ASSEMBLY LANGUAGE PROGRAMMING LAB

RECORD WORK

BACHELOR OF ENGINEERING

in

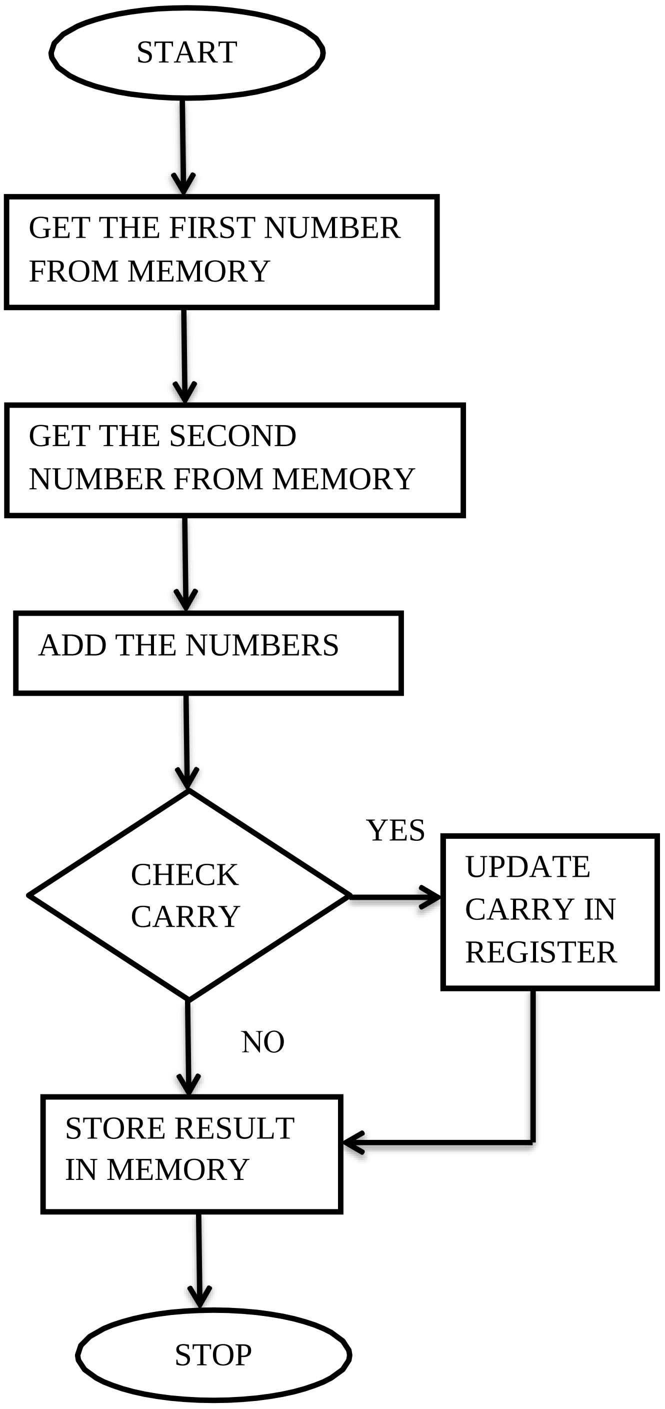
COMPUTER SCIENCE AND ENGINEERING (CSE)

THIAGARAJAR COLLEGE OF ENGINEERING,
MADURAI – 625 015

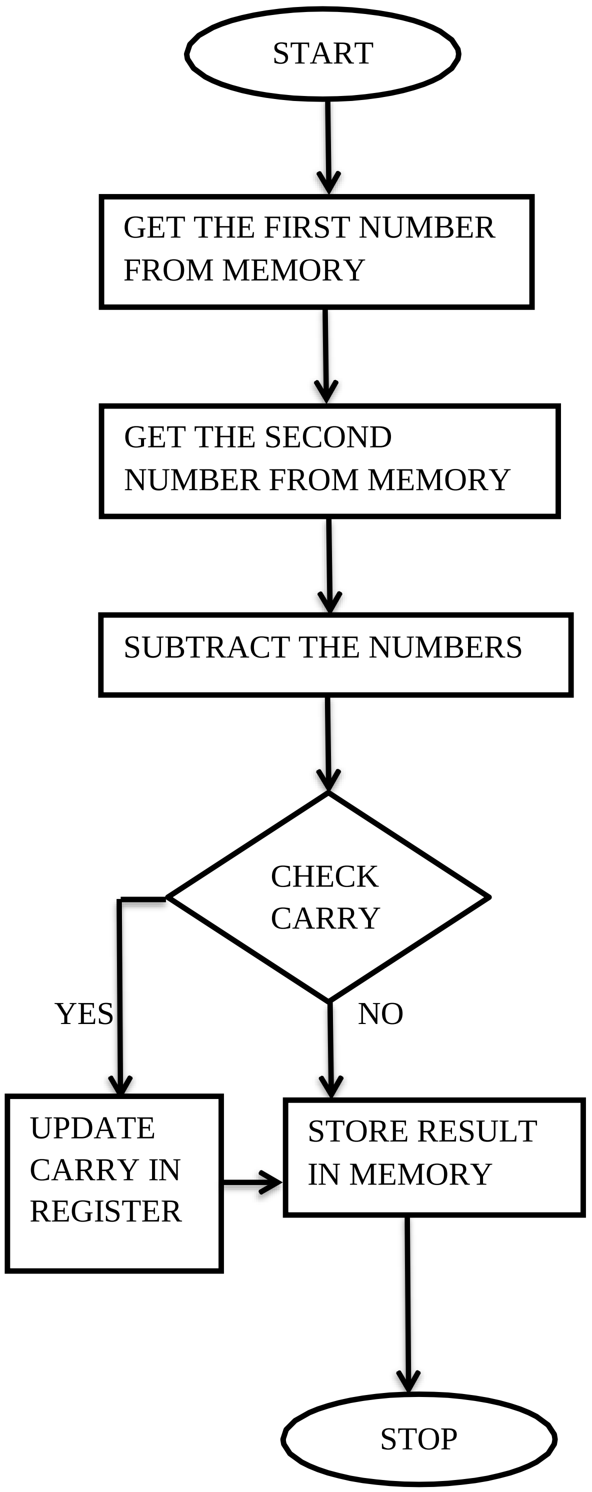
TABLE OF CONTENTS

FLOWCHART:

ADDITION



SUBTRACTION



EX NO 1 ADDITION /SUBRACTION OF 16 BIT NUMBERS

AIM

To develop an Assembly language program for addition/ subtraction of 16 bit numbers.

ALGORITHM

ADDITION

Step 1: Start
Step 2: Get the first 16 bit no from memory
Step 3: Get the second 16 bit number from memory
Step 4: Add the numbers
Step 5: Check carry flag and update in register
Step 6: Store result and carry to memory
Step 7: Stop

SUBTRACTION

Step 1: Start
Step 2: Get the first 16 bit no from memory
Step 3: Get the second 16 bit number from memory
Step 4: Subtract the numbers
Step 5: Check carry flag and update in register
Step 6: Store result and carry to memory
Step 7: Stop

PROGRAM

1. ADDITION OF TWO 16-BIT NUMBERS

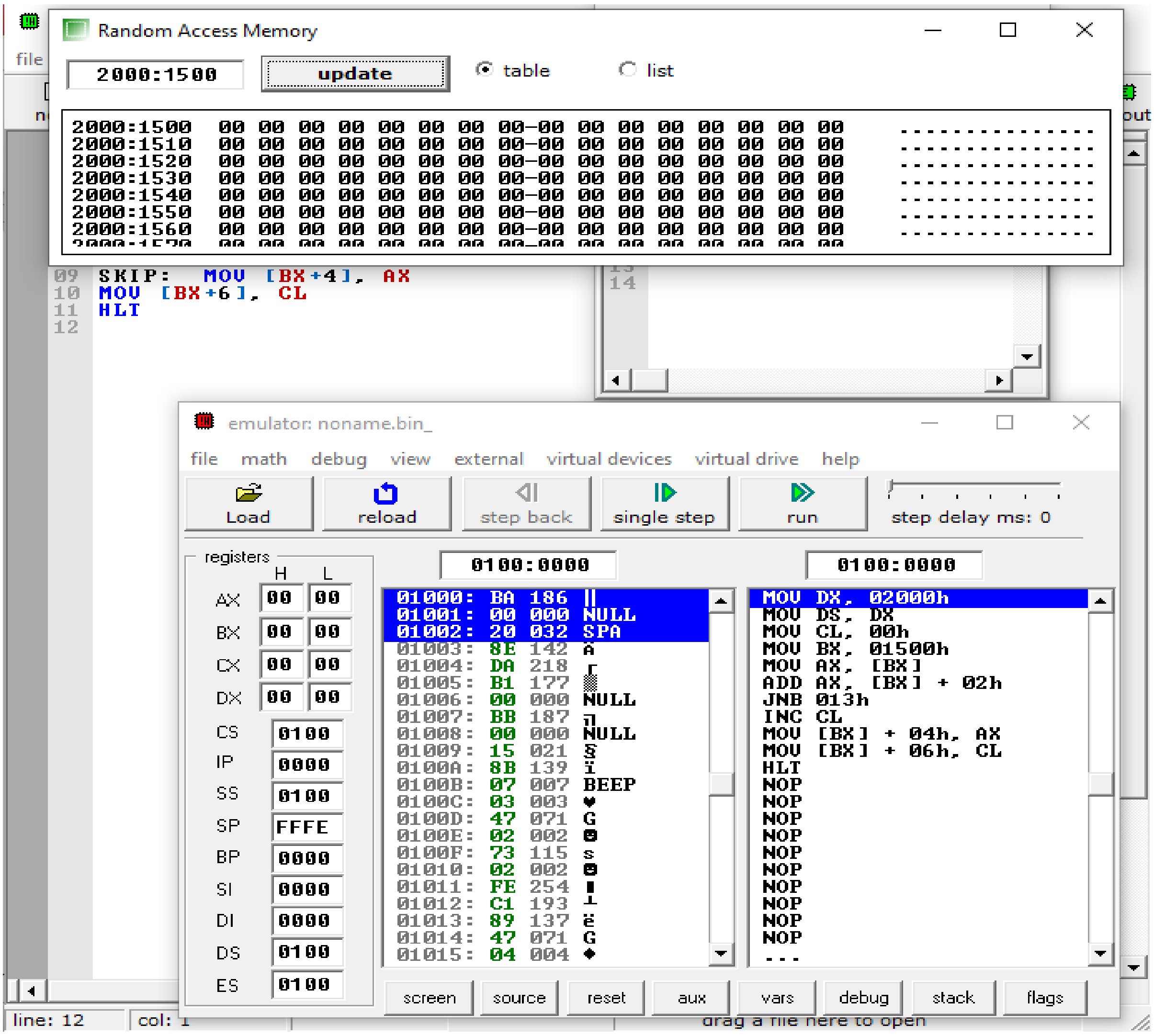
```
MOV DX, 2000H
MOV DS, DX
MOV CL, 00H
MOV BX, 1500H
MOV AX, [BX]
ADD AX, [BX+2]
JNC SKIP
INC CL
SKIP: MOV [BX+4], AX
      MOV [BX+6], CL
HLT
```

2. SUBTRACTION OF TWO 16-BIT NUMBERS

```
MOV BX, 2000H
MOV DS, BX
MOV CL, 00H
MOV AX, [1500H]
MOV BX, [1502H]
SUB AX, BX
MOV [1504H], AX
JNC SKIP
INC CL
SKIP: MOV [1506H], CL
HLT
```

SCREEN SHOTS

BEFORE EXECUTION (ADD) :



```
2000:1500

2000:1500  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

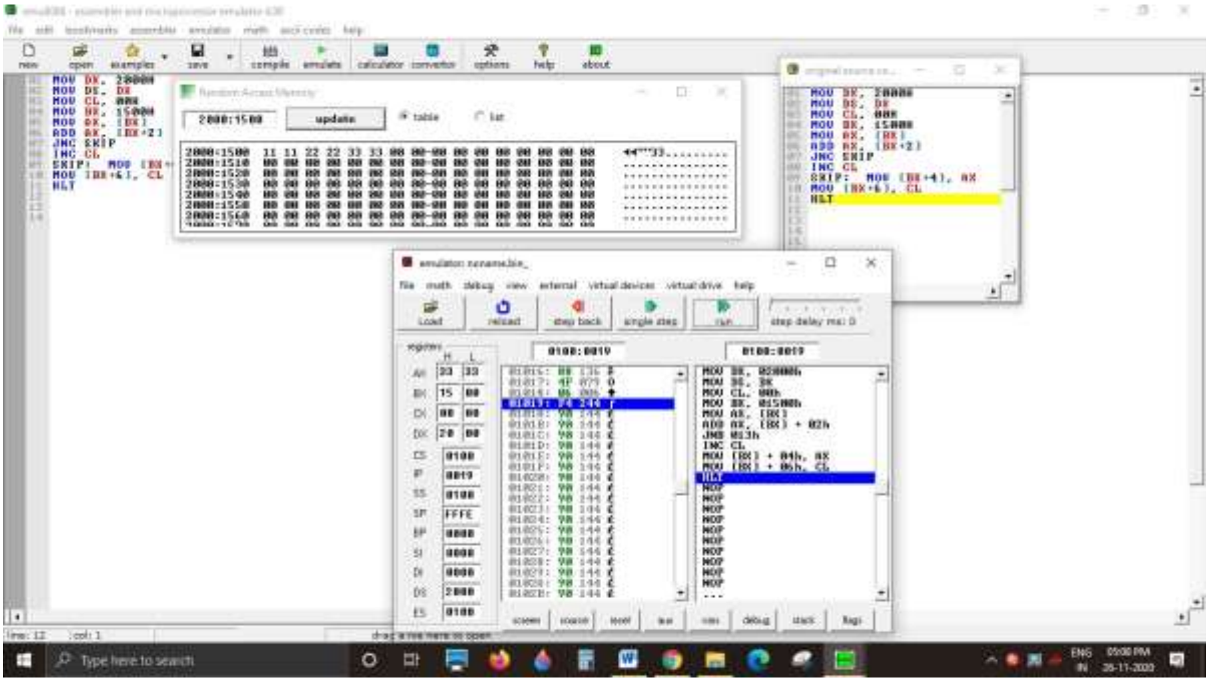
```
SKIP:  MOV [BX+4], AX
MOV [BX+6], CL
HLT
```

```
     H   L
AX   00  00
BX   00  00
CX   00  00
DX   00  00
CS   0100
IP   0000
SS   0100
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0100
ES   0100
```

```
0100:0000

01000: BA 186 ║
01001: 00 000 NULL
01002: 20 032 SPA
01003: 8E 142 Ä
01004: DA 218 ┌
01005: B1 177 ▒
01006: 00 000 NULL
01007: BB 187 ╗
01008: 00 000 NULL
01009: 15 021 §
0100A: 8B 139 ï
0100B: 07 007 BEEP
0100C: 03 003 ♥
0100D: 47 071 G
0100E: 02 002 ☻
0100F: 73 115 s
01010: 02 002 ☻
01011: FE 254 ■
01012: C1 193 ┴
01013: 89 137 ë
01014: 47 071 G
01015: 04 004 ♦
```

```
0100:0000

MOV  DX, 02000h
MOV  DS, DX
MOV  CL, 00h
MOV  BX, 01500h
MOV  AX, [BX]
ADD  AX, [BX] + 02h
JNB  013h
INC  CL
MOV  [BX] + 04h, AX
MOV  [BX] + 06h, CL
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

AFTER EXECUTION(ADD) :

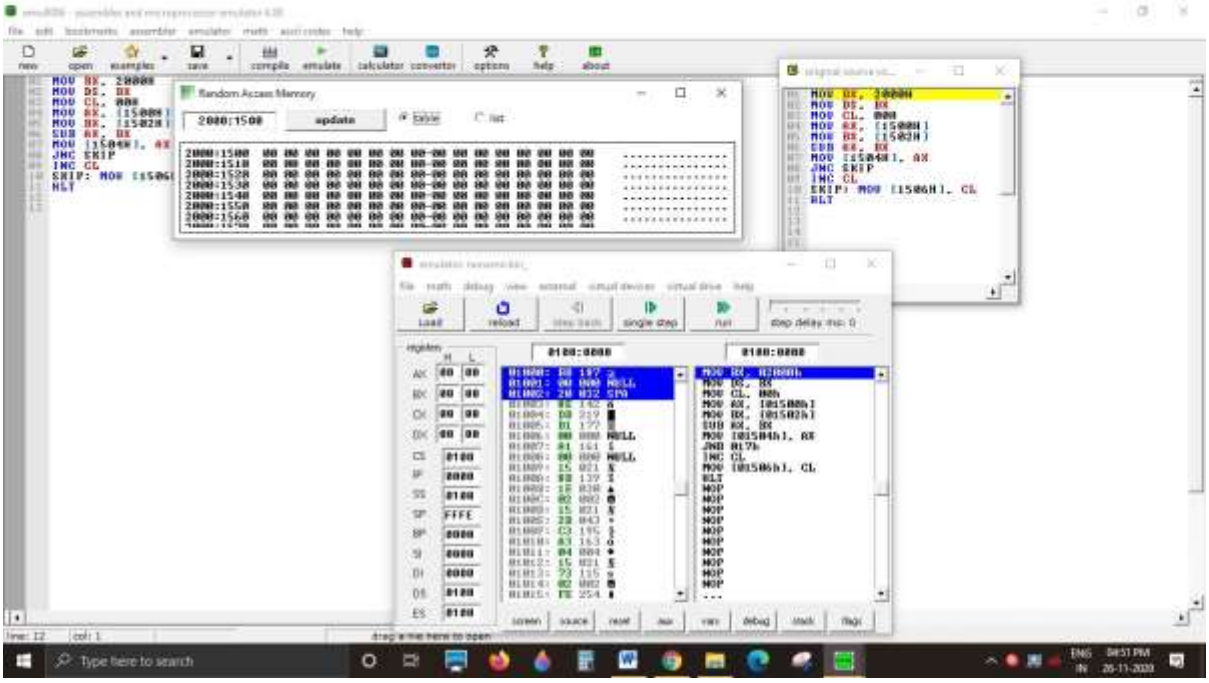


```
MOV DX, 2000H
MOV DS, DX
MOV CL, 00H
MOV BX, 1500H
MOV AX, [BX]
ADD AX, [BX+2]
JNC SKIP
INC CL
SKIP:  MOV [BX+4], AX
MOV [BX+6], CL
HLT
```

```
2000:1500  11 11 22 22 33 33 00 00-00 00 00 00 00 00 00 00
2000:1510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 33 33   BX 15 00   CX 00 00   DX 20 00
CS 0100  IP 0019  SS 0100  SP FFFE  BP 0000  SI 0000  DI 0000  DS 2000  ES 0100

MOV DX, 02000h
MOV DS, DX
MOV CL, 00h
MOV BX, 01500h
MOV AX, [BX]
ADD AX, [BX] + 02h
JNB 013h
INC CL
MOV [BX] + 04h, AX
MOV [BX] + 06h, CL
HLT
```

BEFORE EXECUTION (SUB):

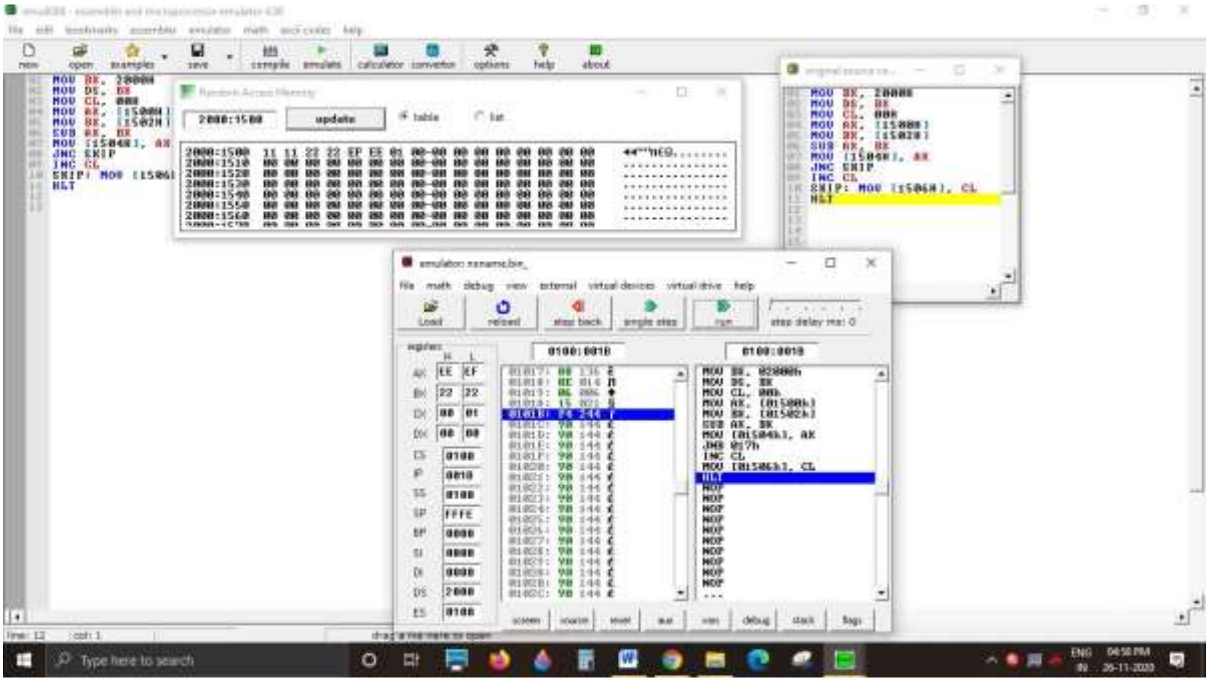


```
MOV DX, 2000H
MOV DS, DX
MOV CL, 00H
MOV AX, [1500H]
MOV BX, [1502H]
SUB AX, BX
MOV [1504H], AX
JNC SKIP
INC CL
SKIP: MOV [1506H], CL
HLT
```

```
2000:1500  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 00 00   BX 00 00   CX 00 00   DX 00 00
CS 0100  IP 0000  SS 0100  SP FFFE  BP 0000  SI 0000  DI 0000  DS 0100  ES 0100

MOV DX, 02000h
MOV DS, DX
MOV CL, 00h
MOV AX, [01500h]
MOV BX, [01502h]
SUB AX, BX
MOV [01504h], AX
JNB 017h
INC CL
MOV [01506h], CL
HLT
```

AFTER EXECUTION (SUB) :



```
MOV BX, 2000H
MOV DS, BX
MOV CL, 00H
MOV AX, [1500H]
MOV BX, [1502H]
SUB AX, BX
MOV [1504H], AX
JNC SKIP
INC CL
SKIP: MOV [1506H], CL
HLT
```

```
2000:1500  11 11 22 22 EF EE 01 00-00 00 00 00 00 00 00 00
```

```
AX  EE EF
BX  22 22
CX  00 01
DX  00 00
CS  0100
IP  0018
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  2000
ES  0100
```

SAMPLE INPUT AND OUTPUT

ADDITION

MEMORY LOCATION					
2000 : 1500	FF	FF	FF	FF	Input
2000 : 1504	FE	FF	01		Output

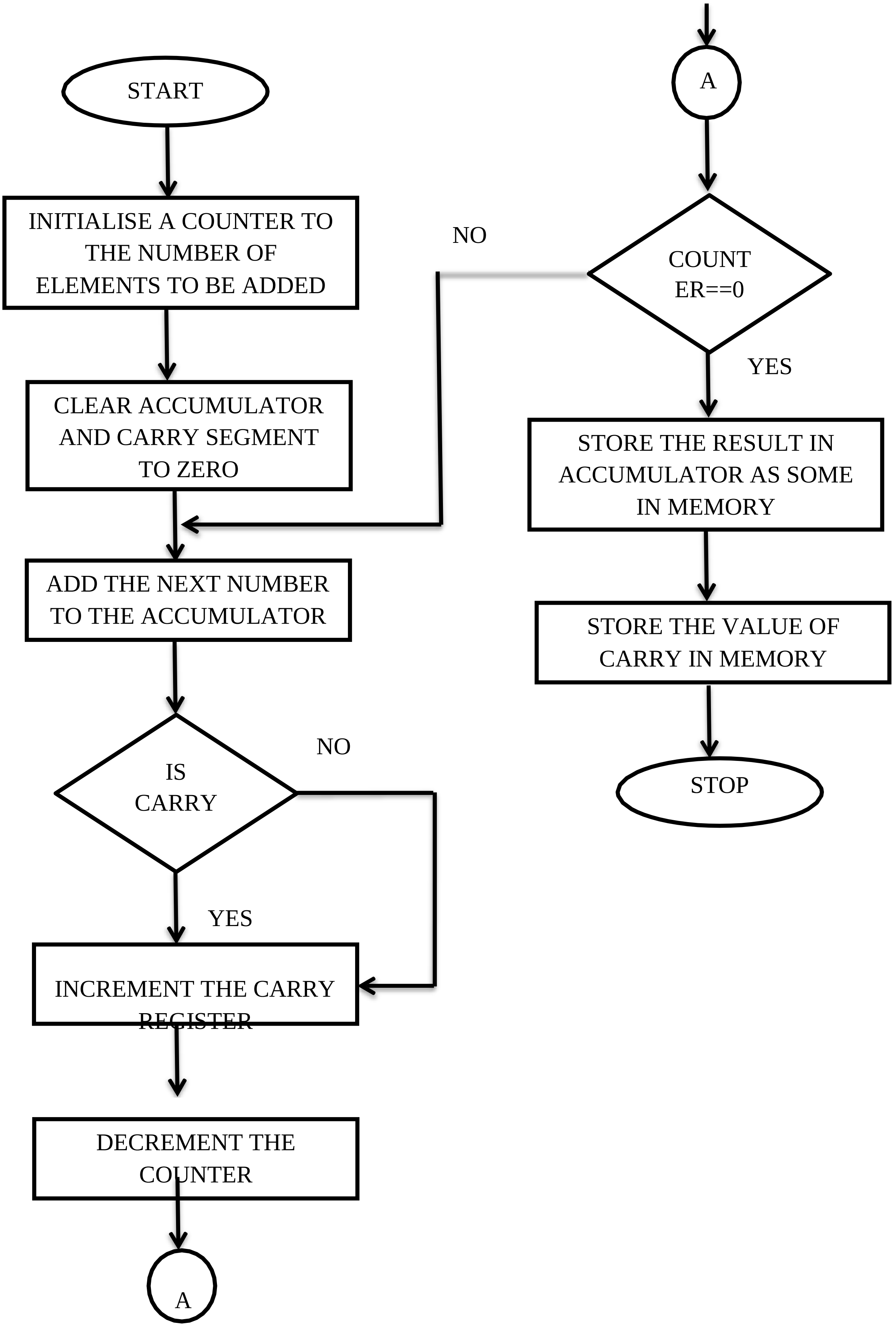
SUBTRACTION

MEMORY LOCATION					
2000 : 1500	11	11	22	22	Input
2000 : 1504	FE	FF	01		output

RESULT

Thus the ALP program for add/sub of 16 bit numbers were developed and output is verified using emu 8086.

FLOWCHART


START
INITIALISE A COUNTER TO THE NUMBER OF ELEMENTS TO BE ADDED
CLEAR ACCUMULATOR AND CARRY SEGMENT TO ZERO
ADD THE NEXT NUMBER TO THE ACCUMULATOR
IS CARRY
NO
YES
INCREMENT THE CARRY REGISTER
DECREMENT THE COUNTER
A
A
COUNT ER==0
NO
YES
STORE THE RESULT IN ACCUMULATOR AS SOME IN MEMORY
STORE THE VALUE OF CARRY IN MEMORY
STOP

EX NO 2 ADDITION OF 16 BIT ARRAY OF NUMBERS

AIM

To develop an Assembly language program for addition of 16 bit array of numbers.

