Program 1 Date 18 July 2012

Aim:

To write an assembly language program to accept two numbers. Calculate and display sum, difference, product and quotient.

Theory:

The program uses the basic ADD, SUB, MUL and DIV instructions to carry out elementary arithmetic operations. The program also utilises an external library (ioasm) to carry out input/output functions. Interrupts are used to print strings and to signal the end of the program.

Algorithm:

```
1> START
2> READ FIRST NUMBER → X
3> READ SECOND NUMBER → Y
4> SUM = X + Y
5> PRINT SUM
6> DIFF = X - Y
7> PRINT DIFF
8> PROD = X * Y
9> PRINT PROD
10>QUO = X / Y
11> PRINT QUO
12> STOP
```

```
.8086
1
     .MODEL SMALL
 2
 3
     .STACK 512
 5
     . DATA
                         DB 13,10,"ENTER VALUE OF X: ",'$'
DB 13,10,"ENTER VALUE OF Y: ",'$'
DB 13,10,"X + Y = ",'$'
 6
              NUM1
 7
              NUM2
 8
              AD
                               DB 13,10,"X - Y = ",'$'
9
              SB
                               DB 13,10,"X * Y = ",'$'
10
              ML
              DV
                                DB 13,10,"X / Y = ",'$'
11
12
              N1
                                DW ?
              N2
                                DW ?
13
14
     .CODE
15
              EXTRN READSINT: NEAR, WRITESINT: NEAR
16
17
     START:
              MOV AX,@DATA
18
              MOV DS, AX
19
```

```
20
21
            LEA DX, NUM1
22
            MOV AH,09H
            INT 21H
23
24
25
            CALL READSINT
26
            MOV N1,AX
27
            LEA DX, NUM2
28
            MOV AH,09H
            INT 21H
29
30
31
            CALL READSINT
            MOV N2,AX
32
33
            LEA DX,AD
34
            MOV AH,09H
35
            INT 21H
36
37
            MOV AX,N1
38
            ADD AX,N2
39
            CALL WRITESINT
40
            LEA DX,SB
41
            MOV AH,09H
42
            INT 21H
43
44
            MOV AX,N1
45
            SUB AX,N2
46
            CALL WRITESINT
47
            LEA DX,ML
48
            MOV AH,09H
49
            INT 21H
50
51
            MOV AX,N1
52
            MOV BX,N2
53
            MUL BX
54
            CALL WRITESINT
55
56
            LEA DX, DV
57
            MOV AH,09H
            INT 21H
58
59
            MOV DX,0
60
            MOV AX,N1
61
            MOV BX,N2
62
            DIV BX
63
64
            CALL WRITESINT
65
66
            MOV AH, 4CH
            INT 21H
67
68
    END START
```

An 8086 assembly language program carrying out summation, subtraction, multiplication and division was written, assembled, linked and debugged to obtain expected output.

Program 2 Date 25 July 2012

Aim:

To write an assembly language program to swap two numbers.

Theory:

The program uses a classic algorithm to swap two numbers using only two variables although 8086 has a native instruction to swap the contents of two locations (xchg). It makes use of basic arithmetic instructions and the ioasm library along with interrupts for input/output.

Algorithm:

```
1> START
2> READ FIRST NUMBER → A
3> READ SECOND NUMBER → B
4> A = A + B
5> B = A - B
6> A = A - B
7> PRINT A
8> PRINT B
```

Source code:

9> STOP

```
.8086
     .MODEL SMALL
 3
     .STACK 512
     .DATA
 5
                               DW ?
 6
              Υ
                               DW ?
7
                               DB 13,10,'$'
              CRLF
                              DB 13,10, "ENTER FIRST NUMBER: ",'$'
              PROMPT1
8
                               DB 13,10, "ENTER THE SECOND NUMBER: ", '$'
9
              PROMPT
                              DB 13,10,"BEFORE SWAPPING: "13,10,'$'
DB 13,10,"AFTER SWAPPING: ",13,10,'$'
10
              PR0MPT2
11
              PROMPT3
12
13
     .CODE
              EXTRN READSINT: NEAR, WRITESINT: NEAR
14
     START:
15
16
              MOV AX,@DATA
17
              MOV DS,AX
18
19
              LEA DX, PROMPT1
             MOV AH,09H
20
             INT 21H
21
22
             CALL READSINT
23
24
             MOV X,AX
```

```
25
            LEA DX, PROMPT
26
            MOV AH,09H
27
            INT 21H
28
29
            CALL READSINT
            MOV Y,AX
30
31
            LEA DX, PROMPT2
            MOV AH,09H
32
            INT 21H
33
34
35
            MOV AX,X
36
            CALL WRITESINT
37
            MOV AX,Y
            CALL WRITESINT
38
39
40
            MOV AX,X
41
42
            ADD AX, BX
43
            MOV BX,AX
44
            SUB BX,Y
45
            SUB AX,BX
46
47
            MOV X,AX
48
            MOV Y,BX
49
50
            LEA DX, PROMPT3
51
            MOV AH,09H
52
            INT 21H
53
54
            MOV AX,X
55
            CALL WRITESINT
56
            MOV AX,Y
57
            CALL WRITESINT
58
59
            MOV AH, 4CH
60
            INT 21H
61
    END START
```

An 8086 assembly language program carrying out swapping of values between two variables was written, assembled, linked and debugged to obtain expected output.

