Engineering and **Information** Technology Faculty

Computer Science Department

Computer Organization

Project Report

Assembly ID Workshop

Basel Abu Hamed 1202397

Mohammad Abaedh 1200368

Samer Farhat 1180574

Dr. Ibrahim Nimer

Section 1/3

19-Jul-23

Abstract

This document presents a detailed analysis of a command-line application developed in Assembly language. The application is designed to handle a collection of seven-character identifiers (IDs), perform various calculations on these IDs, and offer a user-friendly menu-driven interface to display the computed results. The report encompasses the problem definition, requirements, specifications, system architecture, component design, code integration, testing procedures, obtained results, and a conclusive summary.

Table of Contents

Abstract ..... *I*

1 Introduction *1*

2 Architecture *2*

2.1 User Input **2**

2.2 Data Validation **2**

2.3 Computation **2**

2.4 Interactive Menu Generation **2**

2.5 Result Display **2**

2.6 Data Update **3**

2.7 Exit **3**

3 Component Design *4*

3.1 Segments of the System **4**

3.1.1 Data Segment 4

3.1.2 Code Segment 4

3.2 Key Instructions and Components Used **4**

3.2.1 Data Manipulation Instructions 4

3.2.2 Control Flow Instructions 5

3.3 Procedures **5**

3.4 Interrupts **5**

3.5 Registers **5**

3.5.1 Buffer 6

3.5.2 Variables 6

4 Code and Results *7*

4.1 Option Menu **11**

4.1.1 Display IDs 11

4.1.2 Count 12

4.1.3 Sum 12

4.1.4 Mean 13

4.1.5 Median 15

4.1.6 Max 16

4.1.7 Min 17

4.1.8 Add New ID 18

4.1.9 Exit 19

5 Conclusion *21*

Table of Figures

Figure 4-1 Code for asking for input. 7
Figure 4-2 Code for reading ID input. 7
Figure 4-3 Program interface asking user for ID input. 8
Figure 4.1.1-1 Code for showing IDs 11
Figure 4.1.1-2 Code for showing for IDs loop. 11
Figure 4.1.1-3 Program interface showing the entered IDs. 11
Figure 4.1.2-1 Code for counting number of IDs. 12
Figure 4.1.2-2 Program showing the number of IDs. 12
Figure 4.1.3-1 Code for summing the IDs 13
Figure 4.1.3-2 Program showing the sum of IDs. 13
Figure 4.1.4-18 Mean calculation code 14
Figure 4.1.4-2 Program calculating mean. 14
Figure 4.1.5-1 Code for calculating median. 15
Figure 4.1.5-2 Program execution of median 16
Figure 4.1.6-1 Code for max calculation 16
Figure 4.1.6-2 Program execution of max 17
Figure 4.1.7-1 Code for calculating min 17
Figure 4.1.7-2 Program execution of min 18
Figure 4.1.8-1 Code for adding new ID 18
Figure 4.1.8-2 Program execution of adding an ID 19
Figure 4.1.9-1 Code for exiting the program 19
Figure 4.1.9-2 Program executing exit option 20

1 Introduction

This software application is a command line-based program that enables users to input a series of seven-character IDs, perform computations on the IDs, and provide a menu to the user for different functionalities. The requirements for this application include reading user input, performing computations such as count, sum, max, min, average, and median, providing an interactive menu, and enabling users to add new IDs. These requirements translate into specifications including data input and output, mathematical computations, control structures, and string handling. At the initial stage, the application prompts the user for input, specifically a series of seven-character identifiers (IDs). The user interacts with the command line interface to enter these IDs into the application. This phase is crucial for gathering the necessary data that the application uses for further computations and operations within the program.

2 Architecture

2.1 User Input

In the initial stage, the application prompts the user for input. It accepts a series of seven-character identifiers (IDs) entered by the user through the command line interface.

2.2 Data Validation

After the user provides input, the application validates the entered IDs to ensure they adhere to the required format of seven characters. If the IDs are incorrect, the user is prompted to enter valid IDs again.

2.3 Computation

In this stage, the application performs computations on the validated data. It calculates values such as count, sum, max, min, average, and median of the character values in the IDs.

2.4 Interactive Menu Generation

After completing the computations, the system generates an interactive menu. The menu offers the user various options, including displaying the IDs or the calculated results, adding new IDs, or exiting the program.

2.5 Result Display

Depending on the user's selection from the interactive menu, the application displays the corresponding results. If the user chooses to view IDs, the IDs are shown

on the screen. If the user wants to see the results of the computations, such as count, sum, max, min, average, or median, those results are displayed accordingly.

2.6 Data Update

When the user chooses to add new IDs, this block takes control. It prompts the user to enter the new IDs and then validates them to ensure they adhere to the required format. After validation, the data set used for computations is updated to include the newly added IDs.

2.7 Exit

The final stage of the application occurs when the user selects the "exit" option from the menu. In response to this choice, the program gracefully ends, concluding the execution.

3 Component Design

3.1 Segments of the System

3.1.1 Data Segment

The application uses the Data Segment to store both static and dynamic data. This includes messages to be printed to the console, a buffer for storing IDs, and variables for counts, sums, min, max, and average values.

3.1.2 Code Segment

The Code Segment contains the program instructions that are executed. The code follows a procedural structure, where the main procedure coordinates calls to other procedures based on user input. This approach enables the program to perform computations on the input data, display results, and handle user interactions in a well-organized manner.

3.2 Key Instructions and Components Used

3.2.1 Data Manipulation Instructions

Assembly language instructions, such as MOV, ADD, SUB, INC, DEC, MUL, and DIV, are used to process data in the program. The MOV instruction transfers data between registers or memory locations, while the arithmetic operations like ADD, SUB, INC, DEC, MUL, and DIV perform various mathematical computations

on the data. These instructions are fundamental to manipulating and transforming data in the application.

3.2.2 Control Flow Instructions

Conditional instructions in assembly language, such as CMP, JE, JNE, JGE, JLE, JMP, and LOOP, alter the program's execution sequence based on specific conditions. They enable the program to make decisions and execute different code paths as required.

3.3 Procedures

The main procedure serves as the program's entry point. It calls other procedures, such as PrintNumber, to carry out specific tasks and execute various functionalities of the program.

3.4 Interrupts

The INT 21h instruction is utilized to invoke DOS services. It provides access to various functionalities, such as character I/O, string output, and program termination, through different functions available within this interrupt.

3.5 Registers

Registers such as AX, BX, CX, DX, SI, and DI are employed to store and manipulate data during computation. For instance, AX is commonly utilized for arithmetic operations, SI and DI are often used for string and array operations, and so on.