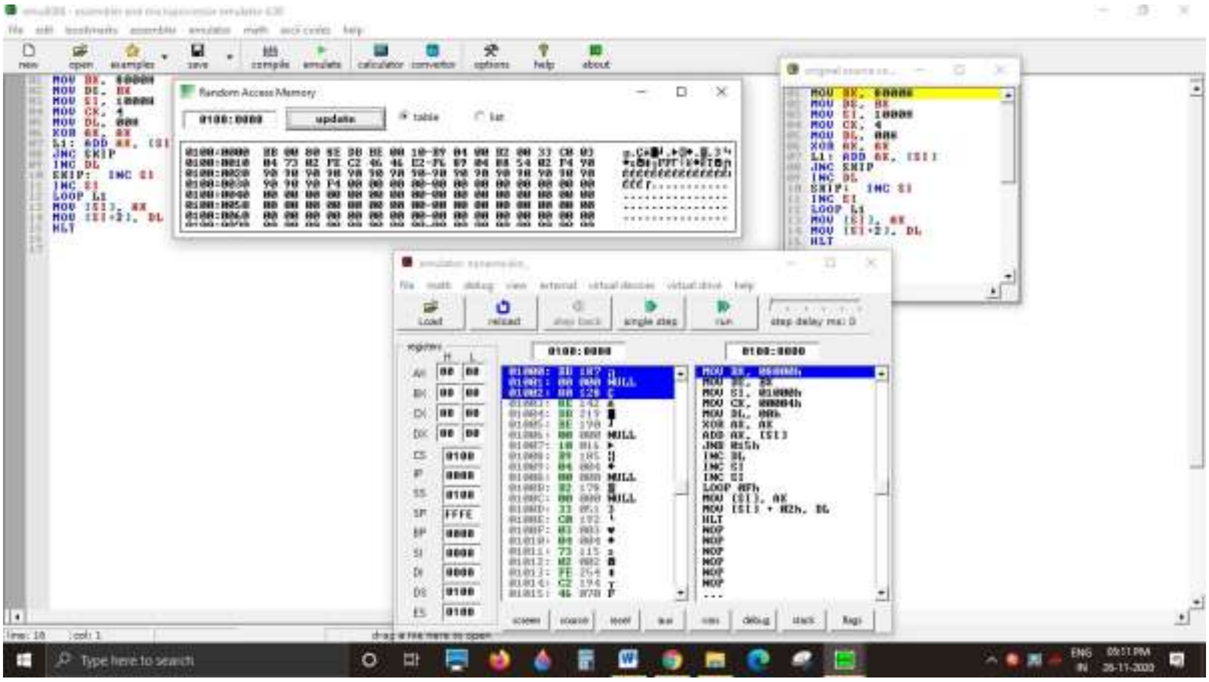
ALGORITHM

Step 1: Start
Step 2: Read the numbers one by one from memory
Step 3: Add the number and get another number from array and add it
Step 4: Store the final result in memory
Step 5: Stop

PROGRAM

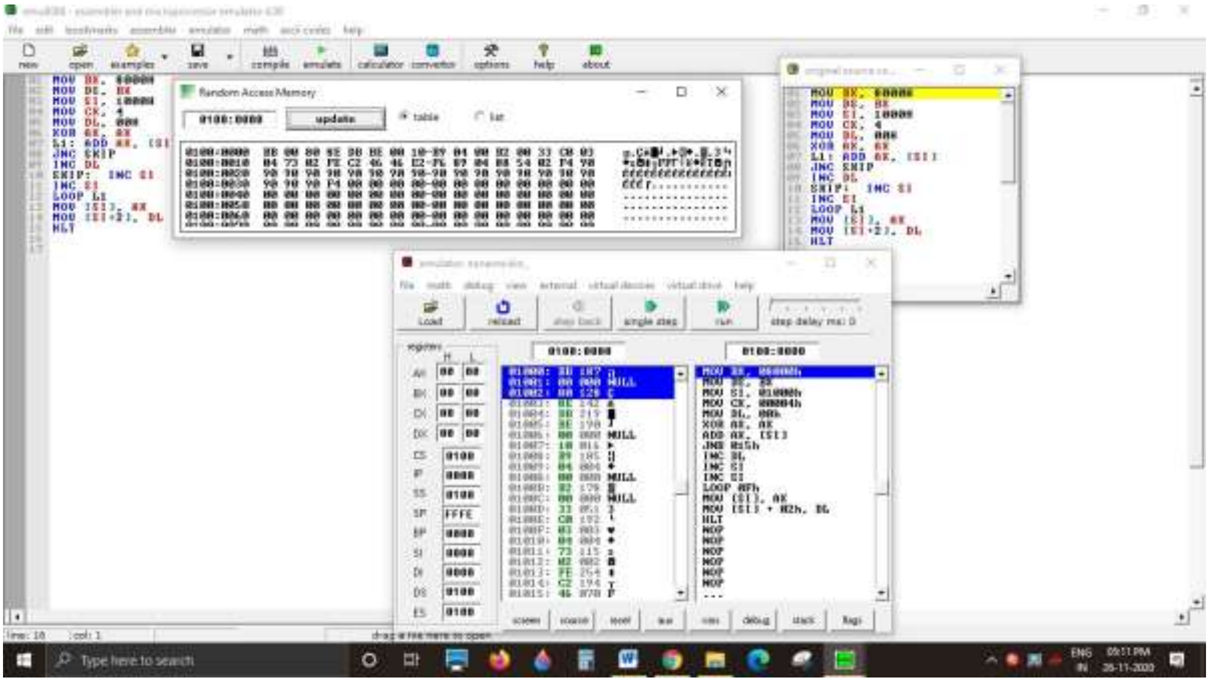
```
MOV BX, 8000H
MOV DS, BX
MOV SI, 1000H
MOV CX, 4
MOV DL, 00H
XOR AX, AX
L1: ADD AX, [SI]
JNC SKIP
INC DL
SKIP: INC SI
      INC SI
      LOOP L1
MOV [SI], AX
MOV [SI+2], DL
HLT
```

SCREENSHOTS:
BEFORE EXECUTION:



```
MOV BX, 8000H
MOV DS, BX
MOV SI, 1000H
MOV CX, 4
MOV DL, 00H
XOR AX, AX
L1: ADD AX, [SI]
JNC SKIP
INC DL
SKIP:  INC SI
INC SI
LOOP L1
MOV [SI], AX
MOV [SI+2], DL
HLT
```

AFTER EXECUTION:



```
MOV BX, 8000H
MOV DS, BX
MOV SI, 1000H
MOV CX, 4
MOV DL, 00H
XOR AX, AX
L1: ADD AX, [SI]
JNC SKIP
INC DL
SKIP:  INC SI
INC SI
LOOP L1
MOV [SI], AX
MOV [SI+2], DL
HLT
```

SAMPLE INPUT AND OUTPUT

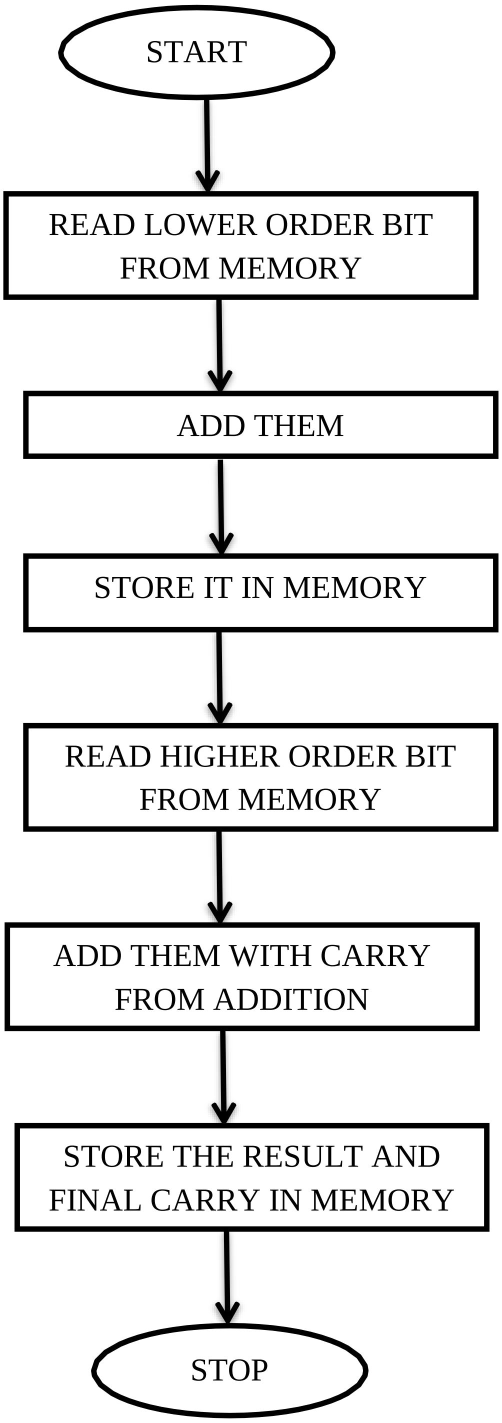
MEMORY LOCATION					
8000 : 1000	10	10	FF	FE	Input
					Output

RESULT:

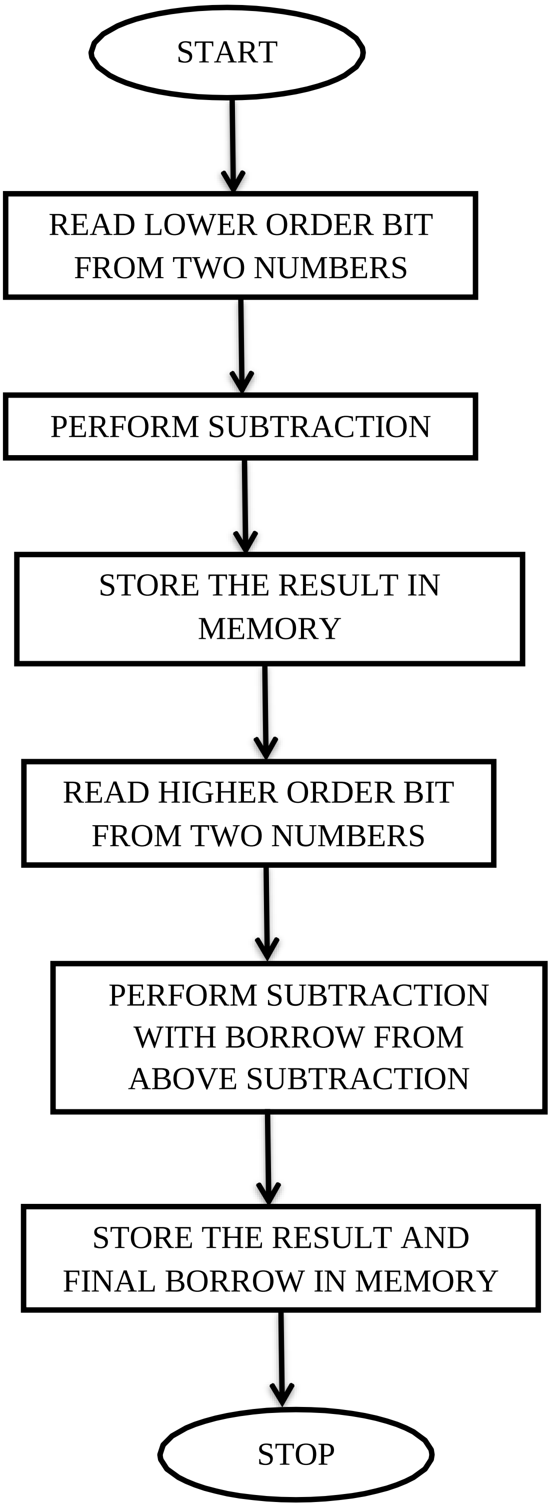
Thus the ALP program for add of 16 bit array of numbers were developed and output is verified using emu 8086.

FLOWCHART

ADDITION


START
READ LOWER ORDER BIT FROM MEMORY
ADD THEM
STORE IT IN MEMORY
READ HIGHER ORDER BIT FROM MEMORY
ADD THEM WITH CARRY FROM ADDITION
STORE THE RESULT AND FINAL CARRY IN MEMORY
STOP

SUBTRACTION


START
READ LOWER ORDER BIT FROM TWO NUMBERS
PERFORM SUBTRACTION
STORE THE RESULT IN MEMORY
READ HIGHER ORDER BIT FROM TWO NUMBERS
PERFORM SUBTRACTION WITH BORROW FROM ABOVE SUBTRACTION
STORE THE RESULT AND FINAL BORROW IN MEMORY
STOP

EX NO 3 ADDITION/ SUBTRACTION OF TWO 32 BIT NUMBERS

AIM

To develop an Assembly language program for addition/ subtraction of two 32 bit numbers.

ALGORITHM

ADDITION

Step 1: Start
Step 2: Read the lower order bits of two numbers
Step 3: Add them
Step 4: Store the result in memory
Step 5: Read the higher order bit of two numbers
Step 6: Add them with the carry of above addition
Step 7: Store the result and final carry/borrow in memory
Step 8: Stop

SUBTRACTION

Step 1: Start
Step 2: Read the lower order bits of two numbers
Step 3: Subtract one from another
Step 4: Store the result in memory
Step 5: Read the higher order bit of two numbers
Step 6: Subtract one number from another with the borrow from above subtraction
Step 7: Store the result and final carry/borrow in memory
Step 8: Stop

PROGRAM

1. ADDITION OF TWO 32-BIT NUMBERS

```
ORG 1000H
MOV BX, 8000H
MOV DS, BX
MOV SI, 1000H
MOV AX, [SI]
MOV CX, [SI+4]
MOV BX, [SI+2]
MOV DX, [SI+6]
ADD AX, CX
ADC BX, DX
MOV CL, 00H
```

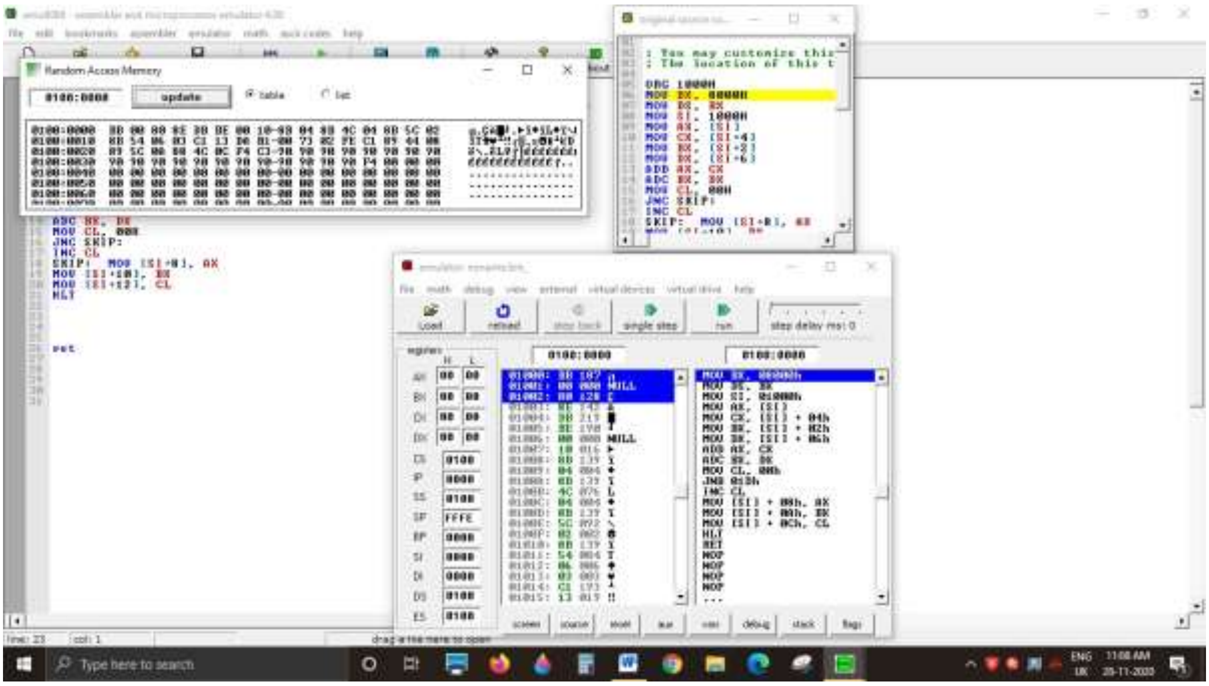
```
JNC SKIP:
INC CL
SKIP: MOV [SI+8], AX
        MOV [SI+10], BX
        MOV [SI+12], CL
HLT
```

2. SUBTRACTION OF TWO 32-BIT NUMBERS

```
ORG 1000H
MOV BX, 8000H
MOV DS, BX
MOV SI, 1000H
MOV AX, [SI]
MOV CX, [SI+4]
MOV BX, [SI+2]
MOV DX, [SI+6]
SUB AX, CX
SBB BX, DX
MOV CL, 00H
JNC SKIP:
INC CL
SKIP: MOV [SI+8], AX
        MOV [SI+10], BX
        MOV [SI+12], CL
HLT
```

SCREENSHOTS:

BEFORE EXECUTION (ADDITION OF TWO 32 BIT NUMBERS):



Random Access Memory: 0100:0000 update ● table ○ list

```
0100:0000  B8 00 00 8E D8 BE 00 10-8B 04 8B 4C 04 8B 5C 02
0100:0010  8B 54 06 03 C1 13 D0 B1-00 73 02 FE C1 89 44 08
0100:0020  89 5C 0A 88 4C 0C F4 C3-90 90 90 90 90 90 90 90
0100:0030  90 90 90 90 90 90 90 90-90 90 90 F4 00 00 00 00
0100:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0100:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0100:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
; You may customize this
; The location of this t

ORG 1000H
MOV AX, 0000H
MOV DS, AX
MOV SI, 1000H
MOV AX, [SI]
MOV CX, [SI+4]
MOV BX, [SI+2]
MOV DX, [SI+6]
ADD AX, CX
ADC BX, DX
MOV CL, 00H
JNC SKIP:
INC CL
SKIP:  MOV [SI+8], AX
MOV [SI+10], BX
MOV [SI+12], CL
HLT

ret
```

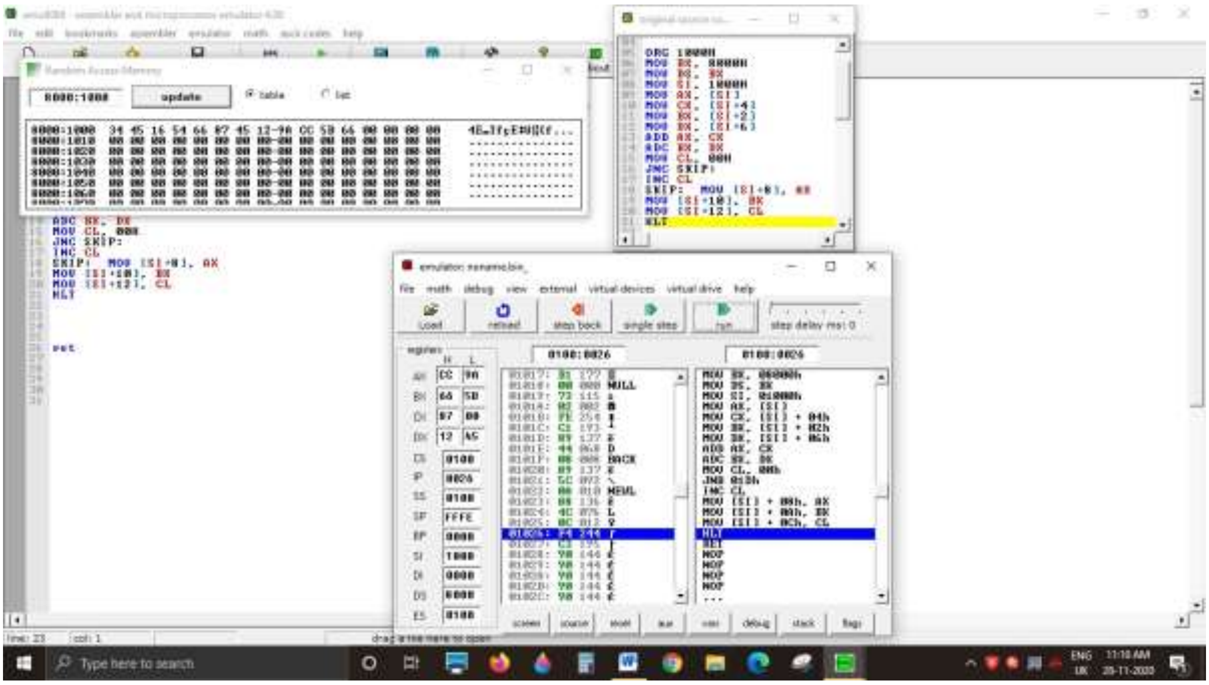
emulator: noname.bin_

Registers: AX 00 00, BX 00 00, CX 00 00, DX 00 00, CS 0100, IP 0000, SS 0100, SP FFFE, BP 0000, SI 0000, DI 0000, DS 0100, ES 0100

```
0100:0000
01000: B8 184 ╕
01001: 00 000 NULL
01002: 00 128 Ç
01003: 8E 142 Ä
01004: D8 216 ╪
01005: BE 190 ╛
01006: 00 000 NULL
01007: 10 016 ►
01008: 8B 139 ï
01009: 04 004 ♦
0100A: 8B 139 ï
0100B: 4C 076 L
0100C: 04 004 ♦
0100D: 8B 139 ï
0100E: 5C 092 \
0100F: 02 002 ☻
01010: 8B 139 ï
01011: 54 084 T
01012: 06 006 ♠
01013: 03 003 ♥
01014: C1 193 ┴
01015: 13 019 ‼
```

```
0100:0000
MOV AX, 00000h
MOV DS, AX
MOV SI, 01000h
MOV AX, [SI]
MOV CX, [SI] + 04h
MOV BX, [SI] + 02h
MOV DX, [SI] + 06h
ADD AX, CX
ADC BX, DX
MOV CL, 00h
JNB 013h
INC CL
MOV [SI] + 08h, AX
MOV [SI] + 0Ah, BX
MOV [SI] + 0Ch, CL
HLT
RET
NOP
NOP
NOP
NOP
...
```

AFTER EXECUTION:



Random Access Memory: 0000:1000 update ● table ○ list

```
0000:1000  34 45 16 54 66 87 45 12-9A CC 5B 66 00 00 00 00
0000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

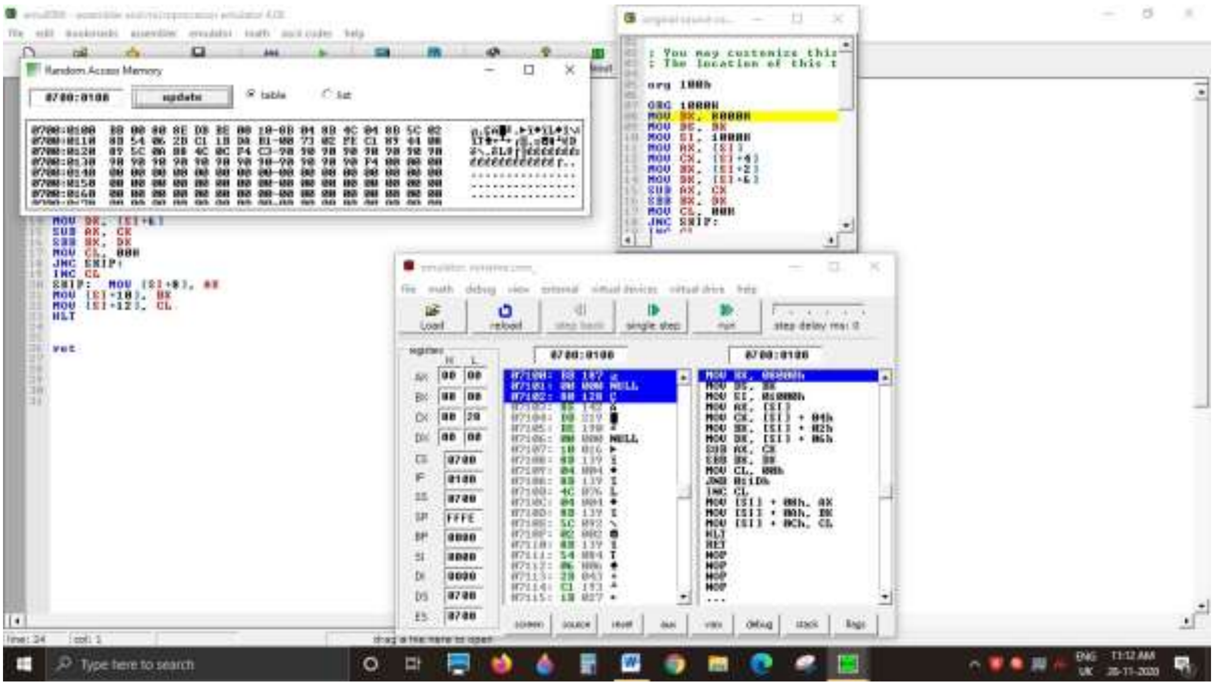
emulator: noname.bin_

Registers: AX CC 9A, BX 66 5B, CX 87 00, DX 12 45, CS 0100, IP 0026, SS 0100, SP FFFE, BP 0000, SI 1000, DI 0000, DS 0000, ES 0100

```
0100:0026
01017: B1 177 ▒
01018: 00 000 NULL
01019: 73 115 s
0101A: 02 002 ☻
0101B: FE 254 ■
0101C: C1 193 ┴
0101D: 89 137 ë
0101E: 44 068 D
0101F: 08 008 BACK
01020: 89 137 ë
01021: 5C 092 \
01022: 0A 010 NEWL
01023: 88 136 ê
01024: 4C 076 L
01025: 0C 012 ♀
01026: F4 244 ⌠
01027: C3 195 ├
01028: 90 144 É
01029: 90 144 É
0102A: 90 144 É
0102B: 90 144 É
0102C: 90 144 É
```

