|  |
| --- |
| **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**: Sum of elements
3. **Arithmetic Mean Calculation**: Calculate the average value of a dataset.
4. **Median**: Middle value in data
5. **Mode Calculation**: Find the most frequent value of a sorted dataset.
6. **Max**: Maximum value in dataset
7. **Min**: Minimum value in dataset
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

; 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\_offset, arr\_size, start offset, end offset)

; arr\_seg\_offset : DI

; arr\_size : SI

; start\_offset : DX

; end\_offset : CX

; NOTE: OFFSETS IN BYTES

STC\_qsort proc

    push bp

    mov bp, sp

    sub sp, 32; 16 nums

    mov bx, di; arr\_seg\_offset

    mov word ptr [bp-2], si; arr\_size

    mov word ptr [bp-4], dx; start offset

    mov word ptr [bp-6], cx; end offset

    ;----

    mov word ptr [bp-8], dx; j

    mov word ptr [bp-10], dx; i

    dec word ptr [bp-10]

    dec word ptr [bp-10]; i = j-1

    mov word ptr [bp-12], cx; pivot

    ; 1st iter guard

    mov dx, word ptr [bp-12]; ld pivot

    cmp word ptr [bp-8], dx; j, pivot

    jge \_end; if j == pivot i.e 1 element: return OR if j > pivot : return

    mov si, cx;pivot

    mov cx, word ptr [bx+si];ld pivot\_val

    mov word ptr [bp-14], cx; pivot value

; while j < pivot; j++

qsort\_loop:

    ; fetch j val & pivot val

    mov si, word ptr [bp-8]; ld j

    mov ax, word ptr [bx+si]; j val

    cmp ax, word ptr [bp-14]; j\_v, piv\_v

    jg  \_skip\_swap   ; if j val > pivot val

    ; swap start ----

    mov di, word ptr [bp-8] ; di = j

    inc word ptr [bp-10]

    inc word ptr [bp-10]; i++

    mov si, word ptr [bp-10]; si = i

    add si, bx

    add di, bx; convert to raw addr

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

    ; swap complete ----

\_skip\_swap:

    inc word ptr [bp-8]

    inc word ptr [bp-8]; j++

    mov ax, word ptr [bp-8]; j

    cmp ax, word ptr [bp-12]; j, pivot

    jl qsort\_loop; loop if j < pivot

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

; swap i <-> pivot ====

    inc word ptr [bp-10]

    inc word ptr [bp-10]; i++

    mov di, word ptr [bp-12]; pivot

    mov si, word ptr [bp-10]; i

    ;sort pivot element

    add si, bx

    add di, bx; convert to raw addr

    call STC\_swap\_in\_buf

    ; swap i & piv

    mov dx, word ptr [bp-12]; pivot

    xchg dx, word ptr [bp-10]; pivot <-> i

    mov word ptr [bp-12], dx

    ; sorted pivot index^

    mov si, word ptr [bp-12]

    mov ax, word ptr [bx+si]; update pivot val

    mov word ptr [bp-14], ax

    cmp word ptr [bp-12], 0

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

    ; rcrse left sub-array-----

    mov dx, word ptr [bp-12]; ld sorted\_pivot

    dec dx

    dec dx; left\_end = pivot - 1

    cmp dx, word ptr [bp-4]

    jle \_skip\_left\_rcrs; left\_end == start\_offset (will be one element sub-arr) OR left\_end < start\_offset

    ; save registers

    push dx

    push cx

    push bx

    push ax

    mov di, bx; arr\_seg\_offset

    mov si, word ptr [bp-2]; arr\_size

    mov cx, dx; left\_end

    mov dx, word ptr [bp-4]; left\_start

    call STC\_qsort

    ;restore registers

    pop ax

    pop bx

    pop cx

    pop dx

\_skip\_left\_rcrs:

    ; rcrse right sub-array-----

    mov dx, word ptr [bp-12]; ld sorted\_pivot

    inc dx

    inc dx; right\_start = pivot + 1

    cmp word ptr [bp-6], dx; end, right\_start

    jle \_skip\_right\_rcrs; right\_start == end OR right\_start > end

    ; save registers

    push dx

    push cx

    push bx

    push ax

    mov di, bx; arr\_seg\_offset

    mov si, word ptr [bp-2]; arr\_size

    ; dx already has right\_start

    mov cx, word ptr [bp-6]; right\_end

    call STC\_qsort

    ;restore registers

    pop ax

    pop bx

    pop cx

    pop dx

\_skip\_right\_rcrs:

\_end:

    add sp, 32

    mov sp, bp

    pop bp

    ret

STC\_qsort 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

org 100h

jmp start

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

.DATA

; MEMORY BLOCK

sample\_buffer dw 10 DUP(0)

buffer\_addr dw 0x0

SCANF\_MAX\_LEN db 20

scanf\_buffer db 21 DUP(0)

int\_buffer db 21 DUP(0)