3.5.1 Buffer

A buffer called id_buffer is utilized to store the IDs entered by the user. This enables the program to access and manipulate the user's data efficiently.

3.5.2 Variables

The application utilizes several variables to store the results of computations, such as the count of IDs (count), maximum value (max_value), minimum value (min_value), and other relevant values. These variables play a crucial role in calculating and displaying the desired outcomes to the user.

4 Code and Results

1. When the program runs it will prompt you to enter an ID: "Enter ID :

```
028 getIds:
029     lea dx, idPrompt ; Prompt for entering ID
030     mov ah, 9h
031     int 21h
032
033     mov si, offset idBuffer
034     add si, idLength ; Point SI to where the new ID should start
035
036     mov cx, 7 ; Length of an ID
037
```

Figure 3.5.2-1 Code for asking for input.

2. You will enter the ID for example "1200368" and press Enter.

```
038 readId:
039     mov ah, 1h ; Read a single character
040     int 21h
041     mov [si], al ; Store the character
042     inc si
043
044     loop readId ; Continue reading until the ID is complete
045
046     add idLength, 7 ; Update total length
047     inc word ptr [idCount] ; Increase count for each ID
048
049     lea dx, partnerPrompt ; Prompt for partners in the team
050     mov ah, 9h
051     int 21h
052     mov ah, 1h
053     int 21h
054     cmp al, 'Y'
055     je newlineAndGetIds ; If yes, proceed to get another ID
056
```

Figure 3.5.2-2 Code for reading ID input.

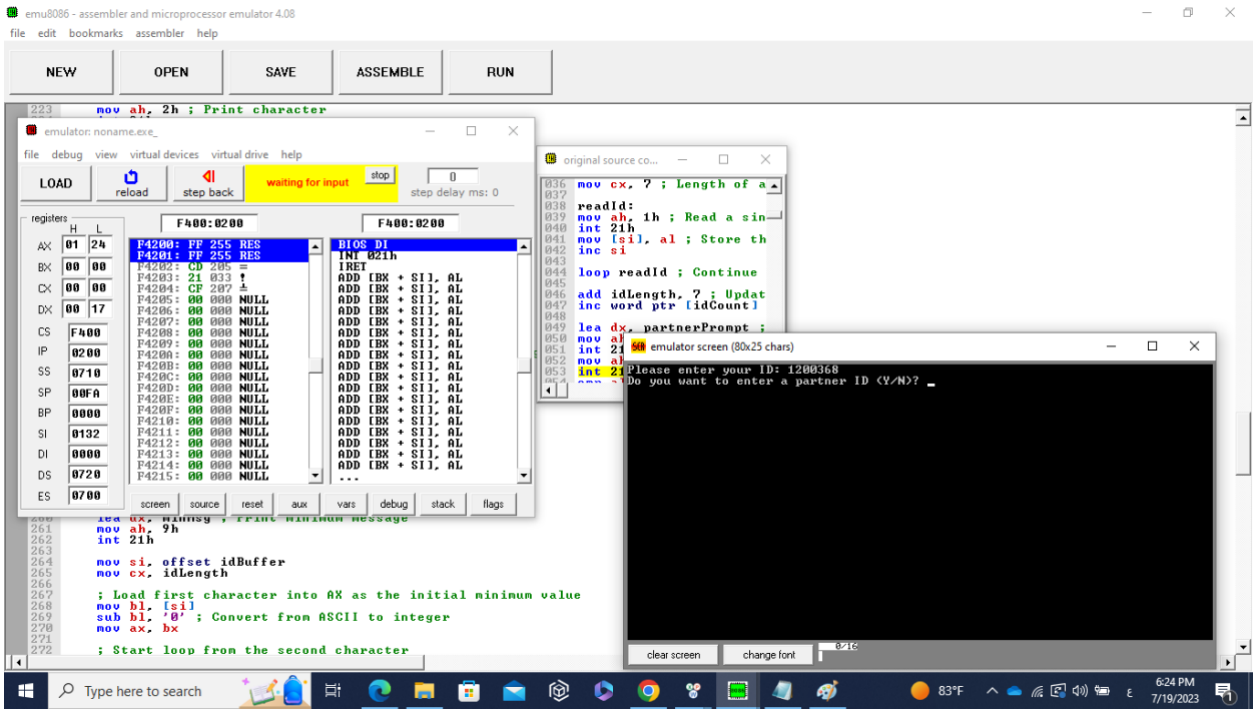

Figure 3.5.2-3 Program interface asking user for ID input.

3. The program will then ask if you have partners in your team: "Do you have partners in your team (Y/N)?"

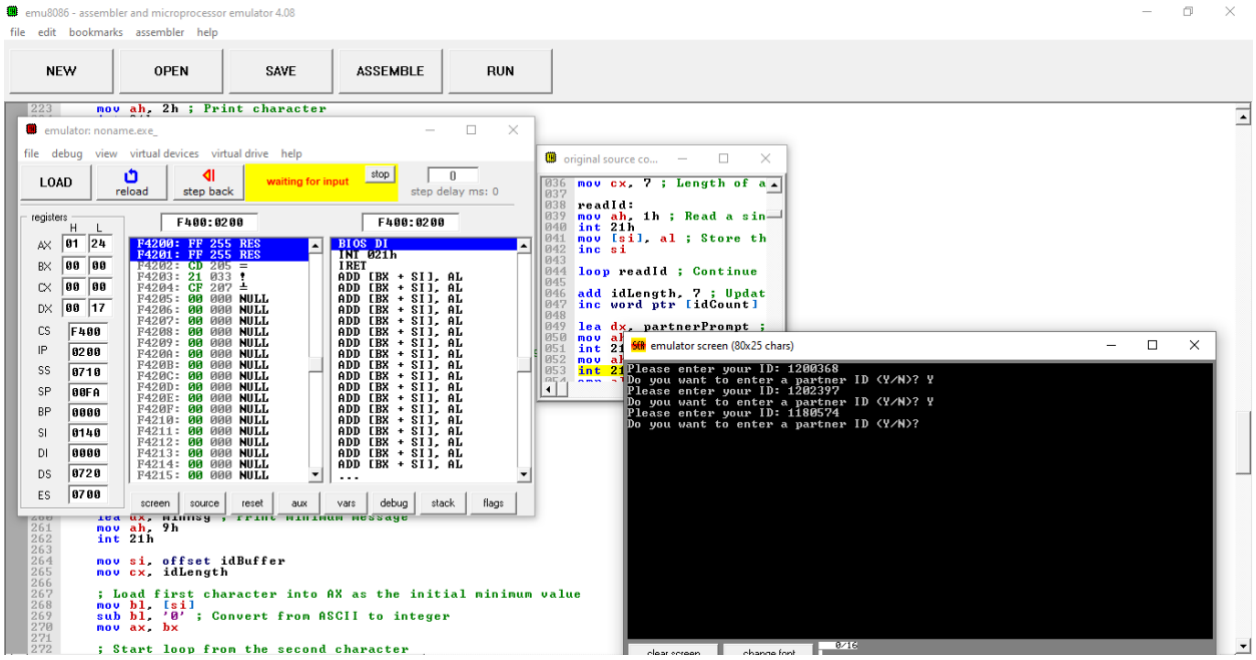

Figure 4-4 Program interface asking for more IDs

If you enter "Y" and press Enter, the program will prompt you to enter the partner's ID: "Enter a new ID: ".

