­­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

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

NUM1 DB 13,10,"ENTER VALUE OF X: ",'$'

NUM2 DB 13,10,"ENTER VALUE OF Y: ",'$'

AD DB 13,10,"X + Y = ",'$'

SB DB 13,10,"X - Y = ",'$'

ML DB 13,10,"X \* Y = ",'$'

DV DB 13,10,"X / Y = ",'$'

N1 DW ?

N2 DW ?

.CODE

EXTRN READSINT:NEAR,WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,NUM1

MOV AH,09H

INT 21H

CALL READSINT

MOV N1,AX

LEA DX,NUM2

MOV AH,09H

INT 21H

CALL READSINT

MOV N2,AX

LEA DX,AD

MOV AH,09H

INT 21H

MOV AX,N1

ADD AX,N2

CALL WRITESINT

LEA DX,SB

MOV AH,09H

INT 21H

MOV AX,N1

SUB AX,N2

CALL WRITESINT

LEA DX,ML

MOV AH,09H

INT 21H

MOV AX,N1

MOV BX,N2

MUL BX

CALL WRITESINT

LEA DX,DV

MOV AH,09H

INT 21H

MOV DX,0

MOV AX,N1

MOV BX,N2

DIV BX

CALL WRITESINT

MOV AH,4CH

INT 21H

END START

Conclusion:

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
9. STOP

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

X DW ?

Y DW ?

CRLF DB 13,10,'$'

PROMPT1 DB 13,10,"ENTER FIRST NUMBER: ",'$'

PROMPT DB 13,10,"ENTER THE SECOND NUMBER: ",'$'

PROMPT2 DB 13,10,"BEFORE SWAPPING: "13,10,'$'

PROMPT3 DB 13,10,"AFTER SWAPPING: ",13,10,'$'

.CODE

EXTRN READSINT:NEAR, WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,PROMPT1

MOV AH,09H

INT 21H

CALL READSINT

MOV X,AX

LEA DX,PROMPT

MOV AH,09H

INT 21H

CALL READSINT

MOV Y,AX

LEA DX,PROMPT2

MOV AH,09H

INT 21H

MOV AX,X

CALL WRITESINT

MOV AX,Y

CALL WRITESINT

MOV AX,X

ADD AX,BX

MOV BX,AX

SUB BX,Y

SUB AX,BX

MOV X,AX

MOV Y,BX

LEA DX,PROMPT3

MOV AH,09H

INT 21H

MOV AX,X

CALL WRITESINT

MOV AX,Y

CALL WRITESINT

MOV AH,4CH

INT 21H

END START

Conclusion:

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:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …

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

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

MSG1 DB "ENTER ORDER: ",$

MSG2 DB "FIBONACCI SERIES: ",13,10.'$'

N DW ?

.CODE

EXTRN READSINT:NEAR,WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,09H

INT 21H

CALL READSINT

MOV N,AX

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,1

LEA DX,MSG2

MOV AH,09H

INT 21H

LOOP0:

CALL WRITESINT

MOV AX,BX

ADD AX,DX

MOV DX,AX

INC CX

CMP CX,N

JNZ LOOP0

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

MSG1 DB "HOW MANY ITEMS? $"

MSG2 DB "ENTER NUMBERS: ",13,10,'$'

CRLF DB 13,10,'$'

SPACE DB " $"

COUNTER DW 1

N DW ?

ARRAY DW 100 DUP(?)

.CODE

EXTRN READSINT:NEAR, WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,09H

INT 21H

CALL READSINT

MOV N,AX

ADD N,1

LEA DX,MSG2

MOV AH,09H

INT 21H

MOV CX,N

LEA BX,ARRAY

LOOP1:

CALL READSINT

MOV [BX],AX

ADD BX,2

INC COUNTER

MOV DX,COUNTER

CMP DX,N

JNE LOOP1

LEA DX,CRLF

MOV AH,09H

INT 21H

SUB BX,2

DEC COUNTER

LOOP2:

MOV AX,[BX]

CALL WRITESINT

SUB BX,2

DEC COUNTER

MOV DX,COUNTER

CMP DX,0

JNE LOOP2

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

MSG1 DB "HOW MANY ITEMS? $"

MSG2 DB "ENTER NUMBERS: ",13,10,'$'

CRLF DB 13,10,'$'

SPACE DB " $"

LOWEST DW ?

HIGHEST DW ?

COUNTER DW 0

N DW ?

ARRAY DW 100 DUP(?)