```
0100:0026
MOV AX, 00000h
MOV DS, AX
MOV SI, 01000h
MOV AX, [SI]
MOV CX, [SI] + 04h
MOV BX, [SI] + 02h
MOV DX, [SI] + 06h
ADD AX, CX
ADC BX, DX
MOV CL, 00h
JNB 013h
INC CL
MOV [SI] + 08h, AX
MOV [SI] + 0Ah, BX
MOV [SI] + 0Ch, CL
HLT
RET
NOP
NOP
NOP
NOP
...
```

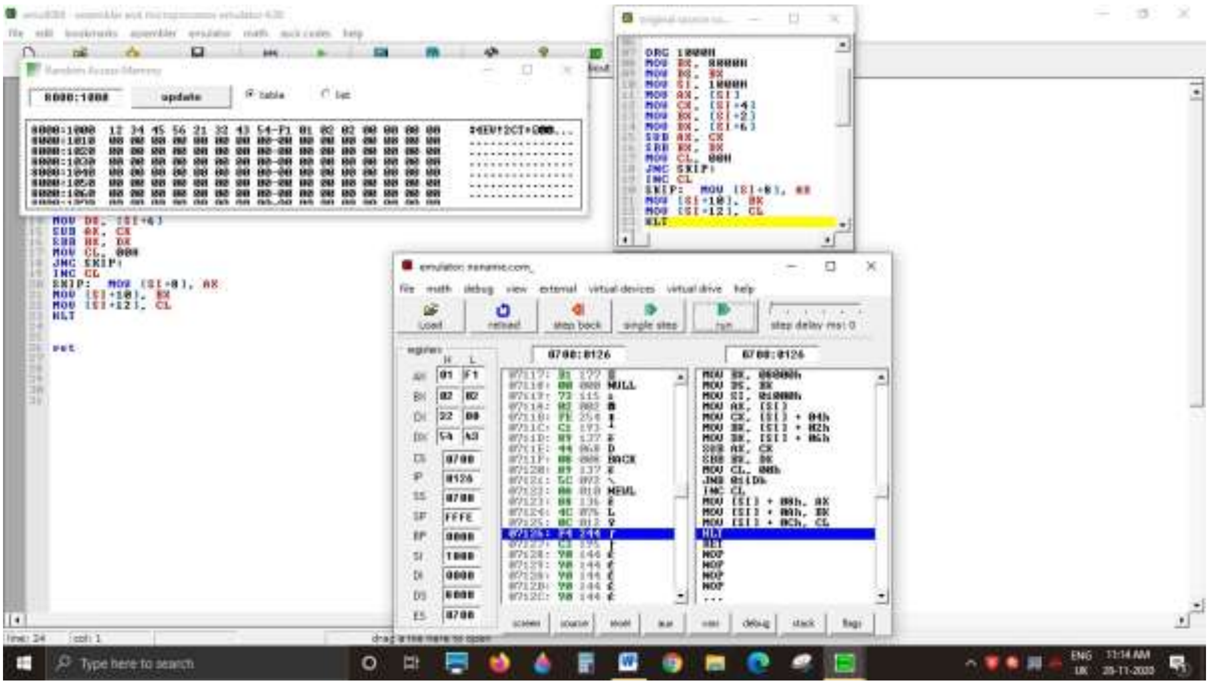
BEFORE EXECUTION(SUBTRACTION OF TWO 32 BIT NUMBERS):



```
; You may customize this
; The location of this t

org 100h

ORG 1000H
MOV AX, 8000H
MOV DS, AX
MOV SI, 1000H
MOV AX, [SI]
MOV CX, [SI+4]
MOV BX, [SI+2]
MOV DX, [SI+6]
SUB AX, CX
SBB BX, DX
MOV CL, 00H
JNC SKIP:
INC CL
SKIP:   MOV [SI+8], AX
MOV [SI+10], BX
MOV [SI+12], CL
HLT

ret
```

```
AX 00 00   BX 00 00   CX 00 28   DX 00 00
CS 0700   IP 0100   SS 0700   SP FFFE   BP 0000   SI 0000   DI 0000   DS 0700   ES 0700
```

AFTER EXECUTION:



```
8000:1000  12 34 45 56 21 32 43 54-F1 01 02 02 00 00 00 00
```

```
AX 01 F1   BX 02 02   CX 22 00   DX 54 43
CS 0700   IP 0126   SS 0700   SP FFFE   BP 0000   SI 1000   DI 0000   DS 8000   ES 0700
```

SAMPLE INPUT AND OUTPUT

ADDITION

MEMORY LOCATION									
8000 : 1000	34	45	16	54	66	87	45	12	Input
	9A	CC	5B	66					Output

SUBTRACTION

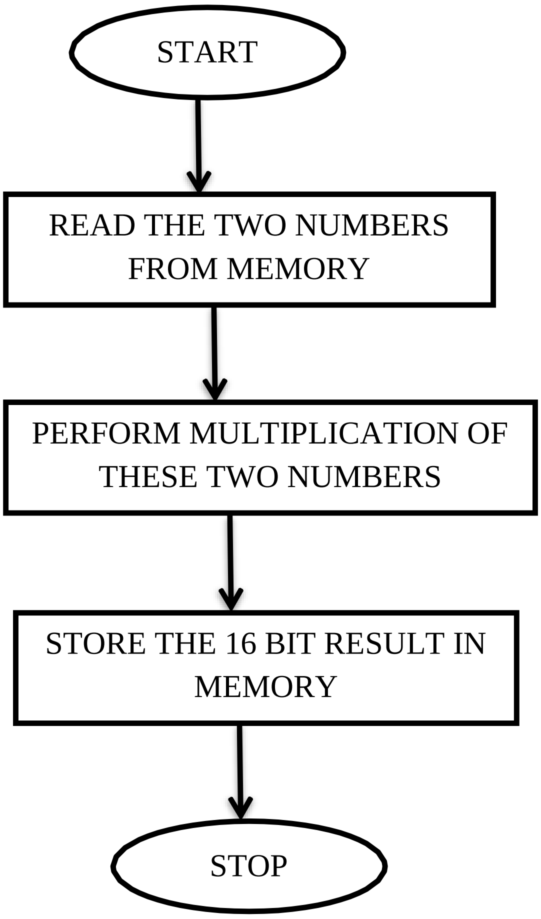
MEMORY LOCATION									
8000 : 1000	12	34	45	56	21	32	43	54	Input
	F1	01	02	02					Output

RESULT

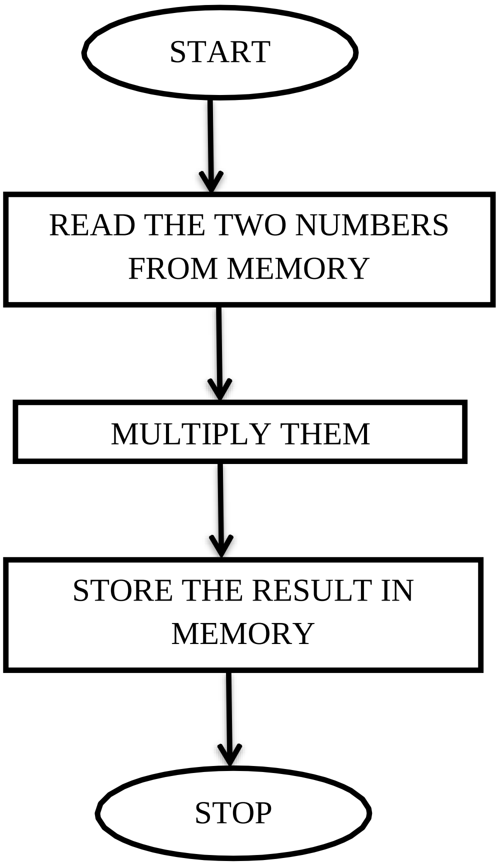
Thus the ALP program for add/sub of two 32 bit numbers were developed and output is verified using emu 8086.

FLOWCHART

TWO 8 BIT NUMBERS


START
READ THE TWO NUMBERS FROM MEMORY
PERFORM MULTIPLICATION OF THESE TWO NUMBERS
STORE THE 16 BIT RESULT IN MEMORY
STOP

TWO 16 BIT NUMBERS



EX NO 4 MULTIPLICATION OF 8/16 BIT NUMBERS

AIM

To develop an Assembly language program for multiplication of 8/16 bit numbers.

ALGORITHM

TWO 8- BIT NUMBERS

```
Step 1: Start
Step 2: Read two numbers from memory
Step 3: Store that two numbers in registers
Step 4: Multiply that two numbers
Step 5: Store the product (result) in the memory
Step 6: Stop
```

TWO 16- BIT NUMBERS

```
Step 1: Start
Step 2: Read two 16- bit numbers from memory
Step 3: Multiply the two 16- bit numbers
Step 4: Store the 32- bit result in the memory
Step 5: Stop
```

PROGRAM

1. MULTIPLICATION OF TWO 8- BIT NUMBERS

```
MOV AL, [1000H]
MOV BL, [1001H]
MUL BL
MOV [1002H], AX
HLT
```

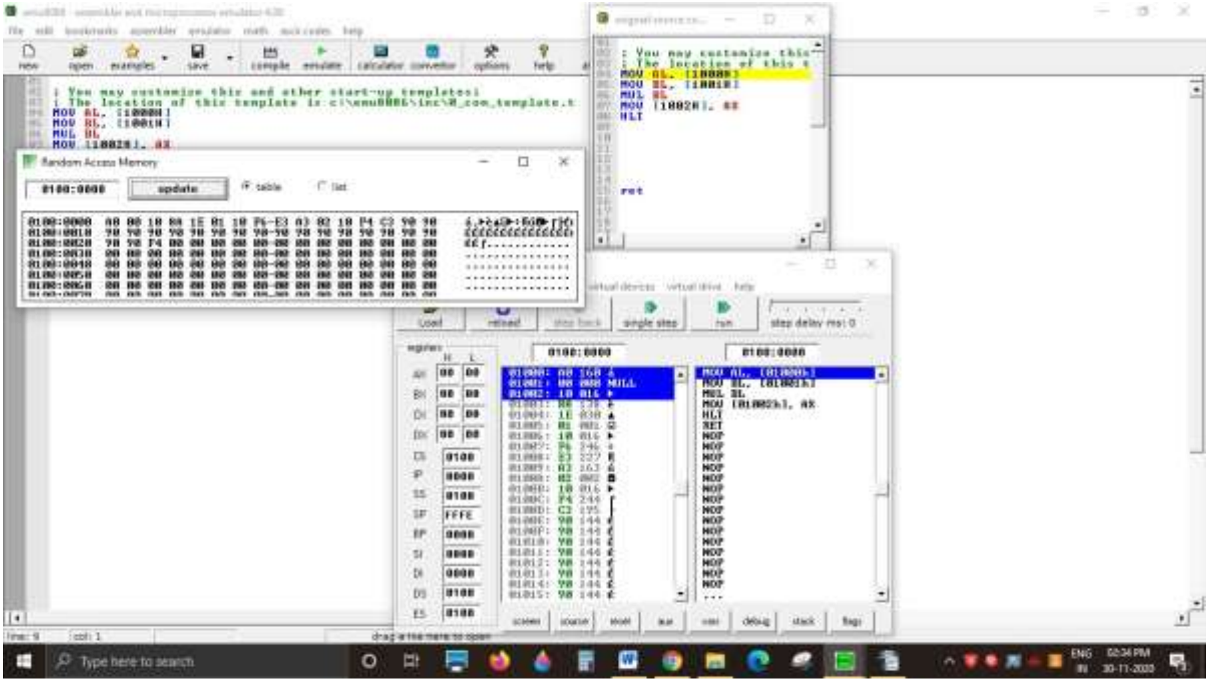
2. MULTIPLICATION OF TWO 16- BIT NUMBERS

```
MOV DX, 2000H
MOV DS, DX
MOV BX, 100H
MOV SI, 00H
MOV AX, [BX] [SI]
MOV CX, [BX] [SI+2]
MUL CX
MOV [BX] [SI+4], AX
```

MOV [BX] [SI+6], DX
HLT

SCREENSHOTS:

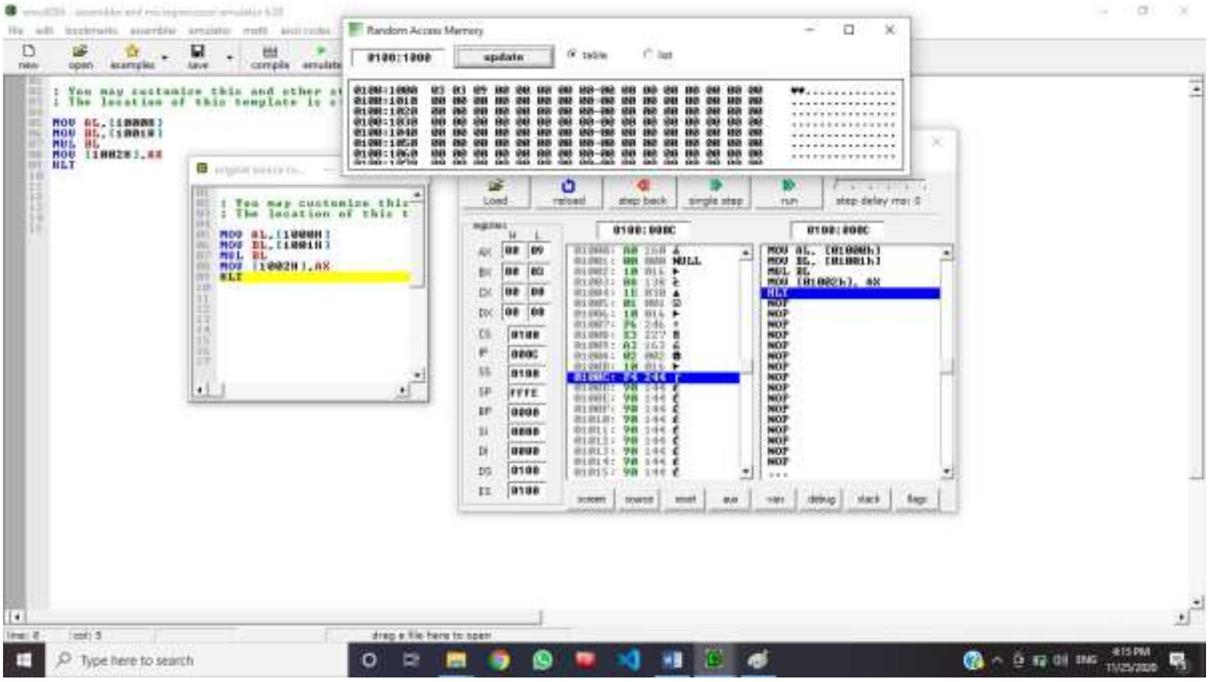
BEFORE EXECUTION(MULTIPLICATION OF 8 BIT NUMBERS):



```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.t
MOV AL, [1000H]
MOV BL, [1001H]
MUL BL
MOV [1002H], AX
HLT

ret
```

```
MOV AL, [01000h]
MOV BL, [01001h]
MUL BL
MOV [01002h], AX
HLT
RET
```

AFTER EXECUTION(MULTIPLICATION OF 8 BIT NUMBERS):

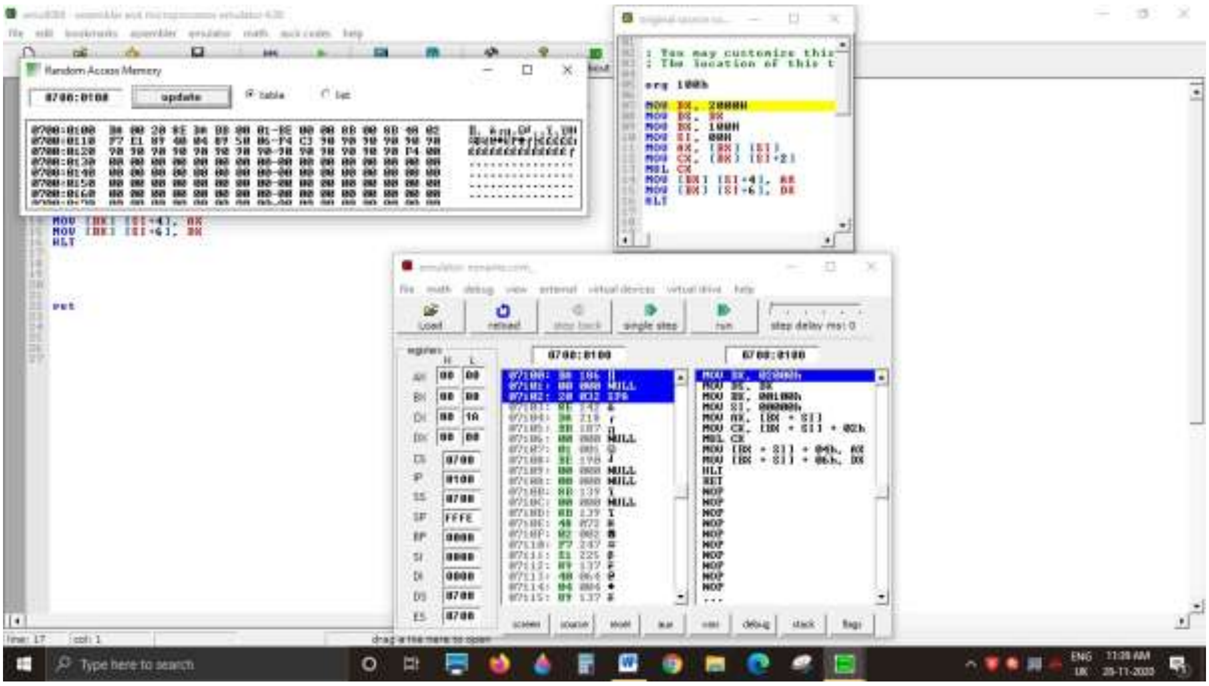


```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.t
MOV AL,[1000H]
MOV BL,[1001H]
MUL BL
MOV [1002H],AX
HLT
```

```
0100:1000  03 03 09 00 00 00 00 00-00 00 00 00 00 00 00 00
```

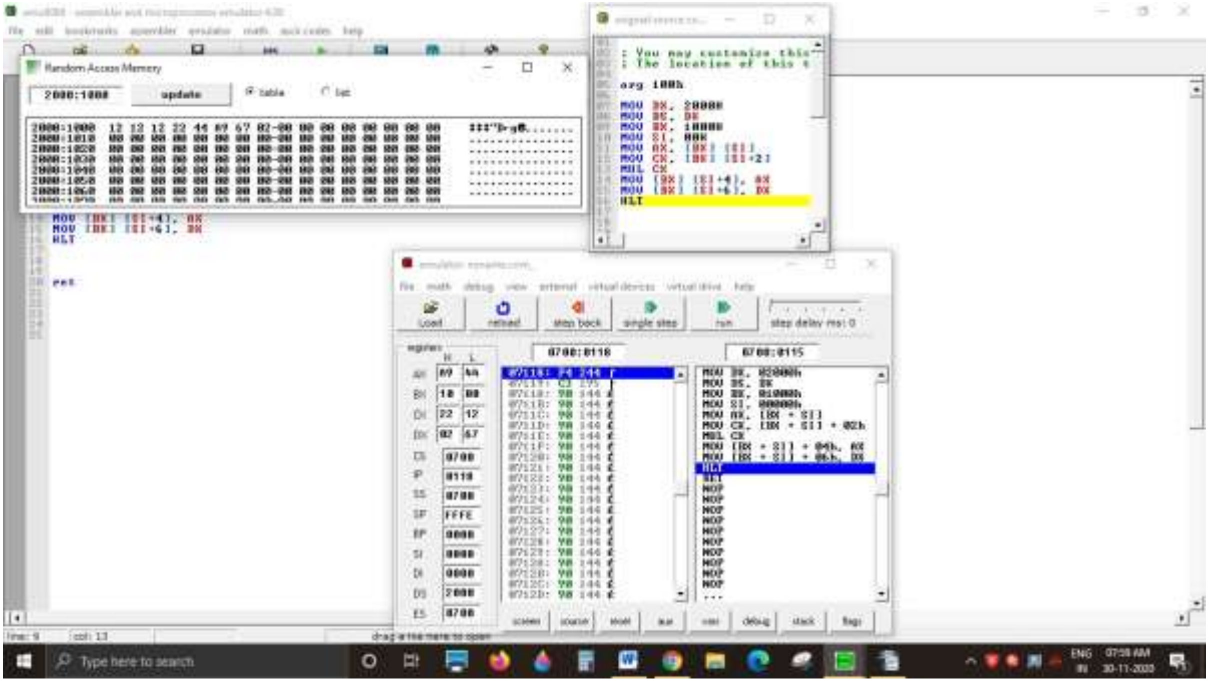
```
AX  00 09
BX  00 03
CX  00 00
DX  00 00
CS  0100
IP  000C
SS  0100
SP  FFFE
```

BEFORE EXECUTION(MULTIPLICATION OF 16 BIT NUMBERS):



```
; You may customize this
; The location of this t

org 100h

MOV BX, 2000H
MOV DS, BX
MOV BX, 1000H
MOV SI, 00H
MOV AX, [BX] [SI]
MOV CX, [BX] [SI+2]
MUL CX
MOV [BX] [SI+4], AX
MOV [BX] [SI+6], DX
HLT
```

AFTER EXECUTION:



```
2000:1000
2000:1000  12 12 12 22 44 89 67 02-00 00 00 00 00 00 00 00   ‡‡‡"Dëg☻........
```

SAMPLE INPUT AND OUTPUT

8- BIT NUMBERS

MEMORY LOCATION: 0710:0000

47	76	BA	20

MULTIPLIER	47H
MULTIPLICAND	76A
RESULT	20BAA

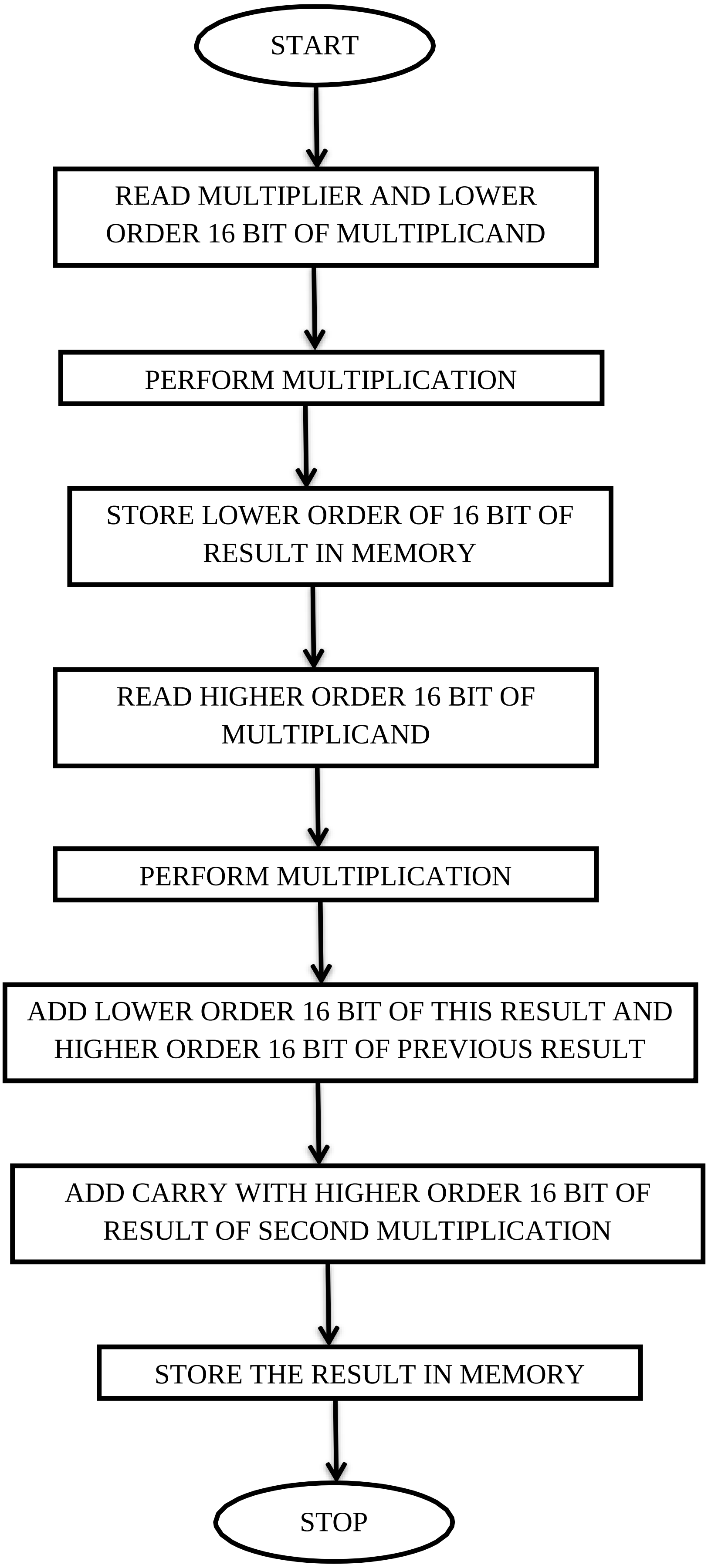
16- BIT NUMBERS

MEMORY LOCATION						
2000 : 1000	12	12	12	22	44	Input
	A9	67	02			Output

RESULT:

Thus the ALP program for multiplication of two 8 and 16 bit numbers are developed and output is verified using emu 8086.

FLOWCHART


START
READ MULTIPLIER AND LOWER ORDER 16 BIT OF MULTIPLICAND
PERFORM MULTIPLICATION
STORE LOWER ORDER OF 16 BIT OF RESULT IN MEMORY
READ HIGHER ORDER 16 BIT OF MULTIPLICAND
PERFORM MULTIPLICATION
ADD LOWER ORDER 16 BIT OF THIS RESULT AND HIGHER ORDER 16 BIT OF PREVIOUS RESULT
ADD CARRY WITH HIGHER ORDER 16 BIT OF RESULT OF SECOND MULTIPLICATION
STORE THE RESULT IN MEMORY
STOP

EX NO 5 MULTIPLICATION OF 32 BIT NUMBER AND 16 BIT NUMBER

AIM

To develop an Assembly language program for multiplication of 32 and 16 bit number.

ALGORITHM

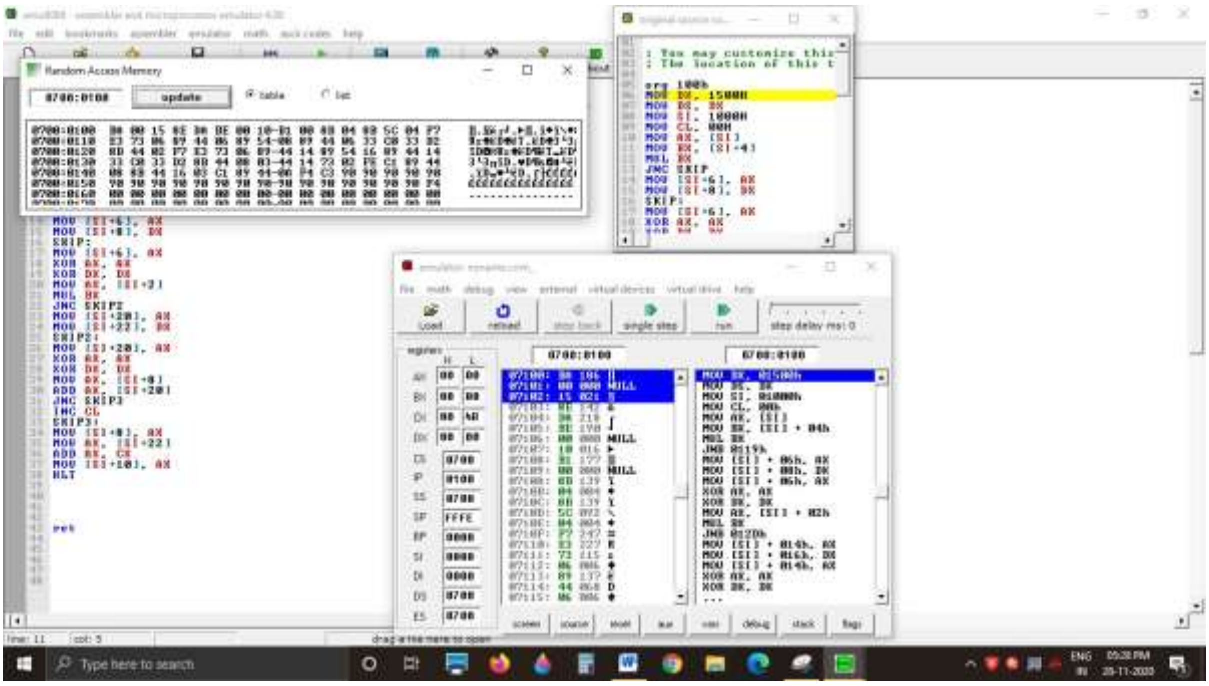
```
Step 1: Start
Step 2: Read multiplier and lower order multiplicand from memory
Step 3: Perform multiplication
Step 4: Store the lower order result in memory
Step 5: Read higher order multiplicand from memory and perform multiplication
Step 6: Add higher order of previous result and lower order of this result and store the result
        in memory
Step 7: Add carry with higher order of result of second multiplication
Step 8: Store the result in memory
Step 9: Stop
```

PROGRAM