4. You will input the partner's ID for example "1202397" and press Enter. Then the program will ask you again if you have partners as in Fig. 4-3.
5. If you enter "N" and press Enter, the program will display the menu options.

```
menu:
    lea dx, menuPrompt ; Prompt for menu choice
    mov ah, 9h
    int 21h
    mov ah, 1h
    int 21h
    cmp al, '1'
    je showIds ; If choice is 1, show IDs
    cmp al, '2'
    je countIds ; If choice is 2, count IDs
    cmp al, '3'
    je sumIds ; If choice is 3, calculate sum
    cmp al, '4'
    je meanId ; If choice is 4, calculate mean
    cmp al, '5'
    je medianId ; If choice is 5, calculate median
    cmp al, '6'
    je maxId ; If choice is 6, find maximum
    cmp al, '7'
    je minId ; If choice is 7, find minimum
    cmp al, '8'
    je addId ; If choice is 8, add new ID
    cmp al, '9'
    je exitProgram ; If choice is 9, exit the program
    jmp menu ; If choice is invalid, show menu again
```

Figure 3.5.2-5 Code for options menu

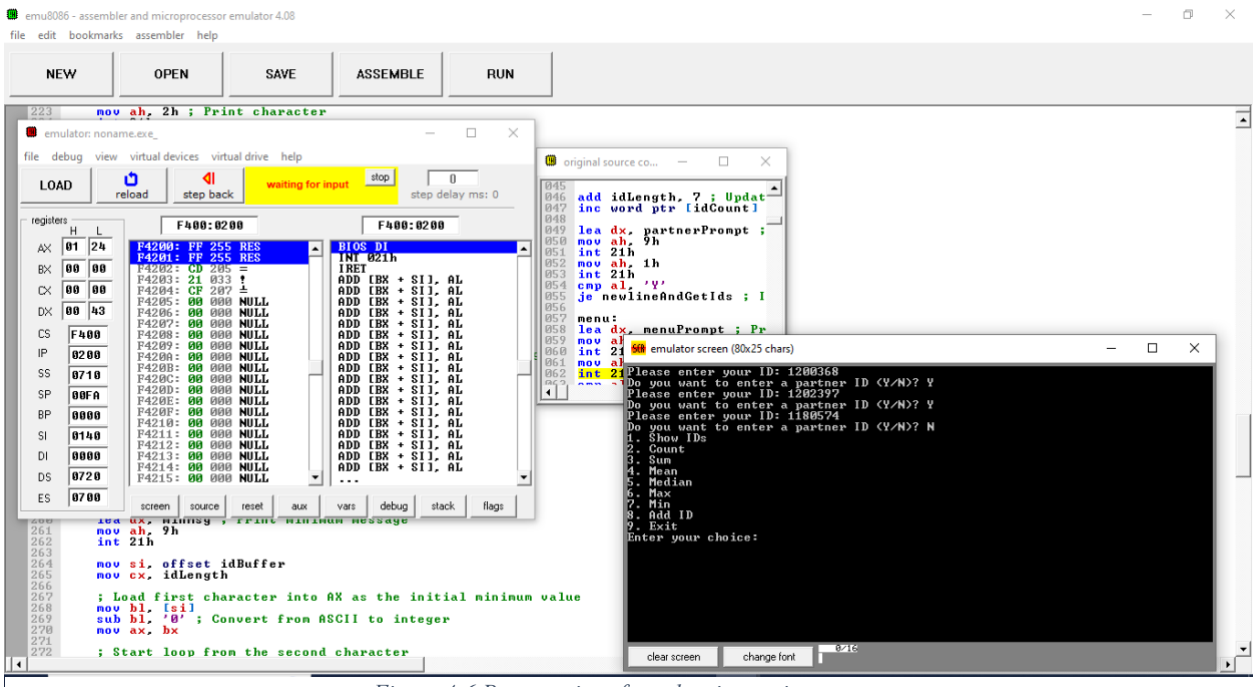

Figure 4-6 Program interface showing options menu.

4.1 Option Menu

4.1.1 Display IDs

The "Show IDs" option will display the entered IDs, each one on a separate line. As observed in the figure below:

```
089 showIds:
090     lea dx, newLine ; Print newline
091     mov ah, 9h
092     int 21h
093     mov si, offset idBuffer
094     mov cx, idLength
095     mov bx, 7 ; Number of characters per ID
```

Figure 4.1.1-1 Code for showing IDs

```
097 showIdsLoop:
098     mov dl, [si] ; Get a character from ID
099     mov ah, 2h ; Print character
100     int 21h
101     inc si
102     dec bx
103     jnz skipNewLine ; If BX != 0, skip printing a newline
104
105     lea dx, newLine ; Print newline after each ID
106     mov ah, 9h
107     int 21h
108     mov bx, 7 ; Reset BX for the next ID
```

Figure 4.1.1-2 Code for showing for IDs loop.

Here we entered 1,2and 3 ID so the output is as in Fig. 4-1-3.