Program 3 Date 25 July 2012

Aim:

To write an assembly language program to generate Fibonacci series. Accept the order of Fibonacci series from the user.

Theory:

The Fibonacci series is a sequence of numbers in which every subsequent number is found by adding the previous two numbers in the series. They can be represented in following integer sequence:

The program reads the number of terms to printed from the user. The program execution continues in a loop using conditional jump labels namely JNZ which jumps if the ZERO flag is reset. The flags are set by the instruction CMP (compare) which compares its operands and appropriately sets the flags.

Algorithm:

```
1> START
2> INITIALISE COUNT = 0, A = 0, B = 1
3> READ ORDER → N
4> PRINT A
5> X = A
6> A = A + B
7> B = X
8> COUNT = COUNT + 1
9> IF NOT COUNT == N GOTO 4
10>STOP
```

```
.8086
1
     .MODEL SMALL
2
3
    .STACK 512
4
     .DATA
5
            MSG1
                            DB "ENTER ORDER: ",$
                            DB "FIBONACCI SERIES: ",13,10.'$'
6
            MSG2
                            DW ?
7
             N
8
    .CODE
             EXTRN READSINT: NEAR, WRITESINT: NEAR
9
10
    START:
11
            MOV AX,@DATA
12
            MOV DS, AX
13
             LEA DX, MSG1
14
15
             MOV AH,09H
```

```
INT 21H
16
17
18
            CALL READSINT
19
            MOV N,AX
20
21
            MOV AX,0
22
            MOV BX,0
23
            MOV CX,0
24
            MOV DX,1
25
26
            LEA DX,MSG2
27
            MOV AH,09H
            INT 21H
28
29
   L00P0:
30
            CALL WRITESINT
31
32
            MOV AX, BX
33
            ADD AX, DX
34
            MOV DX,AX
35
36
            INC CX
37
            CMP CX,N
38
39
            JNZ LOOP0
40
41
            MOV AH,4CH
            INT 21H
42
43
   END START
```

An 8086 assembly language program to print the Fibonacci series of given order was written, assembled, linked and debugged to obtain expected output.

Program 4 Date 1 August 2012

Aim:

To write an assembly language program to accept n numbers and display them in reversed order.

Theory:

The program reads a specified number of items (n) and prints them in reversed order. To do this, it uses two loops. In the first loop, it reads the items, with the counter initially set to 0 and incremented until it reaches the value n. In the second loop, the counter is initially set to n, and decremented until it reaches 0 while printing the items in relation to the index.

Algorithm:

```
1> START

2> READ NUMBER OF ITEMS → N

3> INITIALISE COUNT = 0

4> READ ITEM → [ARR + COUNT]

5> COUNT = COUNT + 1

6> IF NOT COUNT == N, GOTO 4

7> COUNT = N

8> COUNT = COUNT - 1

9> PRINT [ARR + COUNT]

10>IF NOT COUNT == 0, GOTO 8

11>STOP
```

```
1
    .8086
2
    .MODEL SMALL
3
    .STACK 512
    .DATA
5
                           DB "HOW MANY ITEMS? $"
            MSG1
                           DB "ENTER NUMBERS: ",13,10,'$'
6
            MSG2
7
            CRLF
                         DB 13,10,'$'
8
            SPACE
                           DB "
9
                           DW 1
            COUNTER
10
                           DW ?
11
            ARRAY
                           DW 100 DUP(?)
    .CODE
12
            EXTRN READSINT: NEAR, WRITESINT: NEAR
13
    START:
14
15
            MOV AX,@DATA
16
            MOV DS, AX
17
18
            LEA DX, MSG1
            MOV AH,09H
19
            INT 21H
20
```

```
21
22
            CALL READSINT
23
            MOV N, AX
24
            ADD N,1
25
26
            LEA DX,MSG2
27
            MOV AH,09H
            INT 21H
28
29
            MOV CX,N
30
31
            LEA BX, ARRAY
32
    L00P1:
33
34
            CALL READSINT
35
            MOV [BX],AX
36
37
            ADD BX,2
38
            INC COUNTER
39
            MOV DX, COUNTER
40
41
            CMP DX,N
            JNE LOOP1
42
43
44
            LEA DX, CRLF
45
            MOV AH,09H
46
            INT 21H
47
48
            SUB BX,2
            DEC COUNTER
49
50
51
    L00P2:
52
            MOV AX, [BX]
53
            CALL WRITESINT
54
55
            SUB BX,2
56
            DEC COUNTER
57
            MOV DX, COUNTER
58
59
            CMP DX,0
            JNE LOOP2
60
61
            MOV AH,4CH
62
            INT 21H
63
64
65
    END START
```

An 8086 assembly language program to accept a series of numbers and print them in reversed order was written, assembled, linked and debugged to obtain expected output.

