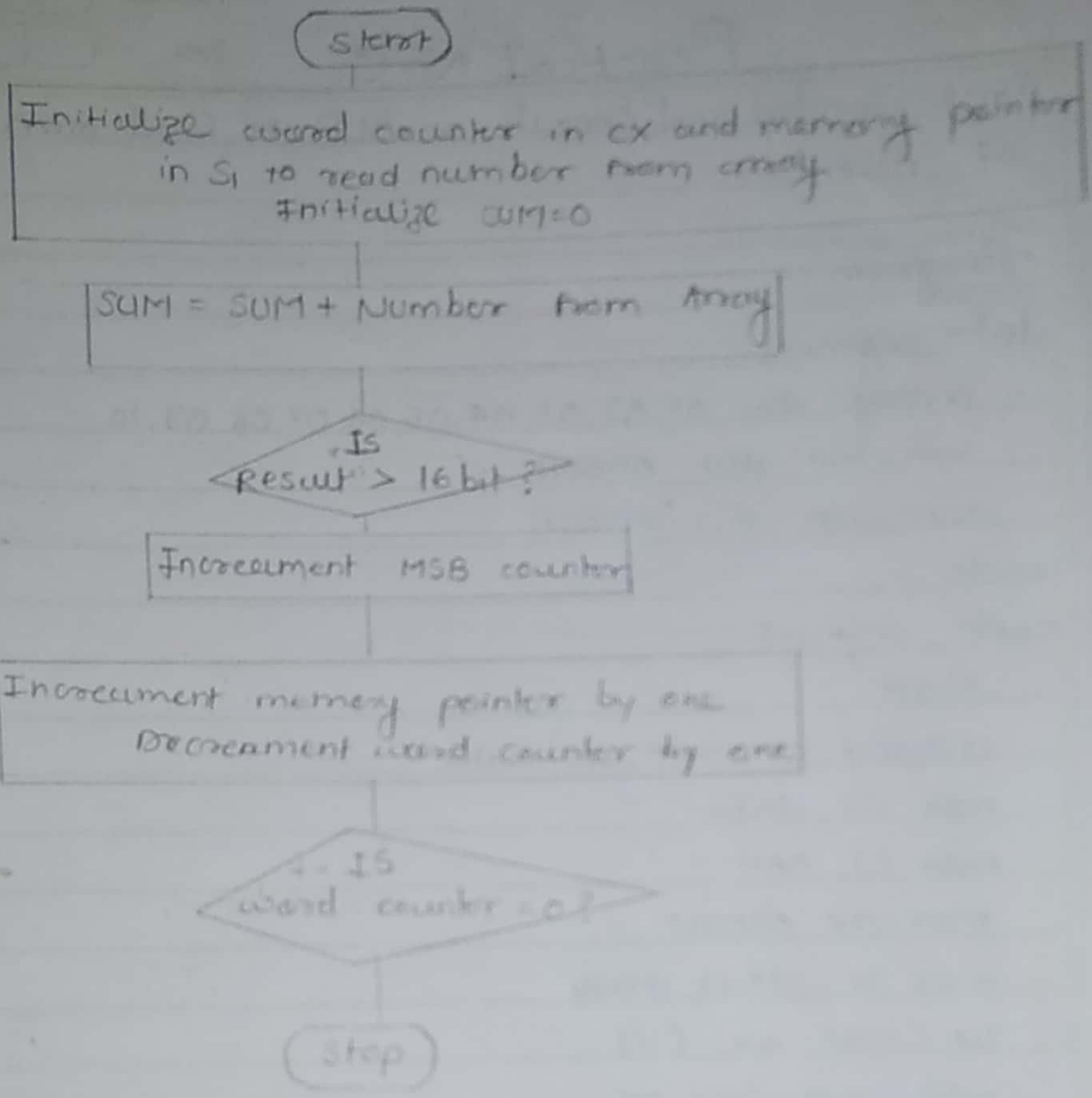
Practical No. 5

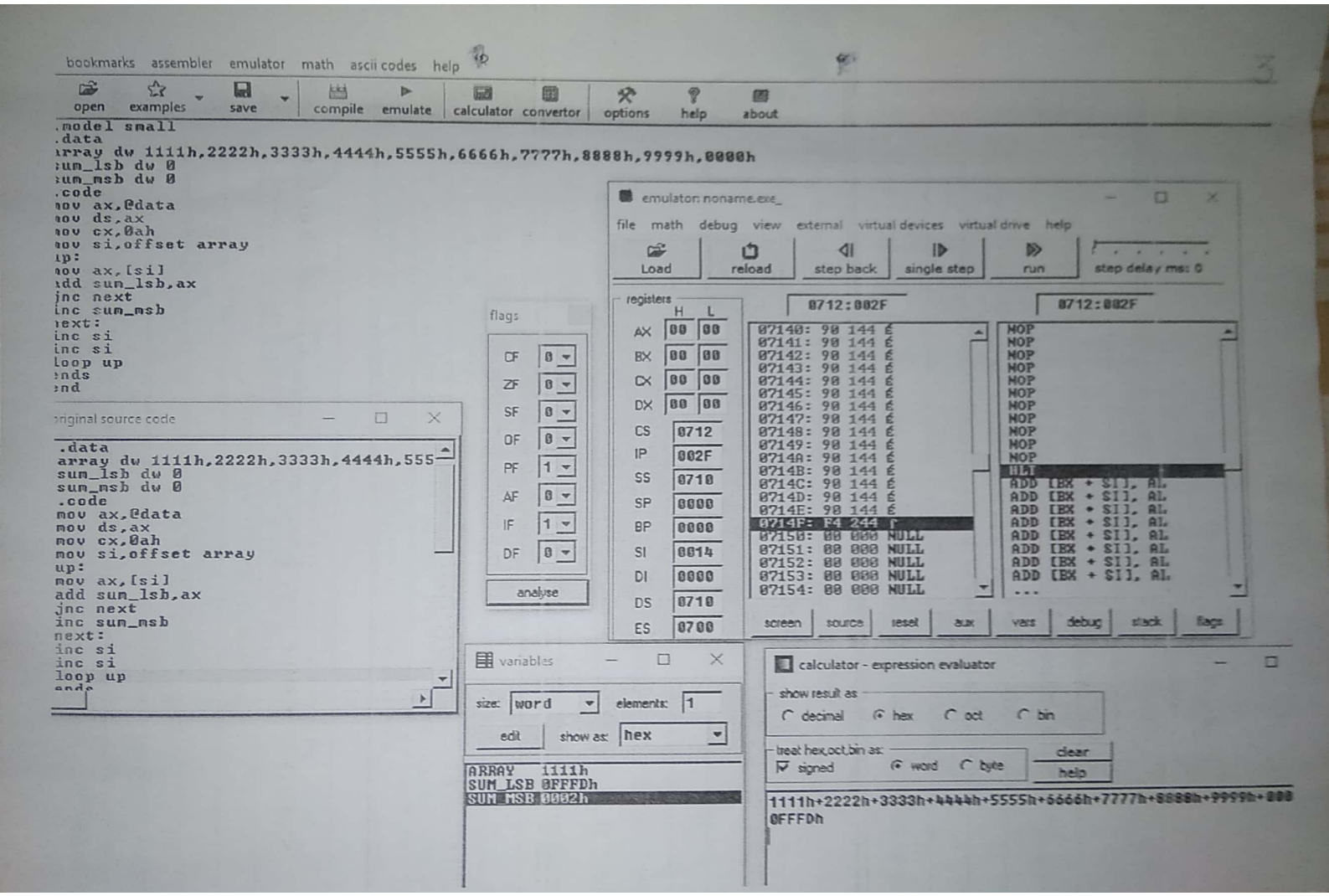
Aim:- Develop and Execute an ALP to find sum of series of number.

