|  |
| --- |
| **External Project Report on**  **Computer Organization and Architecture**  **(EET 2211)** |

**Statistical Calculator in 8086 assembly language**



**Submitted by**

|  |  |  |
| --- | --- | --- |
| **Name 01: Subham Swastik Pradhan** | **Reg. No.:** | **2341016355** |
| **Name 02: Smruti Panigrahi** | **Reg. No.:** | **2341016156** |
| **Name 03: Debabrata Prusty** | **Reg. No.:** | **2341016479** |
| **Name 04: Rishi Priyadarshi** | **Reg. No.:** | **2341016453** |

**B. Tech. CSE 4th Semester (Section – 2341-2P1)**

# Declaration

**INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH**

**(FACULTY OF ENGINEERING)**

**SIKSHA ‘O’ ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA**

We, the undersigned students of B. Tech. of **CSE** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**Statistical Calculator in 8086 assembly language**” submitted to **Siksha ‘O’ Anusandhan Deemed to be University, Bhubaneswar** for the partial fulfillment of the subject **Computer Organization and Architecture (EET 2211)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

|  |  |
| --- | --- |
| **Subham Swastik Pradhan**  **Registration No.: 2341016355** | **Smruti Panigrahi**  **Registration No.: 2341016156** |
| **Rishi Priyadarshi**  **Registration No.: 2341016453** | **Debabrata Prusty**  **Registration No.: 2341016479** |

**DATE: 29.05.2025**

**PLACE: ITER, BBSR,**

**ODISHA**

# Abstract

This project presents the design and implementation of a statistical calculator using 8086 Assembly language. The calculator performs essential statistical operations, including calculation of mean, median, and standard deviation. By leveraging the efficiency and low-level memory management capabilities of Assembly language, this calculator demonstrates the potential for high-performance statistical computations in resource-constrained environments.

The calculator's design focuses on algorithmic efficiency, accurate calculations, and user-friendly input/output operations. This project showcases the application of 8086 Assembly language in statistical analysis and provides a foundation for further development of statistical tools in Assembly language programming.

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial No.** | **Chapter No.** | **Title of the Chapter** | **Page No.** |
|  | 1 | Introduction |  |
|  | 2 | Problem Statement |  |
|  | 3 | Methodology |  |
|  | 4 | Implementation |  |
|  | 5 | Results and interpretation |  |
|  | 6 | Conclusion |  |
|  |  | References |  |
|  |  | Appendices |  |

# **1. INTRODUCTION**

In the realm of statistical analysis, calculators play a vital role in simplifying complex calculations, enabling users to focus on interpreting results rather than tedious computations. This project aims to design a statistical calculator using 8086 Assembly language, a low-level programming language that provides direct access to hardware resources.

**Why 8086 Assembly Language?**

**1. Efficiency**: 8086 Assembly language allows for optimized code, resulting in faster execution times.

**2. Low-Level Memory Management:** Direct access to memory enables efficient data storage and manipulation.

**3. Educational Value:** Learning Assembly language programming helps developers understand computer architecture and software hardware interactions.

**Statistical Calculator Functionalities**:

The calculator will perform the following statistical operations:

1. **Count**: Number of elements
2. **Sum**:
3. **Arithmetic Mean Calculation**: Calculate the average value of a dataset.
4. **Median**:
5. **Mode Calculation**: Find the most frequent value of a sorted dataset.
6. **Max**:
7. **Min**:
8. **Standard Deviation Calculation**: Measure the spread or dispersion of a dataset.
9. **Variance:** Measures the spread of samples

**Project Goals:**

1. Design an efficient statistical calculator using 8086 Assembly language.

2. Implement accurate and reliable statistical calculations.

3. Provide a user-friendly interface for input and output operations.

# **2. Problem Statement**

Design a statistical calculator in 8086 Assembly language to perform mean, median, and standard deviation calculations on a set of numbers.