; STRINGS FOR PRINTING

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

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

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

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

usr\_size\_prompt\_str db "Enter population size (INT): $"

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): $"

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

arr\_print\_str db "Samples: $"

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

count\_str db "Count: $"

sum\_str db "Sum: $"

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]$"

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

.CODE

INCLUDE "qsort.asm"

INCLUDE "mean.asm"

INCLUDE "utils.asm"

INCLUDE "mode.asm"

INCLUDE "median.asm"

INCLUDE "SD.asm"

; returns arr\_size : AX

querySampleArray proc

    push bp

    mov bp, sp

    sub sp, 32; 16 nums

    lea dx, usr\_size\_prompt\_str

    call print

    ;scanf proc

    mov al, SCANF\_MAX\_LEN

    mov si, ax

    lea di, scanf\_buffer

    call scanf

    lea di, [scanf\_buffer+2]

    call sttoi

    mov cx, ax; cx = arr\_size

    mov word ptr [bp-4], cx; arr\_size

    mov word ptr [bp-12], 1; countdown

    mov ah, 0

    mov dx, 2

    mul dx; word size

    mov di, ax; final bytesize

    call malloc; just returns a segment offset

    mov word ptr [bp-6], ax; buff\_addr

    mov word ptr [bp-8], ax; save original addr

    NEWLINE

    lea dx, input\_loop\_heading\_str

    call println

; collection loop

inp\_loop:

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

    call print

    ; printing (cur/cap)

    ; print curr

    mov si, word ptr [bp-12]

    lea di, int\_buffer

    push cx

    call ittost

    pop cx

    lea dx, int\_buffer

    call print

    ; print the slash

    mov dl, '/'

    mov ah, 0x02

    int 21h; print

    ; print capacity

    mov ax, word ptr [bp-4]; ld orig\_capacity

    mov si, ax

    lea di, int\_buffer

    push cx

    call ittost

    pop cx

    lea dx, int\_buffer

    call print

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

    call print

    ; scanf proc

    mov al, SCANF\_MAX\_LEN

    mov si, ax

    lea di, scanf\_buffer

    call scanf

    lea di, [scanf\_buffer+2]

    call sttoi

    ; write to sample\_buffer

    mov si, word ptr [bp-6]; sample\_buffer\_seg\_offset

    mov word [si], ax

    inc word ptr [bp-6]

    inc word ptr [bp-6]

    NEWLINE

    inc word ptr [bp-12]

    dec cl; dec curr

    jnz inp\_loop

inp\_loop\_end:

    lea dx, input\_confirm\_str

    call println

    lea dx, divider\_str

    call println

    mov cx, word ptr [bp-8]; return arr\_seg\_offset

    mov ax, word ptr [bp-4]; return arr\_size to AX

    add sp, 32

    pop bp

    ret

querySampleArray endp

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

start:

    push bp

    mov bp,sp

    sub sp, 32; 16 nums

    ; 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 querySampleArray; will return arr\_size to AX, arr\_seg\_offset to cx

    mov word ptr [bp-2], cx; arr\_seg\_offset

    mov word ptr [bp-4], ax; arr\_size

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

    lea dx, arr\_print\_str

    call print

    ; pass args

    mov bx, word ptr [bp-4]

    mov ax, word ptr [bp-2]

    call printArr

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

    lea dx, arr\_sorted\_print\_str

    call print

    mov ax, word ptr [bp-4]

    ; qsort here

    mov di, word ptr [bp-2]

    mov si, word ptr [bp-4]

    mov dx, 0

    mov cx, word ptr [bp-4]; arr\_size

    mov ax, 2

    mul cx

    mov cx, ax; convert to byte\_size

    call STC\_qsort

    mov ax, word ptr [bp-4]

    ; pass args

    mov bx, ax

    mov ax, word ptr [bp-2]

    call printArr

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

    lea dx, count\_str

    call print

    mov ax, word ptr [bp-4]

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

    ; display SUM ===========================

    mov ax, word ptr [bp-4]

    mov bx, ax ;arr\_sz(words)

    mov ax, word ptr [bp-2];arr\_seg\_offset

    call STC\_computeSum; returns sum in AX

    lea dx, sum\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, ax ;arr\_sz(words)

    mov ax, word ptr [bp-2];arr\_seg\_offset

    call STC\_computeMean; retuns mean in AX

    lea dx, mean\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, word ptr [bp-2]

    xchg ax, bx

    call STC\_computeMedian; returns median in AX

    lea dx, median\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, word ptr [bp-2]

    xchg ax, bx; ax -> arr\_seg\_offset; bx-> arr\_sz

    call STC\_computeMode; returns mode in AX

    lea dx, mode\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, word ptr [bp-2]