```
MOV DX, 1500H
MOV DS, DX
MOV SI, 1000H
MOV CL, 00H
MOV AX, [SI]
MOVBX, [SI+4]
MUL BX
JNC SKIP
MOV [SI+6], AX
MOV [SI+8], DX
SKIP:
MOV [SI+6], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+2]
MUL BX
JNC SKIP2
MOV [SI+20], AX
MOV [SI+22], DX
SKIP2:
MOV [SI+20], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+8]
ADD AX, [SI+20]
JNC SKIP3
INC CL
SKIP3:
MOV [SI+8], AX
MOV AX, [SI+22]
ADD AX, CX
MOV [SI+10], AX
HLT
```

SCREENSHOTS:

BEFORE EXECUTION:



```
0700:0100  BA 00 15 8B DA BE 00 10-B1 00 8B 04 8B 5C 04 F7
0700:0110  E3 73 06 89 44 06 89 54-08 89 44 06 33 C0 33 D2
0700:0120  8B 44 02 F7 E3 73 06 89-44 14 89 54 16 89 44 14
0700:0130  33 C0 33 D2 8B 44 08 03-44 14 73 02 FE C1 89 44
0700:0140  08 8B 44 16 03 C1 89 44-0A F4 C3 90 90 90 90 90
0700:0150  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 F4
0700:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
; You may customize this
; The location of this t

org 100h
MOV DX, 1500H
MOV BX, DX
MOV SI, 1000H
MOV CL, 00H
MOV AX, [SI]
MOV BX, [SI+4]
MUL BX
JNC SKIP
MOV [SI+6], AX
MOV [SI+8], DX
SKIP:
MOV [SI+6], AX
XOR AX, AX
```

```
MOV [SI+6], AX
MOV [SI+8], DX
SKIP:
MOV [SI+6], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+2]
MUL BX
JNC SKIP2
MOV [SI+20], AX
MOV [SI+22], DX
SKIP2:
MOV [SI+20], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+8]
ADD AX, [SI+20]
JNC SKIP3
INC CL
SKIP3:
MOV [SI+8], AX
MOV AX, [SI+22]
ADD AX, CX
MOV [SI+10], AX
HLT



ret
```

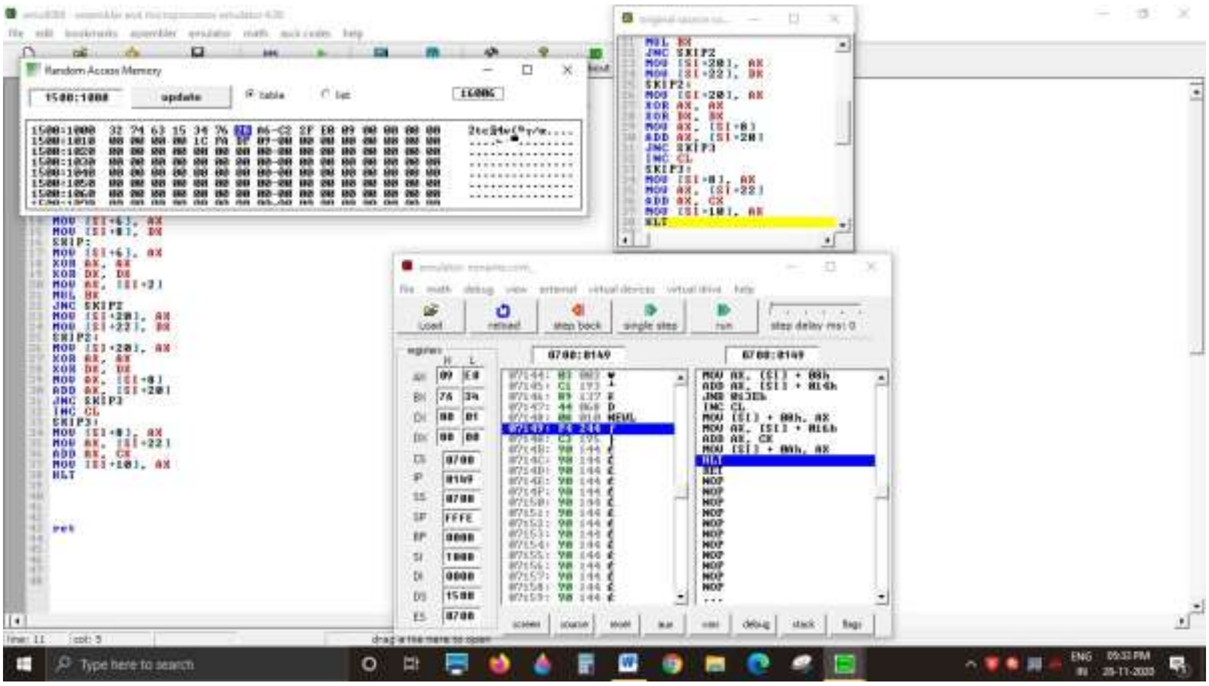
line: 11   col: 5   drag a file here to open

```
AX  00 00
BX  00 00
CX  00 4A
DX  00 00
CS  0700
IP  0100
SS  0700
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  0700
ES  0700
```

```
0700:0100
07100: BA 186
07101: 00 000 NULL
07102: 15 021
07103: 8B 139
07104: DA 218
07105: BE 190
07106: 00 000 NULL
07107: 10 016
07108: B1 177
07109: 00 000 NULL
0710A: 8B 139
0710B: 04 004
0710C: 8B 139
0710D: 5C 092
0710E: 04 004
0710F: F7 247
07110: E3 227
07111: 73 115
07112: 06 006
07113: 89 137
07114: 44 068 D
07115: 06 006
```

```
0700:0100
MOV DX, 01500h
MOV BX, DX
MOV SI, 01000h
MOV CL, 00h
MOV AX, [SI]
MOV BX, [SI] + 04h
MUL BX
JNB 0119h
MOV [SI] + 06h, AX
MOV [SI] + 08h, DX
MOV [SI] + 06h, AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI] + 02h
MUL BX
JNB 012Dh
MOV [SI] + 014h, AX
MOV [SI] + 016h, DX
MOV [SI] + 014h, AX
XOR AX, AX
XOR DX, DX
...
```

AFTER EXECUTION:



```
1500:1000   32 74 63 15 34 76 28 A6-C2 2F E0 09 00 00 00 00   2tc§4v(ª¿/α....
```

```
MOV [SI+6], AX
MOV [SI+8], DX
SKIP:
MOV [SI+6], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+2]
MUL BX
JNC SKIP2
MOV [SI+20], AX
MOV [SI+22], DX
SKIP2:
MOV [SI+20], AX
XOR AX, AX
XOR DX, DX
MOV AX, [SI+8]
ADD AX, [SI+20]
JNC SKIP3
INC CL
SKIP3:
MOV [SI+8], AX
MOV AX, [SI+22]
ADD AX, CX
MOV [SI+10], AX
HLT



ret
```

```
AX  09  E0
BX  76  34
CX  00  01
DX  00  00
CS  0700
IP  01b9
SS  0700
SP  FFFE
BP  0000
SI  1000
DI  0000
DS  1500
ES  0700
```

```
0700:0149

MOV AX, [SI] + 08h
ADD AX, [SI] + 014h
JNB 0013Eh
INC CL
MOV [SI] + 08h, AX
MOV AX, [SI] + 016h
ADD AX, CX
MOV [SI] + 00h, AX
HLT
RET
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

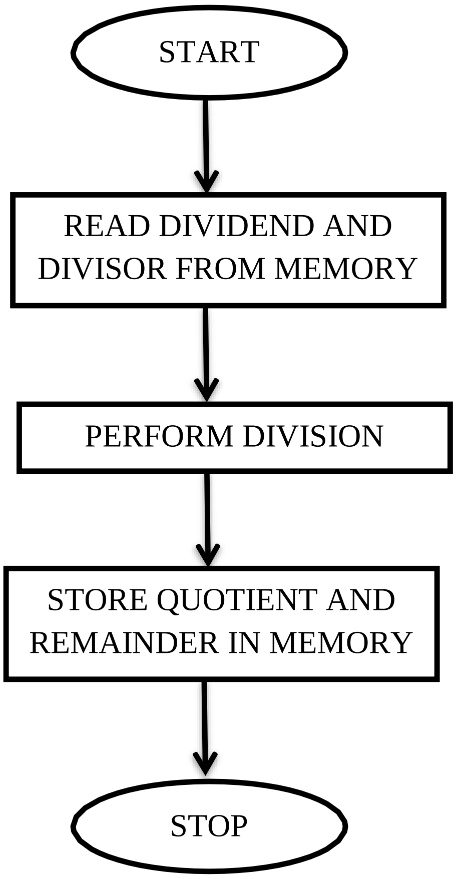
SAMPLE INPUT AND OUTPUT

MEMORY LOCATION							
1500 : 1000	32	74	63	15	34	76	Input
1500 : 1006	28	A6	C2	2F	E0	09	Output

RESULT

Thus the ALP program for multiplication of 32 and 16 bit number were developed and output is verified using emu 8086.

FLOWCHART


START
READ DIVIDEND AND DIVISOR FROM MEMORY
PERFORM DIVISION
STORE QUOTIENT AND REMAINDER IN MEMORY
STOP

EX NO 6 DIVISION OF 16 BIT NUMBER BY 8 BIT NUMBER

AIM

To develop an Assembly language program for division of 16 by 8 bit number.

ALGORITHM

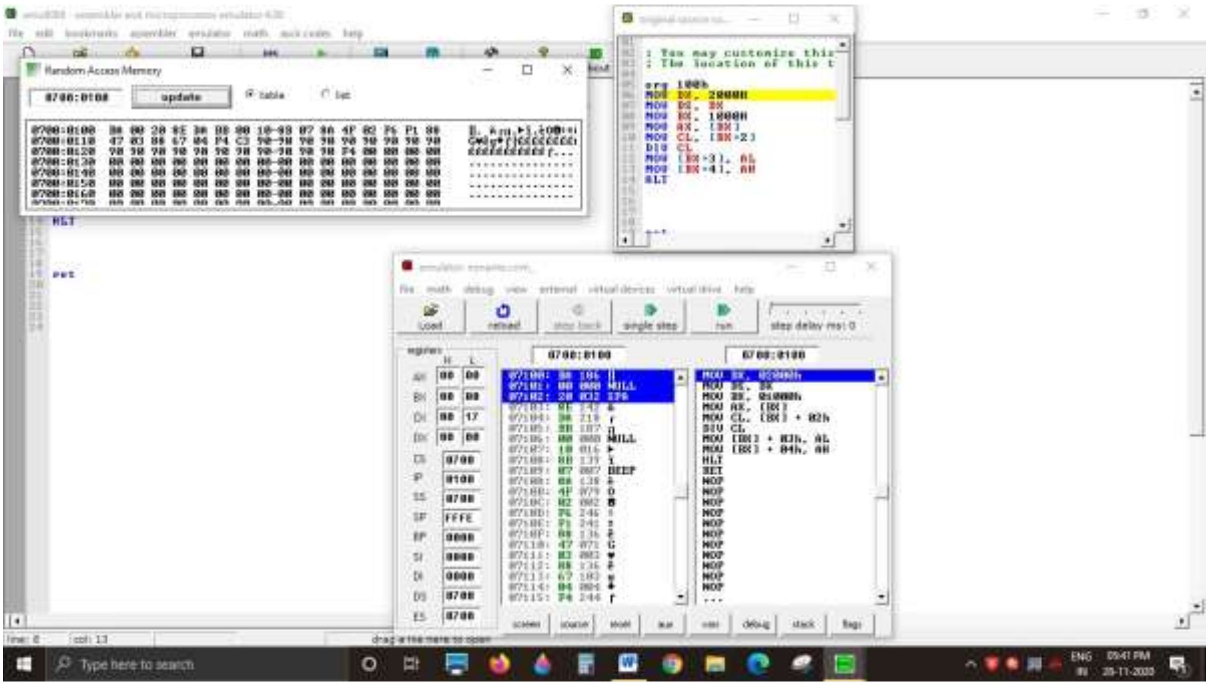
Step 1: Start
Step 2: Read dividend (16 bit) and divisor (8 bit) from memory
Step 3: Perform division
Step 4: Store quotient and remainder in the memory
Step 5: Stop

PROGRAM

```
MOV DX, 2000H
MOV DS, DX
MOV BX, 1000H
MOV AX, [BX]
MOV CL, [BX+2]
DIV CL
MOV [BX+3], AL
MOV [BX+4], AH
HLT
```

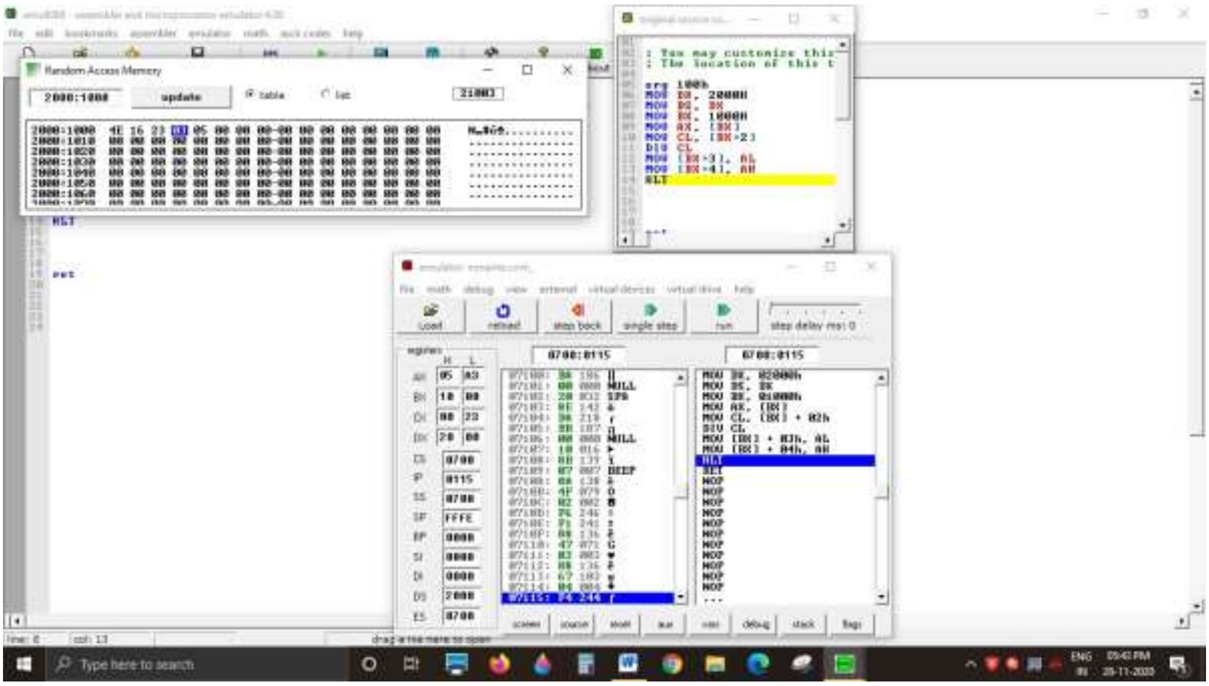
SCREENSHOTS:

BEFORE EXECUTION:



```
; You may customize this
; The location of this t

org 100h
MOV DX, 2000H
MOV DS, DX
MOV BX, 1000H
MOV AX, [BX]
MOV CL, [BX+2]
DIV CL
MOV [BX+3], AL
MOV [BX+4], AH
HLT
```

AFTER EXECUTION:



```
2000:1000   4E 16 23 A3 05 00 00 00-00 00 00 00 00 00 00 00
```

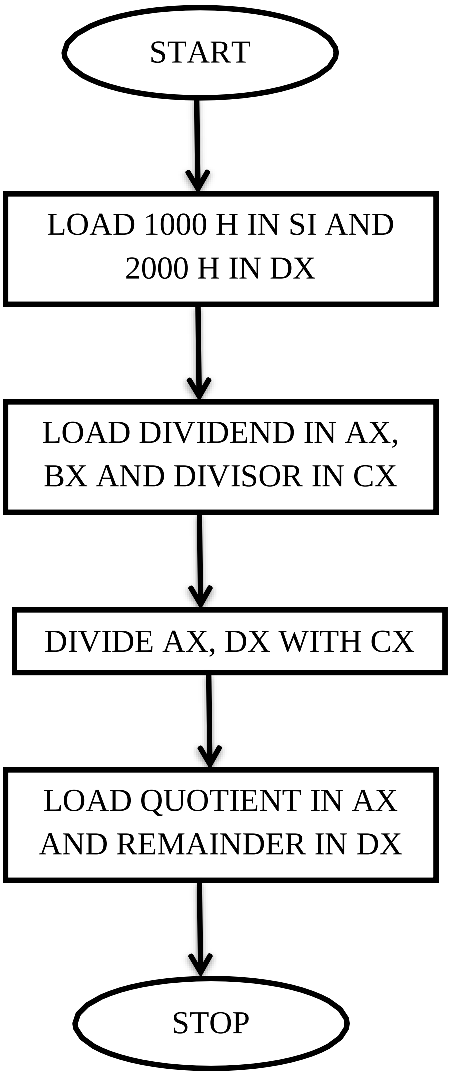
SAMPLE INPUT AND OUTPUT

MEMORY LOCATION					
2000 : 1000	4E	16	23		Input
	A3	05			Output

RESULT:

Thus the ALP program for division of 16 by 8 bit number is developed and output is verified using emu 8086.

FLOWCHART


START
LOAD 1000 H IN SI AND 2000 H IN DX
LOAD DIVIDEND IN AX, BX AND DIVISOR IN CX
DIVIDE AX, DX WITH CX
LOAD QUOTIENT IN AX AND REMAINDER IN DX
STOP

EX NO 7 DIVISION OF 32 BIT NUMBER BY 16 BIT NUMBER

AIM

To develop an Assembly language program for division of 32 by 16 bit number.

ALGORITHM

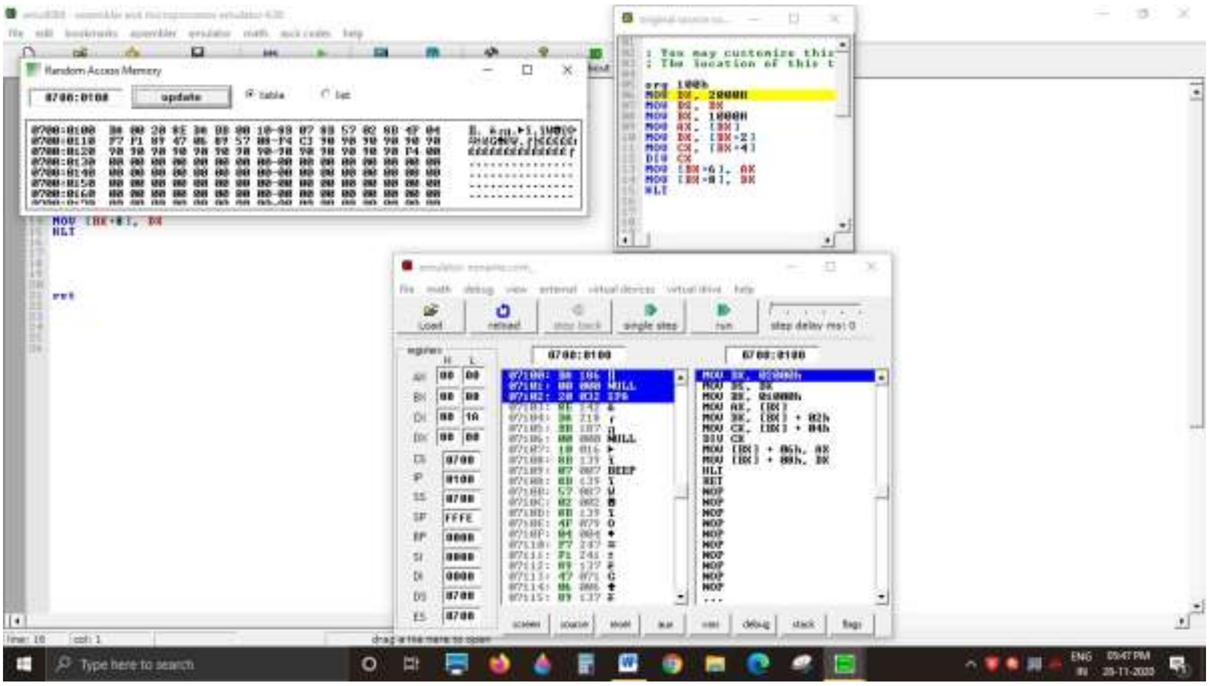
Step 1: Start
Step 2: Read dividend (32 bit) and divisor (16 bit) from memory
Step 3: Perform division
Step 4: Store quotient and remainder in the memory
Step 5: Stop

PROGRAM

```
MOV DX, 2000H
MOV DS, DX
MOV BX, 1000H
MOV AX, [BX]
MOV DX, [BX+2]
MOV CX, [BX+4]
DIV CX
MOV [BX+6], AX
MOV [BX+8], DX
HLT
```

SCREENSHOTS:

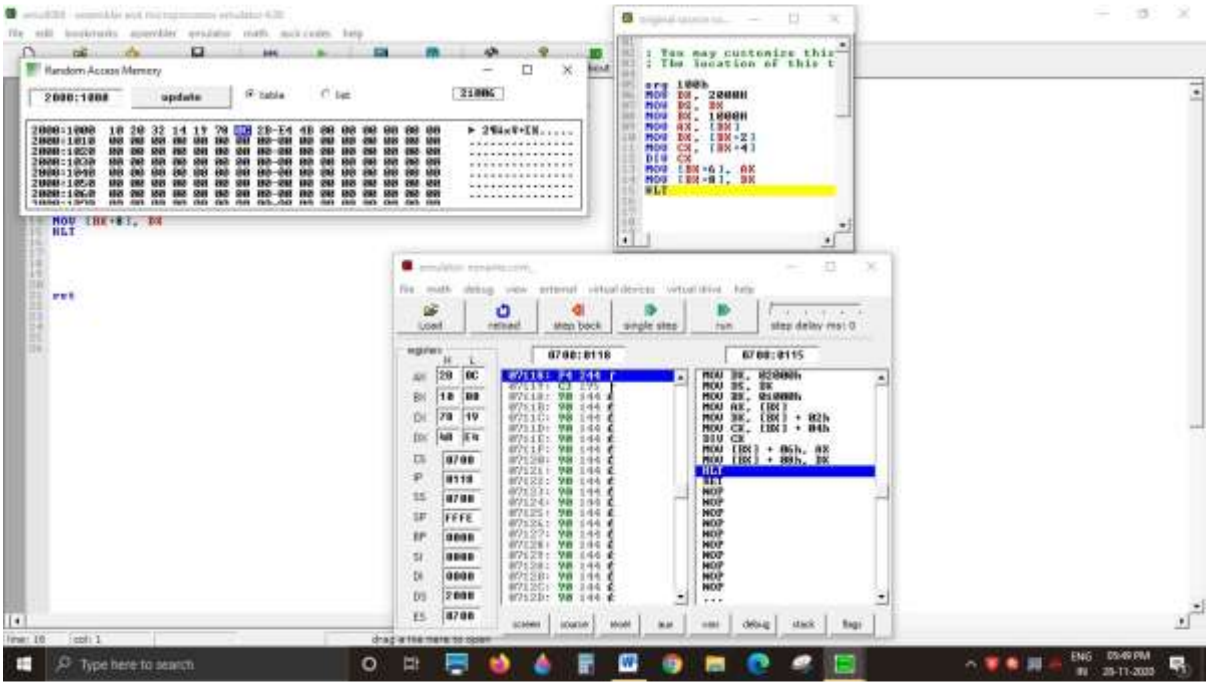
BEFORE EXECUTION:



```
; You may customize this
; The location of this t

org 100h
MOV BX, 2000H
MOV DS, BX
MOV BX, 1000H
MOV AX, [BX]
MOV DX, [BX+2]
MOV CX, [BX+4]
DIV CX
MOV [BX+6], AX
MOV [BX+8], DX
HLT
```

```
0700:0100
MOV BX, 02000h
MOV DS, BX
MOV BX, 01000h
MOV AX, [BX]
MOV DX, [BX] + 02h
MOV CX, [BX] + 04h
DIV CX
MOV [BX] + 06h, AX
MOV [BX] + 08h, DX
HLT
RET
NOP
```

AFTER EXECUTION:



```
2000:1000  10 20 32 14 19 78 0C 2B-E4 4B 00 00 00 00 00 00
```

```
AX 2B 0C
BX 10 00
CX 78 19
DX 4B E4
CS 0700
IP 0118
SS 0700
SP FFFE
BP 0000
SI 0000
DI 0000
DS 2000
ES 0700
```

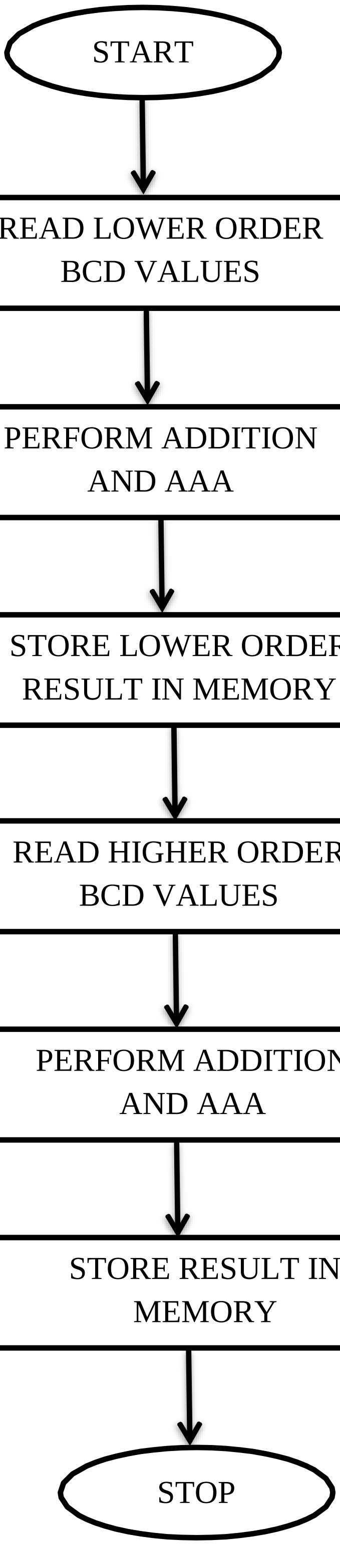
SAMPLE INPUT AND OUTPUT:

MEMORY LOCATION							
2000 : 1000	10	20	32	14	19	78	Input
2000 : 1006	0C	2B	E4	4B			Output

RESULT:

Thus the ALP program for division of 32 by 16 bit number is developed and output is verified using emu 8086.

FLOWCHART


START
READ LOWER ORDER BCD VALUES
PERFORM ADDITION AND AAA
STORE LOWER ORDER RESULT IN MEMORY
READ HIGHER ORDER BCD VALUES
PERFORM ADDITION AND AAA
STORE RESULT IN MEMORY
STOP

EX NO 8 **UNPACKED BCD ADDITION**

AIM

To develop an Assembly language program for unpacked BCD addition.