Program 5 Date 8 August 2012

Aim:

To write an assembly language program to accept n numbers and display the highest and lowest number.

Theory:

The program initially assumes the first element of the list to be the highest and the lowest element. It then traverses through the list, comparing the current high and low values with each element, and changing the variables if required. Once it reaches the end of the list, it print the contents of the highest and the lowest variables as the output.

Algorithm:

```
1> START

2> READ NUMBER OF ITEMS → N

3> INITIALISE COUNT = 0

4> READ ITEM → [ARRAY + COUNT]

5> COUNT = COUNT + 1

6> IF NOT COUNT == N GOTO 4

7> HIGH = [ARR]

8> LOW = [ARR]

9> COUNT = 0

10> IF [ARR + COUNT] > HIGH, HIGH = [ARR + COUNT]

11> IF [ARR + COUNT] < LOW, LOW = [ARR + COUNT]

12> COUNT = COUNT + 1

13> IF NOT COUNT == N, GOTO 10

14> PRINT HIGH, LOW

15> STOP
```

```
.8086
1
     .MODEL SMALL
    .STACK 512
4
    .DATA
                        DB "HOW MANY
DB "ENTER NU
DB 13,10,'$'
5
            MSG1
                           DB "HOW MANY ITEMS? $"
                           DB "ENTER NUMBERS: ",13,10,'$'
6
            MSG2
7
            CRLF
                          DB "
8
            SPACE
                                  $"
                           DW ?
9
            LOWEST
10
           HIGHEST
                           DW ?
           COUNTER
                           DW 0
11
12
            N
                           DW ?
            ARRAY
                           DW 100 DUP(?)
13
```

```
.CODE
14
15
            EXTRN READSINT: NEAR, WRITESINT: NEAR
16
    START:
17
            MOV AX,@DATA
18
            MOV DS,AX
19
20
            LEA DX,MSG1
                                           ; PRINT MSG1
21
            MOV AH,09H
            INT 21H
22
23
24
            CALL READSINT
                                           ;READ NUMBER OF ITEMS
            MOV N,AX
25
26
27
            LEA DX,MSG2
                                           ;PRINT MSG2
28
            MOV AH, 09H
29
            INT 21H
30
31
            LEA BX, ARRAY
32
    L00P1:
33
            CALL READSINT
                                           ; READ A NUMBER
34
            MOV [BX],AX
35
36
            ADD BX,2
                                           ; NEXT ITEM OF ARRAY
37
38
            INC COUNTER
                                           ;INCREMENT COUNTER
39
            MOV DX, COUNTER
40
                                           ; COMPARE COUNTER
41
            CMP DX,N
            JNE LOOP1
42
43
44
            LEA DX, CRLF
                                           ; PRINT NEWLINE
45
            MOV AH,09H
46
            INT 21H
47
48
            LEA BX, ARRAY
                                           ; INIT VALUES OF HIGHEST AND LOWEST
49
            MOV CX,[BX]
            MOV LOWEST, CX
50
51
            MOV CX,[BX]
            MOV HIGHEST, CX
52
53
            MOV BX,2
54
55
   L00P2:
            MOV CX,LOWEST
                                           ; COMPARE LOWEST AND ITEM OF ARRAY
56
57
            CMP CX,[BX]
58
            JNC JUMP1
59
            MOV CX,[BX]
60
            MOV LOWEST, CX
61
    JUMP1:
62
63
            MOV CX, HIGHEST
                                           ;COMPARE HIGHEST AND ITEM OF ARRAY
64
            CMP CX,[BX]
65
            JC JUMP2
66
            MOV CX, [BX]
            MOV HIGHEST, CX
67
68
    JUMP2:
69
            ADD BX,2
70
71
72
            INC COUNTER
73
            MOV DX, COUNTER
            CMP DX,N
74
75
            JL LOOP2
76
```

77	MOV AX,HIGHEST
78	CALL WRITESINT
79	
80	MOV AX,LOWEST
81	CALL WRITESINT
82	
83	MOV AH,4CH
84	INT 21H
85	END START

An 8086 assembly language program to accept a series of numbers and print the highest and the lowest number was written, assembled, linked and debugged to obtain expected output.

Program 6 Date 22 August 2012

Aim:

To write an assembly language program to accept n numbers and display them in

- ascending order
- descending order

using the sorting methods bubble, selection or insertion.

Theory:

A sorting algorithm is an algorithm that puts elements of an array in a certain order.

Bubble sort is a simple sorting algorithm. The algorithm starts at the beginning of the data set. It compares the first two elements, and if the first is greater than the second, it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass.