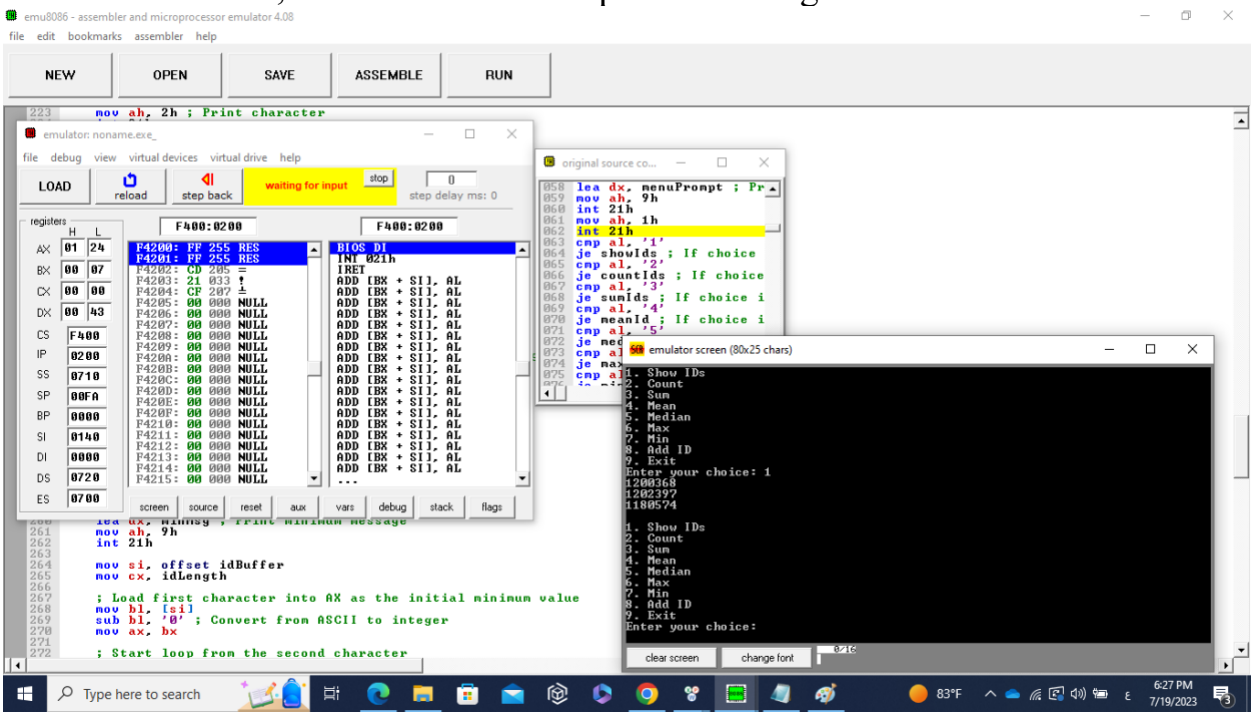

Figure 4.1.1-3 Program interface showing the entered IDs.

4.1.2 Count

The "Count" option will show the overall count of the entered IDs. Since we have entered 3 IDs, the output will be 3, as depicted in the image.

```
115 countIds:
116     lea dx, newLine ; Print newline
117     mov ah, 9h
118     int 21h
119     lea dx, countMsg ; Print count message
120     mov ah, 9h
121     int 21h
122     mov ax, [idCount] ; Load ID count
123     call PrintNumber ; Print the count
124     jmp menu ; Return to menu
```

Figure 4.1.2-1 Code for counting number of IDs.

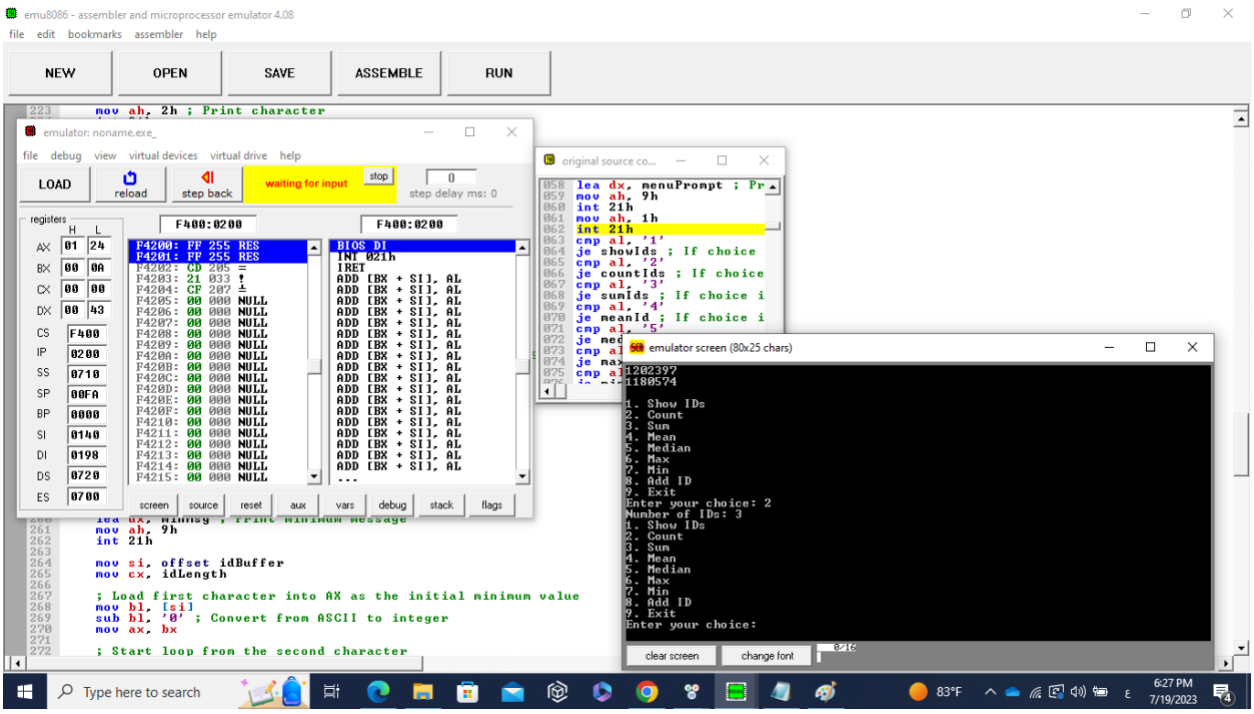

Figure 4.1.2-2 Program showing the number of IDs.