ALGORITHM

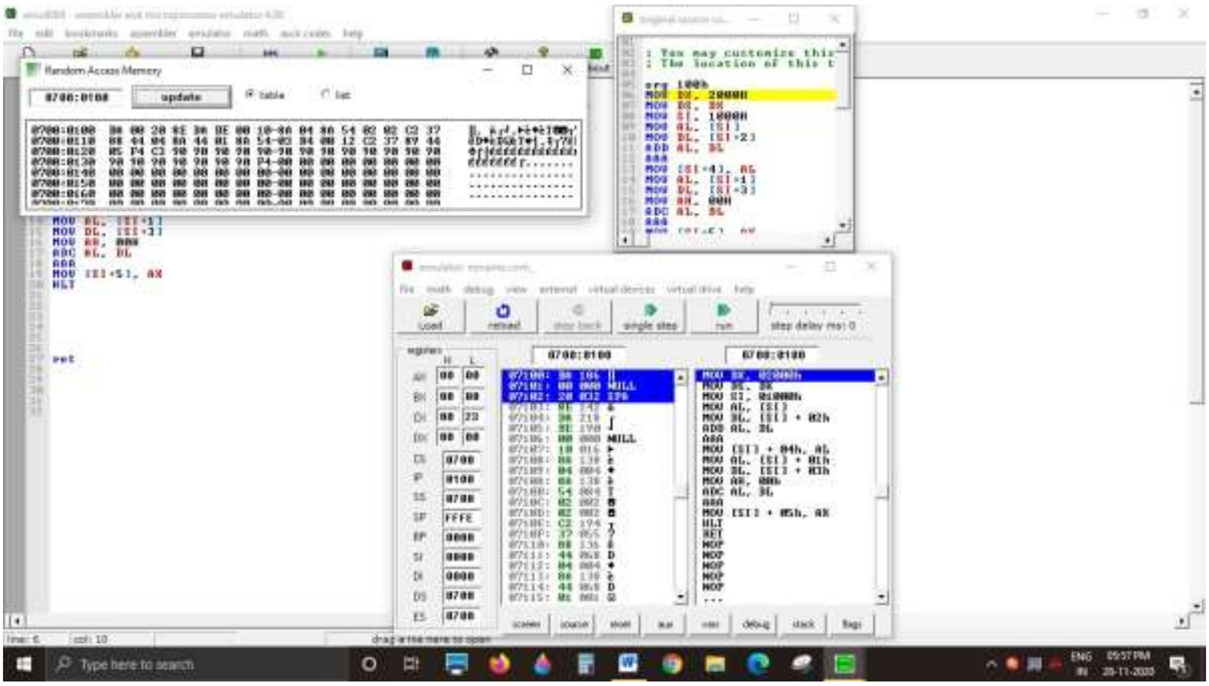
```
Step 1: Start
Step 2: Read lower order BCD values of two numbers
Step 3: Perform addition and ASCII Adjust After Addition
Step 4: Store the lower order in memory
Step 5: Read the higher order BCD values of two numbers
Step 6: Perform addition with higher order of above result and perform ASCII AAA
Step 7: Store the result in memory
Step 8: Stop
```

PROGRAM

```
MOV DX, 2000H
MOV DS, DX
MOV SI, 1000H
MOV AL, [SI]
MOV DL, [SI+2]
ADD AL, DL
AAA
MOV [SI+4], AL
MOV AL, [SI+1]
MOV DL, [SI+3]
MOV AH, 00H
ADC AL, DL
AAA
MOV [SI+5], AX
HLT
```

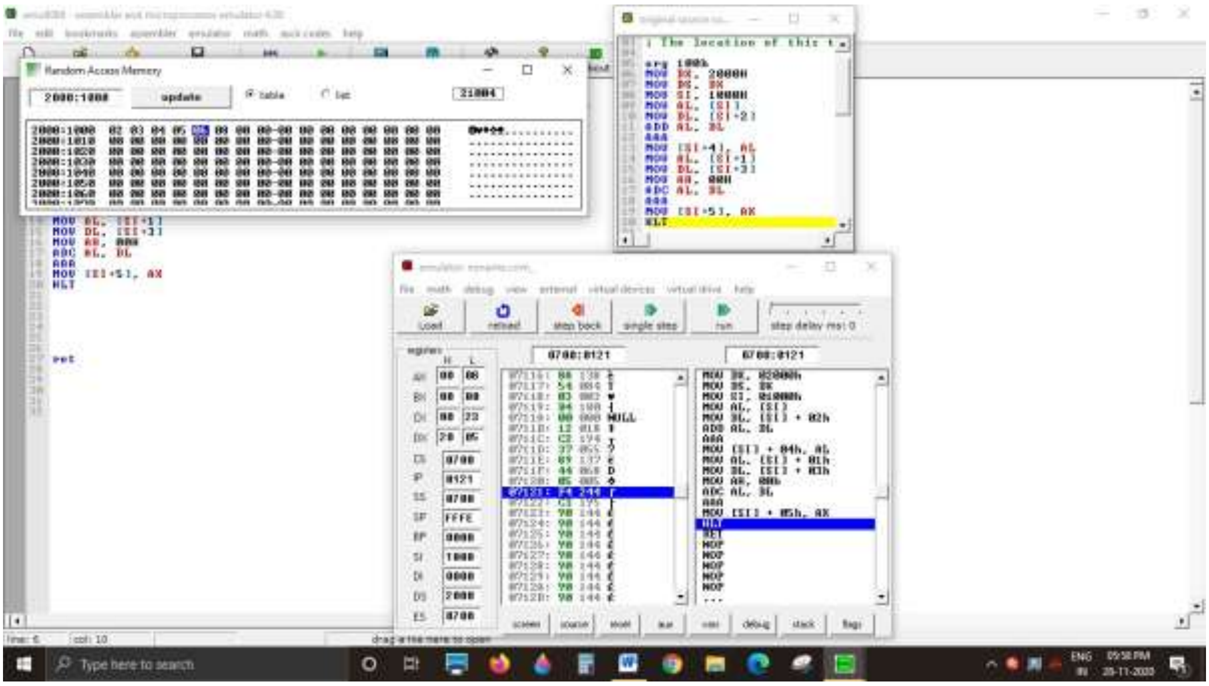
SCREENSHOTS:

BEFORE EXECUTION:



```
; You may customize this
; The location of this t
org 100h
MOV BX, 2000H
MOV DS, BX
MOV SI, 1000H
MOV AL, [SI]
MOV BL, [SI+2]
ADD AL, BL
AAA
MOV [SI+4], AL
MOV AL, [SI+1]
MOV DL, [SI+3]
MOV AH, 00H
ADC AL, DL
AAA
MOV [SI+5], AX
HLT

ret
```

AFTER EXECUTION:



```
2000:1000  02 03 04 05 06 08 00 00-00 00 00 00 00 00 00 00
```

```
; The location of this t
org 100h
MOV BX, 2000H
MOV DS, BX
MOV SI, 1000H
MOV AL, [SI]
MOV BL, [SI+2]
ADD AL, BL
AAA
MOV [SI+4], AL
MOV AL, [SI+1]
MOV DL, [SI+3]
MOV AH, 00H
ADC AL, DL
AAA
MOV [SI+5], AX
HLT
```

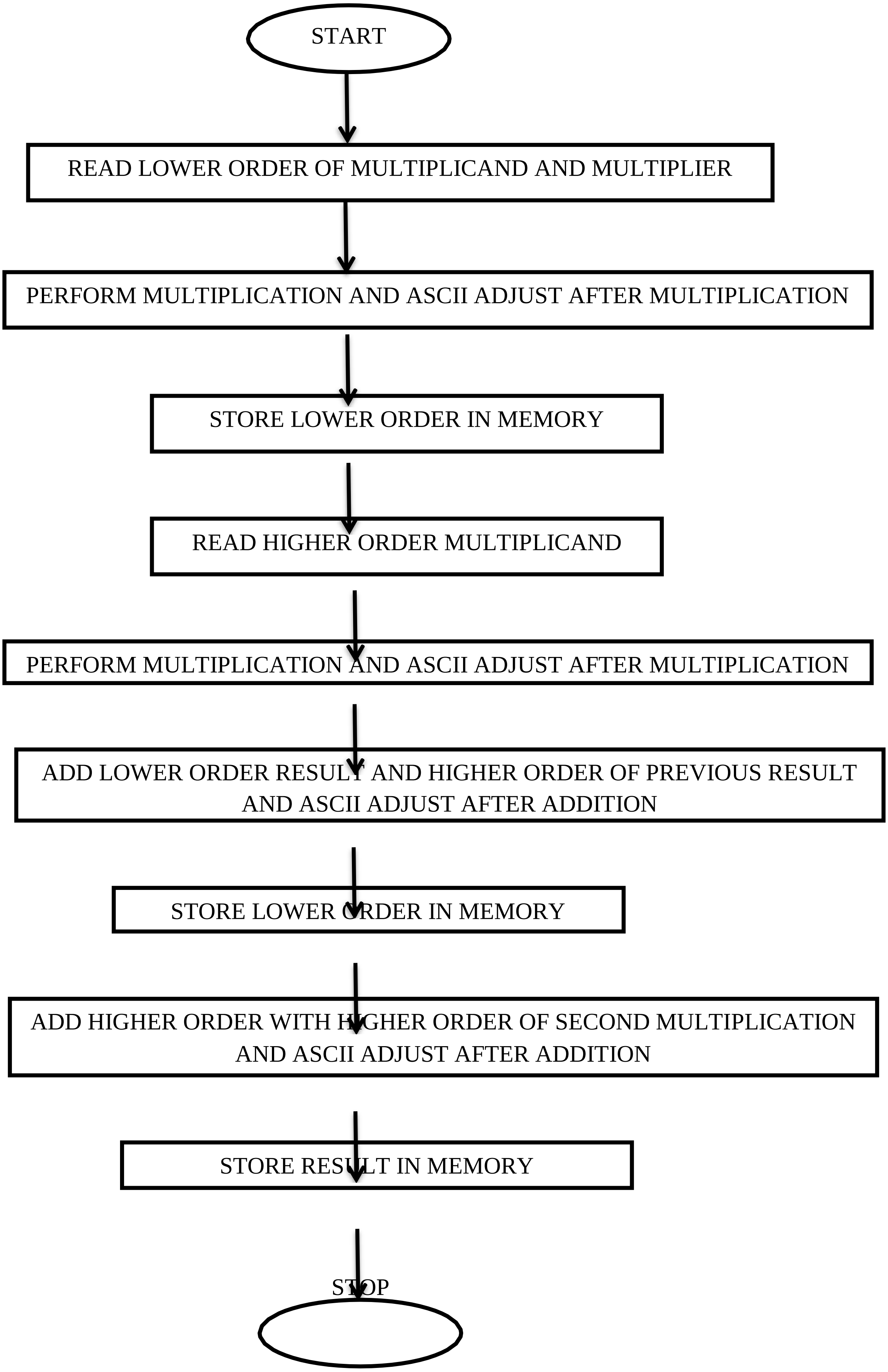
SAMPLE INPUT AND OUTPUT:

MEMORY LOCATION					
2000 : 1000	02	03	04	05	Input
2000 : 1004	06	08			Output

RESULT

Thus the ALP program for unpacked BCD addition is developed and output is verified using emu 8086.

FLOWCHART


START
READ LOWER ORDER OF MULTIPLICAND AND MULTIPLIER
PERFORM MULTIPLICATION AND ASCII ADJUST AFTER MULTIPLICATION
STORE LOWER ORDER IN MEMORY
READ HIGHER ORDER MULTIPLICAND
PERFORM MULTIPLICATION AND ASCII ADJUST AFTER MULTIPLICATION
ADD LOWER ORDER RESULT AND HIGHER ORDER OF PREVIOUS RESULT AND ASCII ADJUST AFTER ADDITION
STORE LOWER ORDER IN MEMORY
ADD HIGHER ORDER WITH HIGHER ORDER OF SECOND MULTIPLICATION AND ASCII ADJUST AFTER ADDITION
STORE RESULT IN MEMORY
STOP

EXP NO 9 **UNPACKED BCD MULTIPLICATION**

AIM

To develop an Assembly language program for unpacked BCD multiplication.

ALGORITHM

Step 1: Start
Step 2: Read lower order of multiplicand and multiplier
Step 3: Perform multiplication and AAM
Step 4: Store lower order of result in memory
Step 5: Read higher order of multiplicand
Step 6: Perform multiplication and AAM
Step 7: Add lower order of this result and higher order of previous result and perform AAA
Step 8: Store lower order in memory
Step 9: Add higher order with higher order of result of second multiplication and AAA
Step 10: Store the result in memory
Step 11: Stop

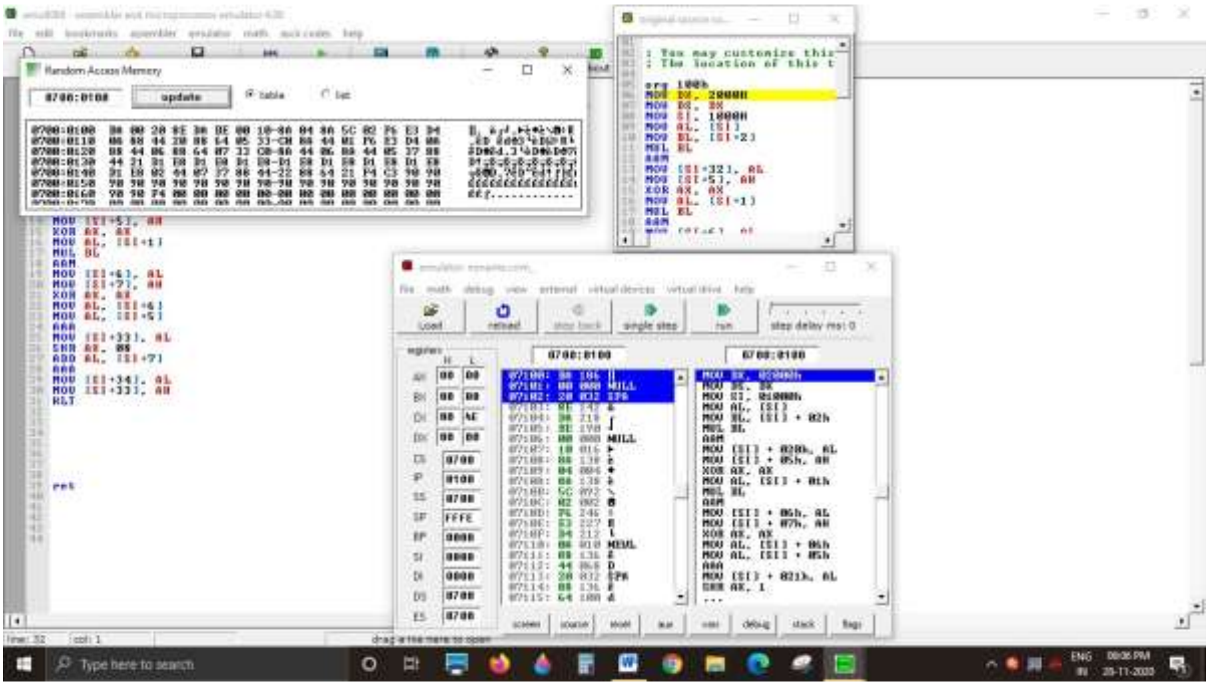
PROGRAM

```
MOV DX, 2000H
MOV DS, DX
MOV SI, 1000H
MOV AL, [SI]
MOV BL, [SI+2]
MUL BL
AAM
MOV [SI+32], AL
MOV [SI+5], AH
XOR AX, AX
MOV AL, [SI+1]
MUL BL
AAM
MOV [SI+6], AL
MOV [SI+7], AH
XOR AX, AX
MOV AL, [SI+6]
MOV AL, [SI+5]
AAA
MOV [SI+33], AL
SHR AX,  08
ADD AL, [SI+7]
```

```
AAA
MOV [SI+34], AL
MOV [SI+33], AH
HLT
```

SCREENSHOTS:

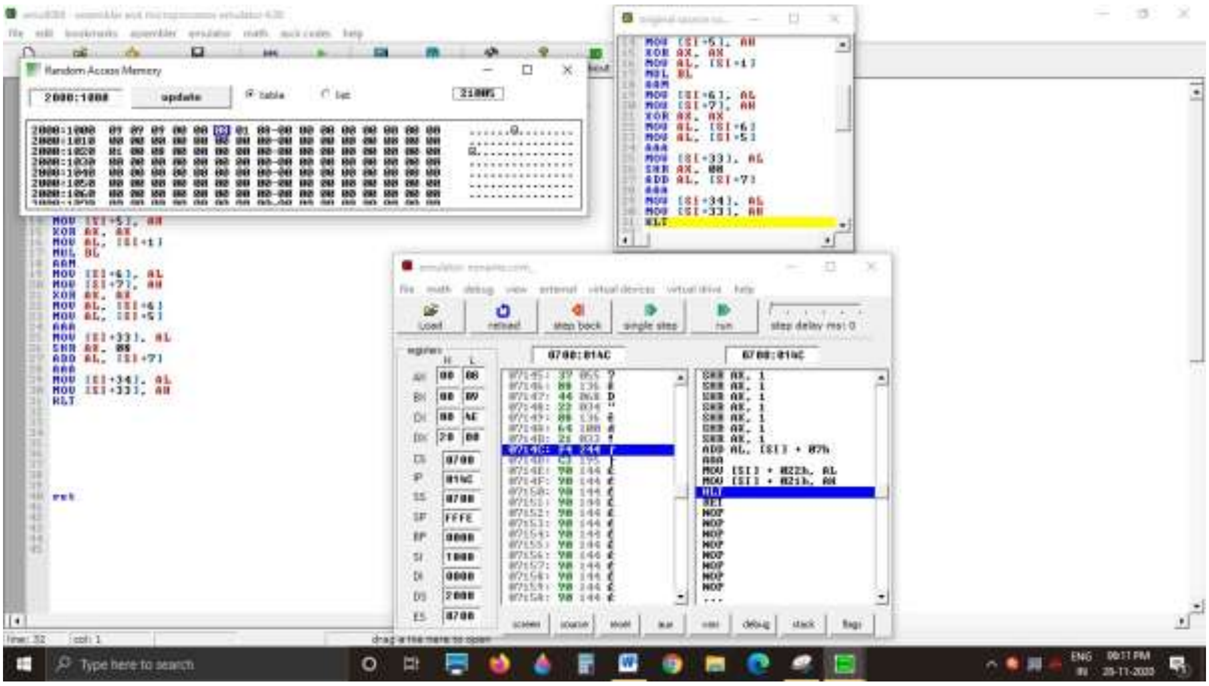
BEFORE EXECUTION:



```
; You may customize this
; The location of this t

org 100h
MOV DX, 2000H
MOV DS, DX
MOV SI, 1000H
MOV AL, [SI]
MOV BL, [SI+2]
MUL BL
AAM
MOV [SI+32], AL
MOV [SI+5], AH
XOR AX, AX
MOV AL, [SI+1]
MUL BL
AAM
```

```
AX 00 00
BX 00 00
CX 00 4E
DX 00 00
CS 0700
IP 0100
SS 0700
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
```

AFTER EXECUTION:



```
MOV [SI+5], AH
XOR AX, AX
MOV AL, [SI+1]
MUL BL
AAM
MOV [SI+6], AL
MOV [SI+7], AH
XOR AX, AX
MOV AL, [SI+6]
MOV AL, [SI+5]
AAA
MOV [SI+33], AL
SHR AX, 08
ADD AL, [SI+7]
AAA
MOV [SI+34], AL
MOV [SI+33], AH
HLT
```

```
AX 00 08
BX 00 09
CX 00 4E
DX 20 00
CS 0700
IP 014C
SS 0700
SP FFFE
BP 0000
SI 1000
DI 0000
DS 2000
ES 0700
```

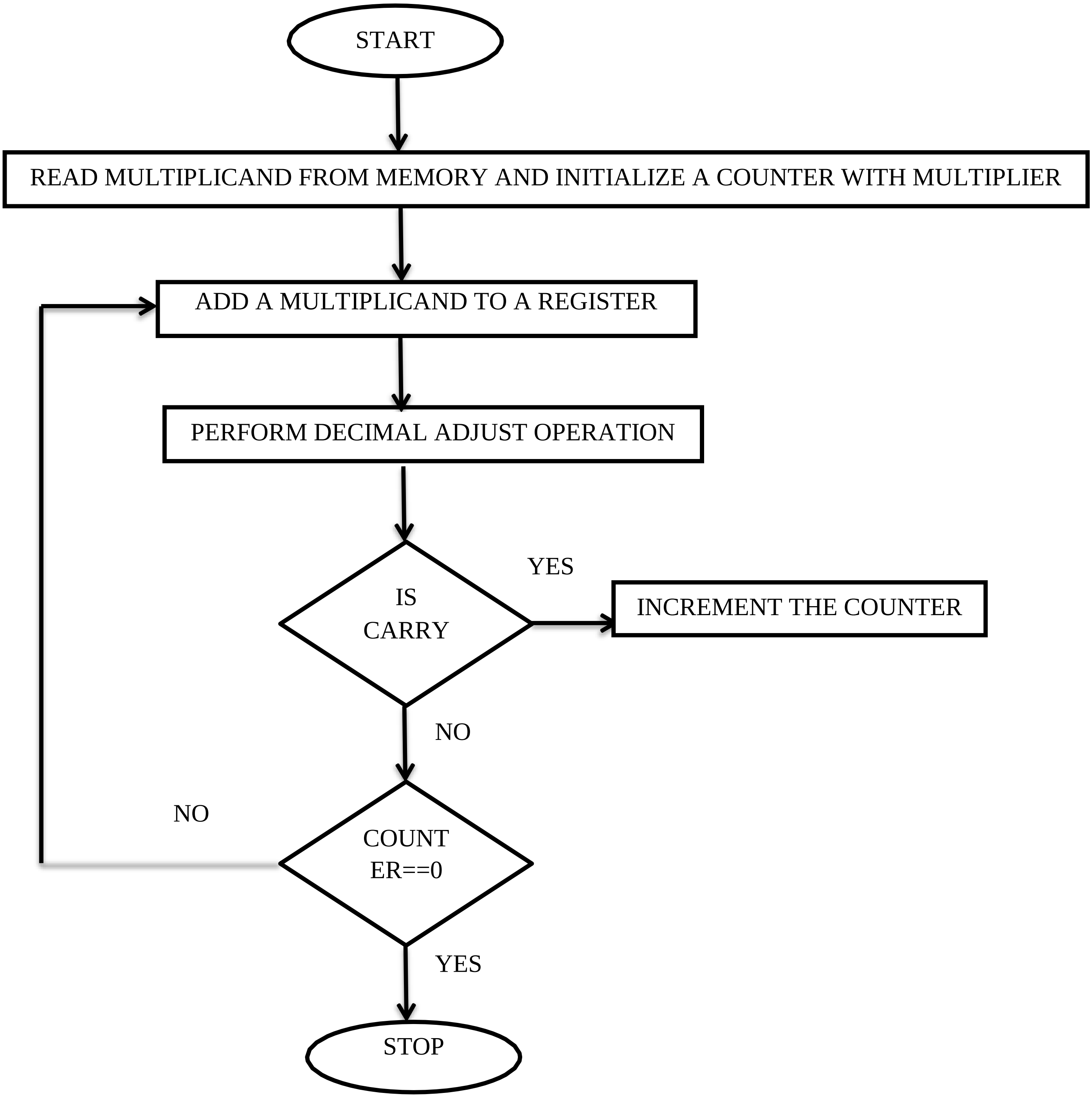
SAMPLE INPUT AND OUTPUT:

MEMORY LOCATION					
2000 : 1000	09	09	09		Input
2000 : 1003	08	01	08		Output

RESULT

Thus the ALP program for unpacked BCD multiplication is developed and output is verified using emu 8086.

FLOWCHART


START
READ MULTIPLICAND FROM MEMORY AND INITIALIZE A COUNTER WITH MULTIPLIER
ADD A MULTIPLICAND TO A REGISTER
PERFORM DECIMAL ADJUST OPERATION
IS CARRY
YES
INCREMENT THE COUNTER
NO
NO
COUNT ER==0
YES
STOP

EXP NO 10 **PACKED BCD MULTIPLICATION**

AIM

To develop an Assembly language program for packed BCD multiplication.