Selection sort is an in-place comparison sort. It has $O(n^2)$ complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists, and often is used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list.

Algorithm:

```
Bubble Sort
```

- 1> START
- 2> READ NUMBER OF ITEMS → N
- 3> READ ITEMS → ARR
- 4> INITIALISE COUNT 1 = 0
- 5> INITIALISE COUNT_2 = 0
- 6> IF [ARR + COUNT] > [ARR + COUNT + 1], SWAP [ARR + COUNT] AND [ARR + COUNT + 1]
- 7> COUNT 1 = COUNT 1 + 1
- 8> IF NOT COUNT_1 == N GOTO 6
- 9> COUNT 2 = COUNT 2 + 1
- 10>IF NOT COUNT_2 == N GOTO 5
- 11>PRINT ARR AS SORTED ITEMS
- 12>STOP

Selection Sort

- 1> START
- 2> READ NUMBER OF ITEMS → N
- 3> READ ITEMS → ARR
- 4> INITIALISE I = 0
- 5> J=I
- 6> K = I
- 7> WHILE K < N
- 8> IF A[K] < A[J], J = K
- 9 > K = K + 1
- 10>END LOOP WHILE
- 11>SWAP A[J], A[K]
- 12>1=1+1
- 13>IF I < N, GOTO 5
- 14>PRINT ARR AS SORTED ITEMS
- 15>STOP

Insertion Sort

- 1> START
- 2> READ NUMBER OF ITEMS → N
- 3> READ ITEMS → ARR
- 4> INITIALISE COUNT = 0
- 5> J = COUNT
- 6> WHILE J > 0 AND ARR[J] < ARR[J-1]
- 7> SWAP A[J], A[J-1]
- 8 > J = J 1
- 9> END LOOP WHILE
- 10>COUNT = COUNT + 1
- 11>IF COUNT < N, GOTO 5
- 12>PRINT ARR AS SORTED ITEMS
- 13>STOP

Source code:

Bubble Sort

- 1 .8086
- 2 .MODEL SMALL
- 3 .STACK 512
- 4 .DATA
- 5 ARR DW 34H,78H,56H,47H
- 6 SZ DW 4
- 7 .CODE
- 8 START:
- 9 MOV AX,@DATA
- 10 MOV DS, AX

```
11
             MOV BX,SZ
12
             DEC BX
13
    OUTLOOP:
14
            MOV CX,BX
15
             MOV SI,0
16
    INLOOP:
17
             MOV AL, ARR[SI]
             INC SI
18
             CMP AL,ARR[SI]
19
20
             JB CONTINUE
21
             XCHG AL,A[SI]
22
             MOV A[SI-1],AL
23
    CONTINUE:
             LOOP INLOOP
24
25
             DEC BX
             JNZ OUTLOOP
26
27
28
             MOV AH, 4CH
29
             INT 21H
30
    END START
    Selection Sort
1
     .8086
2
     .MODEL SMALL
3
     .STACK 512
4
     .DATA
5
             ARR
                            DW 44H,11H,22H,66H
6
                            DW 4
             SZ
7
                                                            ;TOTAL NO. OF COMPARISONS
             NC
                            DW ?
8
     .CODE
9
    START:
10
             MOV AX,@DATA
11
             MOV DS, AX
12
             MOV DX,SZ
13
14
             DEC DX
             MOV NC,0
15
    OUTLOOP:
16
             MOV CX,DX
17
             MOV SI,0
18
             MOV AH, ARR[SI]
19
20
             MOV BX,SI
21
    INLOOP:
22
             INC SI
23
             INC NC
             CMP AH, ARR[SI]
24
25
             JB CONTINUE
26
             MOV AH, ARR[SI]
27
             MOV BX,SI
28
    CONTINUE:
29
             LOOP INLOOP
30
             XCHG AH, ARR[SI]
31
            MOV ARR[BX],AH
32
             DEC DX
33
             JNZ OUTLOOP
34
35
             MOV AH,4CH
36
             INT 21H
37
    END START
```

```
Insertion Sort
1
     .8086
     .MODEL SMALL
3
     .STACK 512
4
     .DATA
5
            ARR
                            DW 78H,34H,12H,56H
6
            SZ
                            DW 4
7
     .CODE
8
    START:
            MOV AX,@DATA
9
            MOV DS,AX
10
11
            MOV CX,2
12
    OUTLOOP:
13
            MOV DX,CX
14
            DEC DX
15
            MOV SI,DX
16
17
            ADD SI,SI
18
            MOV AX, ARR[SI]
    INLOOP:
19
20
            CMP ARR[SI-2],AX
21
            JBE INLOOP_EXIT
22
            MOV DI,ARR[SI-2]
23
            MOV ARR[SI],DI
            DEC SI
24
25
            DEC SI
26
            DEC DX
27
            JNZ INLOOP
    INLOOP_EXIT:
28
29
            MOV ARR[SI],AX
30
            INC CX
31
            CMP CX,SZ
32
            JBE OUTLOOP
33
34
            MOV AH,4CH
35
            INT 21H
36
    END START
```

An 8086 assembly language program to sort a set of numbers using bubble, selection and insertion sort algorithms was written, assembled, linked and debugged to obtain expected output.