- 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
    jnc next
    next inc sum_up
    next : inc si
    inc si
    loop up
    mov ax, 4C00H
    ends
    end start
```

- flowchart of sum of series





Page No.
Date

•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 Increament MSB result counter
vii Increament memory pointer
viii Decreament byte counter
ix If byte counter = 0, then goto step 10, else 4
x Stop

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

•Viva Questions -
1. what is MOV?
→ It is a instruction used to copies the data item reffered 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.

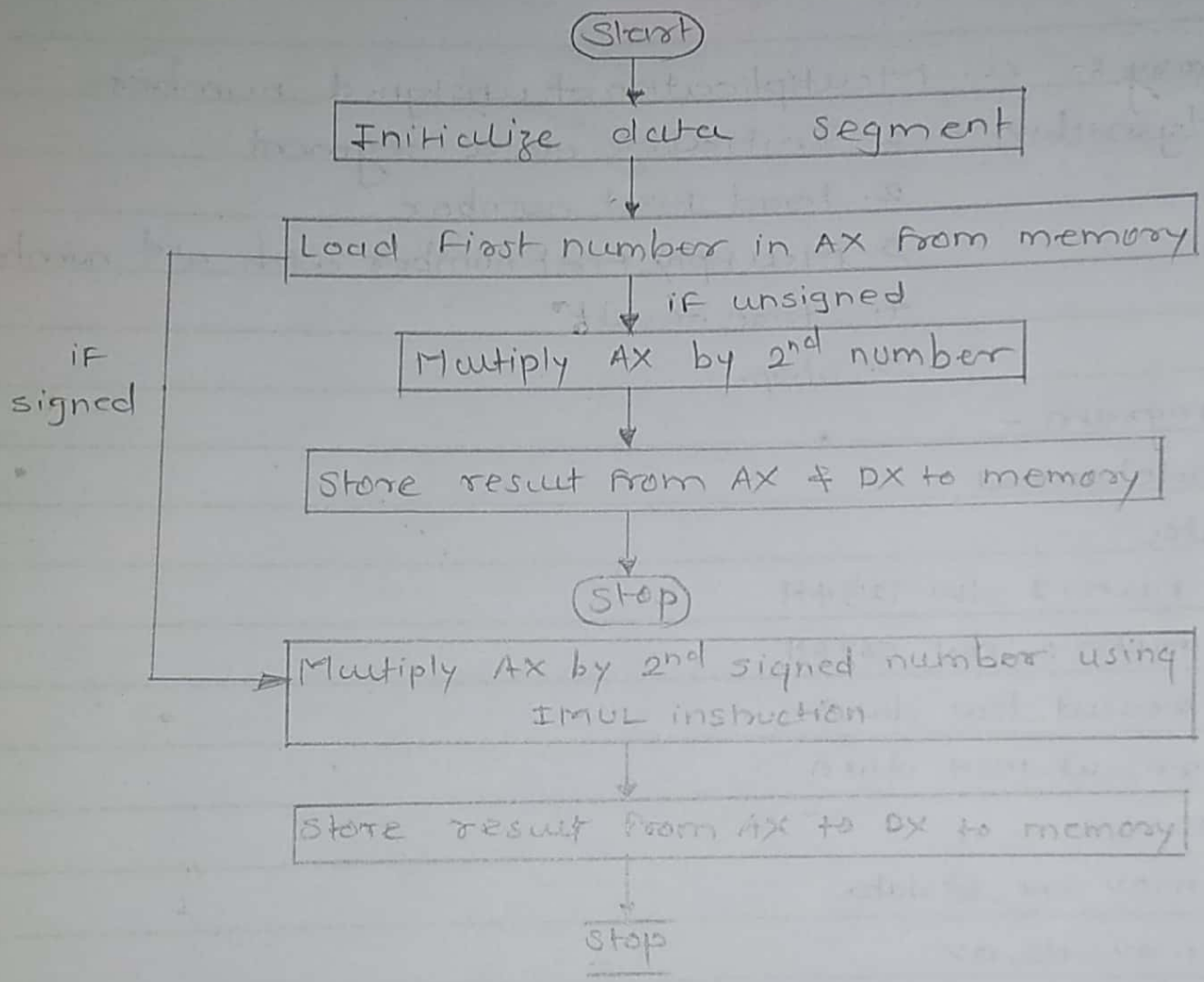
Theory:- a. Multiplication of unsigned numbers

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

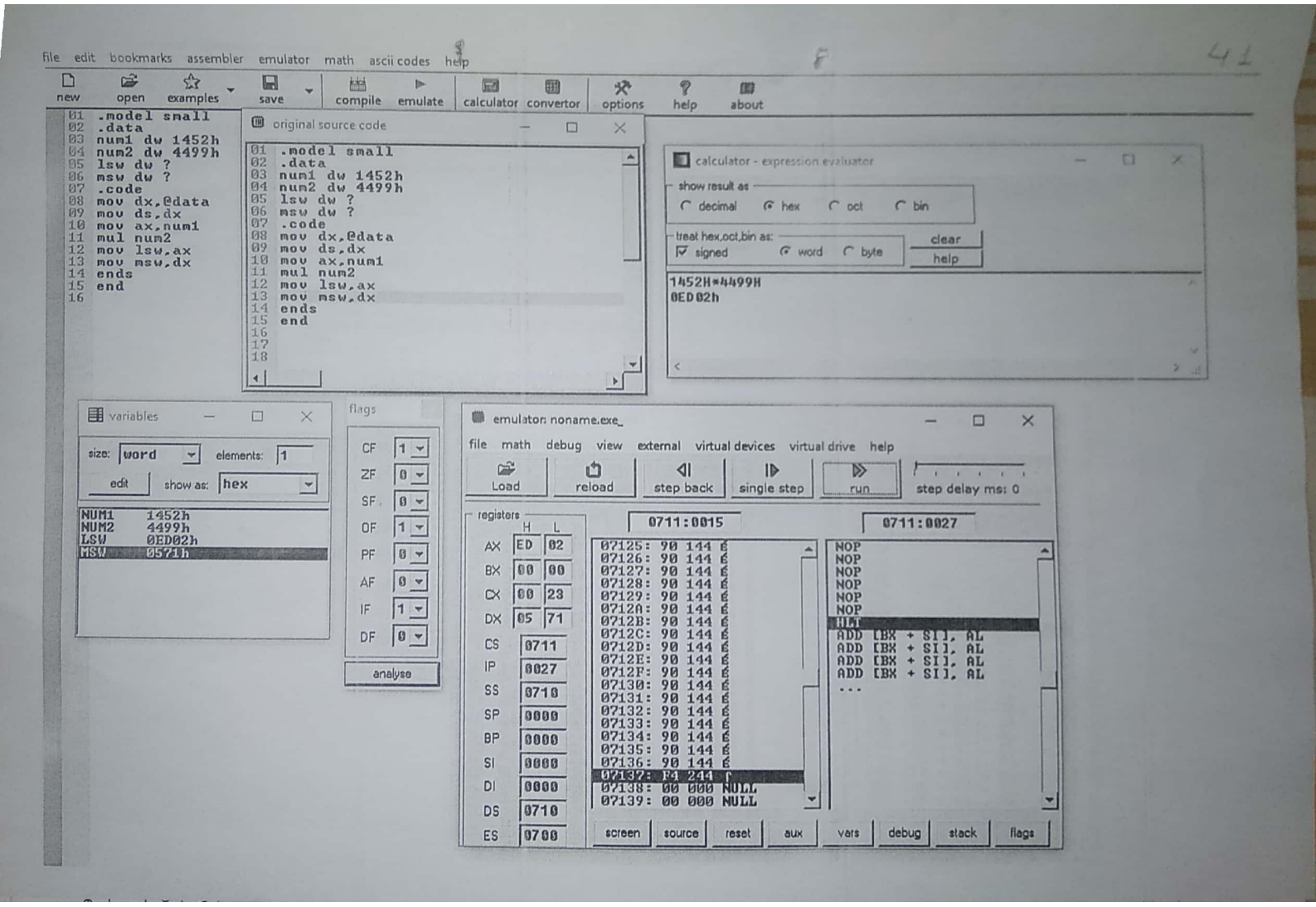
```
.model small
.data
    num1 dw 1234H
    num2 dw 5678H
    result_lsw dw 0
    result_msw dw 0
.code
    mov ax, @data
    mov ds, ax
    mov ax, num1
    mul num2
    mov result_lsw, ax
    mov result_msw, dx
ends
end
```

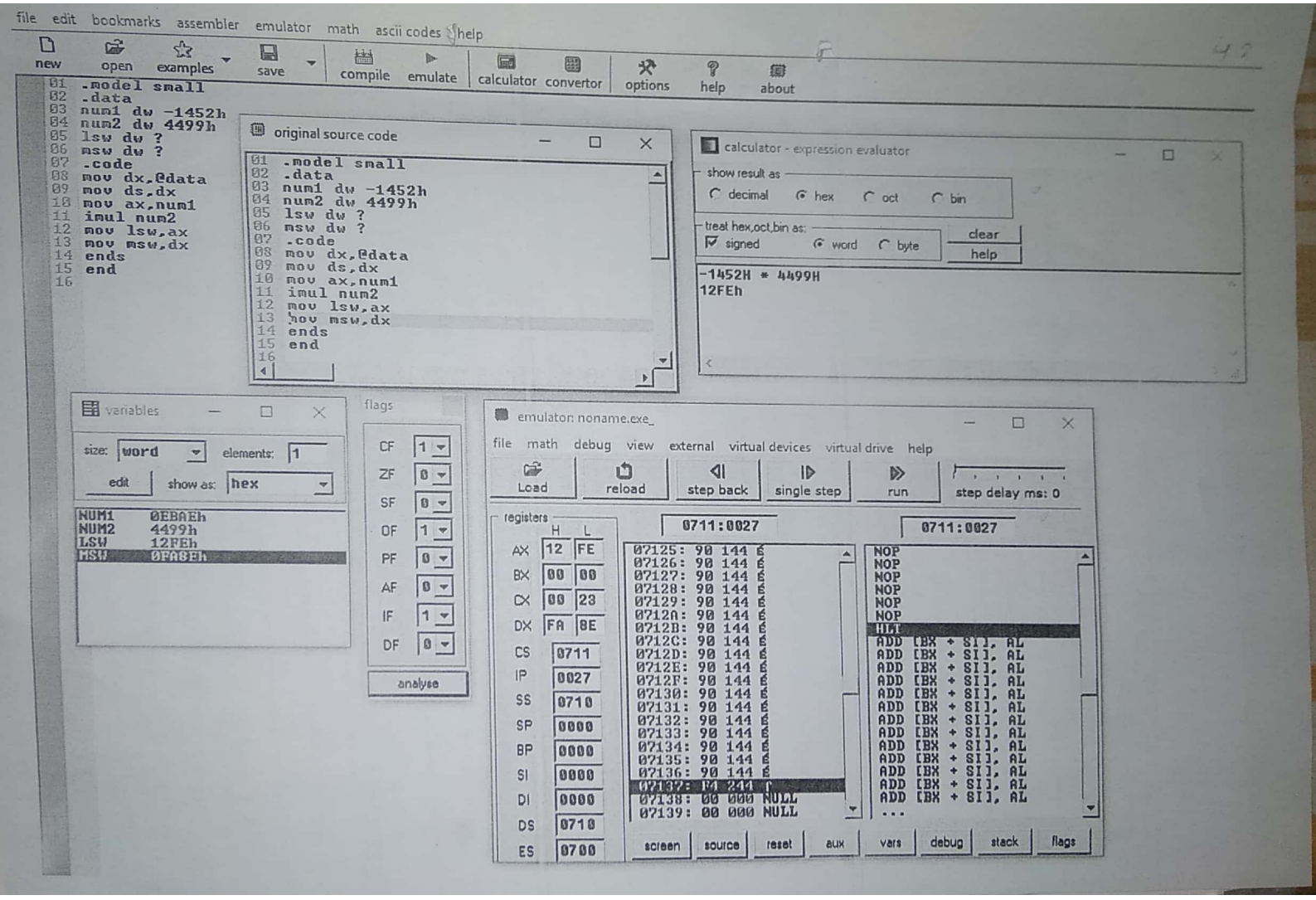
- Instructions - MUL:- The MUL multiplies first unsigned number with the second one Ex:- mul num2.

• flowchart to multiply two unsigned numbers



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





Page No.

Date

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, ax
    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 microprocessor?

→ 5MHz

Result :- 1 Input → num1 : 1234h
num2 : 5678h
output → LSW : 0060h
MSW : 0626h

2 Input → num1 : -1234h
num2 : +5678h
Output → LSW : 0FFA0h
MSW : 0F99h

Result: Hence, we have successfully developed a
executed ALU for multiplication of
16 bit number

Page No.

Date

Practical No. 5

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

Theory :- A. Division of two unsigned numbers.

- Algorithm
1. Initialize data segment
2. Load first number
3. Divide first number by second number
4. Store quotient & remainder
5. Stop.

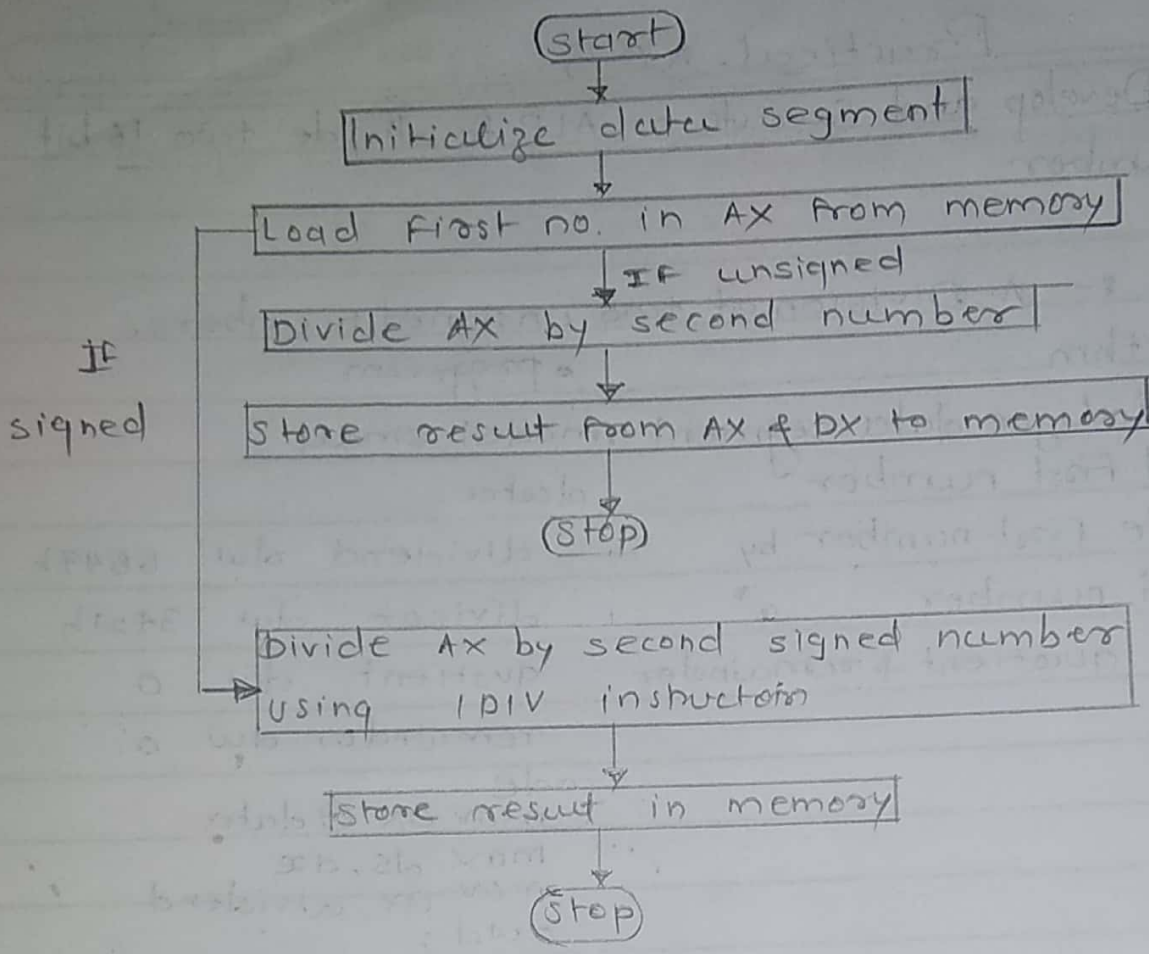
- program