4.1.3 Sum

The "Sum" option will compute the total sum of all the digits in the ID's Array. For the given input ID "1200368", the sum would be calculated as 1 + 2 + 0 + 0 + 3 + 6 + 8 = 16. Consequently, this is for one id, the output will match fig. 4-13 for 3 IDs:

```
126 sumIds:
127     lea dx, newLine ; Print newline
128     mov ah, 9h
129     int 21h
130     lea dx, sumMsg ; Print sum message
131     mov ah, 9h
132     int 21h
133     mov si, offset idBuffer
134     mov cx, idLength
135     xor ax, ax ; Clear AX for the sum
136
```

Figure 4.1.3-1 Code for summing the IDs

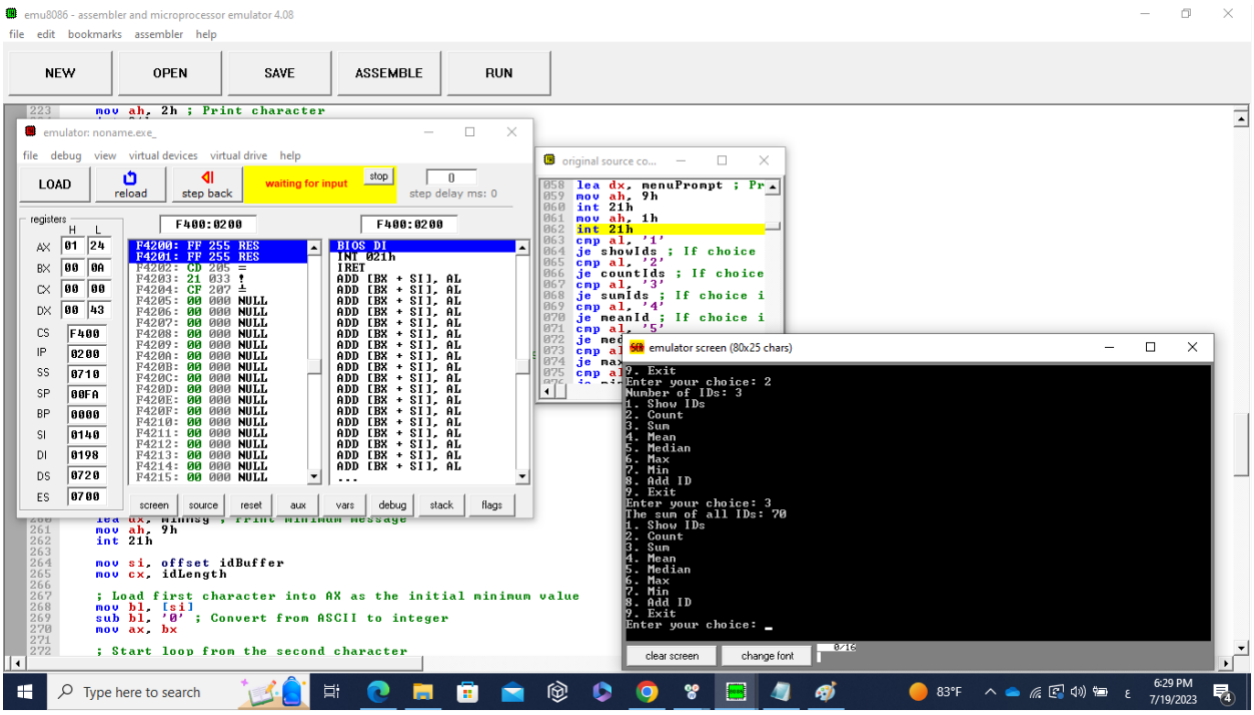

Figure 4.1.3-2 Program showing the sum of IDs.

4.1.4 Mean

The "Mean" option aims to compute the average of all the digits in the ID. For the provided input ID "1200368," the sum of the digits is 20, and the total number of digits is 7. Consequently, the average would be 20 / 7 = 2.8571 (rounded to the nearest integer). As a result, for one ID, the output will match the Fig. 4-15 for 3 IDs.

```
147 meanId:
148     lea dx, newLine ; Print newline
149     mov ah, 9h
150     int 21h
151     lea dx, meanMsg ; Print mean message
152     mov ah, 9h
153     int 21h
154     mov si, offset idBuffer
155     xor ax, ax ; Clear AX for the sum
156     xor bx, bx ; Clear BX as a counter for number of digits
157     mov cx, idLength
158
159 meanIdLoop:
160     mov bl, [si] ; Get a character from ID
161     sub bl, '0' ; Convert from ASCII to integer
162     add al, bl ; Add to the sum
163     inc bx ; Increment digit counter
164     inc si
165     loop meanIdLoop ; Repeat until sum and digit count are calculated
166
167     ; BX now contains total count of digits
168
169     ; Add half of the divisor to the dividend for rounding
170     mov dx, bx
171     shr dx, 1 ; dx = bx / 2
172     add ax, dx ; ax = ax + dx
173
174     ; Divide the sum by the total number of digits to get the mean
175     cwd ; Extend AX into DX:AX
176     idiv bx
177
178     call PrintNumber ; Print the mean
179     jmp menu ; Return to menu
180
```

Figure 4.1.4-18 Mean calculation code

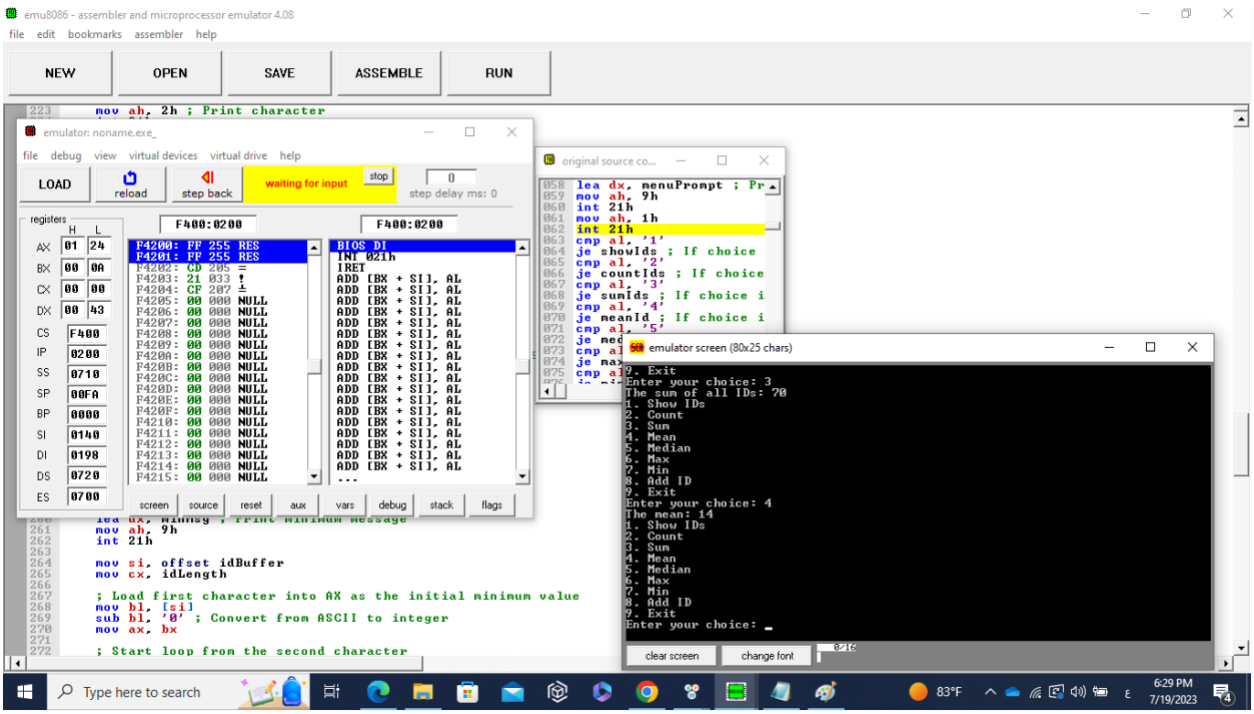

Figure 4.1.4-2 Program calculating mean.