Program 7 Date 29 August 2012

Aim:

Write an assembly language program to implement linear search. Search n numbers for the key. If it is found, display its location or else print error message.

Theory:

Linear search or sequential search is a method for finding a particular value (the key) in a list, that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found. It is the simplest search algorithm; it is a special case of brute-force search.

Algorithm:

```
1> START
```

2> READ NUMBER OF ITEMS → N

3> READ ITEMS → ARR

4> READ KEY → KEY

5> INITIALISE COUNT = 0

6> IF KEY == [ARR+COUNT] GOTO 11

7> COUNT = COUNT + 1

8> IF NOT COUNT == N GOTO 6

9> PRINT KEY NOT FOUND

10>GOTO 12

11>PRINT KEY FOUND

12>STOP

```
.8086
1
    .MODEL SMALL
3
    .STACK 512
    .DATA
5
            ARR
                         DW 12,22,34,45,55
6
                          DW 5
            N
7
            KEY
                           DW 34
8
            PROMPT1
                           DB 13,10, "KEY FOUND AT INDEX $"
9
            PROMPT2
                           DB 13,10, "KEY NOT FOUND $"
10
    .CODE
            EXTRN READSINT: NEAR, WRITESINT: NEAR
11
    START:
12
            MOV AX,@DATA
13
            MOV DS,AX
14
15
16
            MOV SI,0
            MOV CX,0
17
            MOV BX, KEY
18
```

```
19
    AGAIN:
20
            CMP CX,N
21
            JZ NOTFOUND
22
           CMP BX,ARR[SI]
23
            JZ FOUND
24
            INC CX
            INC SI
25
            INC SI
26
            JMP AGAIN
27
28
   FOUND:
           LEA DX,PROMPT1
29
           MOV AH,09H
30
31
            INT 21H
32
           MOV AX,CX
33
            CALL WRITESINT
34
            JMP ENDPROG
35
   NOTFOUND:
           LEA DX,PROMPT2
36
37
            MOV AH,09H
38
            INT 21H
39
   ENDPROG:
40
           MOV AH,4CH
41
            INT 21H
42
   END START
```

An 8086 assembly language program to implement linear search was written, assembled, linked and debugged to obtain expected output.

Aim:

Write an assembly language program to implement binary search. Search n numbers for the key. If it is found, display its location or else print error message.

Theory:

The binary search algorithm finds the position of a specified value (key) within a sorted array. In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match, then a matching element has been found, so its index is returned. Otherwise, if the sought key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a "not found" indication is returned.

Algorithm:

```
1> START
```

2> READ NUMBER OF ITEMS → N

3> READ ITEMS → ARR

4> READ KEY → KEY

5> INITIALISE MIN = 0, MAX = N

6> IF MIN > MAX, GOTO 12

7> MID = (MIN + MAX)/2

8> IF ARR[MID] < KEY

a> MIN = MID + 1

b> GOTO 6

9> IF ARR[MID] > KEY

a> MAX = MID - 1

b> GOTO 6

10>PRINT FOUNT AT INDEX MID

11>GOTO 13

12>PRINT NOT FOUND

13>STOP

Source code:

```
    .8086
    .MODEL SMALL
    .STACK 512
```

4 .DATA

5 ARRAY DW 11,23,33,45,56

```
6
            LEN
                            DW 5
7
            KEY
                            DW 23
8
            SUCCESS
                            DB 13,10, "KEY FOUND AT INDEX $"
9
                            DB 13,10, "KEY WAS NOT FOUND$"
            FAILURE
10
    .CODE
            EXTRN READSINT: NEAR, WRITESINT: NEAR
11
12
    START:
13
            MOV AX,@DATA
14
            MOV DS,AX
15
            MOV BX,1
16
17
            MOV DX, LEN
            MOV CX, KEY
18
19
    AGAIN:
            CMP BX,DX
20
21
            JA NOTFOUND
22
            MOV AX, BX
23
            ADD AX, DX
24
            SHR AX,1
25
            MOV SI,AX
            DEC SI
26
27
            ADD SI,SI
28
            CMP CX,ARRAY[SI]
29
            JAE BIGGER
30
            DEC AX
31
            MOV DX,AX
32
            JMP AGAIN
   BIGGER:
33
            JE FOUND
34
35
            INC AX
            MOV BX,AX
36
37
            JMP AGAIN
    FOUND:
38
39
            MOV SI,AX
40
            LEA DX, SUCCESS
41
            MOV AH,09H
42
            INT 21H
43
            DEC SI
            MOV AX,SI
44
45
            CALL WRITESINT
            JMP ENDPROG
46
47
    NOTFOUND:
48
            LEA DX, FAILURE
49
            MOV AH,09H
50
            INT 21H
51
    ENDPROG:
52
            MOV AH, 4CH
53
            INT 21H
54
    END START
```

An 8086 assembly language program to implement binary search was written, assembled, linked and debugged to obtain expected output.