```
.model small
.data
    dividend dw 6547h
    divisor dw 3421h
    quotient dw 0
    remainder dw 0
.code
    mov ax @data
    mov ds, dx
    mov ax, dividend
    cwd;
    div divisor
    mov quotient, ax
    mov remainder, dx
    ends
    end
```

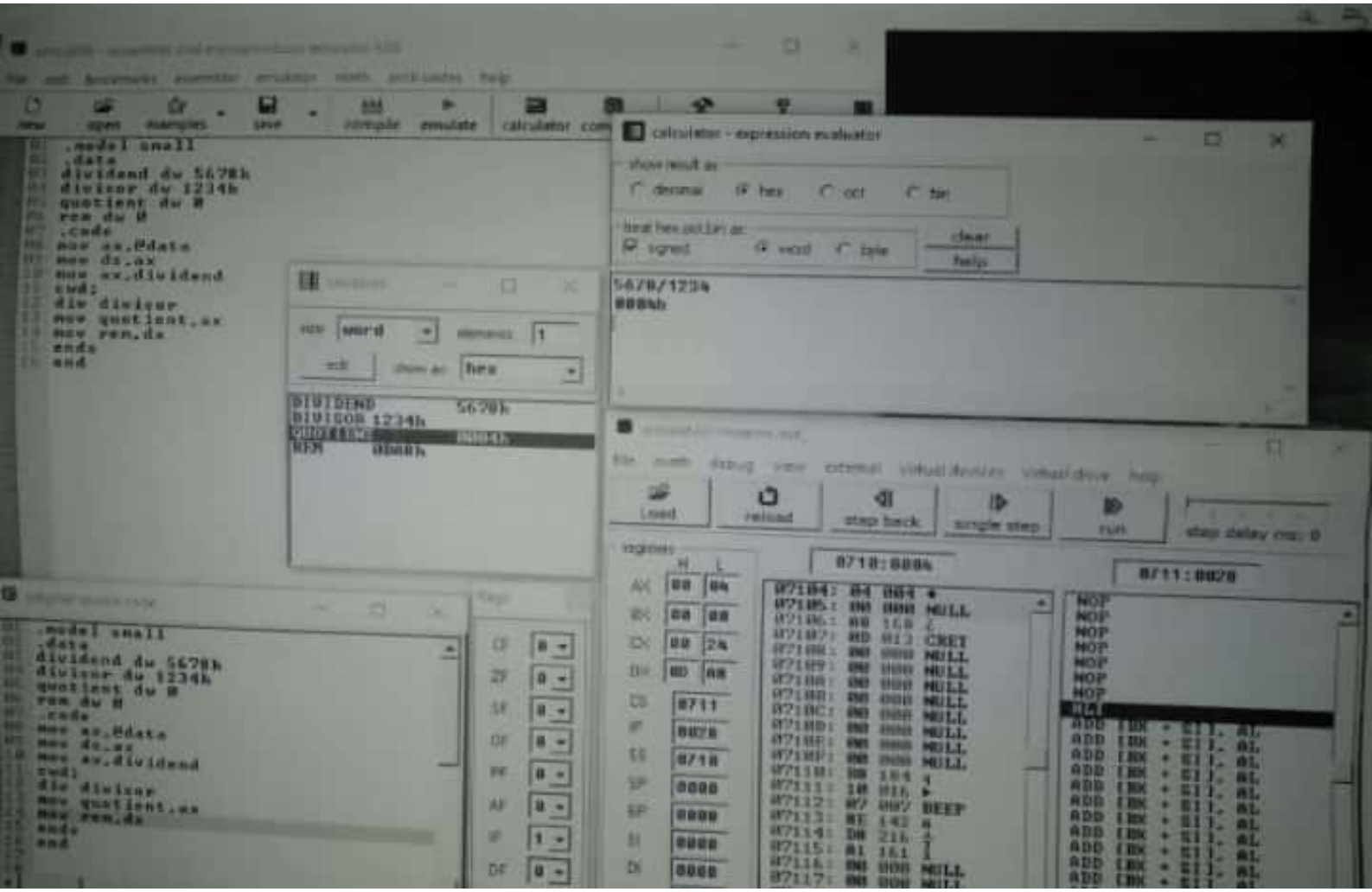
- Instructions - a. DIV :- it is used to divide a number with a number next to it. Ex. div divisor

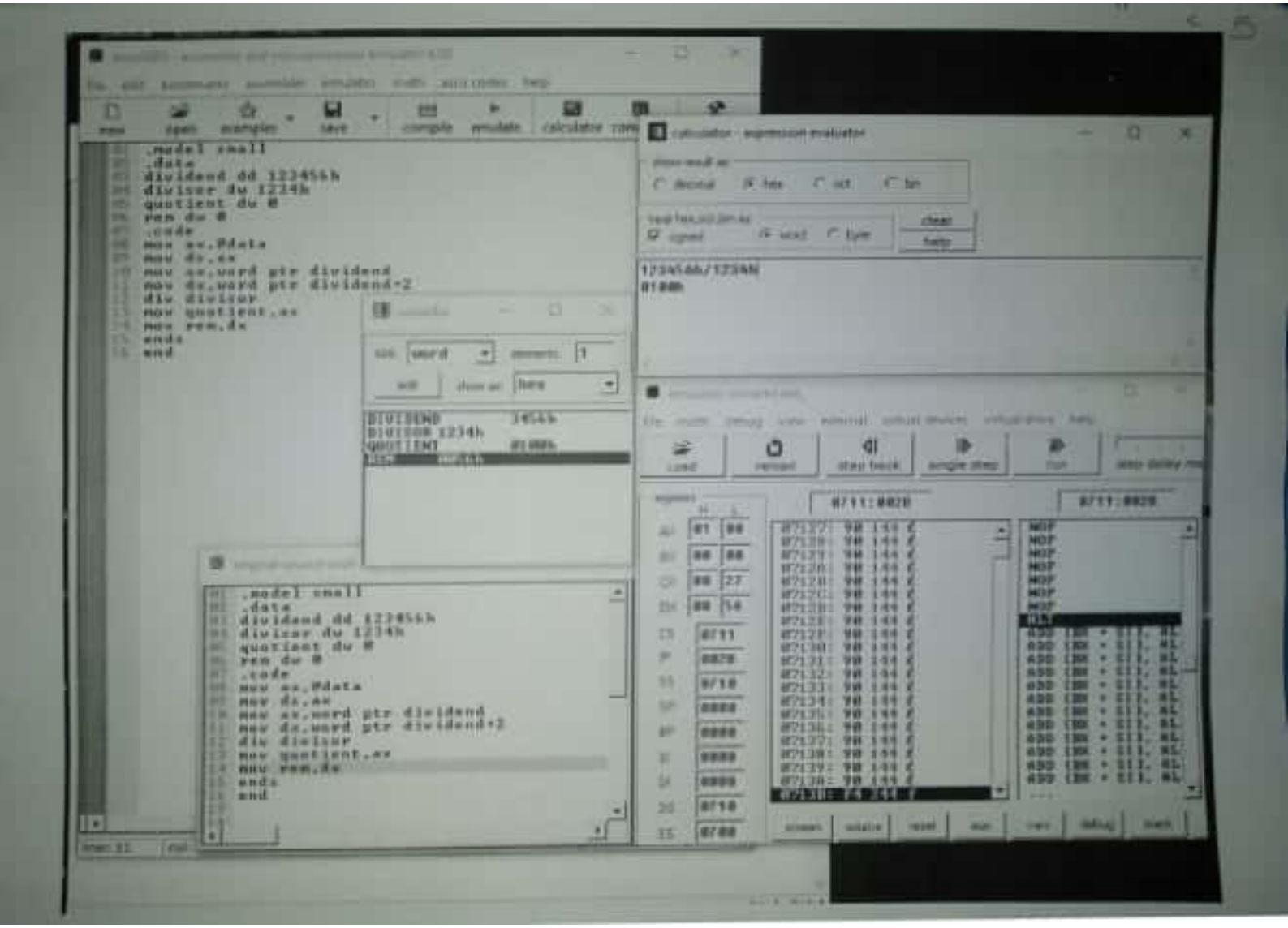
B. CWD :- it is used to convert singed byte to double words

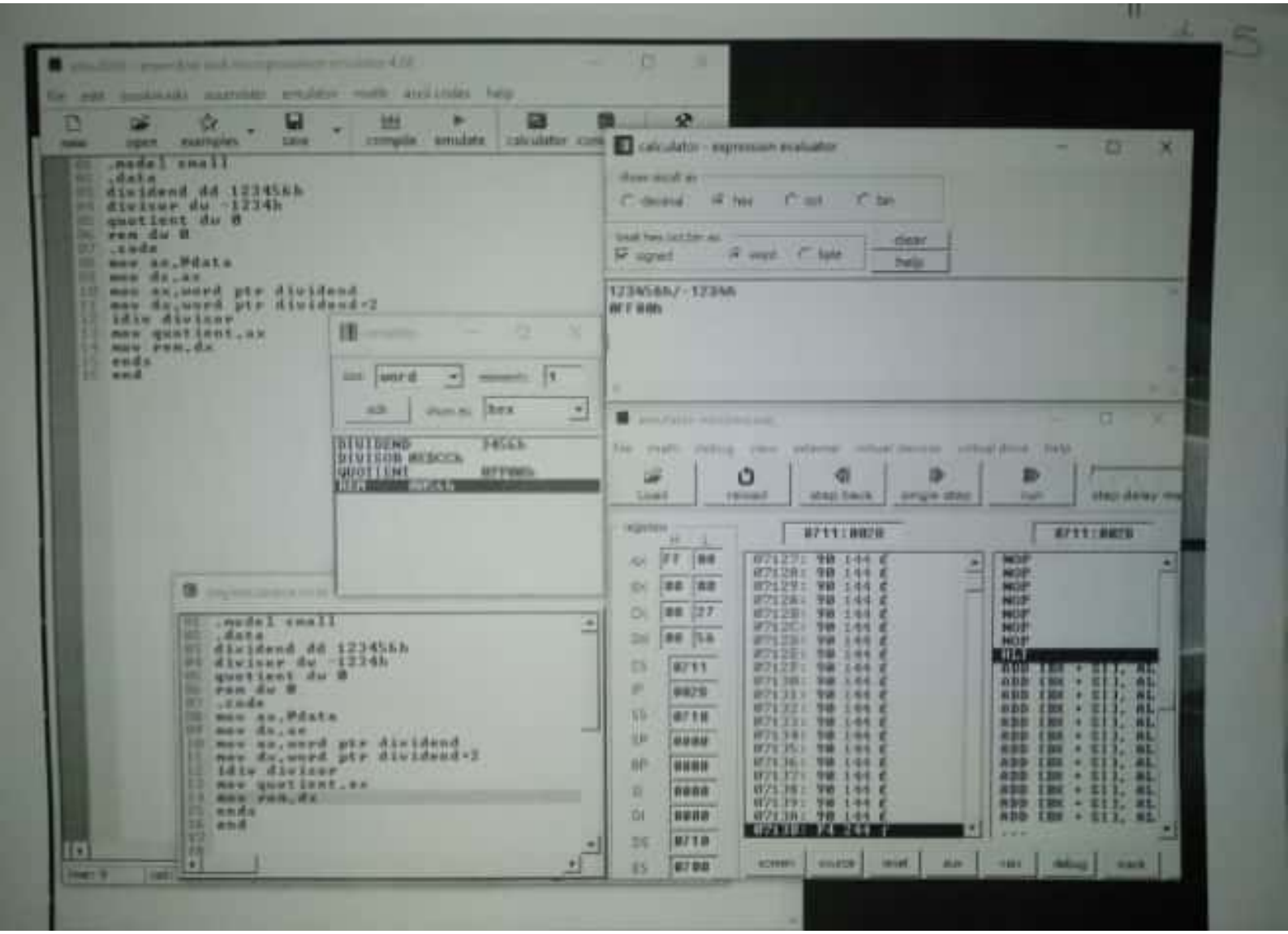
- Flowchart for unsigned & signed devision



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







Page No.

Date

B. Division of signed number with unsigned number.

• Algorithm

1. Initialize data segment
2. Load 16 bit dividend
3. Divide 1st no. with 2nd no.
4. Store quotient & remainder
5. Stop.

• program