**Identification of Input and Output Variables:**

**Input Variables:**

1. Set of numbers (dataset)
   1. Size of dataset
   2. Individual samples

**Output Variables:**

1. Count
2. Calculated mean
3. Calculated median
4. Calculated mode
5. Max value
6. Min value
7. Calculated standard deviation
8. Calculated variance

**Constraints:**

1. Programming Language: 8086 Assembly language
2. Numerical Representation: Integer or fixed-point representation (floating-point operations might be challenging)
3. Limited Memory: 8086's memory constraints
4. Input Size: Limited by memory and programming constraints
5. Error Handling: Handling invalid inputs, division by zero, or calculation overflows .

**Goals and Challenges:**

**Goals:**

1. Accurate calculations
2. Efficient algorithms
3. User-friendly interface

**Challenges:**

1. Implementing complex calculations in Assembly language.
2. Managing memory limitations.
3. Handling potential overflows or underflows.

By understanding these constraints and challenges, we can design an effective statistical calculator in 8086 Assembly language.

# **3. Methodology**

1. Use MASM or TASM assembler.

2. Accept inputs using standard input routines.

3. Store data in an array (using offset and memory addressing).

4. Implement logic using loops, sorting (for median), and arithmetic.

5. Display results using interrupt 21h (MS-DOS API syscall).

# **4. Implementation**

File: mean.asm

**; ----------MEAN---------------**

; Computes the mean (average) of an array of unsigned words.

; Inputs are passed through registers instead of the stack.

;FUNCTION

; cdecl void STC\_computeMean(arr\_seg\_idx, arr\_size)

; arr\_seg\_idx : AX

; arr\_size : BX

; returns mean : AX

;-------------------------------

STC\_computeMean proc

; save registers

push cx

push bx

push dx

push si

; args -> local vars

mov cx, bx ; CX = arr\_size

mov bx, ax ; BX = base arr seg offset

xor ax, ax ; AX = sum = 0

xor si, si ; SI = byte index = 0

push cx; save arr word size

push ax

mov ax, cx

mov cx, 2

mul cx

mov cx, ax; CX= arr bytes size

pop ax

mean\_loop:

mov dx, word ds[bx + si] ; Read word from array

add ax, dx ; Add to sum

inc si

inc si ; byte index++

cmp si, cx

jl mean\_loop ; If byte index < array\_byte\_sz,

; continue loop

end\_loop:

pop cx ; restore arr words size

xor dx, dx; set dividend hb to 0

div cx ; AX / CX -> Quotient in AX, remainder in DX

; restore registers

pop si

pop dx

pop bx

pop cx

ret

STC\_computeMean endp

File: median.asm

**; ----------MEDIAN---------------**

STC\_computeMedian proc

push si

push cx

push dx

push bx

mov si, ax ; SI = base offset (array start)

mov cx, bx ; CX = array size

test cx, cx

jz median\_zero ; if size == 0, median = 0

mov ax, cx

test ax, 1 ; check if odd number of elements

jnz median\_odd

; even size: median = (arr[mid] + arr[mid-1]) / 2

shr ax, 1 ; mid = size / 2

; load arr[mid]

mov bx, si

mov dx, ax

shl dx, 1 ; dx = mid \* 2 (byte offset)

add bx, dx

mov dx, word ptr [bx]

; load arr[mid-1]

mov bx, si

mov dx, ax

dec dx ; mid-1

shl dx, 1

add bx, dx

mov cx, word ptr [bx]

add dx, cx

shr dx, 1 ; average

mov ax, dx

jmp median\_done

median\_odd:

shr ax, 1 ; mid = size / 2

mov bx, si

mov dx, ax

shl dx, 1

add bx, dx

mov ax, word ptr [bx]

median\_done:

jmp median\_exit

median\_zero:

xor ax, ax ; median = 0 if array size 0

median\_exit:

pop bx

pop dx

pop cx

pop si

ret

STC\_computeMedian endp

File: mode.asm

**; ----------MODE---------------**

;problematic inputs: 3 3 1 5 2

; int STC\_computeMode(arr\_seg\_offset, arr\_size)