Aim:

To write an assembly language program to accept n consecutive numbers with one number missing. Determine the missing number.

Theory:

The program maintains a special variable that is initialized to the first value of the array. In every iteration, this value is incremented and compared with the next value of the array. If there is a mismatch, then the current value is printed as the missing number. If the comparisons proceed till the end of the array, then the list has no missing numbers.

Algorithm:

```
1> START

2> READ NUMBER OF ITEMS → N

3> READ ITEMS → ARR

4> INITIALISE COUNT = 0, TEMP = [ARR]

5> IF COUNT == N GOTO 9

6> IF TEMP == [ARR+COUNT]

a> TEMP = TEMP + 1

b> COUNT = COUNT + 1

c> GOTO 5

7> PRINT MISSING NUMBER AS TEMP

8> GOTO 10

9> PRINT NO MISSING NUMBERS FOUND

10>STOP
```

```
1
      .8086
      .MODEL SMALL
 2
 3
     .STACK 512
 4
     .DATA
 5
                                  DW ?
               N
              ARR DW 64 DUP(?)
MSG1 DB 13,10,"HOW MANY ITEMS? $"
MSG2 DB 13,10,"ENTER ITEMS: ",13,10,'$'
MSG3 DB 13,10,"MISSTNG TTEM. #"
 6
               TMP
                                  DW ?
 7
 8
 9
10
                                   DB 13,10,"NO MISSING ITEM ",13,10,'$'
11
               MSG4
12
13
    .CODE
14
               EXTRN READSINT: NEAR, WRITESINT: NEAR
15
     START:
```

```
16
             MOV AX,@DATA
17
             MOV DS,AX
18
             LEA DX,MSG1
19
20
             MOV AH,09H
             INT 21H
21
22
23
             CALL READSINT
24
             MOV N,AX
25
26
             LEA DX,MSG2
27
             MOV AH,09H
28
             INT 21H
29
30
             MOV SI,0
31
             MOV CX, N
32
     RL00P:
33
             CALL READSINT
34
             MOV ARR[SI],AX
35
             INC SI
             INC SI
36
37
             LOOP RLOOP
38
39
             MOV SI,0
40
             MOV CX,N
41
             MOV DX, ARR[SI]
    CL00P:
42
43
             DEC CX
44
             CMP CX,0
45
             JZ NM
                                                     ;NO MISSING ITEMS
46
             INC SI
47
             INC SI
48
             INC DX
49
             CMP DX, ARR[SI]
50
             JZ CLOOP
51
52
             MOV TMP, DX
53
             LEA DX,MSG3
                                                     ;MISSING ITEM FOUND
             MOV AH,09H
54
55
             INT 21H
56
57
             MOV AX, TMP
58
             CALL WRITESINT
59
             JMP ENDPROG
60
61
     NM:
62
             LEA DX, MSG4
             MOV AH,09H
63
             INT 21H
64
65
66
     ENDPROG:
67
             MOV AH, 4CH
68
             INT 21H
     END START
```

An 8086 assembly language program a missing number in a series of continuous numbers was written, assembled, linked and debugged to obtain expected output.

Aim:

To write an 8086/8087 assembly language program to find the following:

- Area of a circle
- Area of a rectangle
- Volume of a cube

Theory:

The 8087 is a floating-point coprocessor for the 8086 line of microprocessors. Its purpose was to speed up computations for floating-point arithmetic, such as addition, subtraction, multiplication, division, and square root. The 8087 does not use a directly addressable register set such as the main registers of 8086; instead, the 8087 registers form an eight-level stack ranging from st(0) to st(7), where st(0) is the top. The 8087 instructions operate by pushing, calculating, and popping values on this stack. They can either take explicit operands or implicitly assume st(0) and st(1) as the operands.

Source code:

```
Area of a circle
     .8087
1
    .MODEL SMALL
2
3
    .STACK 512
4
     .DATA
5
             RAD
                     DD 5.0
6
             AREA
                     DD ?
7
     .CODE
8
    START:
9
             MOV AX,@DATA
             MOV DS, AX
10
11
12
             FLD RAD
             FMUL ST(0),ST(0)
13
14
15
             FLDPI
             FMUL ST(0),ST(1)
16
17
18
             FST AREA
19
             MOV AX,4C00H
20
             INT 21H
21
22
23
    END START
```