.CODE

EXTRN READSINT:NEAR, WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,MSG1 ;PRINT MSG1

MOV AH,09H

INT 21H

CALL READSINT ;READ NUMBER OF ITEMS

MOV N,AX

LEA DX,MSG2 ;PRINT MSG2

MOV AH,09H

INT 21H

LEA BX,ARRAY

LOOP1:

CALL READSINT ;READ A NUMBER

MOV [BX],AX

ADD BX,2 ;NEXT ITEM OF ARRAY

INC COUNTER ;INCREMENT COUNTER

MOV DX,COUNTER

CMP DX,N ;COMPARE COUNTER

JNE LOOP1

LEA DX,CRLF ;PRINT NEWLINE

MOV AH,09H

INT 21H

LEA BX,ARRAY ;INIT VALUES OF HIGHEST AND LOWEST

MOV CX,[BX]

MOV LOWEST,CX

MOV CX,[BX]

MOV HIGHEST,CX

MOV BX,2

LOOP2:

MOV CX,LOWEST ;COMPARE LOWEST AND ITEM OF ARRAY

CMP CX,[BX]

JNC JUMP1

MOV CX,[BX]

MOV LOWEST,CX

JUMP1:

MOV CX,HIGHEST ;COMPARE HIGHEST AND ITEM OF ARRAY

CMP CX,[BX]

JC JUMP2

MOV CX,[BX]

MOV HIGHEST,CX

JUMP2:

ADD BX,2

INC COUNTER

MOV DX,COUNTER

CMP DX,N

JL LOOP2

MOV AX,HIGHEST

CALL WRITESINT

MOV AX,LOWEST

CALL WRITESINT

MOV AH,4CH

INT 21H

END START

Conclusion:

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(n2) 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. I = I + 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

.8086

.MODEL SMALL

.STACK 512

.DATA

ARR DW 34H,78H,56H,47H

SZ DW 4

.CODE

START:

MOV AX,@DATA

MOV DS,AX

MOV BX,SZ

DEC BX

OUTLOOP:

MOV CX,BX

MOV SI,0

INLOOP:

MOV AL,ARR[SI]

INC SI

CMP AL,ARR[SI]

JB CONTINUE

XCHG AL,A[SI]

MOV A[SI-1],AL

CONTINUE:

LOOP INLOOP

DEC BX

JNZ OUTLOOP

MOV AH,4CH

INT 21H

END START

Selection Sort

.8086

.MODEL SMALL

.STACK 512

.DATA

ARR DW 44H,11H,22H,66H

SZ DW 4

NC DW ? ;TOTAL NO. OF COMPARISONS

.CODE

START:

MOV AX,@DATA

MOV DS,AX

MOV DX,SZ

DEC DX

MOV NC,0

OUTLOOP:

MOV CX,DX

MOV SI,0

MOV AH,ARR[SI]

MOV BX,SI

INLOOP:

INC SI

INC NC

CMP AH,ARR[SI]

JB CONTINUE

MOV AH,ARR[SI]

MOV BX,SI

CONTINUE:

LOOP INLOOP

XCHG AH,ARR[SI]

MOV ARR[BX],AH

DEC DX

JNZ OUTLOOP

MOV AH,4CH

INT 21H

END START

Insertion Sort

.8086

.MODEL SMALL

.STACK 512

.DATA

ARR DW 78H,34H,12H,56H

SZ DW 4

.CODE

START:

MOV AX,@DATA

MOV DS,AX

MOV CX,2

OUTLOOP:

MOV DX,CX

DEC DX

MOV SI,DX

ADD SI,SI

MOV AX,ARR[SI]

INLOOP:

CMP ARR[SI-2],AX

JBE INLOOP\_EXIT

MOV DI,ARR[SI-2]

MOV ARR[SI],DI

DEC SI

DEC SI

DEC DX

JNZ INLOOP

INLOOP\_EXIT:

MOV ARR[SI],AX

INC CX

CMP CX,SZ

JBE OUTLOOP

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

ARR DW 12,22,34,45,55

N DW 5

KEY DW 34

PROMPT1 DB 13,10,"KEY FOUND AT INDEX $"

PROMPT2 DB 13,10,"KEY NOT FOUND $"

.CODE

EXTRN READSINT:NEAR,WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

MOV SI,0

MOV CX,0

MOV BX,KEY

AGAIN:

CMP CX,N

JZ NOTFOUND

CMP BX,ARR[SI]

JZ FOUND

INC CX

INC SI

INC SI

JMP AGAIN

FOUND:

LEA DX,PROMPT1

MOV AH,09H

INT 21H

MOV AX,CX

CALL WRITESINT

JMP ENDPROG

NOTFOUND:

LEA DX,PROMPT2

MOV AH,09H

INT 21H

ENDPROG:

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Program 8 Date 5 September 2012

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
   1. MIN = MID + 1
   2. GOTO 6
9. IF ARR[MID] > KEY
   1. MAX = MID – 1
   2. GOTO 6
10. PRINT FOUNT AT INDEX MID
11. GOTO 13
12. PRINT NOT FOUND
13. STOP

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

ARRAY DW 11,23,33,45,56

LEN DW 5

KEY DW 23

SUCCESS DB 13,10,"KEY FOUND AT INDEX $"

FAILURE DB 13,10,"KEY WAS NOT FOUND$"

.CODE

EXTRN READSINT:NEAR,WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

MOV BX,1

MOV DX,LEN

MOV CX,KEY

AGAIN:

CMP BX,DX

JA NOTFOUND

MOV AX,BX

ADD AX,DX

SHR AX,1

MOV SI,AX

DEC SI

ADD SI,SI

CMP CX,ARRAY[SI]

JAE BIGGER

DEC AX

MOV DX,AX

JMP AGAIN

BIGGER:

JE FOUND

INC AX

MOV BX,AX

JMP AGAIN

FOUND:

MOV SI,AX

LEA DX,SUCCESS

MOV AH,09H

INT 21H

DEC SI

MOV AX,SI

CALL WRITESINT

JMP ENDPROG

NOTFOUND:

LEA DX,FAILURE

MOV AH,09H

INT 21H

ENDPROG:

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Program 9 Date 12 September 2012

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]
   1. TEMP = TEMP + 1
   2. COUNT = COUNT + 1
   3. GOTO 5
7. PRINT MISSING NUMBER AS TEMP
8. GOTO 10
9. PRINT NO MISSING NUMBERS FOUND
10. STOP

Source code:

.8086

.MODEL SMALL

.STACK 512

.DATA

N DW ?

TMP DW ?

ARR DW 64 DUP(?)

MSG1 DB 13,10,"HOW MANY ITEMS? $"

MSG2 DB 13,10,"ENTER ITEMS: ",13,10,'$'

MSG3 DB 13,10,"MISSING ITEM: $"

MSG4 DB 13,10,"NO MISSING ITEM ",13,10,'$'

.CODE

EXTRN READSINT:NEAR,WRITESINT:NEAR

START:

MOV AX,@DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,09H

INT 21H

CALL READSINT

MOV N,AX

LEA DX,MSG2

MOV AH,09H

INT 21H

MOV SI,0

MOV CX,N

RLOOP:

CALL READSINT

MOV ARR[SI],AX

INC SI

INC SI

LOOP RLOOP

MOV SI,0

MOV CX,N

MOV DX,ARR[SI]

CLOOP:

DEC CX

CMP CX,0

JZ NM ;NO MISSING ITEMS

INC SI

INC SI

INC DX

CMP DX,ARR[SI]

JZ CLOOP

MOV TMP,DX

LEA DX,MSG3 ;MISSING ITEM FOUND

MOV AH,09H

INT 21H

MOV AX,TMP

CALL WRITESINT

JMP ENDPROG

NM:

LEA DX,MSG4

MOV AH,09H

INT 21H

ENDPROG:

MOV AH,4CH

INT 21H

END START

Conclusion:

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

Program 10 Date 19 September 2012

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

.MODEL SMALL

.STACK 512

.DATA

RAD DD 5.0

AREA DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FLD RAD

FMUL ST(0),ST(0)

FLDPI

FMUL ST(0),ST(1)

FST AREA

MOV AX,4C00H

INT 21H

END START

Area of a rectangle

.8087

.MODEL SMALL

.STACK 512

.DATA

L DD 5.0

B DD 3.0

AREA DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FLD L

FLD B

FMUL ST(0),ST(1)

FST AREA

MOV AH,4CH

INT 21H

END START

Volume of a cube

.8087

.MODEL SMALL

.STACK 512

.DATA

LENGTH DD 2.0

VOLUME DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FLD LENGTH

FMUL ST,ST(0)

FMUL ST,ST(0)

FST VOLUME

MOV AH,4CH

INT 21H

END START

Conclusion:

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.

Program 11 Date 26 September 2012

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.

Source code:

sin2(x) + cos2(x) = 1

.8087

.MODEL SMALL

.STACK 512

.DATA

X DD 1.2

A DD ?

O DD ?

H DD ?