4.1.5 Median

The median option will sort the digits of all IDs in ascending order then will divide the count all the digits (n) over 2 if the count of digits is even it will calculate the median as follows, median = the element with the index [(n/2 +(n/2 +1))/2]; otherwise if the count of digits is odd then the median = the element with the index [n/2 +1].

```
medianId:
    lea dx, newLine ; Print newline
    mov ah, 9h
    int 21h
    lea dx, medianMsg ; Print median message
    mov ah, 9h
    int 21h

    mov si, offset idBuffer
    mov cx, idLength

    mov dx, cx ; Save the length of IDs
    shr dx, 1 ; Divide by 2 to get the middle position
    jnc medianEven ; If the length is even, jump to medianEven

    medianOdd:
    mov bx, dx ; BX = middle position
    jmp medianLoop ; Jump to the median calculation loop

    medianEven:
    dec cx ; Decrement length by 1
    mov bx, cx ; BX = length - 1 (0-based indexing)
    shr bx, 1 ; Divide by 2 to get the first middle position
    add bx, 2 ; Add 2 to skip the newline characters
    jmp medianLoop ; Jump to the median calculation loop

    medianLoop:
    mov si, offset idBuffer ; Start from the beginning of IDs

    medianLoopStart:
    cmp bx, 0 ; Check if BX reached 0
    je medianFound ; If yes, median found

    inc si ; Increment SI to the next character
    cmp byte ptr [si], 13 ; Check for newline character
    je medianLoopStart ; If newline, skip

    dec bx ; Decrement BX
    jmp medianLoopStart ; Repeat until median is found

    medianFound:
    mov dl, [si] ; Get the median character
    mov ah, 2h ; Print character
    int 21h

    jmp menu ; Return to menu
```

Figure 4.1.5-1 Code for calculating median.

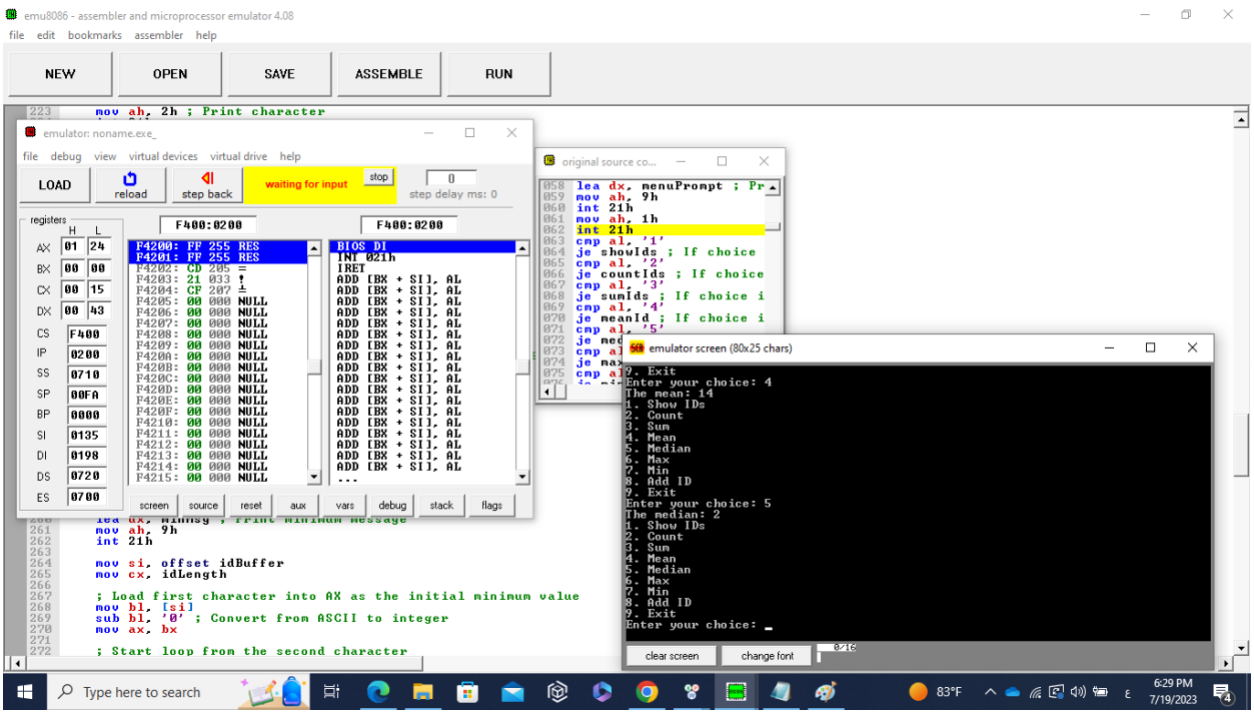

Figure 4.1.5-2 Program execution of median

4.1.6 Max

The "Max" option aims to determine the highest digit among all the digits in the ID's array. In the provided input IDs "1200368,12002397,1180574" the maximum digit is9. As a result, the output will mirror the image.

```
maxId:
    lea dx, newLine ; Print newline
    mov ah, 9h
    int 21h
    lea dx, maxMsg ; Print maximum message
    mov ah, 9h
    int 21h

    mov si, offset idBuffer
    mov cx, idLength

    mov ax, 0 ; Clear AX for the maximum value
```