```
.model small
.data
dividend dw -6547h
divisor dw +3421h
quotient dw 0
remainder dw 0
.code
mov dx @data
mov ds, ax
mov ax, dividend
cwd;
idiv divisor
mov quotient, ax
ends
end
```

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

• Viva Questions

1. What is difference between DIV and IDIV in 8086?

→ DIV : It only used in unsigned numbers

IDIV: It only used in signed numbers

2. 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
Output → Quotient : 0001h
Remainder : 3126h

2 Input → num1 : -6547h
num2 : +3421h
Output → Quotient : OFFFFh
Remainder : CCEDAh

Page No.

Date

Practical No. 6

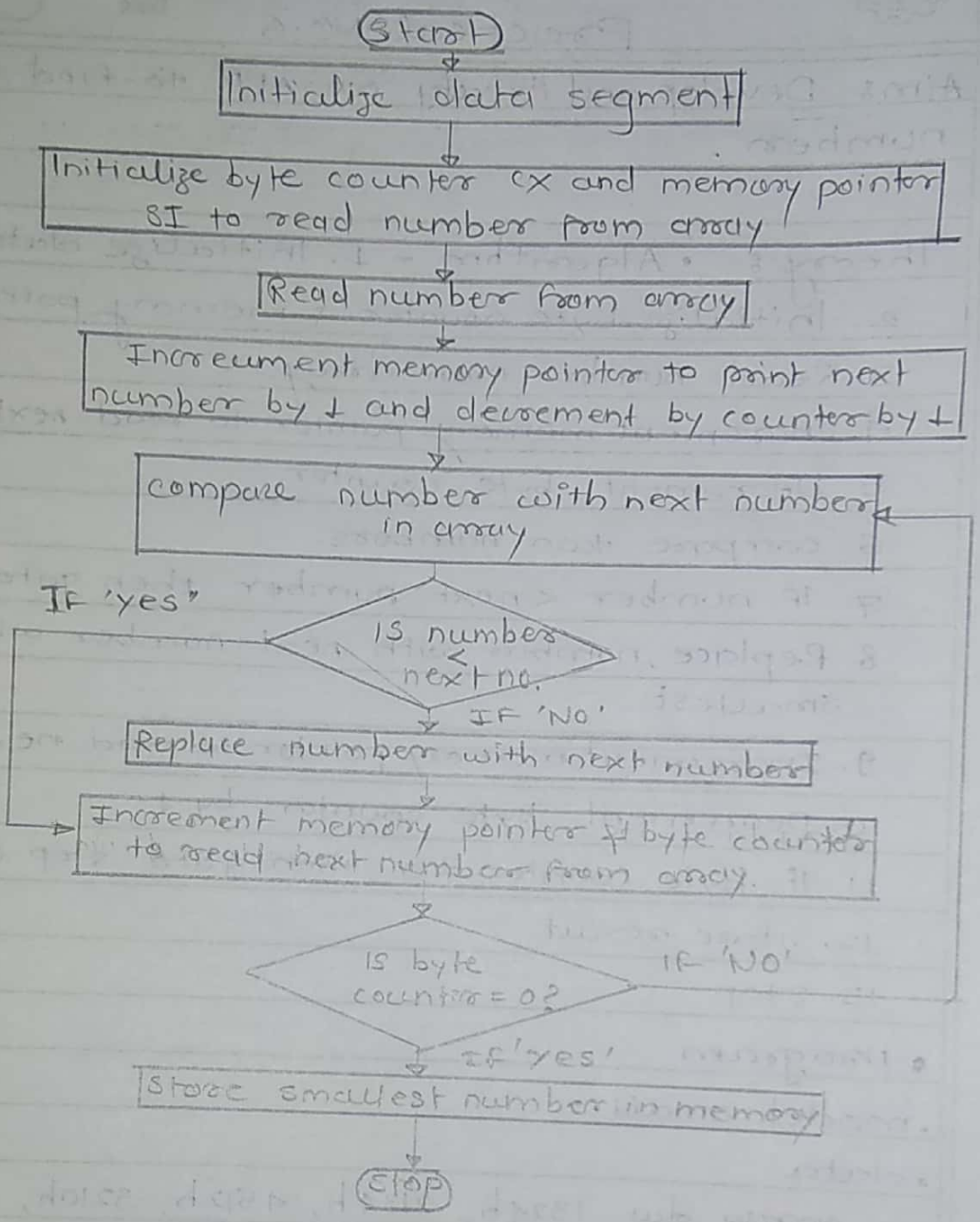
Aim:- Develop and Execute an ALP to find smallest number.

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 1
11. If byte counter ≠ 0, then goto step 6
12. store result.
13. Stop.

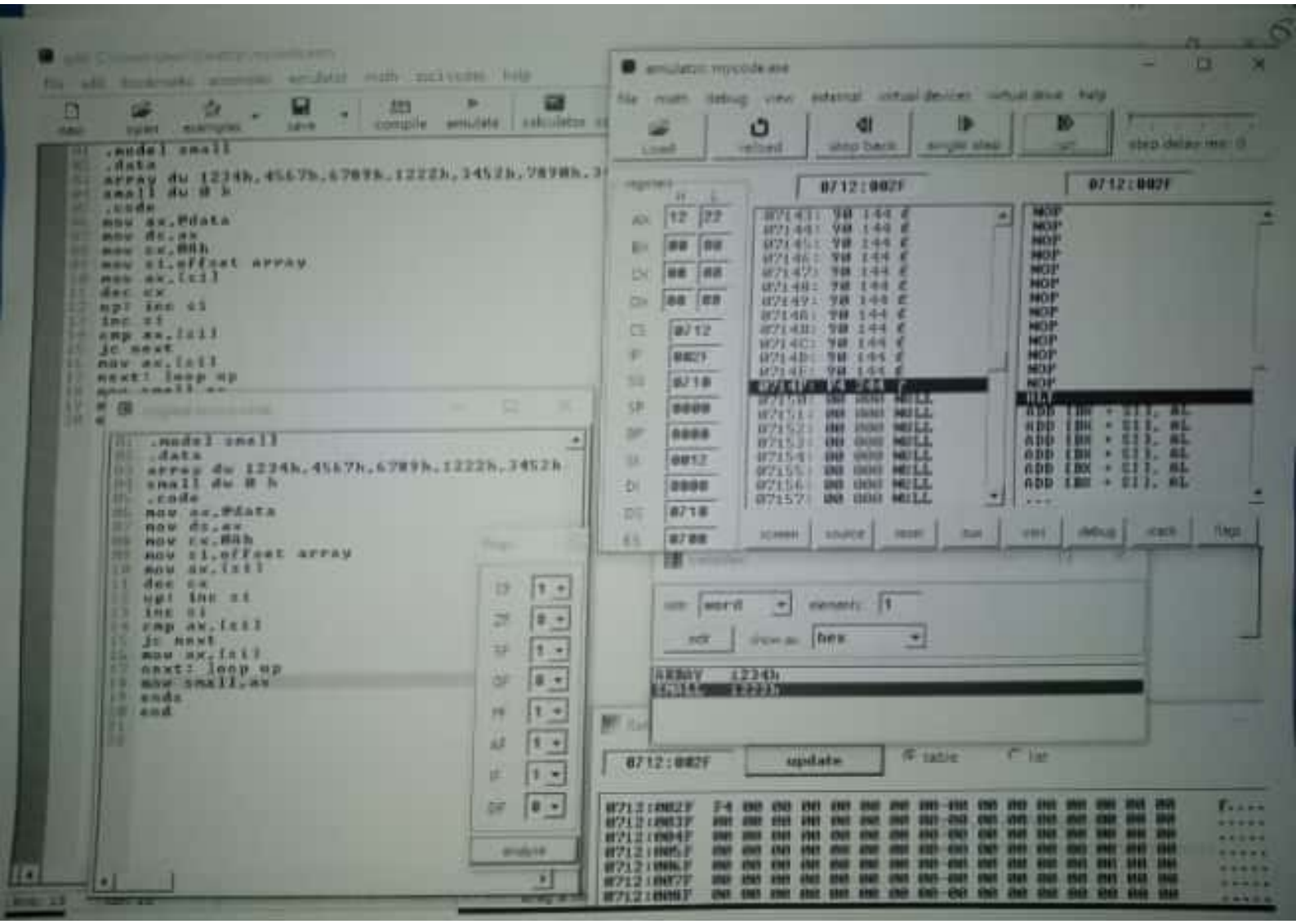
• program