; arr\_seg\_offset : AX

; arr\_size : BX

; returns median : AX

STC\_computeMode proc

push bp; save prev bp

mov bp, sp; setup frame pointer

; save scratch regs

push dx

push cx

push bx

push si

mov cx, bx; cx = arr\_sz;

mov bx, ax; bx = arr\_seg\_offset;

mov si, 0; idx

push bx; save arr\_seg\_offset to CONSTANT DATA [bp-10]

mov ax, 0; last val

mov dx, 0; cur reps

push ax; mode val = 0

push dx; mode reps = 0

mov ax, 0xFFFF; init last val (-1)

modeloop:

push cx; save arr\_sz

mov cx, word ds[bx+si]; fetch cx = curr val

cmp cx, ax

je seq\_repeat ; last == curr

jne seq\_break ; last != curr

seq\_repeat:

inc dx; curr\_reps++

jmp end\_seq\_break

seq\_break:

mov dx, 1

end\_seq\_break:

mov ax, cx; last = curr

pop cx; restore arr\_sz

pop bx; bx = modereps

cmp bx, dx

jg skip\_mode\_update; if modereps > currreps

mov bx, dx; modereps = curreps

pop dx; dx = modeval

mov dx, ax; mode = curr; redundant ig

push dx; save modeval

skip\_mode\_update:

push bx; save modereps

mov bx, [bp-10]; restore arr\_seg\_offset from CONSTANT\_DATA

inc si

inc si

dec cx

jnz modeloop

endmodeloop:

pop ax ; modereps

pop ax; mode val

pop bx; clean CONSTANT DATA

;restore scratch regs

pop si

pop bx

pop cx

pop dx

pop bp

ret

STC\_computeMode endp

File: qsort.asm

**; ----------QUICKSORT ---------------**

; swap si and di vals from ds

; cdecl void STC\_swap\_in\_buf(si, di)

STC\_swap\_in\_buf proc

push ax

push bx

mov ax, word ds[si]

mov bx, word ds[di]

mov word ds[si], bx

mov word ds[di], ax

pop bx

pop ax

ret

STC\_swap\_in\_buf endp

; cdecl void STC\_qsort(arr\_seg\_idx, arr\_size, start offset, end offset)

; scratch registers: AX, BX, CX, DX

; Caller should save these ^

STC\_qsort proc

push bp

mov bp, sp

; fetch args

mov si, [bp+4] ; load src arr seg offset: arr\_seg\_idx

mov cx, [bp+6] ; load size: arr\_size

mov ax, [bp+8]; j : start offset

mov bx, ax; i

dec bx

dec bx; i = j - 1

mov dx, [bp+10] ; pivot : end offset

; 1st iter guard

cmp dx, ax

jz \_end; if j == pivot i.e 1 element: return

jl \_end; if j > pivot : return

push ax; save original j

; while j < pivot; j++

\_loop:

;save scratch registers

push cx; size

push ax; j

; fetch j val & pivot val

push si

add si, ax

mov cx, word ds[si]; j val

sub si, ax

add si, dx

mov ax, word ds[si]; pivot val

pop si

; cx & ax are modified now

cmp ax, cx; if j val > pivot val

pop ax; j

js \_skip\_swap

; swap start

mov di, si

add di, ax; di = j

push si; save si

inc bx

inc bx; i++

add si, bx; si = i

call STC\_swap\_in\_buf; swap i & j vals from ds

; swap complete

pop si; arr\_seg\_idx

\_skip\_swap:

pop cx; arr\_sz

inc ax

inc ax; j++

cmp dx, ax; j < pivot

jnz \_loop; only if dx and ax are aligned to two bytes

\_loop\_end:; j == pivot (assuming AX will never overshoot)