RESULT DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FINIT

FLD X

FPTAN

FSTP A

FST O

FMUL ST,ST(0)

FLD A

FMUL ST,ST(0)

FADD

FSQRT

FST H

FINIT

FLD O

FLD H

FDIV

FMUL ST,ST(0)

FLD A

FLD H

FDIV

FMUL ST,ST(0)

FADD

FST RESULT

MOV AH,4CH

INT 21H

END START

tan2(x) - sec2(x) = -1

.8087

.MODEL SMALL

.STACK 512

.DATA

X DD 1.2

A DD ?

O DD ?

H DD ?

RESULT DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FINIT

FLD X

FPTAN

FSTP A

FST O

FMUL ST,ST(0)

FLD A

FMUL ST,ST(0)

FADD

FSQRT

FSTP H

FINIT

FLD X

FPTAN

FDIV

FMUL ST,ST(0)

FLD H

FLD A

FDIV

FMUL ST,ST(0)

FSUB

FST RESULT

MOV AH,4CH

INT 21H

END START

cot2(x) - cosec2(x) = -1

.8087

.MODEL SMALL

.STACK 512

.DATA

X DD 1.2

A DD ?

O DD ?

H DD ?

RESULT DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FINIT

FLD X

FPTAN

FSTP A

FST O

FMUL ST,ST(0)

FLD A

FMUL ST,ST(0)

FADD

FSQRT

FSTP H

FINIT

FLD A

FLD O

FDIV

FMUL ST,ST(0)

FLD H

FLD O

FDIV

FMUL ST,ST(0)

FSUB

FST RESULT

MOV AH,4CH

INT 21H

END START

cos(a+b) - cos(a)cos(b) + sin(a)sin(b) = 0

.8087

.MODEL SMALL

.STACK 512

.DATA

A DD 1.2

B DD 1.0

ADJ\_AB DD ?

OPP\_AB DD ?

HYP\_AB DD ?

ADJ\_A DD ?

OPP\_A DD ?

HYP\_A DD ?

ADJ\_B DD ?

OPP\_B DD ?

HYP\_B DD ?

RESULT DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FINIT

FLD A

FLD B

FADD

FPTAN

FSTP ADJ\_AB

FST OPP\_AB

FMUL ST,ST(0)

FLD ADJ\_AB

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_AB

FINIT

FLD A

FPTAN

FSTP ADJ\_A

FST OPP\_A

FMUL ST,ST(0)

FLD ADJ\_A

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_A

FINIT

FLD B

FPTAN

FSTP ADJ\_B

FST OPP\_B

FMUL ST,ST(0)

FLD ADJ\_B

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_B

FINIT

FLD ADJ\_AB

FLD HYP\_AB

FDIV

FLD ADJ\_A

FLD HYP\_A

FDIV

FLD ADJ\_B

FLD HYP\_B

FDIV

FMUL

FSUB

FLD OPP\_A

FLD HYP\_A

FDIV

FLD OPP\_B

FLD HYP\_B

FDIV

FMUL

FADD

FST RESULT

MOV AH,4CH

INT 21H

END START

sin(a+b) - sin(a)cos(b) - cos(a)sin(b) = 0

.8087

.MODEL SMALL

.STACK 512

.DATA

A DD 1.2

B DD 1.0

ADJ\_AB DD ?

OPP\_AB DD ?

HYP\_AB DD ?

ADJ\_A DD ?

OPP\_A DD ?

HYP\_A DD ?

ADJ\_B DD ?

OPP\_B DD ?

HYP\_B DD ?

RESULT DD ?

.CODE

START:

MOV AX,@DATA

MOV DS,AX

FINIT

FLD A

FLD B

FADD

FPTAN

FSTP ADJ\_AB

FST OPP\_AB

FMUL ST,ST(0)

FLD ADJ\_AB

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_AB

FINIT

FLD A

FPTAN

FSTP ADJ\_A

FST OPP\_A

FMUL ST,ST(0)

FLD ADJ\_A

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_A

FINIT

FLD B

FPTAN

FSTP ADJ\_B

FST OPP\_B

FMUL ST,ST(0)

FLD ADJ\_B

FMUL ST,ST(0)

FADD

FSQRT

FST HYP\_B

FINIT

FLD OPP\_AB

FLD HYP\_AB

FDIV

FLD OPP\_A

FLD HYP\_A

FDIV

FLD ADJ\_B

FLD HYP\_B

FDIV

FMUL

FSUB

FLD ADJ\_A

FLD HYP\_A

FDIV

FLD OPP\_B

FLD HYP\_B

FDIV

FMUL

FSUB

FST RESULT

MOV AH,4CH

INT 21H

END START

Conclusion:

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