Figure 4.1.6-1 Code for max calculation

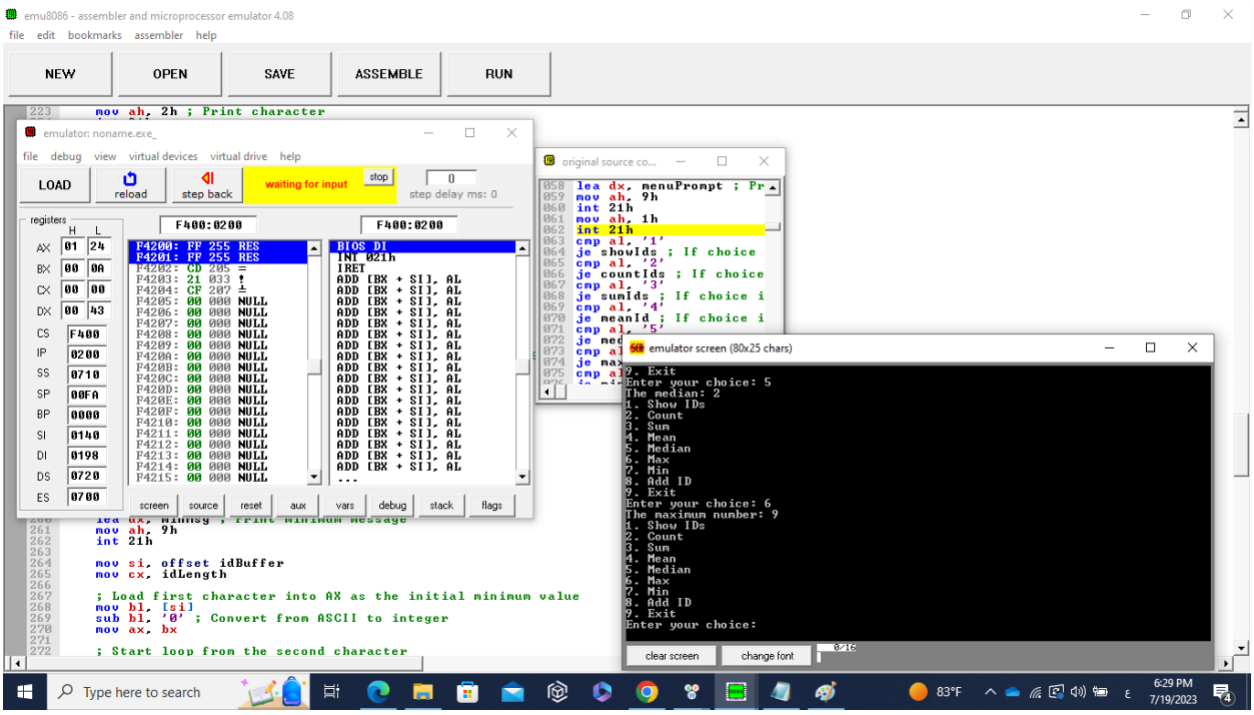

Figure 4.1.6-2 Program execution of max

4.1.7 Min

The "Min" option will find the minimum digit among all the digits in the ID's array. In the input ID "1200368", the minimum digit is 0. So, the output will be as in the image.

```
minId:
    lea dx, newLine ; Print newline
    mov ah, 9h
    int 21h
    lea dx, minMsg ; Print minimum message
    mov ah, 9h
    int 21h

    mov si, offset idBuffer
    mov cx, idLength

    ; Load first character into AX as the initial minimum value
    mov bl, [si]
    sub bl, '0' ; Convert from ASCII to integer
    mov ax, bx

    ; Start loop from the second character
    inc si
    dec cx
```

Figure 4.1.7-1 Code for calculating min

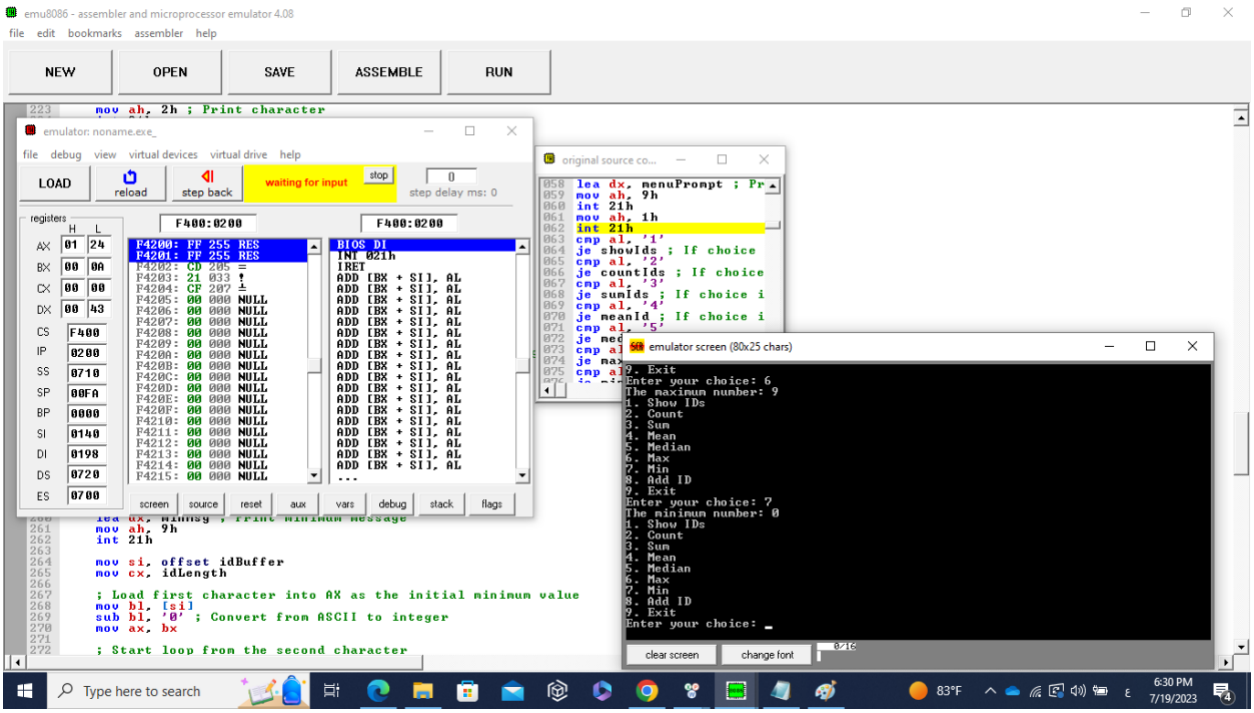

Figure 4.1.7-2 Program execution of min

4.1.8 Add New ID

The "New ID" option allows the user to enter a new ID. the output will be as in the image.

```
addId:
    lea dx, newLine ; Print newline
    mov ah, 9h
    int 21h
    lea dx, newIdPrompt ; Prompt for new ID
    mov ah, 9h
    int 21h

    mov si, offset idBuffer
    add si, idLength ; Point SI to where the new ID should start

    mov cx, 7 ; Length of an ID
```