pop ax; restore original j

; swap i <-> pivot ====

inc bx

inc bx; i++

mov di, si

add di, dx; pivot

push si

add si, bx; i

call STC\_swap\_in\_buf

pop si

xchg dx, bx; pivot <-> i

; dx is modified now; sorted pivot

; bx: end

cmp dx, 0

;jz \_skip\_left\_rcrs; if pivot==0: skip

; rcrse left sub-array-----

dec dx

dec dx; left\_end = pivot - 1

cmp dx, ax

jz \_skip\_left\_rcrs; left\_end == start (will be one element sub-arr)

jl \_skip\_left\_rcrs; left\_end < start

; save registers

push dx

push cx

push bx

push ax

;args; will be popped inside function

push dx; left\_end

push ax; left\_start

push cx; arr\_size

push si; arr\_seg\_idx

call STC\_qsort; will restore regs

add sp, 8;cleanup args from stack

;restore registers

pop ax

pop bx

pop cx

pop dx

\_skip\_left\_rcrs:

inc dx

inc dx; restore pivot

; rcrse right sub-array-----

inc dx

inc dx; right\_start = pivot + 1

cmp bx, dx; end < right\_start

jz \_skip\_right\_rcrs; right\_start == arr\_size

jl \_skip\_right\_rcrs; right\_start > arr\_size

; save registers

push dx

push cx

push bx

push ax

push bx; end

push dx; start

push cx; arr\_size

push si; arr\_seg\_idx

call STC\_qsort

add sp, 8;cleanup args from stack

;restore registers

pop ax

pop bx

pop cx

pop dx

\_skip\_right\_rcrs:

dec dx

dec dx; restore pivot;

\_end:

mov sp, bp

pop bp

ret

STC\_qsort endp

callQsort proc

; prep to call qsort-----

; save registers ==

push dx

push cx

push bx

push ax; arr\_sz

mov bx, 2

mul bx ; 2 bytes(1 word)(BX) \* count(AX)

mov bx, ax; bx = arr\_byte\_sz

dec ax

dec ax; minus 1 word (cuz its an offset)

; pass args (cdecl) ==

push ax; arg: end offset

mov ax, 0; 2 bytes(1 word) \* 0 offset

push ax; arg: start offset

mov ax, bx ; arg: arr\_size

mov bx, 2

div bx; ax = arr\_sz

push ax

mov ax, OFFSET sample\_buffer; get src arr offset from seg

push ax; arg: arr\_seg\_idx

xor ax, ax; clear

;cdecl void qsort(arr\_seg\_idx, arr\_size, start offset, end offset)

call STC\_qsort

add sp, 8; cleanup args from stack

; restore

pop ax

pop bx

pop cx

pop dx

ret

callQsort endp

File: SD.asm

**; ----------STANDARD DEVIATION---------------**

computePopulationStandardDeviation proc

;TEMP=================

;mov ax, 3

;ret

;TEMP=================

push dx

push cx

push si

push ax; arr\_seg\_offset

push bx

call STC\_computeMean

mov dx, ax; dx = mean (mu)

pop cx; cx = arr\_sz

xor ax, ax; numerator

pop bx; bx = arr\_seg\_offset

mov si, 0; byte idx

push cx; save orig arr\_sz

sdloop:

push cx; save arr\_sz

mov cx, word ds[bx+si]; xi

sub cx, dx; xi - mu

xchg cx, ax; ax -> diff, cx -> numer

push dx

mul ax; diff squared

pop dx

xchg cx, ax; cx -> diff, ax -> numer

add ax, cx; numerator += diff

pop cx

inc si

inc si

dec cx

jnz sdloop:

endsdloop:

pop cx; restore orig arr\_sz

xor dx, dx

div cx; numerator/arr\_sz

; the quotient is in AX

call STC\_sqrt

pop si

pop cx

pop dx

ret

computePopulationStandardDeviation endp

File: stcg12.asm

**; ----------MAIN FUNCTION ---------------**

org 100h

jmp start

;=============CONSTANT DATA=====================================================================================

.DATA

; MEMORY BLOCK

sample\_buffer dw 10 DUP(0)