```
.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.



Page No.

Date

```
up:
    inc si
    inc si
    cmp ax, [si]
    ja next
    mov ax, [si]
next:
    loop up
    mov small, dx
    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

SMALL 1112h

Page No.

Date

Practical. NO. 7

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

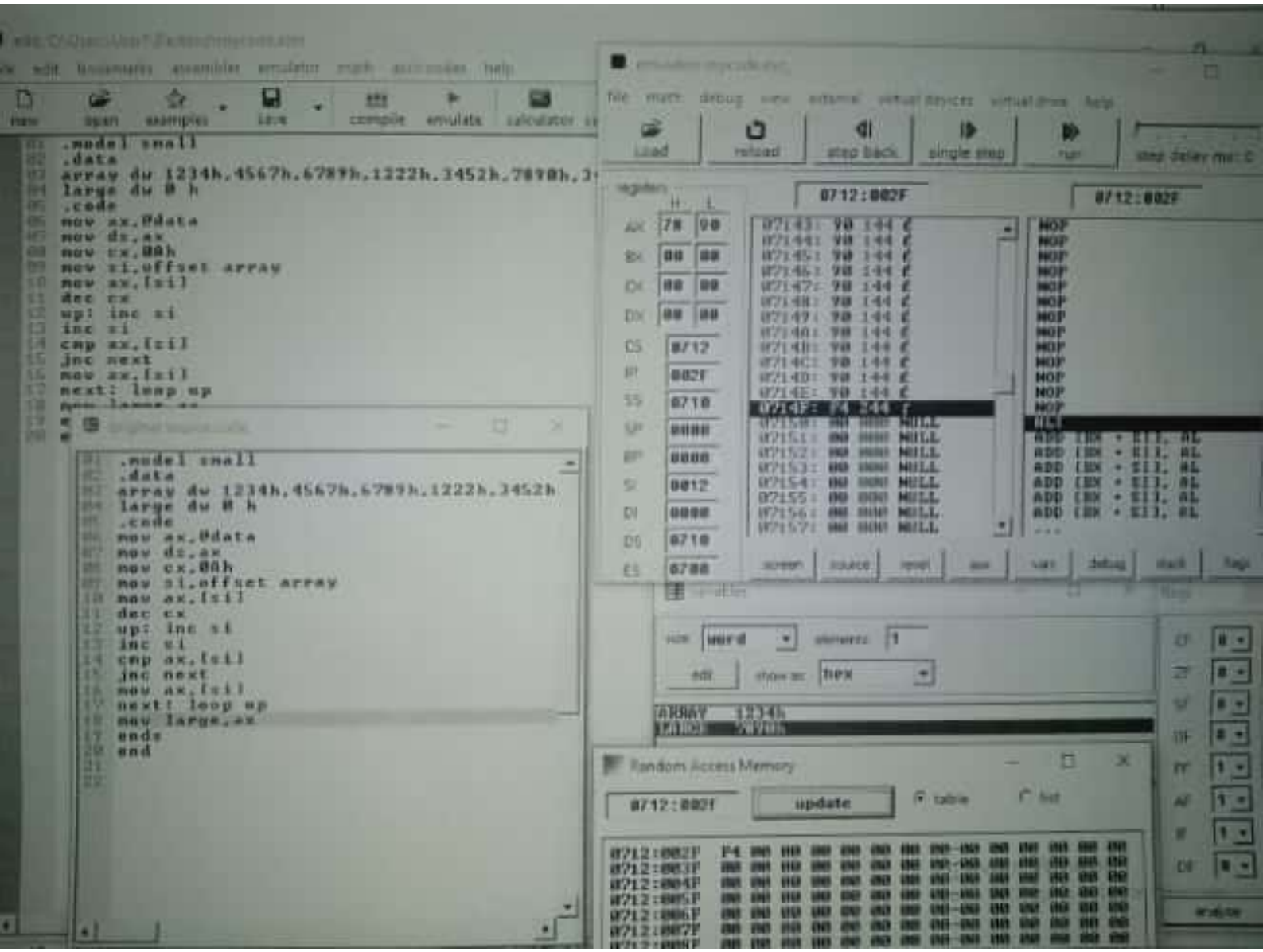
Theory :-

- Algorithm.

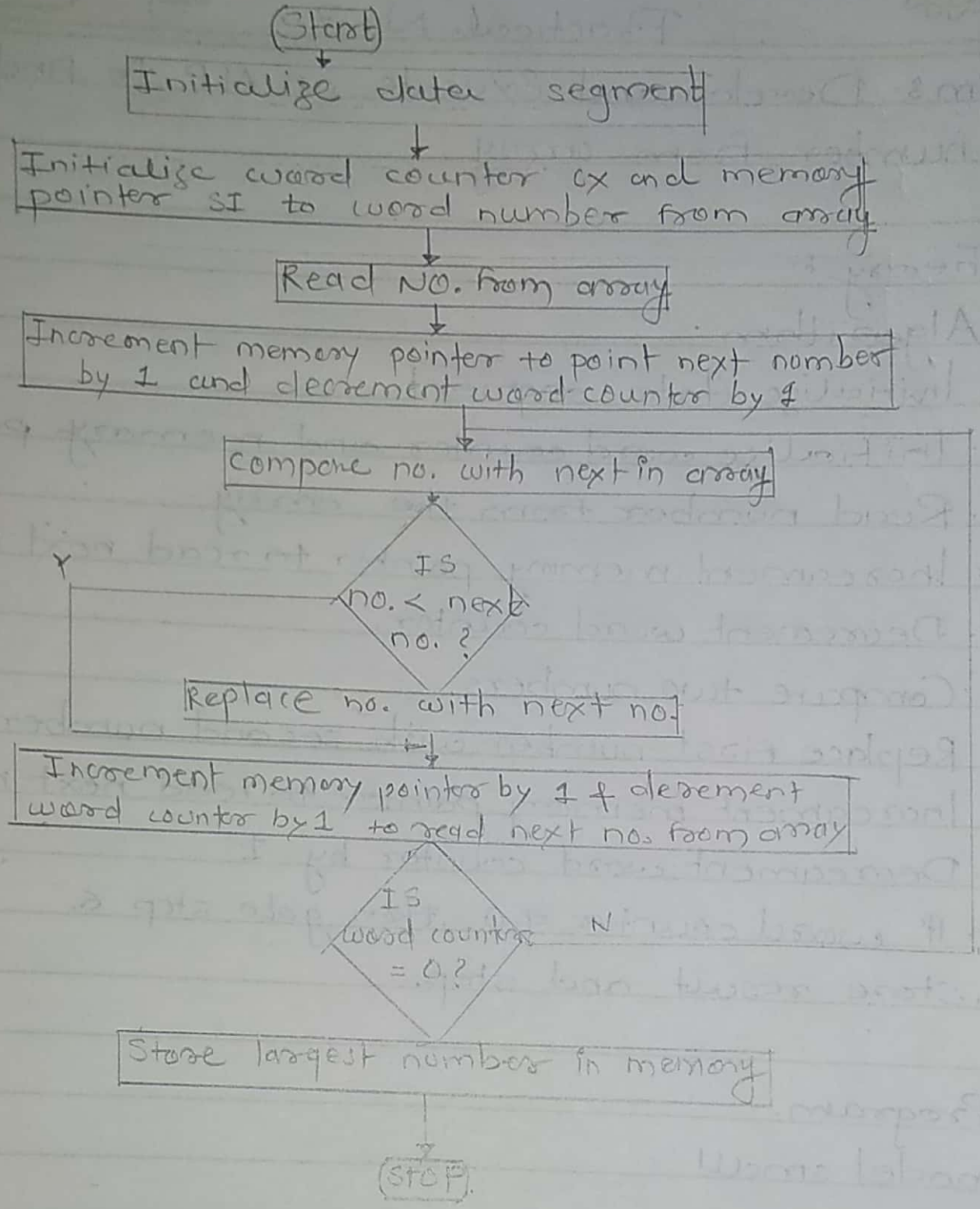
1. Initialize data segment.
2. Initialize word counter and memory pointer
3. Read number from the array
4. Increament memory pointer to read next number
5. Decrement word counter.
6. Compare two numbers
7. Replace first number with second number.
8. Increament memory pointer to read next number.
9. Decreament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

- Program.