    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

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, word ptr [bp-2]

    mov ax, word ds[bx]; beginning element

    lea dx, arr\_min\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    mov ax, word ptr [bp-4]

    mov bx, word ptr [bp-2]

    xchg ax, bx

    call STC\_computePopulationStandardDeviation

    lea dx, standard\_deviation\_str

    call print

    push ax; save SD for variance

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    pop ax; restore SD

    mul ax; VAR = SQR(SD)

    lea dx, variance\_str

    call print

    mov si, ax

    lea di, int\_buffer

    call ittost

    lea dx, int\_buffer

    call println

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

    ; decorative prints

    lea dx, divider\_str

    call println

    lea dx, prog\_end\_str

    call println

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

    add sp, 32

    pop bp

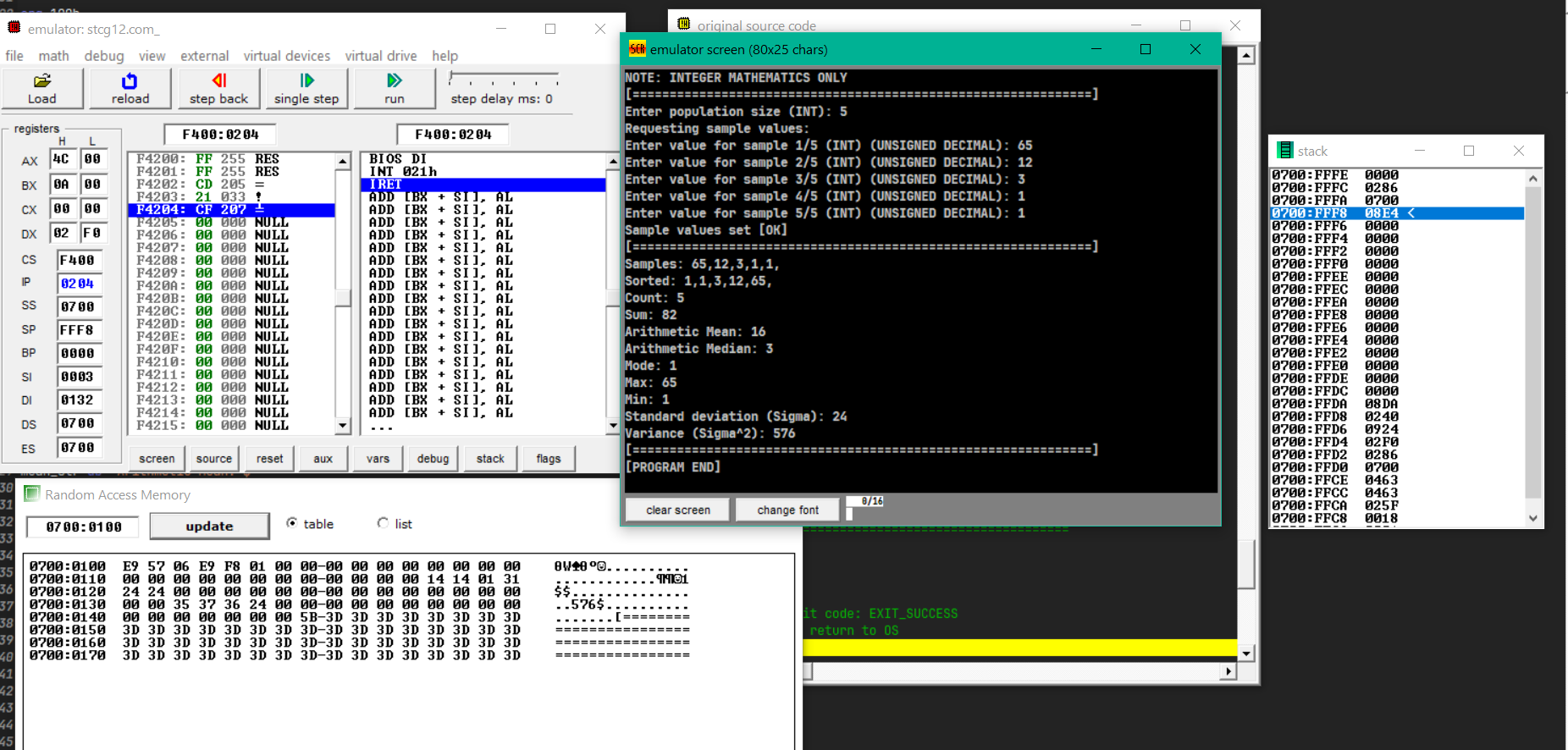
    mov al, 0; exit code: EXIT\_SUCCESS

    mov ah, 0x4C; 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.