; STRINGS FOR PRINTING

prog\_title\_str db '[Statistical Calculator v1.0.1.0]$'

usr\_size\_prompt\_str db 'Enter population size (INT) (MAX=9): $'

input\_loop\_heading\_str db 'Requesting sample values:$'

usr\_sample\_prompt\_beg\_str db "Enter value for sample $"

usr\_sample\_prompt\_end\_str db " (INT) (UNSIGNED DECIMAL) (MAX=9): $"

arr\_print\_str db "Samples: $"

arr\_sorted\_print\_str db "Sorted: $"

input\_confirm\_str db "Sample values set [OK]$"

divider\_str db "[==============================================================]$"

count\_str db "Count: $"

mean\_str db "Arithmetic Mean: $"

median\_str db "Arithmetic Median: $"

mode\_str db "Mode: $"

arr\_max\_str db "Max: $"

arr\_min\_str db "Min: $"

standard\_deviation\_str db "Standard deviation (Sigma): $"

variance\_str db "Variance (Sigma^2): $"

prog\_end\_str db "[PROGRAM END]$"

team\_name\_str db "[Software by: GROUP 12]$"

note1\_str db "NOTE: INTEGER MATHEMATICS ONLY$"

;==================================================================================================

.CODE

INCLUDE "qsort.asm"

INCLUDE "mean.asm"

INCLUDE "utils.asm"

INCLUDE "mode.asm"

INCLUDE "median.asm"

INCLUDE "SD.asm"

; returns arr\_size : AX

getSampleArr proc

lea dx, usr\_size\_prompt\_str

call print

; parse int

mov ah, 0x1

int 0x21

call chtoi

mov cx, 0

mov cl, al; set arr\_size

push cx

mov si, OFFSET sample\_buffer

NEWLINE

lea dx, input\_loop\_heading\_str

call println

inp\_loop:

lea dx, usr\_sample\_prompt\_beg\_str

call print

mov al, cl

call itoch

mov dl, al

mov ah,02h

int 21h

mov dl, '/'

int 21h

; fmt nums

pop ax

push ax

call itoch

mov dl, al

mov ah, 02h

int 21h

lea dx, usr\_sample\_prompt\_end\_str

call print

; single inputs

mov ah, 01h

int 21h

call chtoi

mov word ds[si], ax

inc si

inc si

NEWLINE

dec cl

jnz inp\_loop

inp\_loop\_end:

lea dx, input\_confirm\_str

call println

lea dx, divider\_str

call println

pop ax; return arr\_size

ret

getSampleArr endp

;=================================== PROGRAM START==============================

start:

; setup ds

mov ax, cs

mov ds, ax

xor ax, ax

; decorative prints

lea dx, divider\_str

call println

lea dx, prog\_title\_str

call println

lea dx, team\_name\_str

call println

lea dx, note1\_str

call println

lea dx, divider\_str

call println

; call sample collection procedure

call getSampleArr; will return arr\_size to AX

push ax; for print

;=====PRINT ARRAY========================

lea dx, arr\_print\_str

call print

; pass args

mov bx, ax

mov ax, OFFSET sample\_buffer

call printArr

;=======PRINT SORTED ARRAY=====================

lea dx, arr\_sorted\_print\_str

call print

pop ax; arr\_sz

; qsort here

call callQsort

push ax; save arr\_sz

; pass args

mov bx, ax

mov ax, OFFSET sample\_buffer

call printArr

; display COUNT =============================

lea dx, count\_str

call print

pop ax; restore arr\_sz

call printNum

NEWLINE

; display MEAN =========================

push ax; save arr\_sz

mov bx, ax ;arr\_sz(words)