```
.model small
.data
    array dw 1324h, 2134h, 4512h, 3210h, 1112h,
             4563h, 3122h, 9999h, 765th, 3216h
.code
    mov ax,@data
    mov ds , ax
    mov CX, 0Ah
    mov si, offset array
    mov ax,[si]
    dec cx
```



• Flowchart to find largest number from array.



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

Page No.

Date

```
up:
    inc si
    inc si
    cmp ax, [si]
    jnc next
    mov ax, [si]
nextidec cx
    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

i 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

Page No.

Date

Practical No. 8

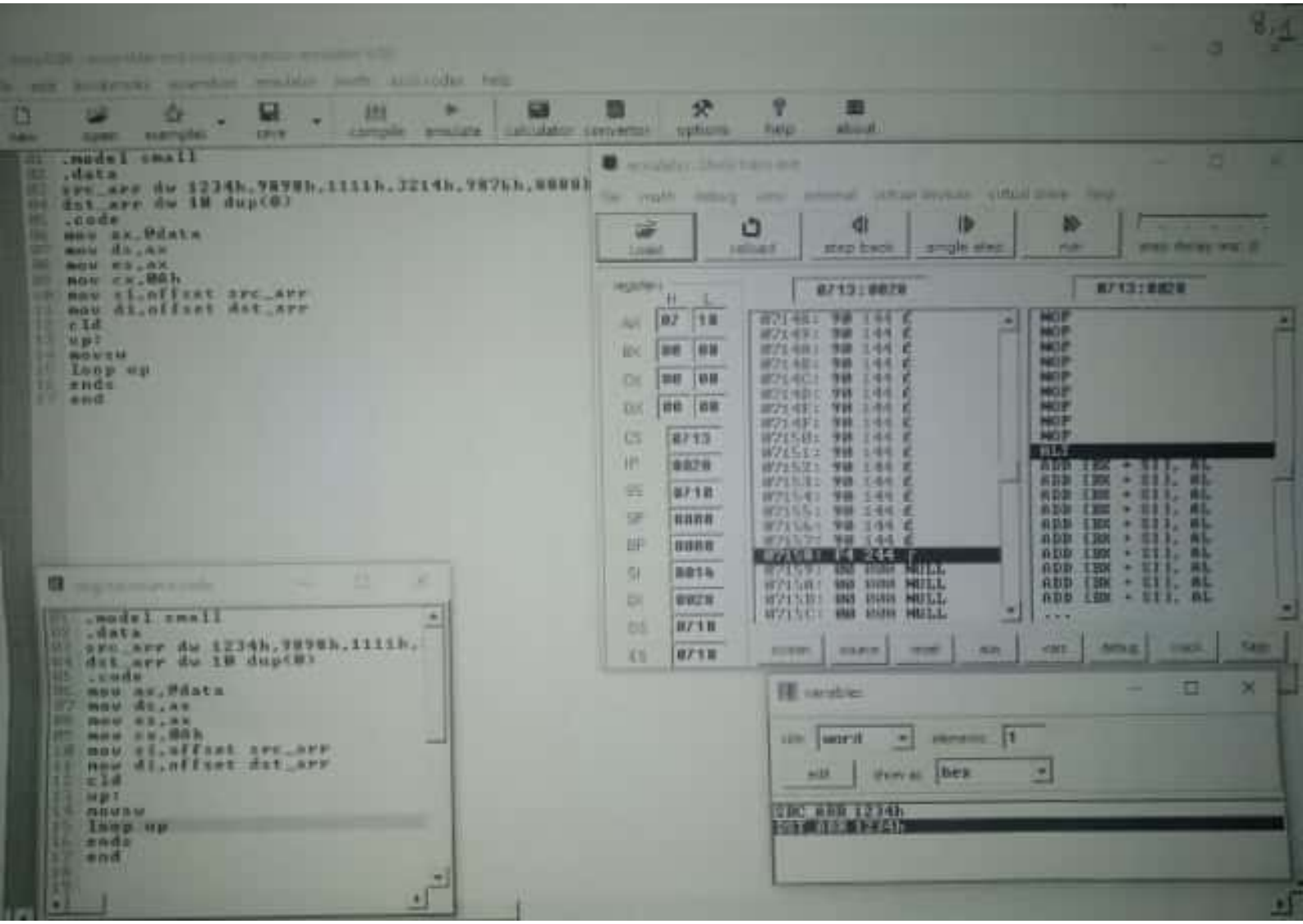
Aim :- Develop and Execute an ALP to perform block transfer data.

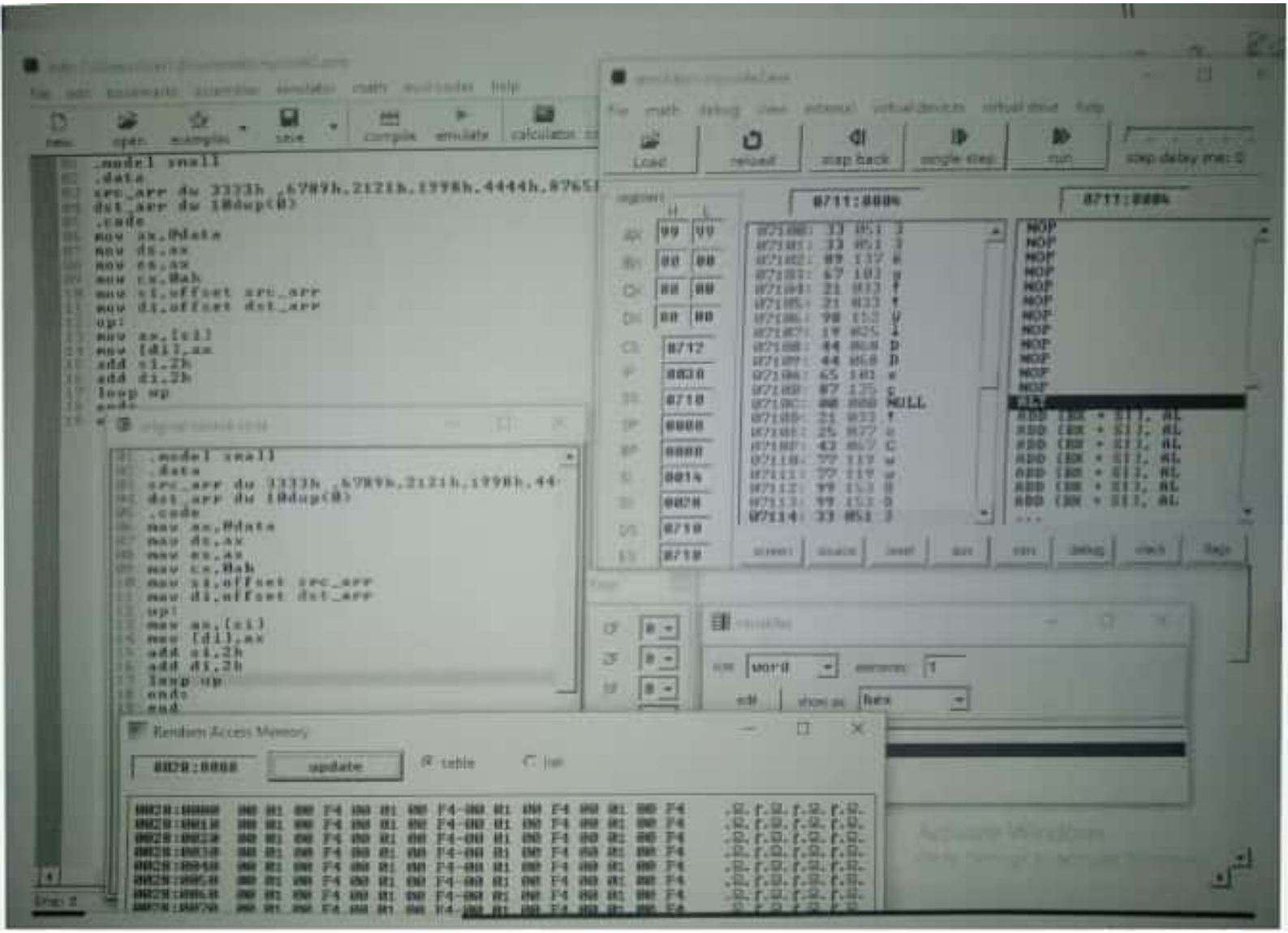
Theory :- • Algorithm :- 1. Initialize data and extra segmen

2. Initialize word counter.
3. Initialize memory pointer for source and destination array
4. Read number for source array
5. Copy it to destination array.
6. Increament memory pointer for source & destination array for next number
7. Decreament word counter by 1
8. If word counter ≠ 0, then goto step 4. else Stop.