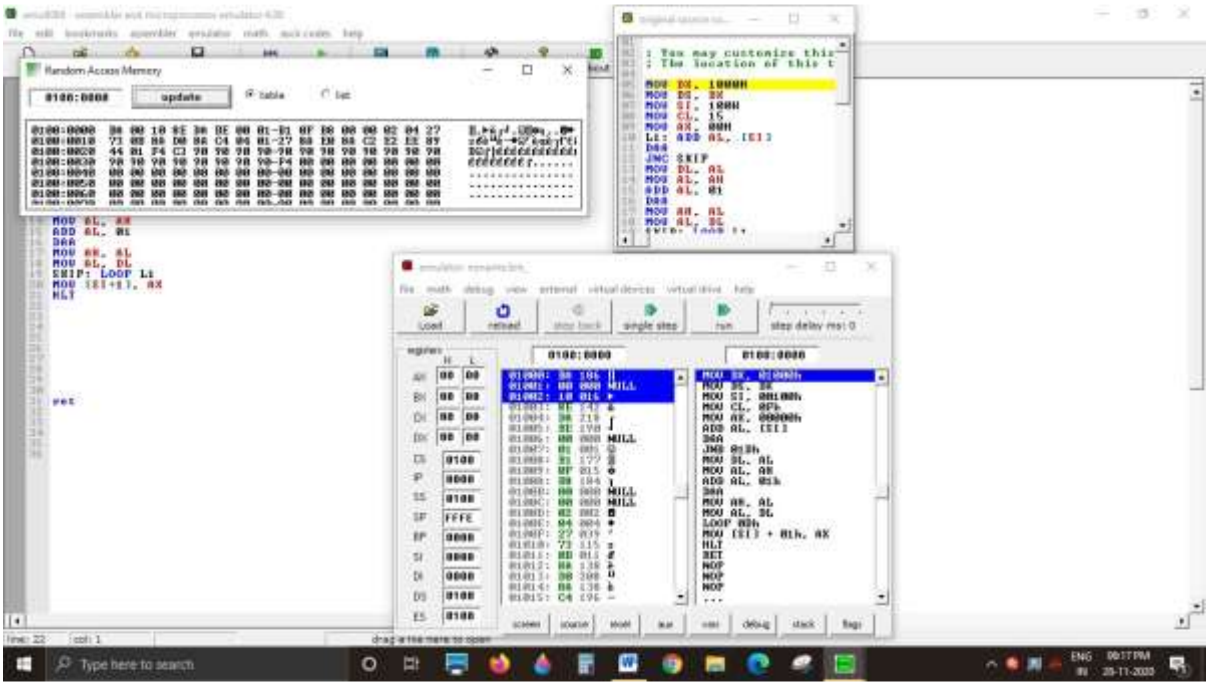
ALGORITHM

Step 1: Start
Step 2: Initialize the memory and read multiplicand
Step 3: Load multiplier into register
Step 4: Add the same number repeatedly for multiplier number of times using loop
Step 5: Store the result in memory with carry
Step 6: Stop

PROGRAM

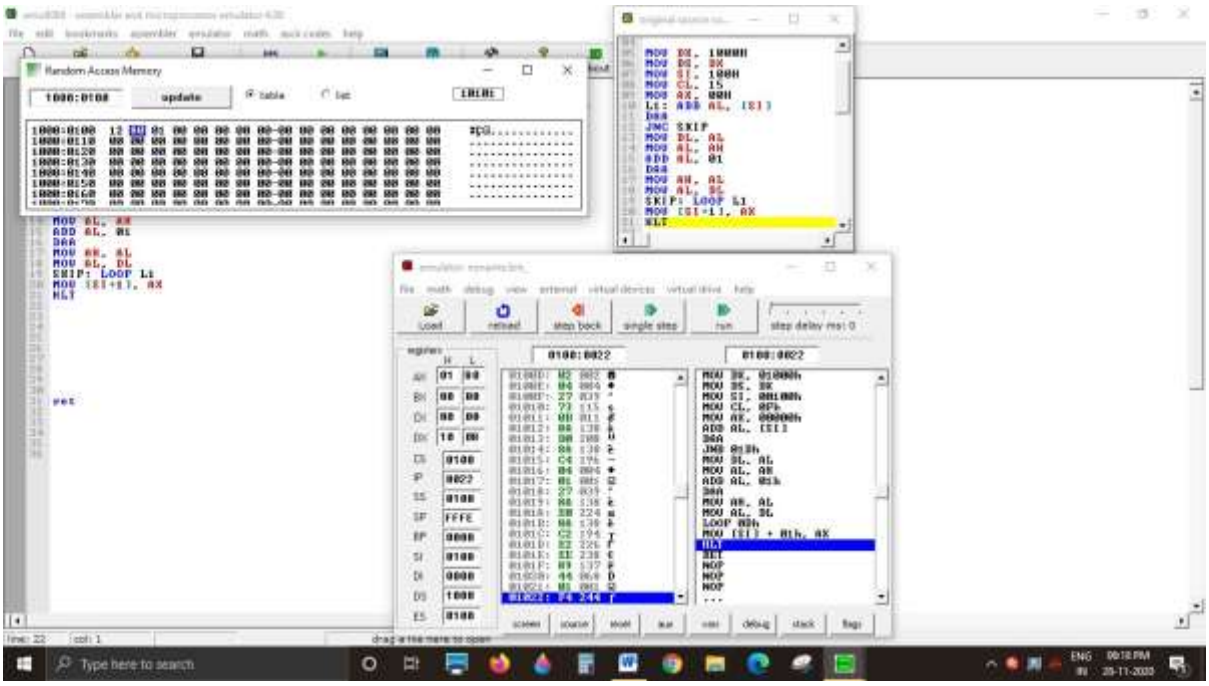
```
MOV DX, 1000H
MOV DS, DX
MOV SI, 100H
MOV CL, 15
MOV AX, 00H
L1: ADD AL, [SI]
    DAA
    JNC SKIP
MOV DL, AL
MOV AL, AH
ADD AL, 01
DAA
MOV AH, AL
MOV AL, DL
SKIP: LOOP L1
MOV [SI+1], AX
HLT
```

SCREENSHOTS:
BEFORE EXECUTION:



```
; You may customize this
; The location of this t

MOV DX, 1000H
MOV DS, DX
MOV SI, 100H
MOV CL, 15
MOV AX, 00H
L1: ADD AL, [SI]
DAA
JNC SKIP
MOV DL, AL
MOV AL, AH
ADD AL, 01
DAA
MOV AH, AL
MOV AL, DL
SKIP: LOOP L1
```

AFTER EXECUTION:



```
MOV DX, 1000H
MOV DS, DX
MOV SI, 100H
MOV CL, 15
MOV AX, 00H
L1: ADD AL, [SI]
DAA
JNC SKIP
MOV DL, AL
MOV AL, AH
ADD AL, 01
DAA
MOV AH, AL
MOV AL, DL
SKIP: LOOP L1
MOV [SI+1], AX
HLT
```

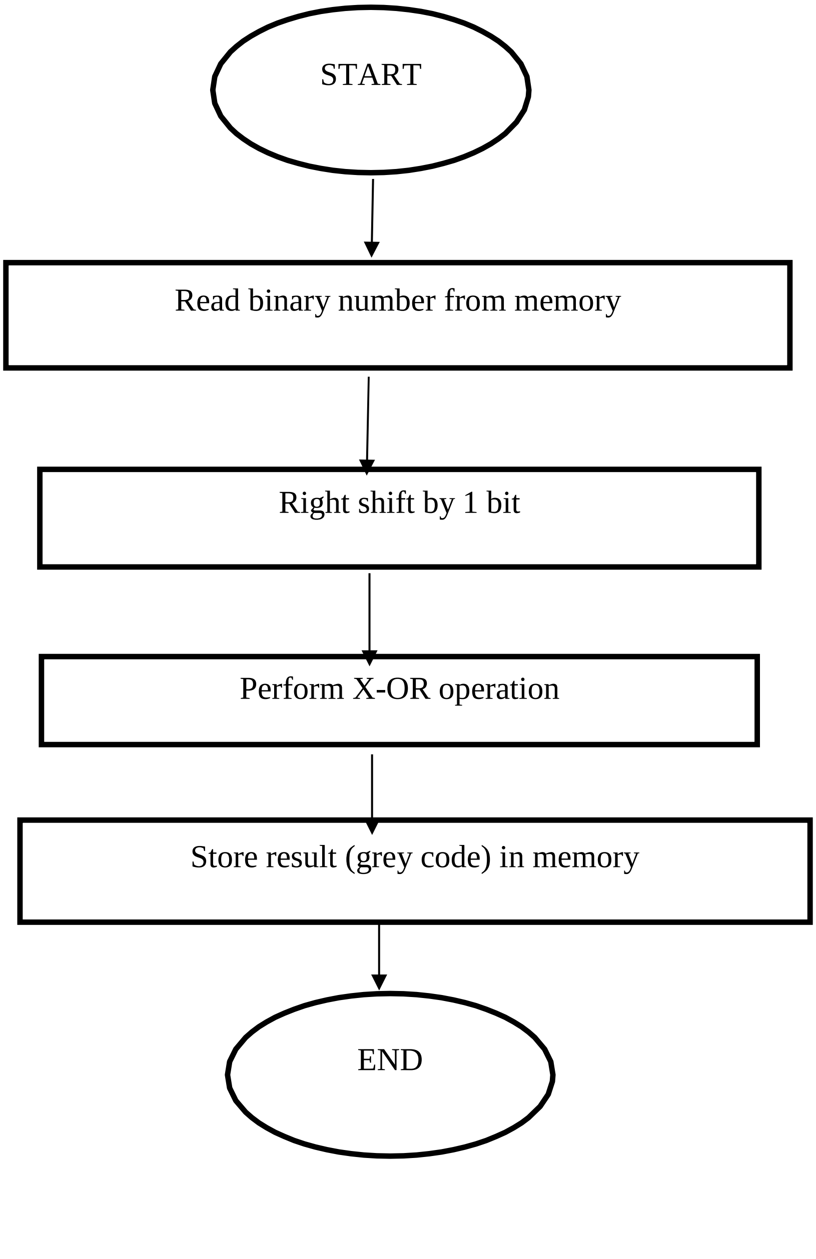
SAMPLE INPUT AND OUTPUT:

MEMORY LOCATION					
1000 : 0100	12				Input
1000 : 0101	80	01			Output

RESULT

Thus the ALP program for unpacked BCD multiplication is developed and output is verified using emu 8086.

FLOWCHART:


START
Read binary number from memory
Right shift by 1 bit
Perform X-OR operation
Store result (grey code) in memory
END

EX NO 11

BINARY TO GRAY CODE CONVERSION

AIM

To develop an Assembly language program to convert binary to gray code.

ALGORITHM

STEP 1: Start

STEP 2: Read the binary (in hexadecimal form) as input.

STEP 3: Shift the binary number right by 1 bit.

STEP 4: Then perform the exclusive OR function with the original binary number.

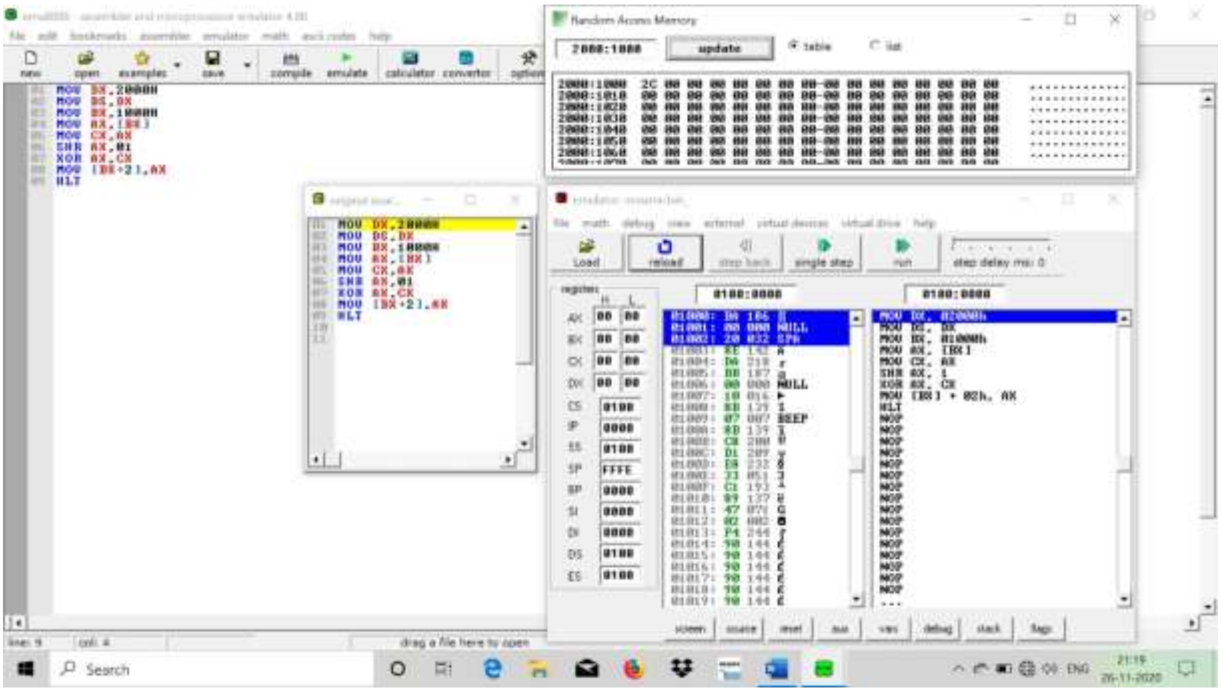
STEP 5: Store the result (gray code) in memory.

STEP 6: End

PROGRAM

```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV [BX+2],AX
HLT
```

BEFORE EXECUTION:



```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV [BX+2],AX
HLT
```

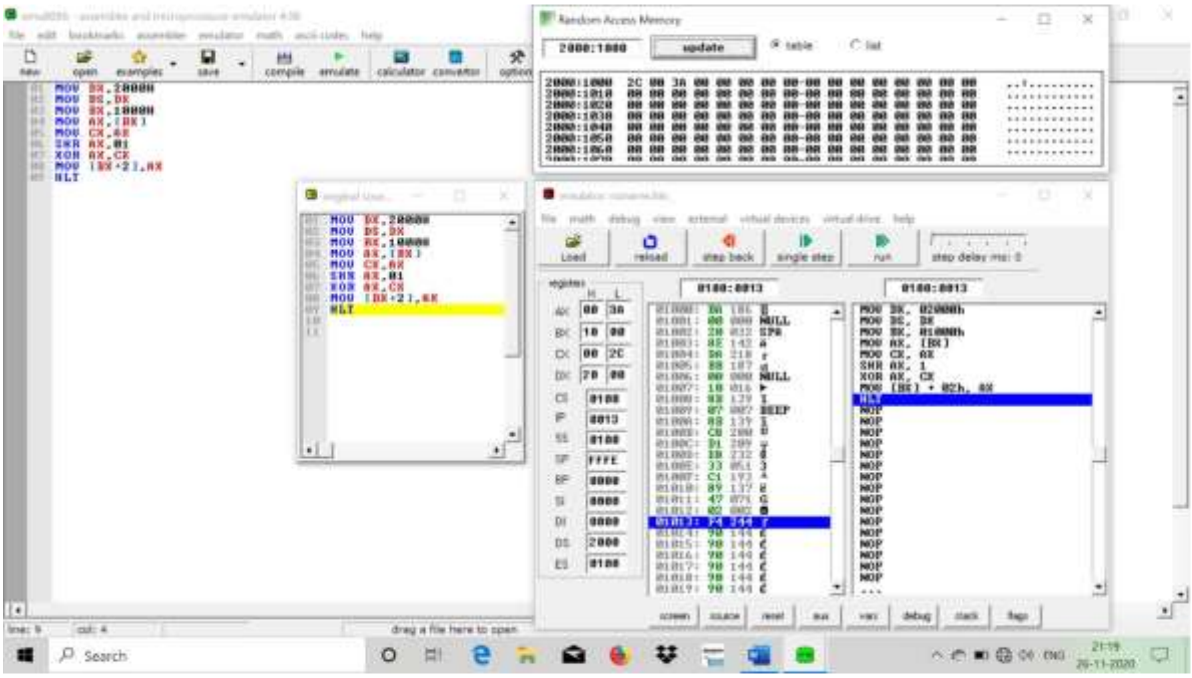
```
2000:1000
2000:1000  2C 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
registers
     H   L
AX   00  00
BX   00  00
CX   00  00
DX   00  00
CS   0100
IP   0000
SS   0100
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0100
ES   0100
```

```
0100:0000

01000: BA 186
01001: 00 000 NULL
01002: 20 032 SPA
01003: 8E 142
01004: DA 218
01005: BB 187
01006: 00 000 NULL
01007: 10 016
01008: 8B 139
01009: 07 007 BEEP
0100A: 8B 139
0100B: C8 200
0100C: D1 209
0100D: E8 232
0100E: 33 051 3
0100F: C1 193
01010: 89 137
01011: 47 071 G
01012: 02 002
01013: F4 244
01014: 90 144
01015: 90 144
01016: 90 144
01017: 90 144
01018: 90 144
01019: 90 144
```

```
0100:0000

MOV DX, 02000h
MOV DS, DX
MOV BX, 01000h
MOV AX, [BX]
MOV CX, AX
SHR AX, 1
XOR AX, CX
MOV [BX] + 02h, AX
HLT
NOP
...
```

AFTER EXECUTION:



```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV [BX+2],AX
HLT
```

```
2000:1000
2000:1000  2C 00 3A 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
registers
     H   L
AX   00  3A
BX   10  00
CX   00  2C
DX   20  00
CS   0100
IP   0013
SS   0100
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   2000
ES   0100
```

```
0100:0013

01000: BA 186
01001: 00 000 NULL
01002: 20 032 SPA
01003: 8E 142
01004: DA 218
01005: BB 187
01006: 00 000 NULL
01007: 10 016
01008: 8B 139
01009: 07 007 BEEP
0100A: 8B 139
0100B: C8 200
0100C: D1 209
0100D: E8 232
0100E: 33 051 3
0100F: C1 193
01010: 89 137
01011: 47 071 G
01012: 02 002
01013: F4 244
01014: 90 144
01015: 90 144
01016: 90 144
01017: 90 144
01018: 90 144
01019: 90 144
```

```
0100:0013

MOV DX, 02000h
MOV DS, DX
MOV BX, 01000h
MOV AX, [BX]
MOV CX, AX
SHR AX, 1
XOR AX, CX
MOV [BX] + 02h, AX
HLT
NOP
...
```

SAMPLE INPUT AND OUTPUT:

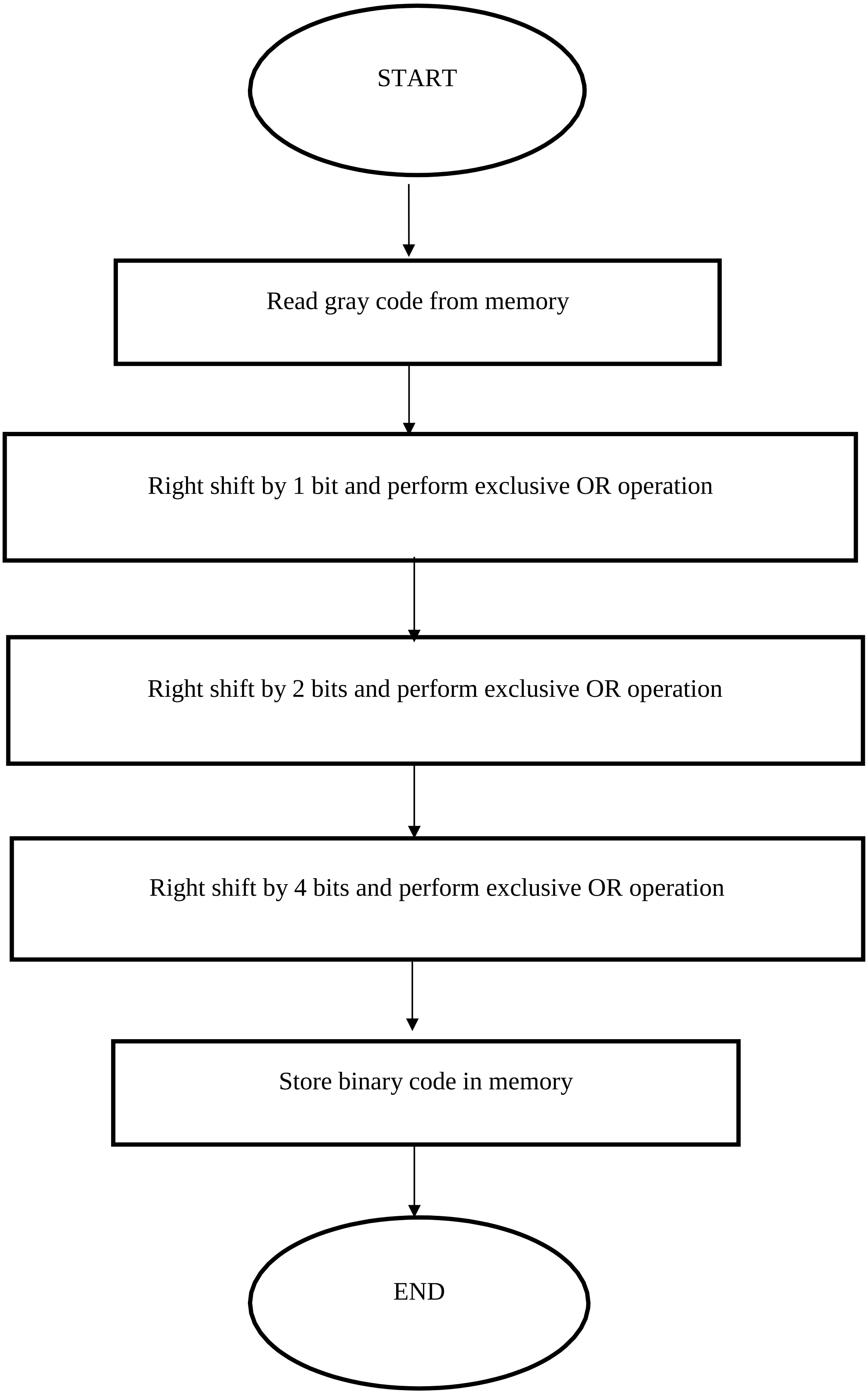
Memory location		
2000:1000	2C	Input
2000:1002	3A	Output

RESULT

Thus, the ALP program to convert binary code to gray code were developed and the output is verified using emu 8086.

GRAY TO BINARY CODE CONVERSION

FLOWCHART


START
Read gray code from memory
Right shift by 1 bit and perform exclusive OR operation
Right shift by 2 bits and perform exclusive OR operation
Right shift by 4 bits and perform exclusive OR operation
Store binary code in memory
END

EX NO 12

GRAY TO BINARY CODE CONVERSION

AIM

To develop an Assembly language program to convert gray code to binary code.

ALGORITHM

STEP 1: Start

STEP 2: Read the code gray code stored in memory as hexadecimal as input.

STEP 3: Right shift by 1 bit and perform exclusive OR operation.

STEP 4: Then again, right shift by 2 bits and perform exclusive OR operation.

STEP 5: And again, right shift by 4 bits and perform exclusive OR operation and binary code is generated.

STEP 6: Store the result in memory

STEP 7: End

PROGRAM

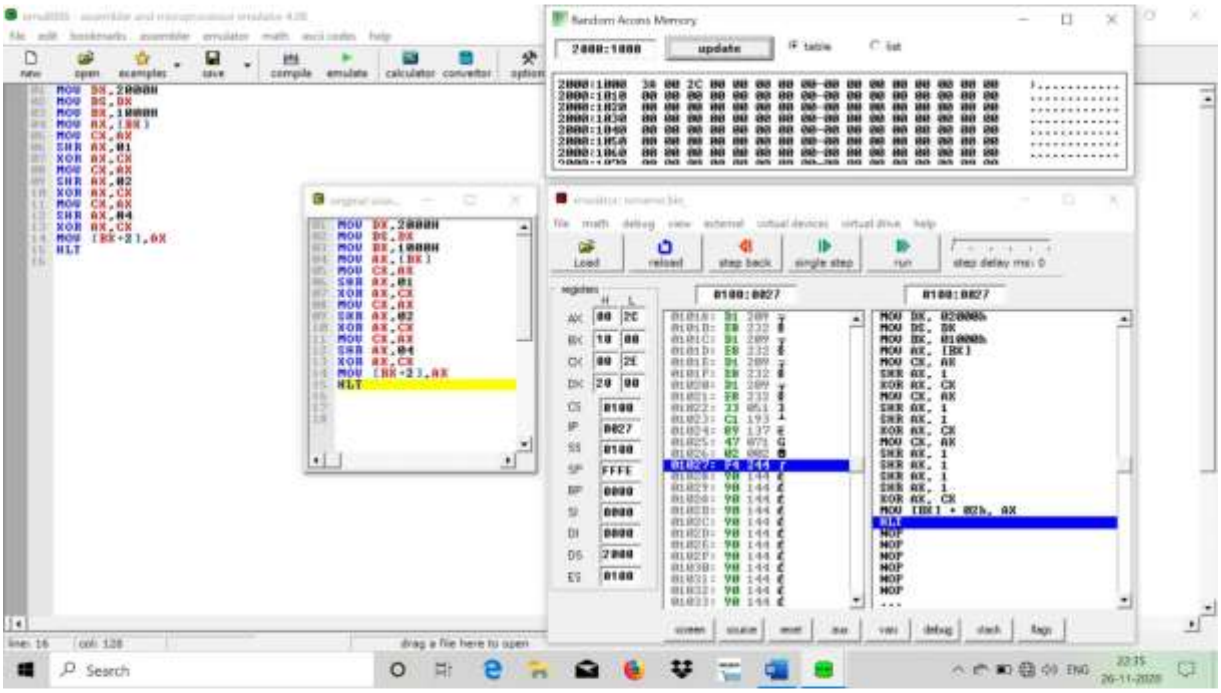
```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV CX,AX
SHR AX,02
XOR AX,CX
MOV CX,AX
SHR AX,04
XOR AX,CX
```

MOV [BX+2],AX

HLT

SCREENSHOT:

BEFORE EXECUTION:



```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV CX,AX
SHR AX,02
XOR AX,CX
MOV CX,AX
SHR AX,04
XOR AX,CX
MOV [BX+2],AX
HLT
```

Random Access Memory

2000:1000 ● table ○ list

```
2000:1000  3A 00 2C 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

registers

	H	L
AX	00	2C
BX	10	00
CX	00	2E
DX	20	00
CS	0100	
IP	0027	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	2000	
ES	0100	

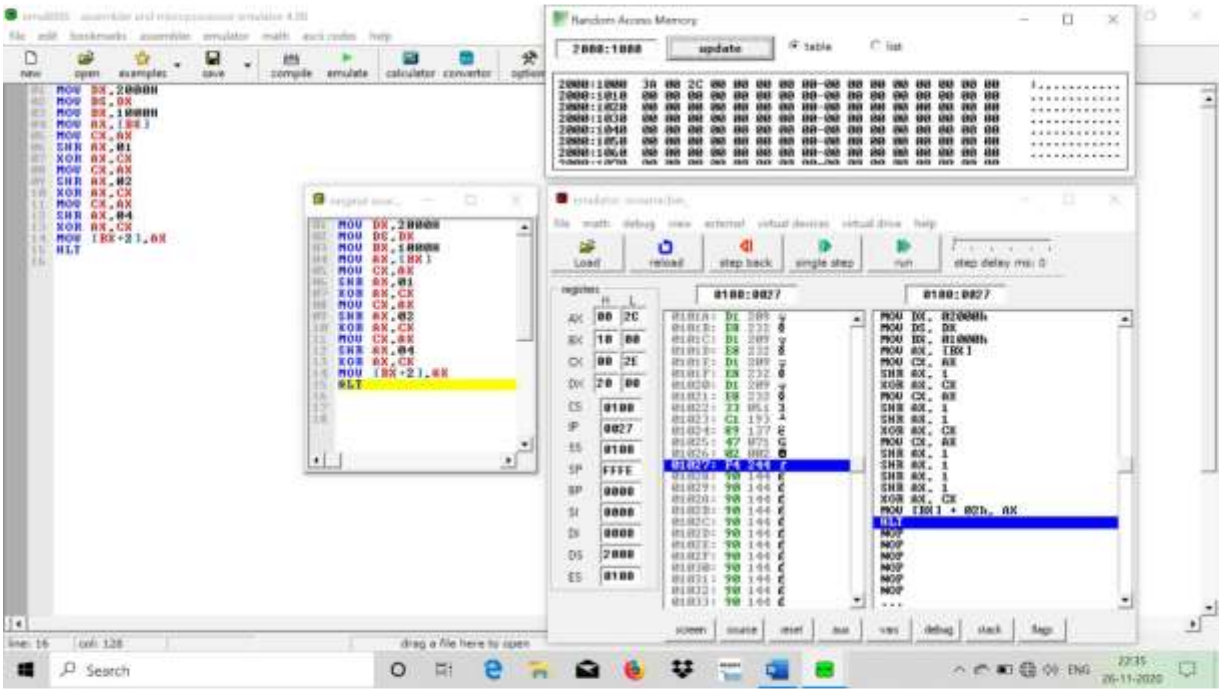
0100:0027

```
01027: F4 244 (
01028: 90 144 É
01029: 90 144 É
0102A: 90 144 É
0102B: 90 144 É
0102C: 90 144 É
0102D: 90 144 É
0102E: 90 144 É
0102F: 90 144 É
01030: 90 144 É
01031: 90 144 É
01032: 90 144 É
01033: 90 144 É
```

0100:0027

```
MOV DX, 02000h
MOV DS, DX
MOV BX, 01000h
MOV AX, [BX]
MOV CX, AX
SHR AX, 1
XOR AX, CX
MOV CX, AX
SHR AX, 1
SHR AX, 1
XOR AX, CX
MOV CX, AX
SHR AX, 1
SHR AX, 1
SHR AX, 1
SHR AX, 1
XOR AX, CX
MOV [BX] + 02h, AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
...
```

AFTER EXECUTION:



```
MOV DX,2000H
MOV DS,DX
MOV BX,1000H
MOV AX,[BX]
MOV CX,AX
SHR AX,01
XOR AX,CX
MOV CX,AX
SHR AX,02
XOR AX,CX
MOV CX,AX
SHR AX,04
XOR AX,CX
MOV [BX+2],AX
HLT
```