mov ax, OFFSET sample\_buffer ;arr\_seg\_offset

call STC\_computeMean

lea dx, mean\_str

call print

call printNum

NEWLINE

; display MEDIAN =====================

pop ax; arr\_sz

push ax; save arr\_sz

mov bx, OFFSET sample\_buffer

xchg ax, bx

call STC\_computeMedian; returns median in AX

lea dx, median\_str

call print

call printNum; AX has Median value

NEWLINE

; display MODE =========================

pop ax; arr\_sz

push ax; save arr\_sz

mov bx, OFFSET sample\_buffer

xchg ax, bx

call STC\_computeMode; returns mode in AX

lea dx, mode\_str

call print

call printNum;AX has mode valu

NEWLINE

; display MAX ===========================

pop ax; arr\_sz

push ax; save arr\_sz

mov bx, OFFSET sample\_buffer

mov si, 2

mul si; ax = bytesize = 2 \* arr\_size

dec ax; ax--

dec ax; ax = offset (size-1)

mov si, ax

mov ax, word ds[bx+si]; end element

lea dx, arr\_max\_str

call print

call printNum

NEWLINE

; display MIN =============================

pop ax; arr\_sz

push ax; save arr\_sz

mov bx, OFFSET sample\_buffer

mov ax, word ds[bx]; beginning element

lea dx, arr\_min\_str

call print

call printNum

NEWLINE

; display SD =============================

pop ax; arr\_sz

mov bx, OFFSET sample\_buffer

xchg ax, bx

call computePopulationStandardDeviation

lea dx, standard\_deviation\_str

call print

push ax; save SD for variance

call printNum

NEWLINE

; display VARIANCE =============================

pop ax; restore SD

mul ax; VAR = SQR(SD)

lea dx, variance\_str

call print

call printNum

NEWLINE

; END =========================================

; decorative prints

lea dx, divider\_str

call println

lea dx, prog\_end\_str

call println

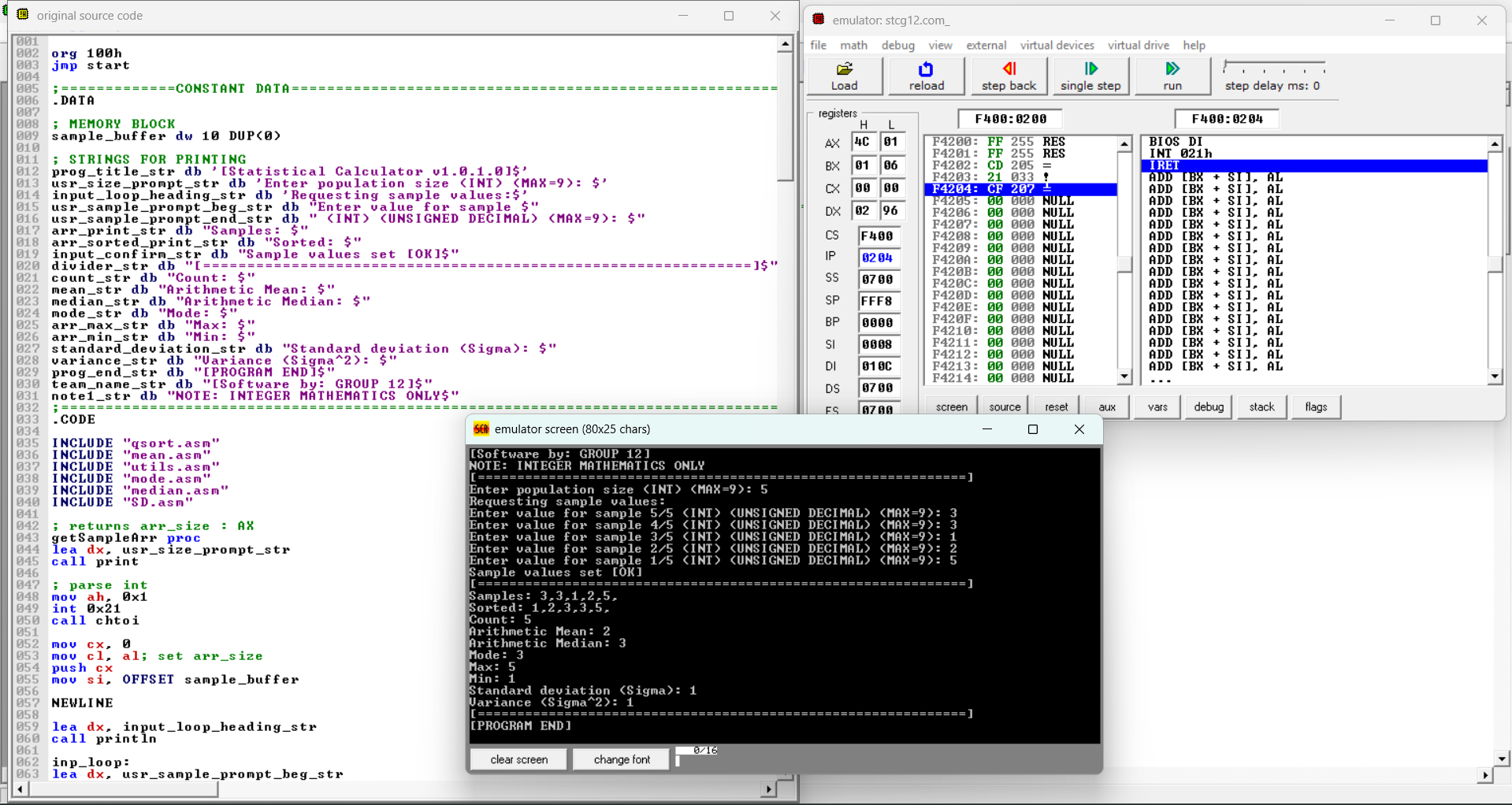
;================================================