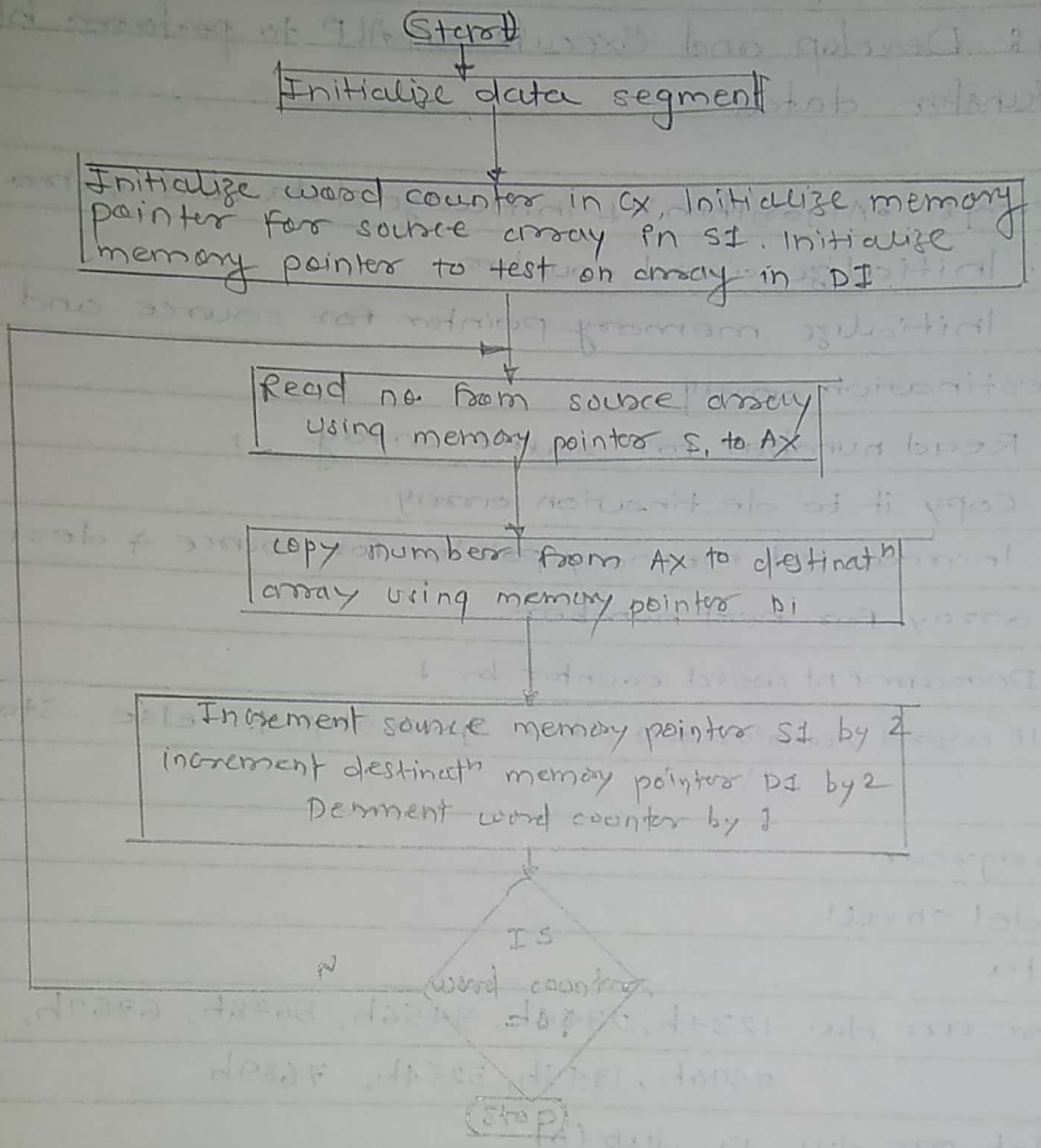
• Program

```
.model small
.data
   src_arr dw 1234h, 2345h, 3456h, 5678h, 6789h,
              7890h, 1345h, 3254h, 7689h
   dst_arr dw 10 dup (0)
.code
   mov ax, @data
   mov ds, ax
   mov cs, ax
   mov cx, 0ah
   mov si, offset src_arr
   mov di, offset dst_arr
   eid
```





• Flowchart



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

Page No.

Date

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 instruct 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

1. what is instruction set ?

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

Result :- (i) SRC_ARR 1234h

DST_ARR 1234h

Page No.

Date

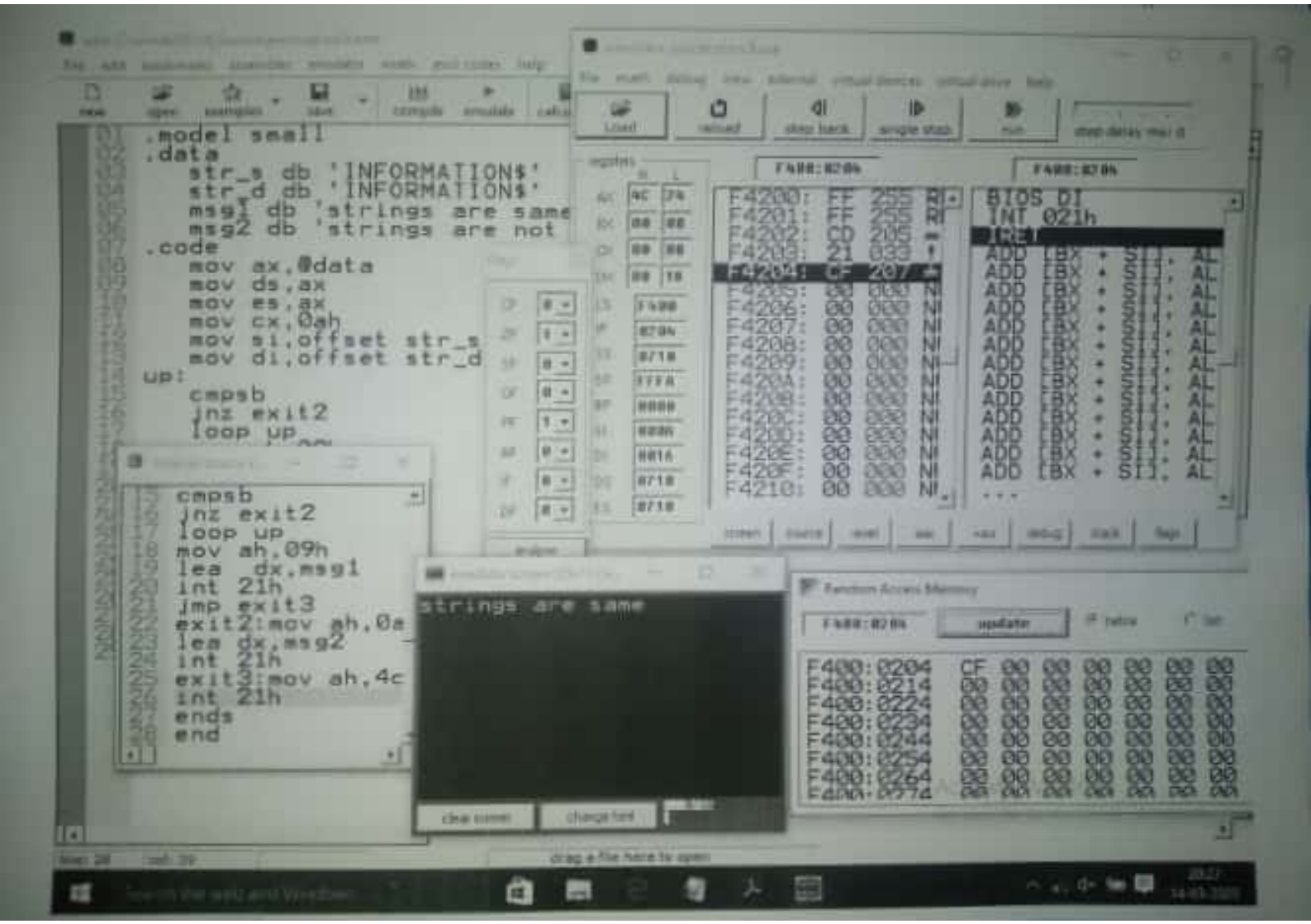
Practical. No. 9

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 character 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 'reset TheWalkingDead $'
   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
    jng exitz
    loop up
    mov ah, 09h
    lea dx, msg1
```



Date

Practical - 10

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

Theory -

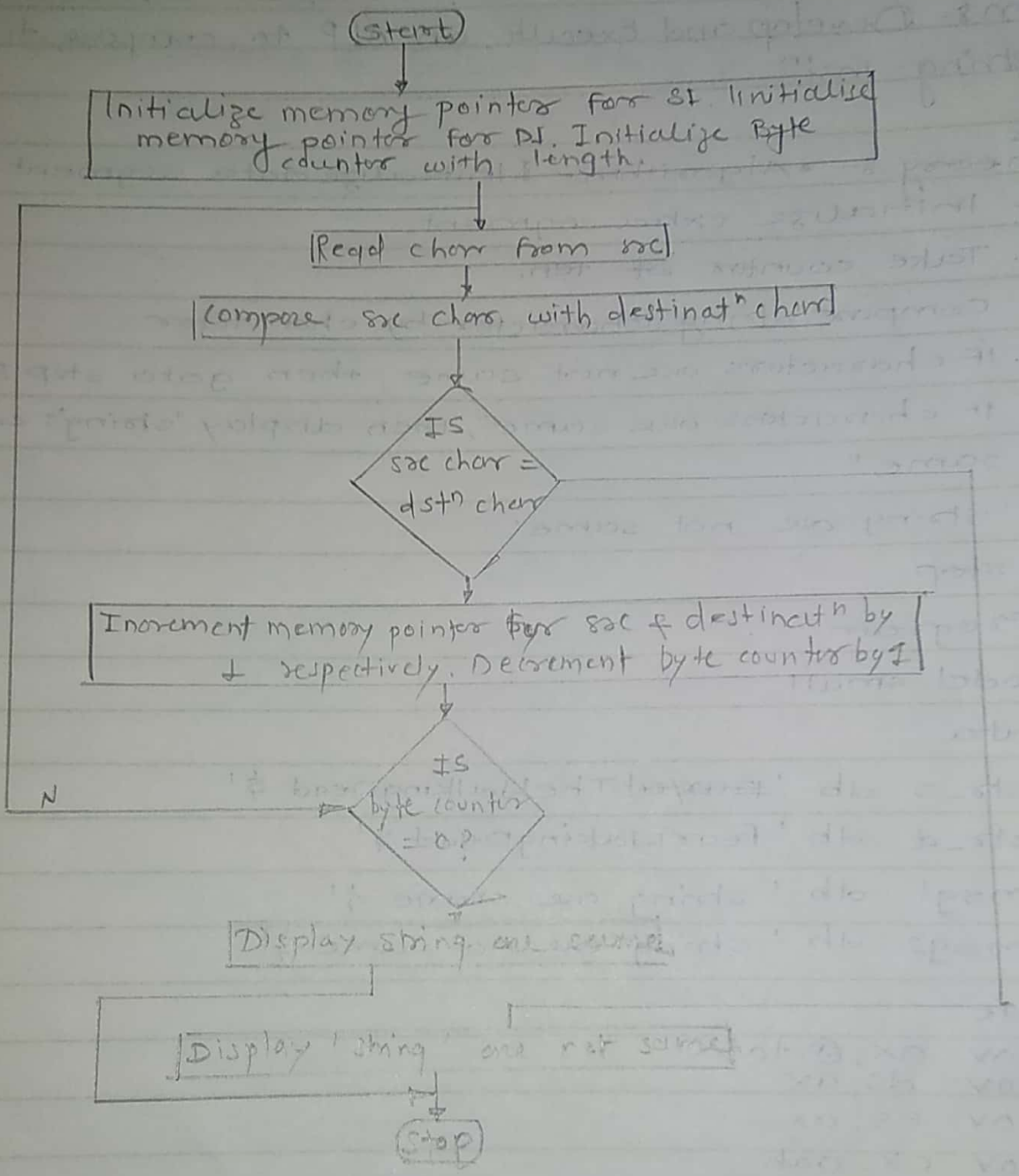
Algorithm

1. Initialize data segment.
2. Initialize memory pointer in SI and a length counter with a for source string
3. Read character 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_s db 'INFORMATION$'
```

•flowchart



Conclusion:- Hence, we successfully developed ALP to compare two string.