Area of a rectangle

```
1 .8087
2 .MODEL SMALL
3 .STACK 512
4 .DATA
```

```
5
             L
                     DD 5.0
6
                     DD 3.0
7
             AREA
                     DD ?
8
    .CODE
9
    START:
10
             MOV AX,@DATA
11
             MOV DS,AX
12
             FLD L
13
             FLD B
14
15
16
             FMUL ST(0),ST(1)
17
             FST AREA
18
19
             MOV AH, 4CH
20
21
             INT 21H
22
    END START
    Volume of a cube
1
     .8087
2
     .MODEL SMALL
3
     .STACK 512
4
    .DATA
5
             LENGTH
                     DD 2.0
6
             VOLUME
                       DD ?
7
     .CODE
8
    START:
9
             MOV AX,@DATA
10
             MOV DS, AX
11
             FLD LENGTH
12
13
             FMUL ST,ST(0)
14
15
             FMUL ST, ST(0)
16
17
             FST VOLUME
18
19
             MOV AH,4CH
20
             INT 21H
```

END START

21

An 8086/8087 assembly language programs to find area of a circle, area of a rectangle and volume of a cube were written, assembled, linked and debugged to obtain expected output.

Aim:

To write 8086/8087 assembly language program to prove the following trigonometric identities:

```
    sin²(x) + cos²(x) = 1
    tan²(x) + 1 = sec²(x)
    cot²(x) + 1 = cosec²(x)
    cos(a+b) = cos(a) cos(b) - sin(a) sin(b)
    sin(a+b) = sin(a) cos(b) + cos(a) sin(b)
```

Theory:

Several trigonometric identities can be proven using the 8087s fptan instruction. The fptan replaces st(1) with numerator and st(0) with the denominator of the partial tangent of the operand. These values are further utilized to obtain other trigonometric ratios as required.

```
\sin^2(x) + \cos^2(x) = 1
     .8087
1
 2
     .MODEL SMALL
 3
     .STACK 512
 4
     .DATA
 5
             Χ
                             DD 1.2
                             DD ?
 6
             Α
7
                             DD ?
             0
8
                             DD ?
             Н
             RESULT
                             DD ?
9
     .CODE
10
11
     START:
12
             MOV AX,@DATA
13
             MOV DS,AX
14
             FINIT
15
16
             FLD X
17
18
             FPTAN
19
             FSTP A
20
             FST 0
21
             FMUL ST, ST(0)
22
23
             FLD A
24
             FMUL ST, ST(0)
25
             FADD
26
             FSQRT
27
             FST H
28
             FINIT
29
30
             FLD 0
31
             FLD H
32
             FDIV
```

```
33
             FMUL ST, ST(0)
34
35
             FLD A
36
             FLD H
37
             FDIV
38
             FMUL ST,ST(0)
39
40
             FADD
41
             FST RESULT
42
             MOV AH,4CH
43
44
             INT 21H
45
     END START
    tan^{2}(x) - sec^{2}(x) = -1
     .8087
1
     .MODEL SMALL
2
3
     .STACK 512
4
     .DATA
5
                            DD 1.2
             Χ
6
                            DD ?
             Α
7
             0
                            DD ?
8
                            DD ?
             Н
9
             RESULT
                            DD ?
10
     .CODE
11
     START:
12
             MOV AX,@DATA
13
             MOV DS,AX
14
15
             FINIT
             FLD X
16
             FPTAN
17
18
             FSTP A
19
20
             FST 0
21
22
             FMUL ST, ST(0)
23
             FLD A
24
             FMUL ST, ST(0)
25
             FADD
26
             FSQRT
27
             FSTP H
28
             FINIT
29
30
             FLD X
31
             FPTAN
32
             FDIV
33
             FMUL ST,ST(0)
34
35
             FLD H
36
             FLD A
37
             FDIV
38
             FMUL ST,ST(0)
39
             FSUB
             FST RESULT
40
41
             MOV AH,4CH
42
43
             INT 21H
44
     END START
```

```
\cot^2(x) - \csc^2(x) = -1
1
     .8087
2
3
     .MODEL SMALL
     .STACK 512
 4
     .DATA
5
             Χ
                             DD 1.2
6
             Α
                             DD ?
7
             0
                             DD ?
8
                             DD ?
             Н
9
             RESULT
                             DD ?
10
     .CODE
11
     START:
             MOV AX,@DATA
12
             MOV DS,AX
13
14
15
             FINIT
             FLD X
16
17
             FPTAN
18
19
             FSTP A
20
             FST 0
21
22
             FMUL ST,ST(0)
             FLD A
23
24
             FMUL ST,ST(0)
25
             FADD
26
             FSQRT
27
             FSTP H
28
29
             FINIT
30
             FLD A
31
             FLD 0
32
             FDIV
33
             FMUL ST,ST(0)
34
35
             FLD H
36
             FLD 0
37
             FDIV
38
             FMUL ST,ST(0)
39
40
             FSUB
41
             FST RESULT
42
43
             MOV AH,4CH
44
             INT 21H
45
     END START
     cos(a+b) - cos(a)cos(b) + sin(a)sin(b) = 0
1
     .8087
     .MODEL SMALL
2
3
     .STACK 512
4
     .DATA
5
             Α
                             DD 1.2
                             DD 1.0
6
             В
7
             ADJ_AB
                             DD ?
8
             OPP_AB
                             DD ?
9
             HYP_AB
                             DD ?
10
             ADJ_A
                             DD ?
11
             OPP_A
                             DD ?
                             DD ?
12
             HYP_A
13
             ADJ_B
                             DD ?
             OPP_B
                             DD ?
14
```

```
HYP B
15
                            DD ?
16
             RESULT
                            DD ?
17
     .CODE
18
    START:
19
             MOV AX,@DATA
            MOV DS,AX
20
21
22
             FINIT
23
             FLD A
24
             FLD B
25
             FADD
26
             FPTAN
27
28
             FSTP ADJ_AB
29
             FST OPP_AB
             FMUL ST,ST(0)
30
31
             FLD ADJ AB
32
             FMUL ST,ST(0)
33
             FADD
34
             FSQRT
35
             FST HYP_AB
36
37
             FINIT
38
             FLD A
             FPTAN
39
40
             FSTP ADJ_A
41
             FST OPP_A
42
             FMUL ST,ST(0)
43
             FLD ADJ_A
44
             FMUL ST,ST(0)
45
             FADD
46
             FSQRT
            FST HYP_A
47
48
49
             FINIT
50
             FLD B
51
             FPTAN
52
             FSTP ADJ_B
53
             FST OPP_B
54
             FMUL ST,ST(0)
55
             FLD ADJ_B
            FMUL ST,ST(0)
56
57
             FADD
58
             FSQRT
59
             FST HYP_B
60
61
             FINIT
62
             FLD ADJ AB
63
             FLD HYP_AB
             FDIV
64
65
66
             FLD ADJ A
67
             FLD HYP_A
68
             FDIV
69
             FLD ADJ_B
70
71
             FLD HYP_B
72
             FDIV
73
74
            FMUL
75
             FSUB
76
77
             FLD OPP_A
```

```
78
             FLD HYP_A
79
             FDIV
80
81
             FLD OPP_B
             FLD HYP_B
82
83
             FDIV
84
85
             FMUL
             \mathsf{FADD}
86
87
88
             FST RESULT
89
             MOV AH,4CH
90
91
             INT 21H
92
     END START
     sin(a+b) - sin(a)cos(b) - cos(a)sin(b) = 0
     .8087
1
2
     .MODEL SMALL
3
     .STACK 512
4
     .DATA
5
             Α
                             DD 1.2
6
             В
                             DD 1.0
7
             ADJ_AB
                             DD ?
8
             OPP_AB
                             DD ?
                             DD ?
9
             HYP_AB
10
             ADJ_A
                             DD ?
11
             OPP_A
                             DD ?
12
             HYP_A
                             DD ?
                             DD ?
13
             ADJ_B
14
             OPP B
                             DD ?
15
             HYP B
                             DD ?
             RESULT
16
                             DD ?
17
     .CODE
18
     START:
19
             MOV AX,@DATA
20
             MOV DS,AX
21
22
             FINIT
23
             FLD A
24
             FLD B
25
             FADD
26
             FPTAN
27
28
             FSTP ADJ_AB
             FST OPP_AB
29
             FMUL ST,ST(0)
30
31
             FLD ADJ_AB
32
             FMUL ST, ST(0)
33
             FADD
34
             FSQRT
35
             FST HYP_AB
36
37
             FINIT
38
             FLD A
39
             FPTAN
40
             FSTP ADJ_A
41
             FST OPP_A
42
             FMUL ST,ST(0)
43
             FLD ADJ_A
44
             FMUL ST,ST(0)
45
             FADD
```

```
46
             FSQRT
47
             FST HYP_A
48
49
             FINIT
50
             FLD B
51
             FPTAN
52
             FSTP ADJ_B
             FST OPP_B
53
             FMUL ST,ST(0)
54
55
             FLD ADJ_B
56
             FMUL ST,ST(0)
57
             FADD
58
             FSQRT
59
             FST HYP_B
60
61
             FINIT
62
             FLD OPP_AB
63
             FLD HYP_AB
64
             FDIV
65
66
             FLD OPP_A
67
             FLD HYP_A
68
             FDIV
69
70
             FLD ADJ_B
71
             FLD HYP_B
72
             FDIV
73
             FMUL
74
75
             FSUB
76
77
             FLD ADJ A
78
             FLD HYP_A
79
             FDIV
80
81
             FLD OPP B
82
             FLD HYP_B
83
             FDIV
84
85
             FMUL
             FSUB
86
87
             FST RESULT
88
89
            MOV AH,4CH
90
91
             INT 21H
92
    END START
```

8086/8087 assembly language programs to prove common trigonometric identities were written, assembled, linked and debugged to obtain expected output.