mov ah, 4Ch; return to OS

int 0x21

; PROGRAM END ===================================================================

# **5. Results & Interpretation**

****

**6. Conclusion**

The statistical calculator designed using 8086 Assembly language provides a functional tool for calculating mean, median, and standard deviation. This project demonstrates the application of Assembly language programming in statistical analysis, showcasing its potential for efficient numerical computations.

**Key Takeaways:**

1. Assembly language programming can be used for statistical calculations. 2. Efficient algorithms and data representation are crucial for accurate results.

3. The calculator can be extended to include additional statistical functions.

**Future Enhancements:**

1. Support for floating-point numbers

2. Additional statistical functions (e.g., moments, correlation)

3. Improved user interface and input/output handling

The statistical calculator serves as a foundation for further development and exploration in statistical analysis using Assembly language programming.

# **References**

1. **Barry B. Brey** - "The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processors"
2. X86-64 registers cheat sheet: <https://math.hws.edu/eck/cs220/f22/registers.html>
3. MS-DOS interrupts: https://mrszeto.net/CIT/interrupts.htm
4. **Muhammad Ali Mazidi** - "The 8051 Microcontroller and Embedded Systems"
5. **Kenneth Morse** - "The 8086/8088 Primer"
6. Online resources and documentation on 8086 Assembly language programming
7. Statistical analysis and calculation resources (e.g., textbooks, online tutorials)

These references provide a foundation for understanding 8086 Assembly language programming and statistical analysis, supporting the development of the statistical calculator

# **Appendices**

## Appendix A: 8086 Assembly Language Syntax and Instructions

## - Instruction set architecture

## - Syntax and semantics

## - Examples of common instructions

## Appendix B: Statistical Formulas

## - Mean: https://en.wikipedia.org/wiki/Arithmetic\_mean

## - Mode: most frequent value of sorted dataset

## - Standard Deviation: https://en.wikipedia.org/wiki/Standard\_deviation

## Appendix C: Sample Datasets

## - Example datasets for testing the calculator

## - Expected results for mean, median, and standard deviation calculations

## Appendix D: Code Listings

## - Complete code listings for the statistical calculator

## - Comments and explanations for key sections of code

## These appendices provide supplementary information and resources to support understanding and implementation of the statistical calculator.