Figure 4.1.8-1 Code for adding new ID

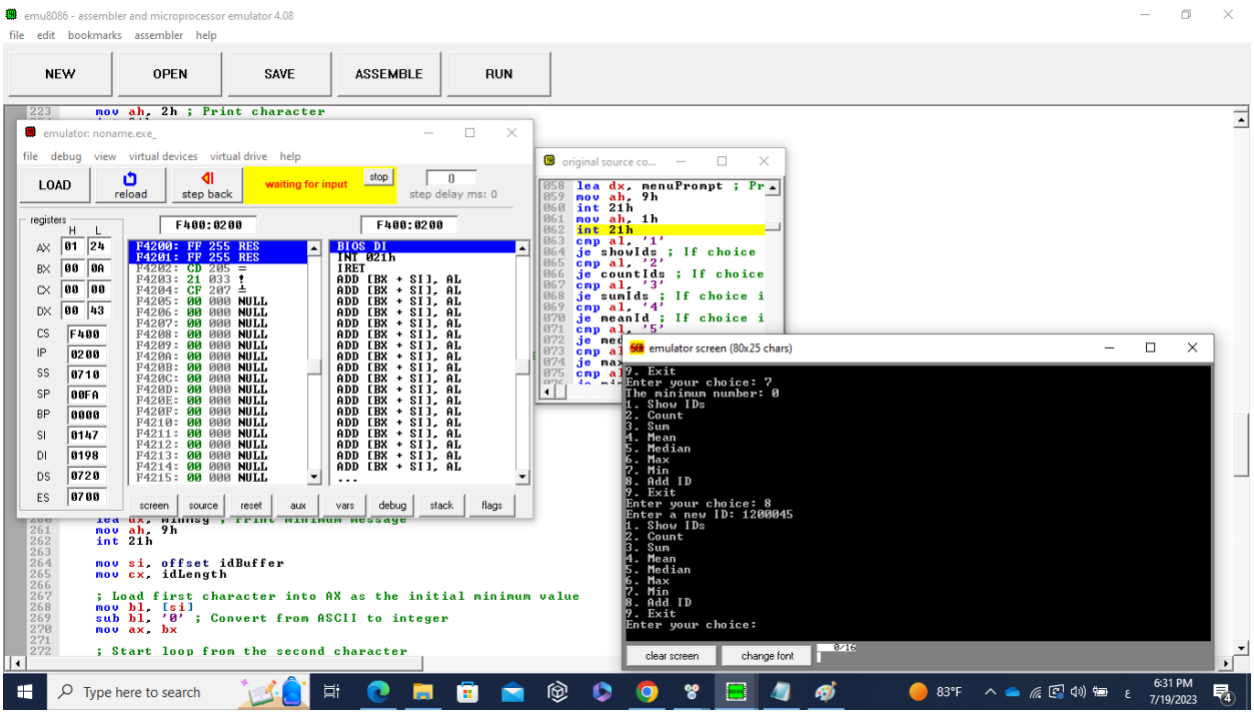

Figure 4.1.8-2 Program execution of adding an ID

4.1.9 Exit

The "Exit" option will exit the program and terminate the execution.

```
317 exitProgram:
318     mov ax, 4C00h ; Exit the program
319     int 21h
320
321 main endp
322
```

Figure 4.1.9-1 Code for exiting the program

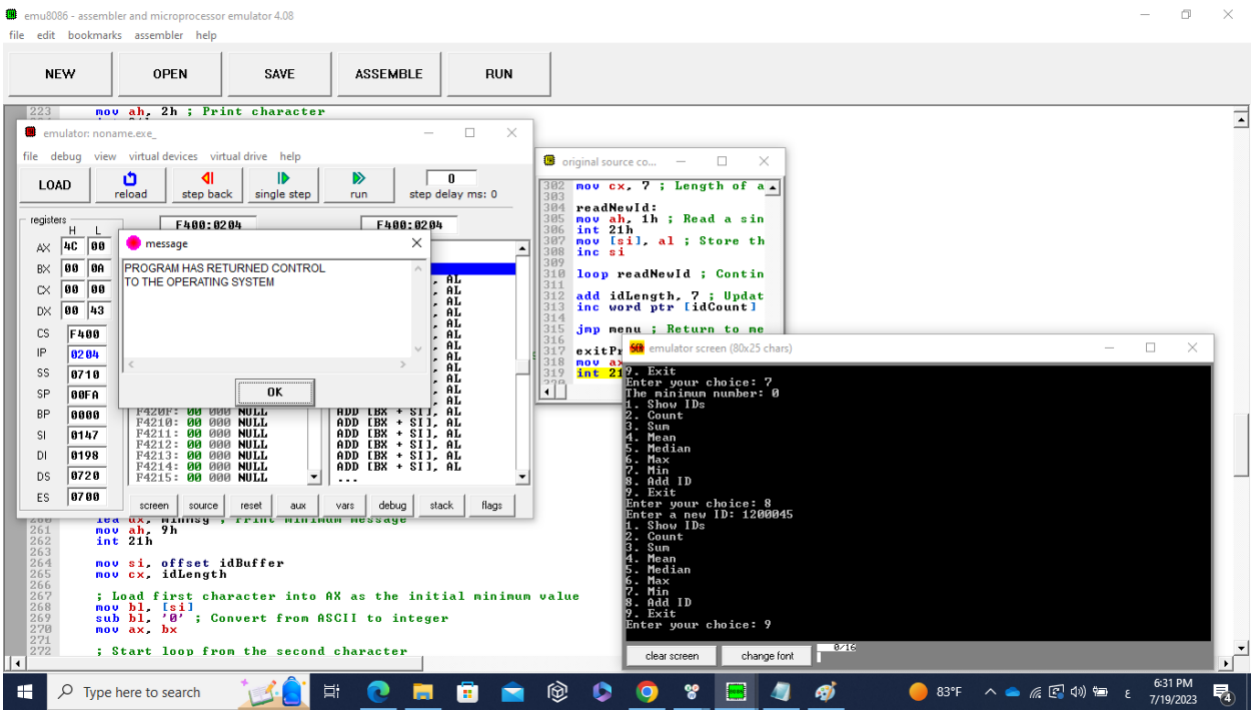

Figure 4.1.9-2 Program executing exit option

5 Conclusion

The project offered a comprehensive exploration of assembly language programming, delving into various aspects such as procedures, data storage, user inputs, control structures, and CPU interrupts. It provided valuable insights and a solid learning experience in working with assembly language.

The code demonstrated a straightforward user interface application that operates on user-entered IDs. It showcased effective management of user inputs, performing calculations, and presenting user-friendly responses. Overall, the project served as an excellent resource for understanding assembly language programming concepts and practical implementation.