```
2000:1000  3A 00 2C 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 00 2C
BX 10 00
CX 00 2E
DX 20 00
CS 0100
IP 0027
SS 0100
SP FFFE
BP 0000
SI 0000
DI 0000
DS 2000
ES 0100
```

SAMPLE INPUT AND OUTPUT:

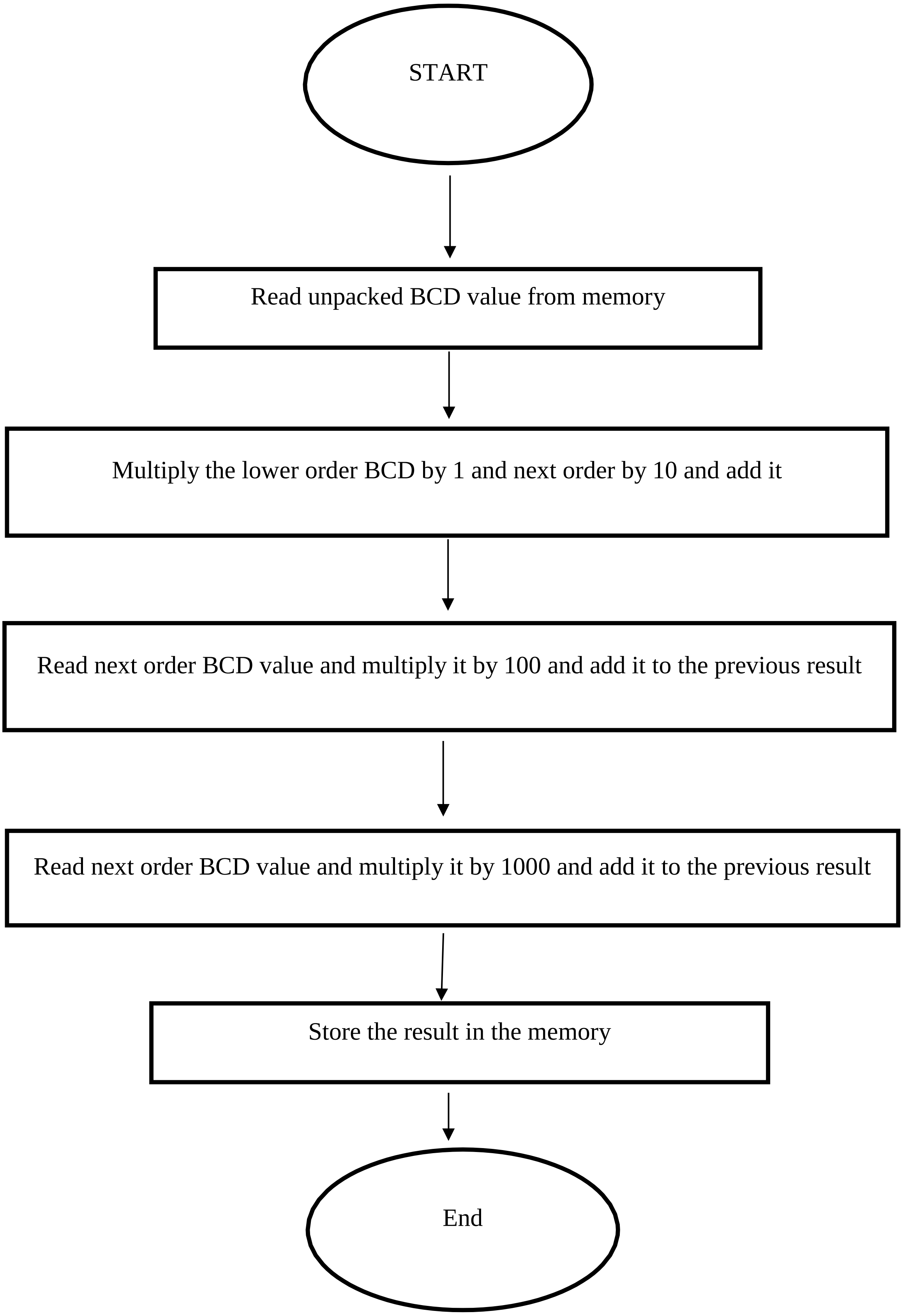
Memory location		
2000:1000	3A	Input
2000:1002	2C	Output

Result

Thus, the ALP program to convert gray code to binary code were developed and the output is verified using emu 8086.

UNPACKED BCD TO HEXA DECIMAL CONVERSION

FLOWCHART


START
Read unpacked BCD value from memory
Multiply the lower order BCD by 1 and next order by 10 and add it
Read next order BCD value and multiply it by 100 and add it to the previous result
Read next order BCD value and multiply it by 1000 and add it to the previous result
Store the result in the memory
End

EX NO 13

UNPACKED BCD TO HEXA DECIMAL CONVERSION

AIM

To develop an Assembly language program to convert unpacked BCD to hexadecimal value.

ALGORITHM

STEP 1: Start

STEP 2: Read the unpacked BCD values from the memory as inputs.

STEP 3: Multiply the lower order BCD value from 1 and next order BCD value by 10.

STEP 4: Perform addition of both the values.

STEP 5: Multiply the next order BCD value by 100 and add it to the previous result.

STEP 6: Multiply the next order BCD value by 1000 and add it to the previous result.

STEP 7: Multiply the next order BCD value by 10000 and add it to the previous result.

STEP 8: Store the final result in memory.

STEP 9: End

PROGRAM

```
ORG 1000H

MOV DX,2000H

MOV DS,DX

MOV SI,1000H

MOV BL,[SI]

MOV AL,[SI+1]

MOV CX,10

MUL CX

ADD BX,AX

MOV AL,[SI+2]

MOV CX,100
```

MUL CX

ADD BX,AX

XOR AX,AX

MOV AL,[SI +4]

MOV CX,10000

MUL CX

ADD BX,AX

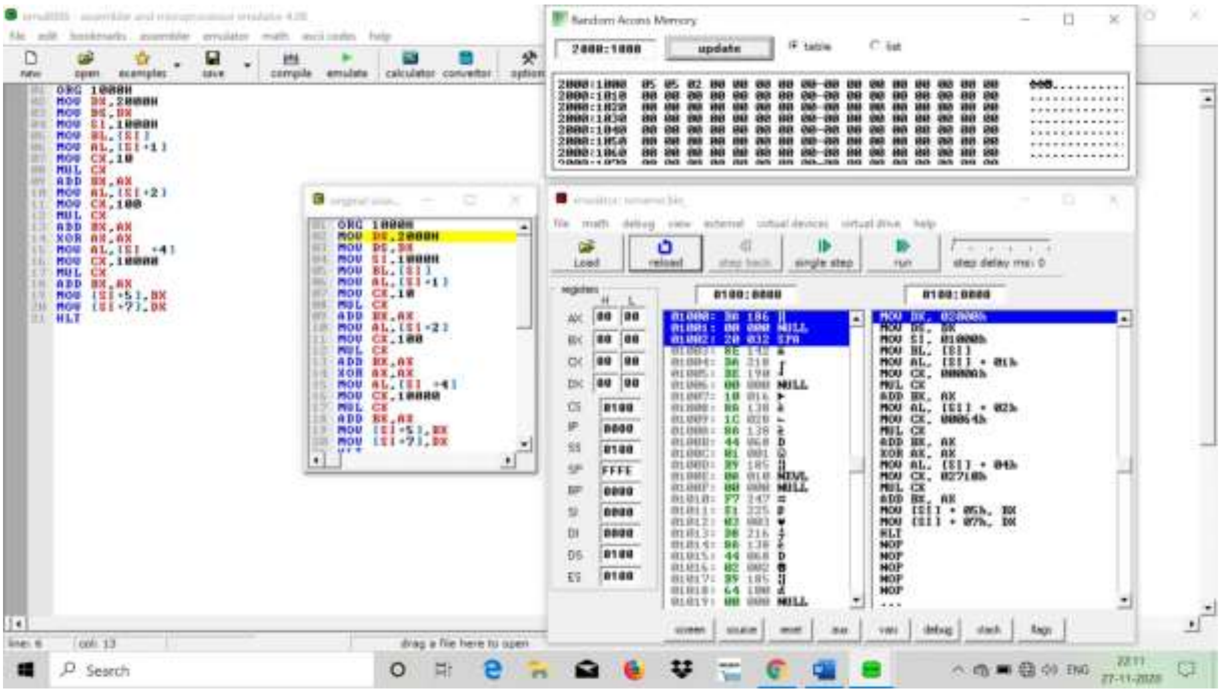
MOV [SI+5],BX

MOV [SI+7],DX

HLT

SCREENSHOTS:

BEFORE EXECUTION



```
ORG 1000H
MOV DX,2000H
MOV DS,DX
MOV SI,1000H
MOV BL,[SI]
MOV AL,[SI+1]
MOV CX,10
MUL CX
ADD BX,AX
MOV AL,[SI+2]
MOV CX,100
MUL CX
ADD BX,AX
XOR AX,AX
MOV AL,[SI +4]
MOV CX,10000
MUL CX
ADD BX,AX
MOV [SI+5],BX
MOV [SI+7],DX
HLT
```

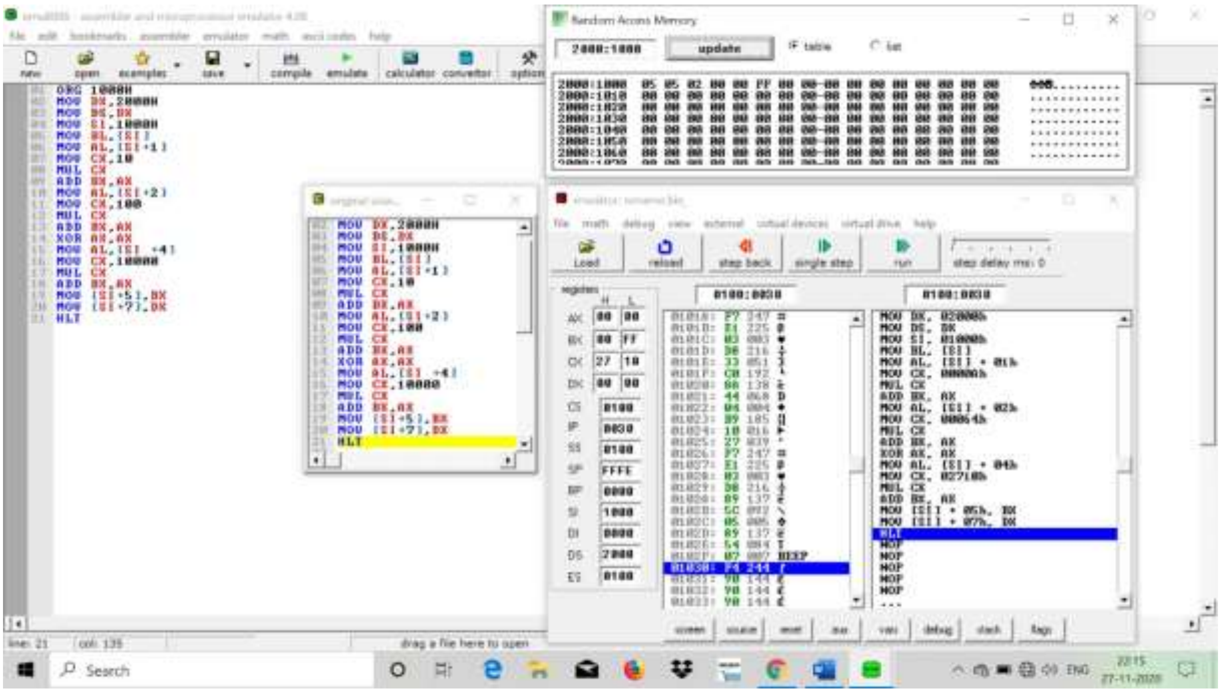
```
2000:1000
2000:1000  05 05 02 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 00 00
BX 00 00
CX 00 00
DX 00 00
CS 0100
IP 0000
SS 0100
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0100
ES 0100
```

```
0100:0000
01000: BA 186 ║
01001: 00 000 NULL
01002: 20 032 SPA
01003: 8E 142 Ä
01004: DA 218 ┌
01005: BE 190 ╛
01006: 00 000 NULL
01007: 10 016 ►
01008: 8A 138 è
01009: 1C 028 ∟
0100A: 8A 138 è
0100B: 44 068 D
0100C: 01 001 ☺
0100D: B9 185 ╣
0100E: 0A 010 NEWL
0100F: 00 000 NULL
01010: F7 247 ≈
01011: E1 225 ß
01012: 03 003 ♥
01013: D8 216 ╪
01014: 8A 138 è
01015: 44 068 D
01016: 02 002 ☻
01017: B9 185 ╣
01018: 64 100 d
01019: 00 000 NULL
```

```
0100:0000
MOV DX, 02000h
MOV DS, DX
MOV SI, 01000h
MOV BL, [SI]
MOV AL, [SI] + 01h
MOV CX, 0000Ah
MUL CX
ADD BX, AX
MOV AL, [SI] + 02h
MOV CX, 00064h
MUL CX
ADD BX, AX
XOR AX, AX
MOV AL, [SI] + 04h
MOV CX, 02710h
MUL CX
ADD BX, AX
MOV [SI] + 05h, BX
MOV [SI] + 07h, DX
HLT
NOP
NOP
NOP
NOP
NOP
...
```

AFTER EXECUTION



```
ORG 1000H
MOV DX,2000H
MOV DS,DX
MOV SI,1000H
MOV BL,[SI]
MOV AL,[SI+1]
MOV CX,10
MUL CX
ADD BX,AX
MOV AL,[SI+2]
MOV CX,100
MUL CX
ADD BX,AX
XOR AX,AX
MOV AL,[SI +4]
MOV CX,10000
MUL CX
ADD BX,AX
MOV [SI+5],BX
MOV [SI+7],DX
HLT
```

```
2000:1000  05 05 02 00 00 FF 00 00-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

	H	L
AX	00	00
BX	00	FF
CX	27	10
DX	00	00
CS	0100	
IP	0030	
SS	0100	
SP	FFFE	
BP	0000	
SI	1000	
DI	0000	
DS	2000	
ES	0100	

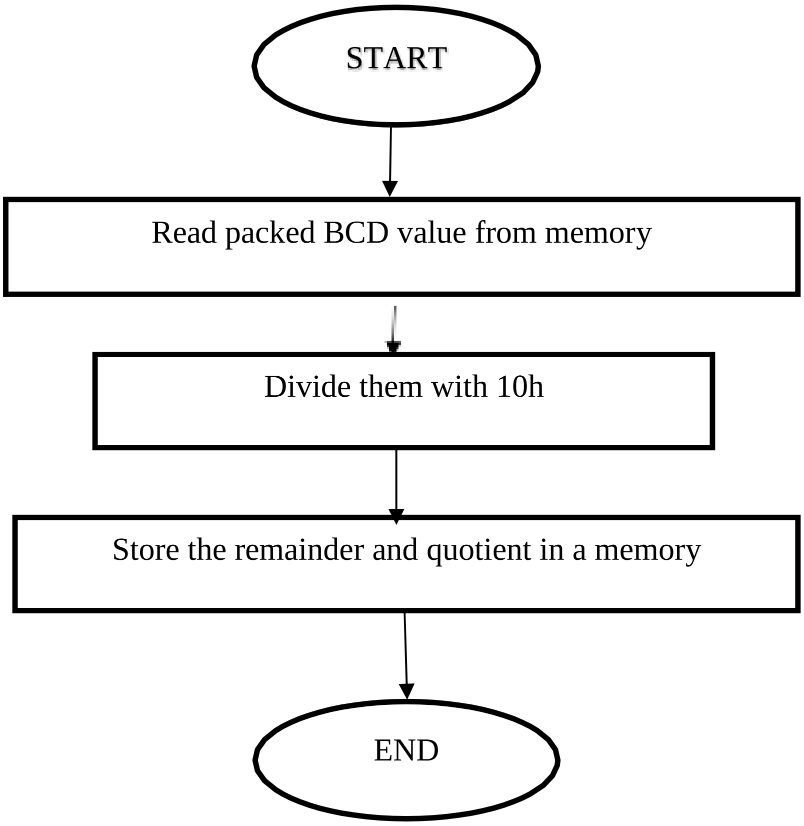
SAMPLE INPUT AND OUTPUT

Memory location						
2000:1000	05	05	02	00	00	Input
2000:1005	FF					Output

RESULT

Thus, the ALP program to convert unpacked BCD to hexadecimal value were developed and the output is verified using emu 8086.

FLOWCHART


START
Read packed BCD value from memory
Divide them with 10h
Store the remainder and quotient in a memory
END

EX NO 14

PACKED BCD TO UNPACKED BCD CONVERSION

AIM

To develop an Assembly language program to convert packed BCD to unpacked BCD.

ALGORITHM

STEP 1: Start

STEP 2: Read packed BCD number from memory

STEP 3: Divide the packed BCD with 10

STEP 4: Store the remainder and quotient in the memory

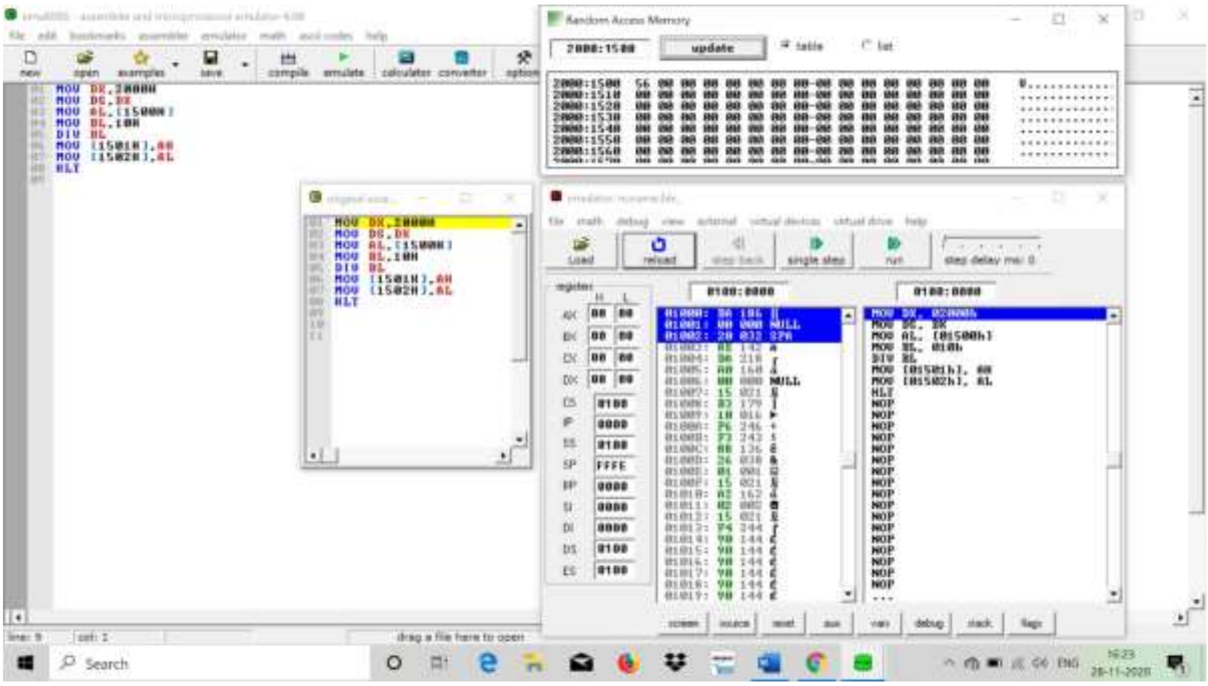
STEP 5: End

PROGRAM

```
MOV DX,2000H

MOV DS,DX

MOV AL,[1500H]

MOV BL,10H

DIV BL

MOV [1501H],AH

MOV  [1502H],AL

HLT
```

SCREENSHOTS:

BEFORE EXECUTION



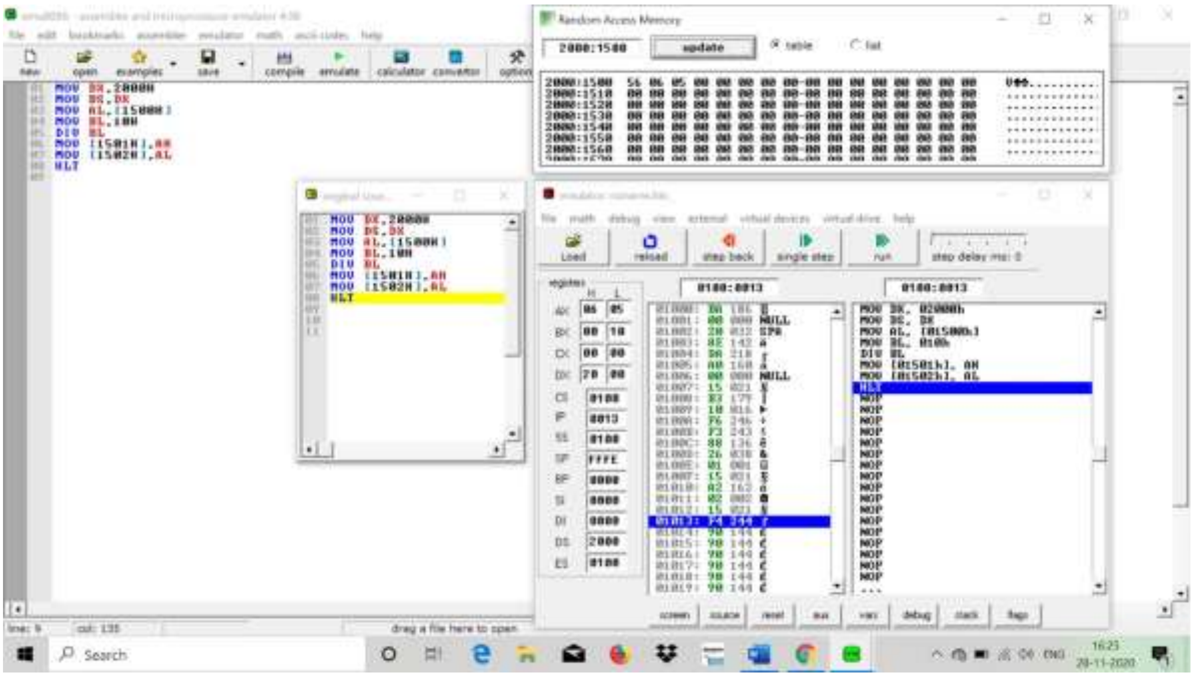
```
MOV DX,2000H
MOV DS,DX
MOV AL,[1500H]
MOV BL,10H
DIV BL
MOV [1501H],AH
MOV [1502H],AL
HLT
```

```
2000:1500  56 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 00 00
BX 00 00
CX 00 00
DX 00 00
CS 0100
IP 0000
SS 0100
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0100
ES 0100
```

```
MOV DX, 02000h
MOV DS, DX
MOV AL, [01500h]
MOV BL, 010h
DIV BL
MOV [01501h], AH
MOV [01502h], AL
HLT
```

AFTER EXECUTION



```
MOV DX,2000H
MOV DS,DX
MOV AL,[1500H]
MOV BL,10H
DIV BL
MOV [1501H],AH
MOV [1502H],AL
HLT
```

```
2000:1500  56 06 05 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
AX 06 05
BX 00 10
CX 00 00
DX 20 00
CS 0100
IP 0013
SS 0100
SP FFFE
BP 0000
SI 0000
DI 0000
DS 2000
ES 0100
```

```
MOV DX, 02000h
MOV DS, DX
MOV AL, [01500h]
MOV BL, 010h
DIV BL
MOV [01501h], AH
MOV [01502h], AL
HLT
```

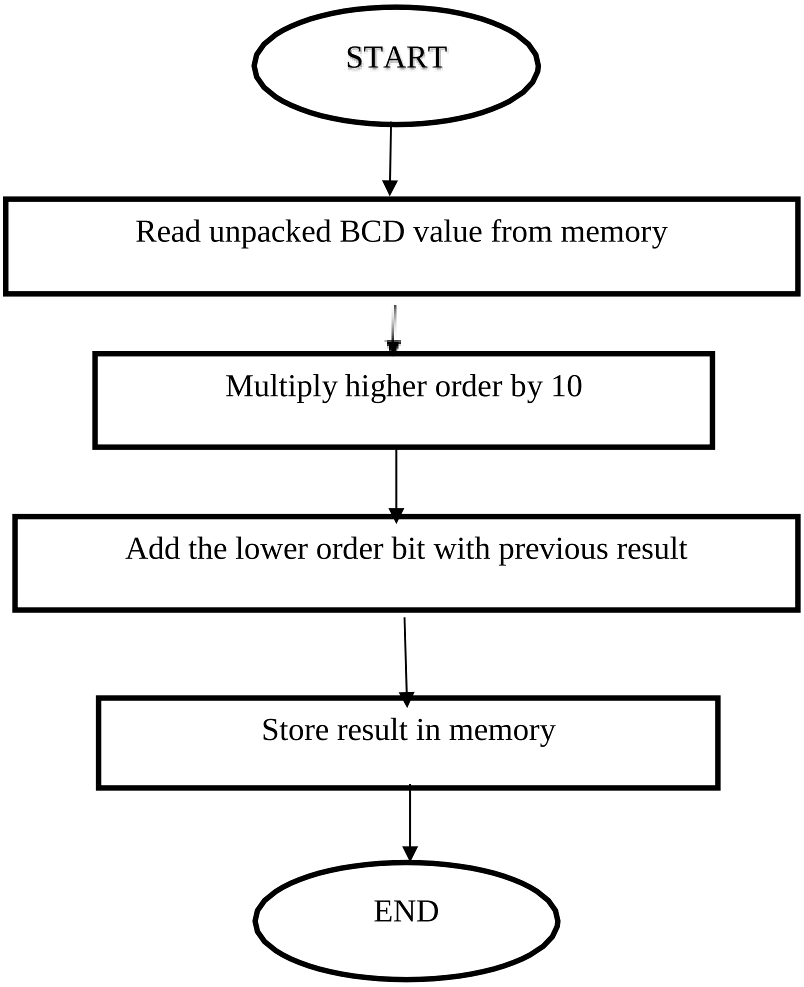
SAMPLE INPUT AND OUTPUT

Memory location			
2000:1500	56		Input
2000:1501	06	05	Output

RESULT

Thus, the ALP program to convert packed BCD into unpacked BCD were developed and the output is verified using emu 8086.

FLOWCHART


START
Read unpacked BCD value from memory
Multiply higher order by 10
Add the lower order bit with previous result
Store result in memory
END

EX NO 15

UNPACKED BCD TO PACKED BCD CONVERSION

AIM

To develop an Assembly language program to convert unpacked BCD to packed BCD.

ALGORITHM

STEP 1: Start

STEP 2: Read unpacked BCD number from memory.