Page No.
Date

```
length db dup(0)
.code
    mov ax, @data
    mov ds, ax
    mov si, offset str1
next:
    mov al, [si]
    cmp al, '$'
    JE exit
    inc si
    inc length
    jmp next
exit:
    ends
    end
```

Instructions:

1. LEA

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

Page No.
Date

* Viva Questions

1- Which character defines ~~character~~ 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
exit2:
    mov ab, 09h
    lea dx, msg2
    int 21h
    exit3:
exit3:
    mov ab, 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 register.

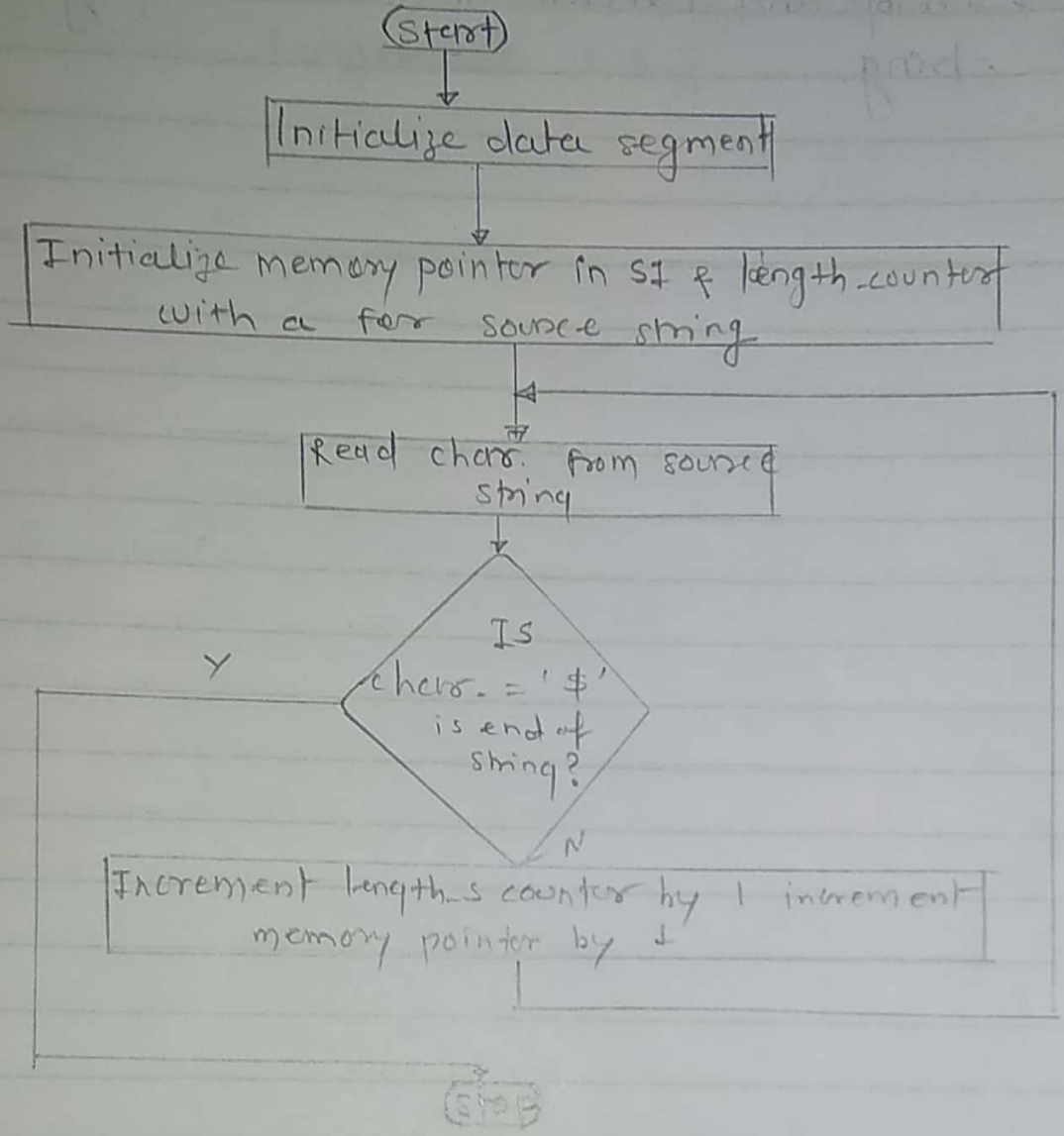
2. What is compare instructor?

→ This instruction is used to compare two string bytes

Result :-

Strings are not same.

• Flowchart



Page No.

Date

Practical No. 11

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

Apparatus:- 1. Microprocessor bit 8086
2. Power supply +5V dc, +12V dc
3. ADC Interface board - 1.

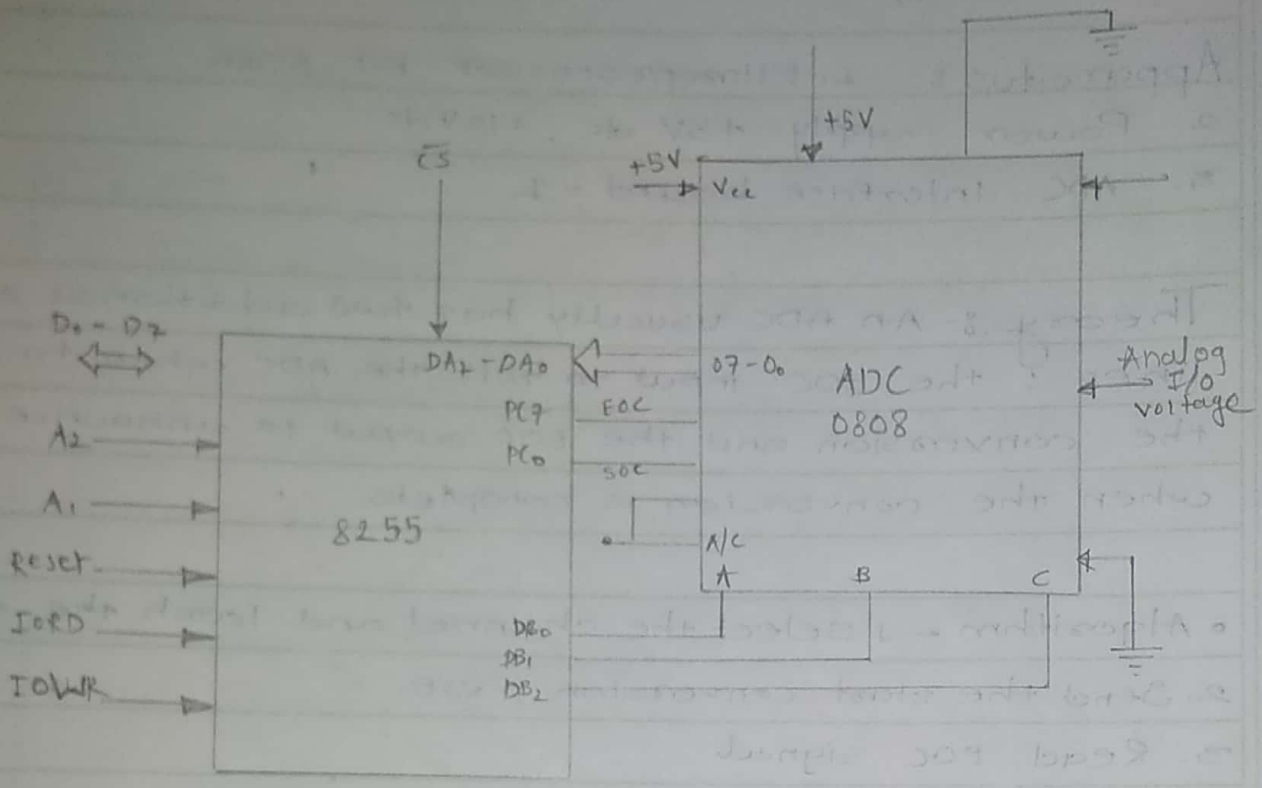
Theory:- An ADC usually has two additional control lines : the SOC input to tell the ADC when to start the conversion and the EOC output to announce when the conversion is complete.

- Algorithm - 1. Select the channel and latch 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 0808 with 8086

Page No.

Date

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
```

(Students Co- op Stores G.P.N.)

Teacher's Signature

Page No.

Date

```
Wait : 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 practica successfully

Page No.

Date

Practical. NO. 12

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

Theory:- • Stepper motor - A motor in which the motor is able to assume only discreate 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~~ indirect alignment with the stator poles. A partial but symmetric allignment 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 out 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 more data into accumat^n
3. Drive the stepper motor circuitry & interface delay
4. Decrement the counter is not zero repeat from 3
5. Repeat the above producer 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

Page No.

Date

• Program –

```
# start = stepper_motor.exe #
name "stepper"
# make_bin #
steps_before_direction_change = 20h;
jmp start
; bin data for clockwize
dat cw  db  0000_0110b
        db  0000_0100b
        db  0000_0011b
        db  0000_0010b
; bin data for counter clock_wise
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 dat cw;
  mov si, 0
  mov cx, 0
  next step:
  wait: in al, 7
        test al, 10000000b
        jz wait
  mov  al, [bx][si]
  out  7, al
  inc si
  cmp si, 4
  jb next_step
```

(Students Co-op. Stores G.P.N.)

Teacher's Signature

Page No.

Date

```
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 - i 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. JZ :- The JZ command is used to jump when zero flag is se

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

v Wait :- It checks pending unmasked floating point explains

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

- Viva Question

i What is meant by stepper motor?

→ It's a DC motor that move in discrete steps. It have multiple coils, organized in group 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.

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

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

- It is used to mark the end of sub-routine It has no parameter. 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 (80×25). 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 next row block increment dh and compare dh to 16.
8. If zero flag is zero jump to stop-point (step 12) and move a to dl
9. In next_char block set the cursor position <dl, dh> and move the point to al

Page No.
Date

10. Increment bl (i.e next attribute) and dl also.
11. Again compare dl to 16 if zero flag comes to zero jump to next-row (step 7) or else to next-char (step 9)
12. Or 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
```

(Students Co-op Stores G.P.N.)

Teacher's Signature

```
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 bl
inc dl
cmp dl, 16
je nextrow
jmp next_char

stop print:
mov. al, 10
```

Date

```
mov  dh, 5
mov  ah, 02h
int  10h
mov  al, 'x'
mov  ah, 0eh
int  10h
mov  ah, 0
int  16h
ret.
```

Result - Hence I have prepared a miniproject successfully