STEP 3: Multiply the higher order by 10

STEP 4: Add the lower order bit with the previous result

STEP 5: Store the final result in the memory

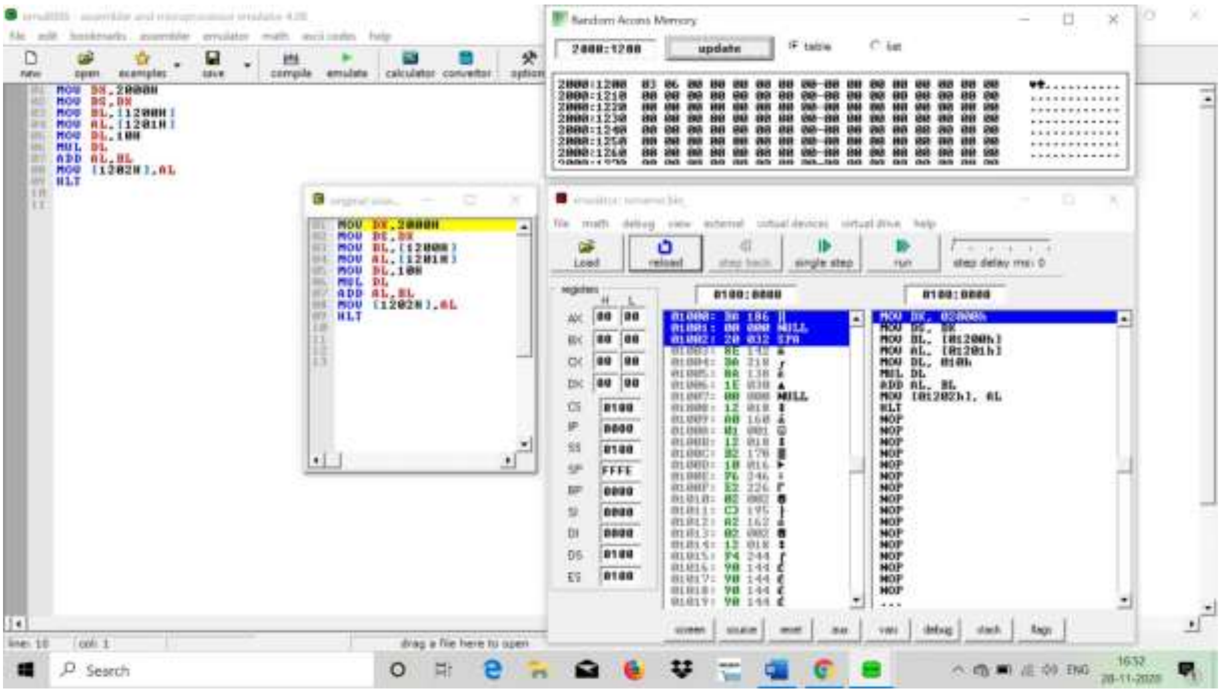
STEP 6: End

PROGRAM

```
MOV DX,2000H

MOV DS,DX

MOV BL,[1200H]

MOV AL,[1201H]

MOV DL,10H

MUL DL

ADD AL,BL

MOV [1202H],AL

HLT
```

SCREENSHOTS:

BEFORE EXECUTION



```
MOV DX,2000H
MOV DS,DX
MOV BL,[1200H]
MOV AL,[1201H]
MOV DL,10H
MUL DL
ADD AL,BL
MOV [1202H],AL
HLT
```

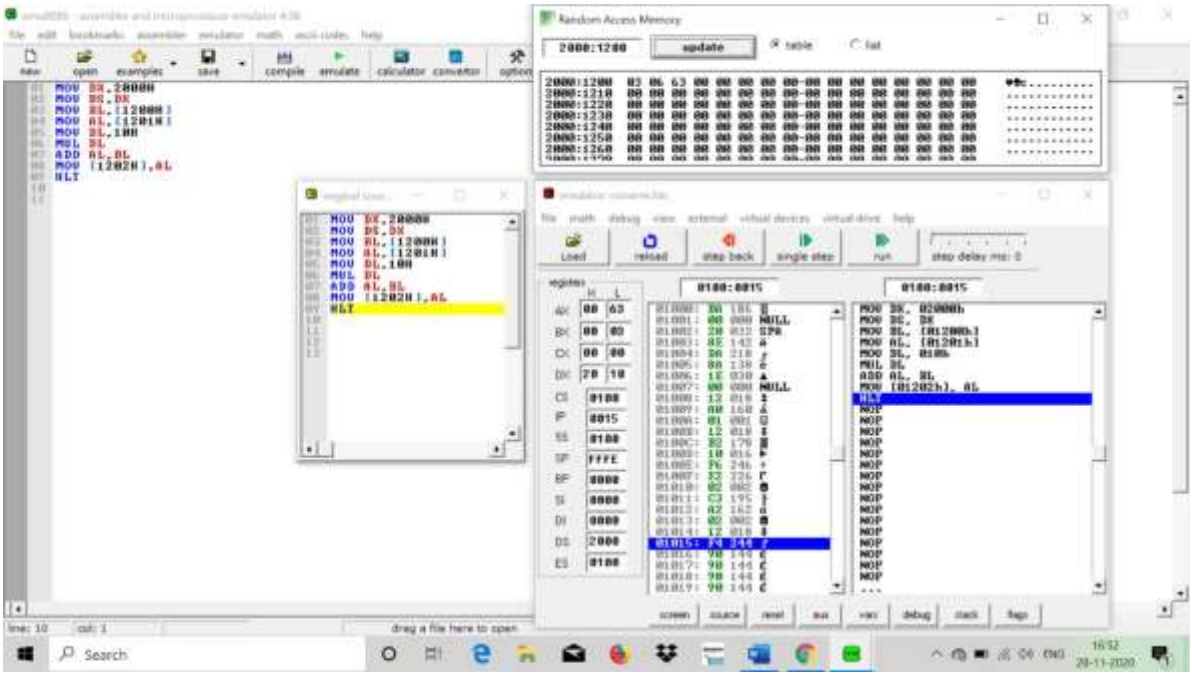
Random Access Memory: 2000:1200 update ● table ○ list

```
2000:1200  03 06 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
registers
     H  L
AX  00 00
BX  00 00
CX  00 00
DX  00 00
CS  0100
IP  0000
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  0100
ES  0100
```

```
0100:0000
MOV DX, 02000h
MOV DS, DX
MOV BL, [01200h]
MOV AL, [01201h]
MOV DL, 010h
MUL DL
ADD AL, BL
MOV [01202h], AL
HLT
NOP
```

AFTER EXECUTION:



Random Access Memory: 2000:1200 update ● table ○ list

```
2000:1200  03 06 63 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
registers
     H  L
AX  00 63
BX  00 03
CX  00 00
DX  20 10
CS  0100
IP  0015
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  2000
ES  0100
```

```
0100:0015
MOV DX, 02000h
MOV DS, DX
MOV BL, [01200h]
MOV AL, [01201h]
MOV DL, 010h
MUL DL
ADD AL, BL
MOV [01202h], AL
HLT
NOP
```

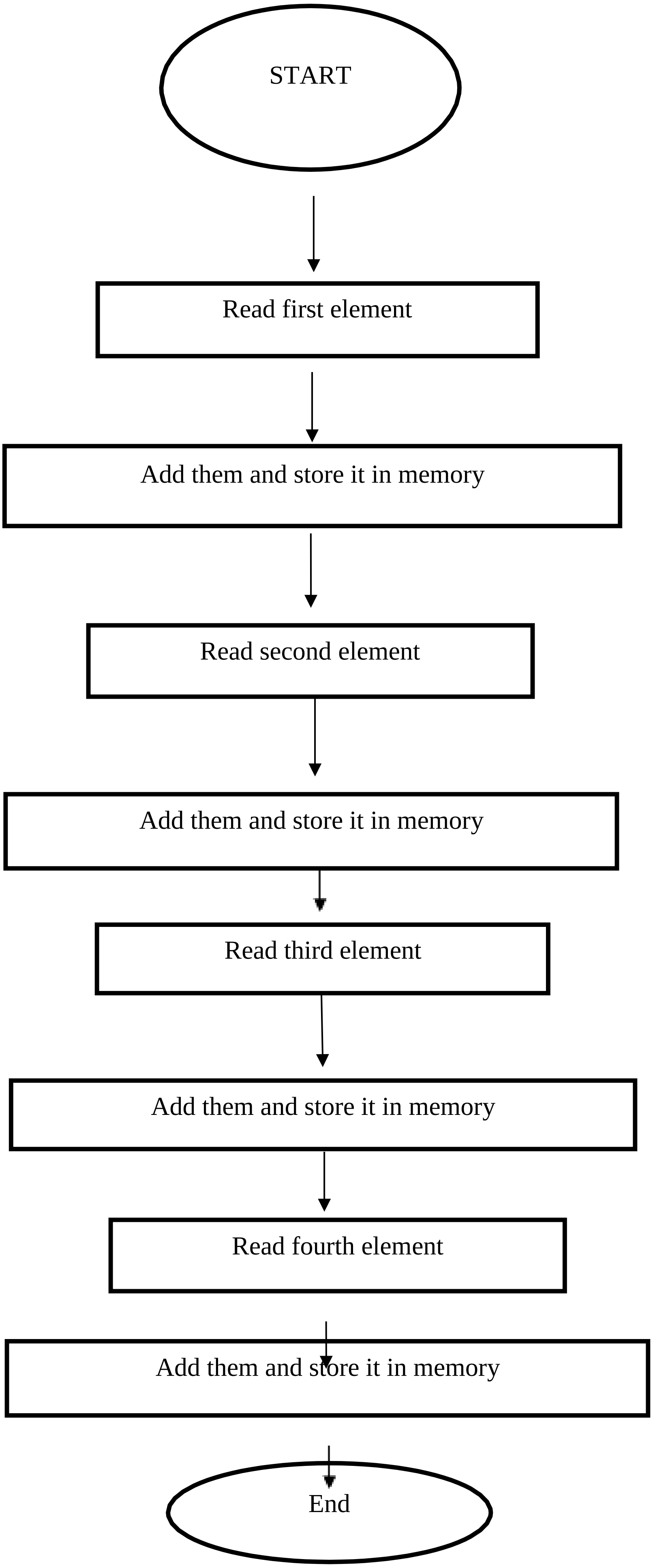
SAMPLE INPUT AND OUTPUT

Memory location			
2000:1200	03	06	Input
2000:1202	63		Output

RESULT

Thus, the ALP program to convert unpacked BCD into packed BCD were developed and the output is verified using emu 8086.

FLOWCHART


START
Read first element
Add them and store it in memory
Read second element
Add them and store it in memory
Read third element
Add them and store it in memory
Read fourth element
Add them and store it in memory
End

EX NO 15

2*2 MATRIX ADDITION OF 16 BIT NUMBERS

AIM

To develop an Assembly language program to perform 2*2 matrix addition of 16 bit numbers.

ALGORITHM

STEP 1: Start

STEP 2: Read the first element of both matrices.

STEP 3: Add them and store it in a memory

STEP 4: Read the second element of both matrices

STEP 5: Add them and store it in a memory

STEP 6: Read the third element of both matrices

STEP 7: Add them and store it in a memory

STEP 8: Read the fourth element of both matrices

STEP 9: Add them and store it in a memory

STEP 10: End

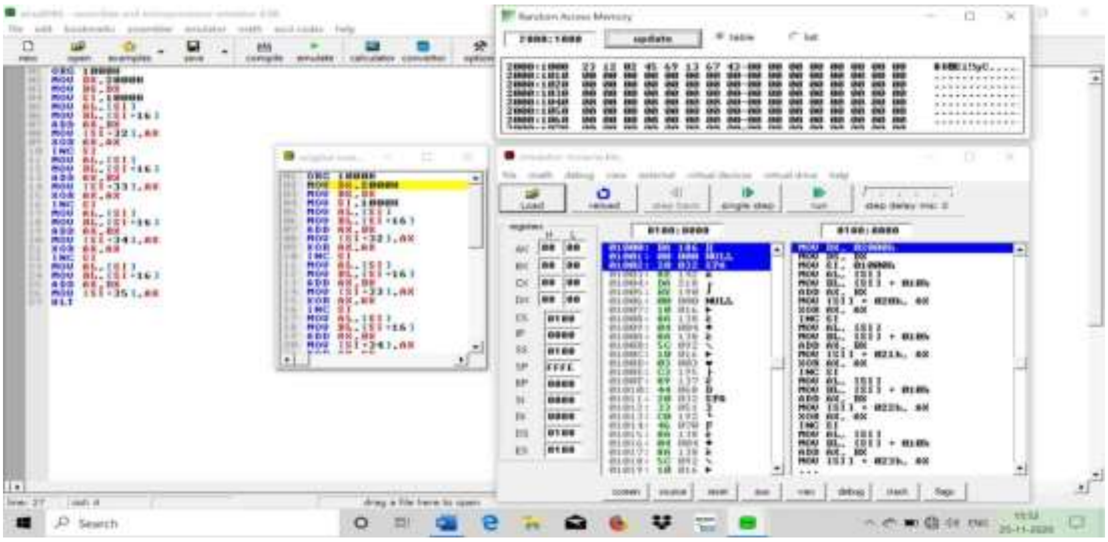
PROGRAM

```
ORG 1000H

MOV DX,2000H

MOV DS,DX

MOV SI,1000H

MOV AL,[SI]

MOV BL,[SI+16]

ADD AX,BX

MOV [SI+32],AX

XOR AX,AX

INC SI
```

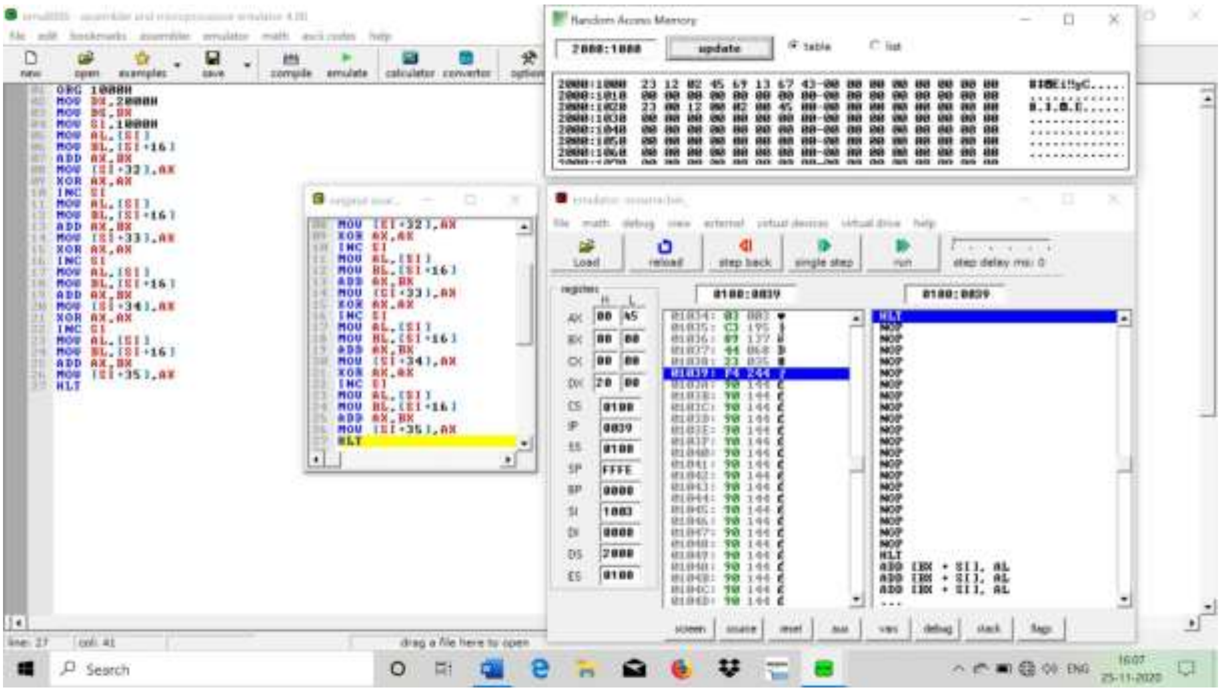
```
MOV AL,[SI]

MOV BL,[SI+16]

ADD AX,BX

MOV [SI+33],AX

XOR AX,AX

INC SI

MOV AL,[SI]

MOV BL,[SI+16]

ADD AX,BX

MOV [SI+34],AX

XOR AX,AX

INC SI

MOV AL,[SI]

MOV BL,[SI+16]

ADD AX,BX

MOV [SI+35],AX

HLT
```

SCREENSHOT:

BEFORE EXECUTION



AFTER EXECUTION



```
ORG 1000H
MOV DX,2000H
MOV DS,DX
MOV SI,1000H
MOV AL,[SI]
MOV BL,[SI+16]
ADD AX,BX
MOV [SI+32],AX
XOR AX,AX
INC SI
MOV AL,[SI]
MOV BL,[SI+16]
ADD AX,BX
MOV [SI+33],AX
XOR AX,AX
INC SI
MOV AL,[SI]
MOV BL,[SI+16]
ADD AX,BX
MOV [SI+34],AX
XOR AX,AX
INC SI
MOV AL,[SI]
MOV BL,[SI+16]
ADD AX,BX
MOV [SI+35],AX
HLT
```

```
2000:1000  23 12 02 45 69 13 67 43-00 00 00 00 00 00 00 00
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1020  23 00 12 00 02 00 45 00-00 00 00 00 00 00 00 00
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

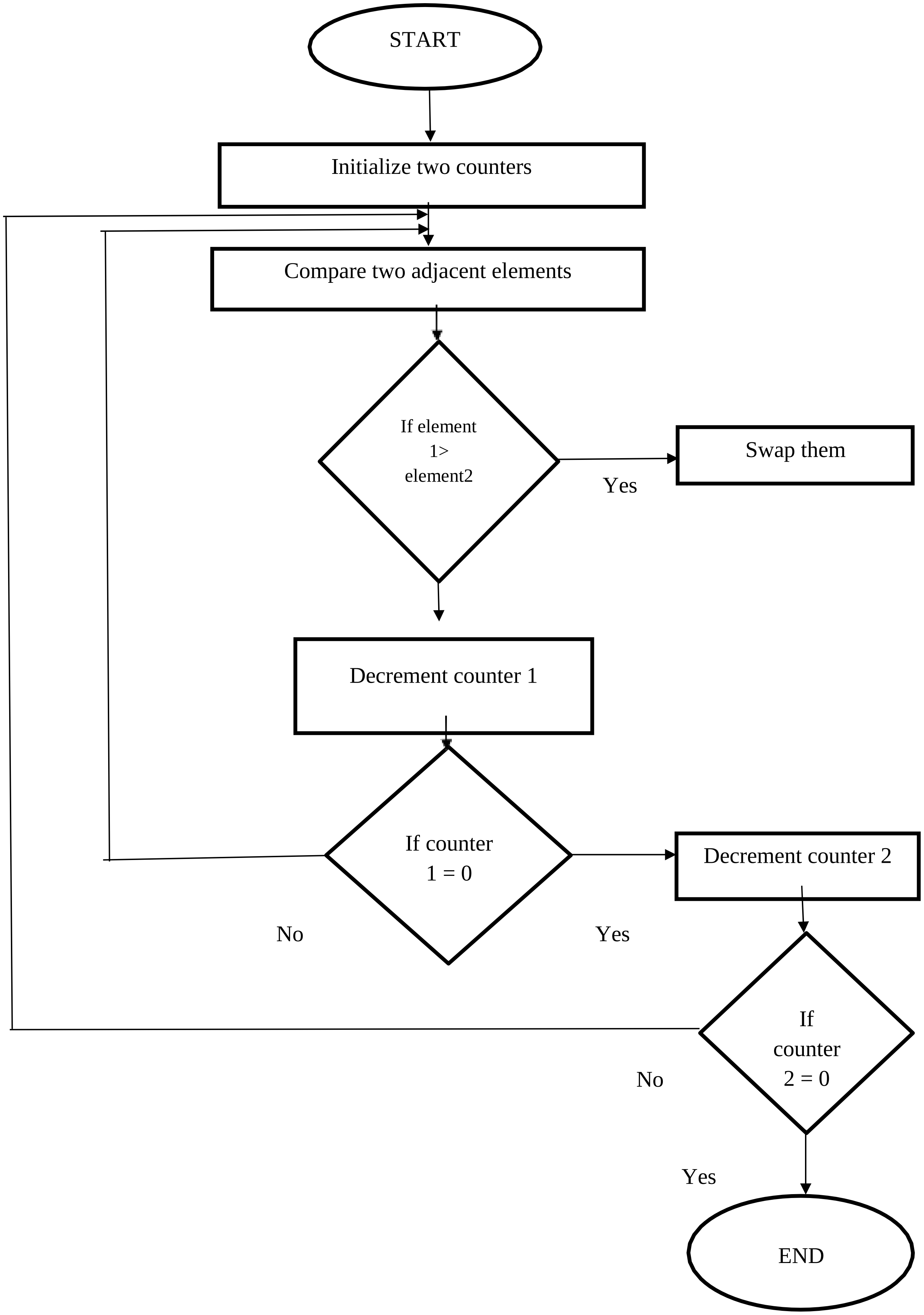
SAMPLE INPUT AND OUTPU

Memory location									
2000:1000	23	12	02	45	69	13	67	43	Input
2000:1032	00	2C	25	00	69	00	88		Output

RESULT

Thus, the ALP program to perform 2*2 matrix addition were developed and the output is verified using emu 8086.

FLOWCHART:


START
Initialize two counters
Compare two adjacent elements
If element 1> element2
Yes
Swap them
Decrement counter 1
If counter 1 = 0
No
Yes
Decrement counter 2
If counter 2 = 0
No
Yes
END

EX NO 16

SORTING OF 8 BIT NUMBERS

AIM

To develop an Assembly language program for sorting of 8-bit numbers.

ALGORITHM

STEP 1: Start

STEP 2: Initialize a counter

STEP 3: From starting address to last, compare two numbers, if the first number is greater than the second number, then swap the numbers.

STEP 4: Then decrement the counter

STEP 5: Repeat the above two steps until the counter is zero.

STEP 6: End

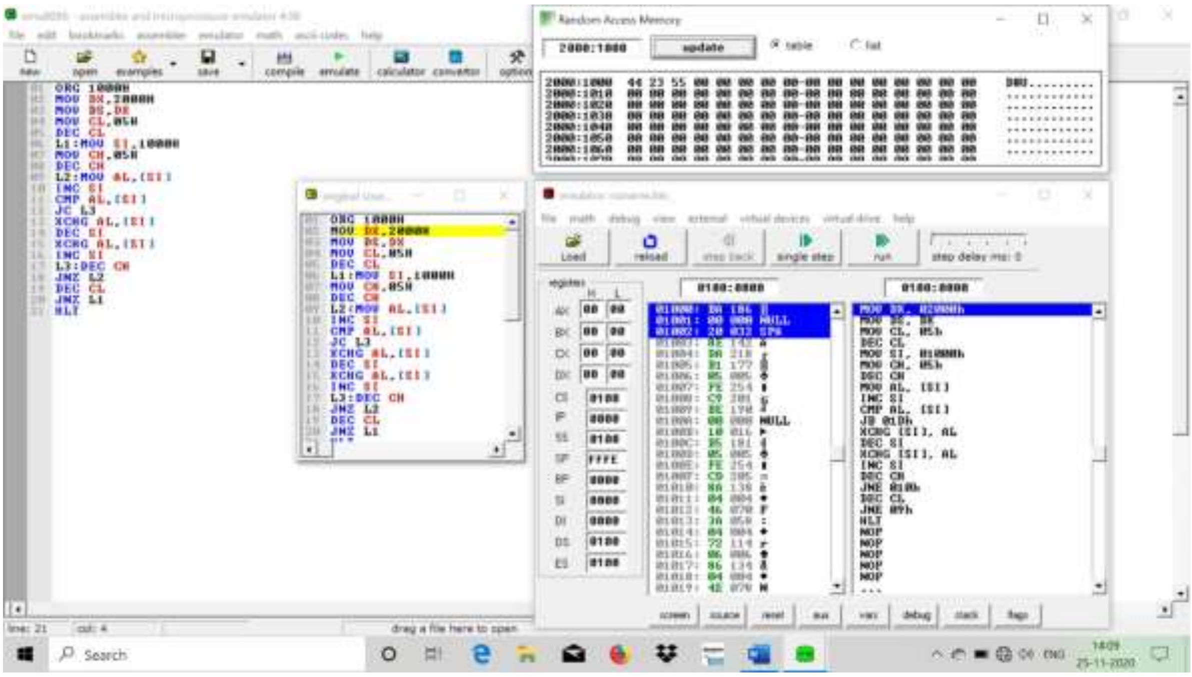
PROGRAM

```
ORG 1000H
MOV DX,2000H
MOV DS, DX
MOV CL,05H
DEC CL
L1:MOV SI,1000H
MOV CH,05H
DEC CH
L2:MOV AL,[SI]
INC SI
CMP AL,[SI]
JC L3
XCHG AL,[SI]
```

```
DEC  SI

XCHG AL,[SI]

INC SI

L3:DEC CH

JNZ L2

DEC CL

JNZ L1

HLT
```

SCREENSHOTS:

BEFORE EXECUTION



```
ORG 1000H
MOV DX,2000H
MOV DS,DX
MOV CL,05H
DEC CL
L1:MOV SI,1000H
MOV CH,05H
DEC CH
L2:MOV AL,[SI]
INC SI
CMP AL,[SI]
JC L3
XCHG AL,[SI]
DEC SI
XCHG AL,[SI]
INC SI
L3:DEC CH
JNZ L2
DEC CL
JNZ L1
HLT
```

Random Access Memory

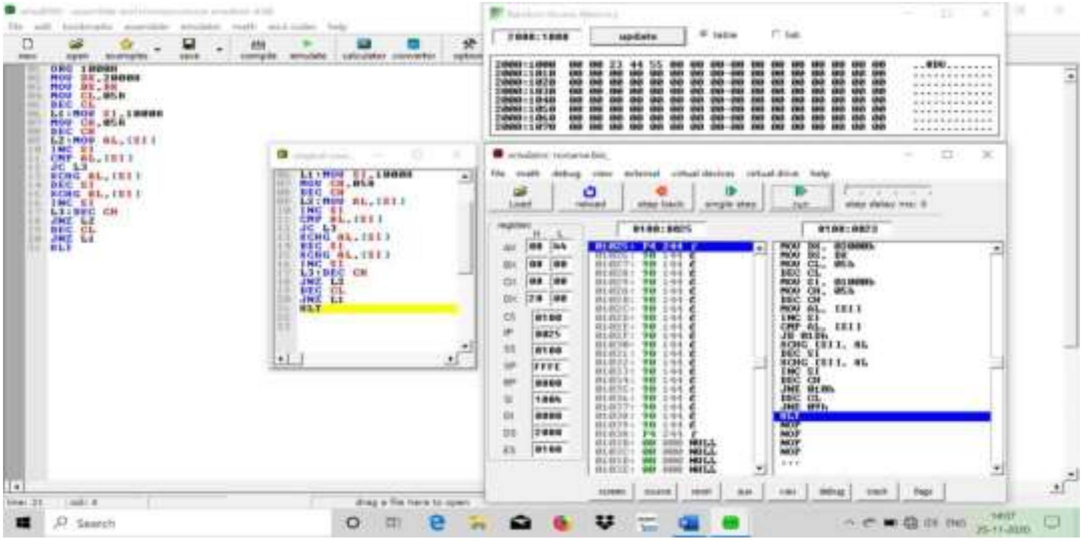
2000:1000

```
2000:1000  44 23 55 00 00 00 00 00-00 00 00 00 00 00 00 00   D#U.............
2000:1010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2000:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

registers	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	0000	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

```
MOV DX, 02000h
MOV DS, DX
MOV CL, 05h
DEC CL
MOV SI, 01000h
MOV CH, 05h
DEC CH
MOV AL, [SI]
INC SI
CMP AL, [SI]
JB 01Dh
XCHG [SI], AL
DEC SI
XCHG [SI], AL
INC SI
DEC CH
JNE 010h
DEC CL
JNE 0Fh
HLT
NOP
NOP
NOP
NOP
NOP
```

AFTER EXECUTION:



```
ORG 1000H
MOV AX,2000H
MOV DS,AX
MOV CL,05H
DEC CL
L1:MOV SI,1000H
MOV CH,05H
DEC CH
L2:MOV AL,[SI]
INC SI
CMP AL,[SI]
JC L3
XCHG AL,[SI]
DEC SI
XCHG AL,[SI]
INC SI
L3:DEC CH
JNZ L2
DEC CL
JNZ L1
HLT
```

```
2000:1000  00 00 23 44 55 00 00 00-00 00 00 00 00 00 00 00
```

SAMPLE INPUT AND OUTPUT

Memory location						
2000:1000	44	23	55	00	00	Input
2000:1000	00	00	23	44	55	Output

RESULT

Thus, the ALP program for sorting of 8-bit numbers were developed and output is verified using emu 8086.