Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department

A logo for a university

Description automatically generated

|  |
| --- |
| **Implementation of a Convolution-based 2D Moving 3x3 Average Filter for the 8086 Microprocessor** |

|  |  |  |
| --- | --- | --- |
| *Authors:* |  | *Supervisor:* |
| Leen Almousa | 20210268 | Dr.Heba Abdel Nabi |

*December,15, 2024*

***Abstract***

*This project uses the 8086 microprocessor to construct a 2D moving average filter for image processing. The input is an 8x8 grayscale matrix obtained from a university ID, and a 3x3 filter is used to create a 6x6 output matrix. The 8086 assembly-based application is made up procedures for computation, memory allocation, and hexadecimal output formatting. Timing analysis is used to determine the number of clock cycles and time  necessary for the procedure. This project demonstrates the application of assembly language to image processing and microprocessor-based tasks.*

Table of Contents

[1 Introduction 2](#_Toc185597376)

[2 Description of Work 2](#_Toc185597377)

[3 Output Results 7](#_Toc185597387)

4 [Flowchart 8](#_Toc185597388)

[5 Clock Cycle Calculation 8](#_Toc185597389)

[6 Conclusion 9](#_Toc185597390)

7 [Resources 9](#_Toc185597390)

Table of Figures:

[Figure 1:Code snippet to initialize the indata array 2](#_Toc185599613)

[Figure 2:Code snippet to reserve space in memory for outdata array 3](#_Toc185599614)

[Figure 3:Code snippet to initialize the strings I wanna print to the user. 3](#_Toc185599615)

[Figure 4:Procedures to print the strings 3](#_Toc185599616)

[Figure 5: Code to create a 3x3 filter 4](#_Toc185599617)

[Figure 6: Procedure to print output in correct form 5](#_Toc185599618)

[Figure 7: Procedure to print the input matrix in the correct format. 5](#_Toc185599619)

[Figure 8 : Code of the main segment calling all the procedures. 6](#_Toc185599620)

[Figure 9: Output of the program. 7](#_Toc185599621)

# Introduction

Using the 8086 CPU, this project shows how to apply a 3x3 moving average filter to an 8x8 picture matrix. The matrix is created using a university ID, and the filter computes average values to yield a 6x6 result. The software is written in 8086 assembly and handles memory management, computation, and performance analysis. The goal is to use a fundamental convolution technique to better understand microprocessor functionality in image processing.

# Description of Work

1. **Initializing an 8x8 Matrix Based on the University ID (20210268):**

* Each row of the matrix is generated by multiplying each digit of the ID by the corresponding row number.
* Values greater than 15 are replaced, as they require two hex digits to represent.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | **0** | **2** | **1** | **0** | **2** | **6** | **8** |
| **4** | **0** | **4** | **2** | **0** | **4** | **12** | **15** |
| **6** | **0** | **6** | **3** | **0** | **6** | **7** | **10** |
| **8** | **0** | **8** | **4** | **0** | **8** | **2** | **4** |
| **10** | **0** | **10** | **5** | **0** | **10** | **0** | **6** |
| **12** | **0** | **12** | **6** | **0** | **12** | **6** | **8** |
| **14** | **0** | **14** | **7** | **0** | **14** | **2** | **4** |
| **12** | **0** | **12** | **8** | **0** | **6** | **8** | **4** |

A number grid with black numbers

Description automatically generated with medium confidence

Figure :Code snippet to initialize the indata array

1. **Reserving Memory Space for the Output Array:**

An array named outdata is allocated to store the results of computations.



Figure :Code snippet to reserve space in memory for outdata array

1. **Setting Display Strings:**

Initialize the text strings that will be shown to the user during the execution of the program.



Figure :Code snippet to initialize the strings I wanna print to the user.

1. **Creating Procedures for String Display:**

Implement two procedures to display the content of the pre-defined strings on the screen.

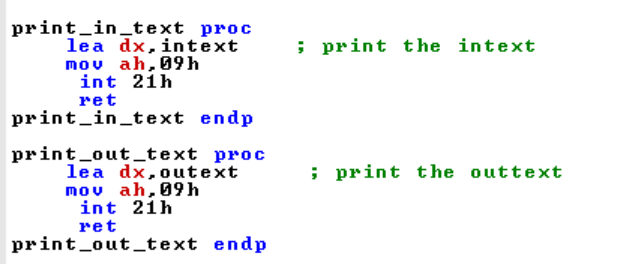


Figure :Procedures to print the strings

1. **Developing a Producer for a 3x3 Moving Average Filter:**

* Use a nested loop to iterate over the indata matrix.
* Compute the average for each 3x3 window and store the result in the corresponding cell of the outdata 6x6 matrix.

1. **Initialization:**

* lea si, indata: Loads the effective address of the input matrix (indata) into the si register, which will be used to access the matrix elements.
* lea di, outdata: Loads the effective address of the output matrix (outdata) into the di register, serving as the destination for the calculated averages.
  + mov cx, 6: Initializes the outer loop counter (cx) to 6, representing the number of rows to process.

1. **Outer Loop (rows\_loop):**
   * push cx: Saves the current value of the outer loop counter on the stack to preserve it for later restoration.
   * mov cx, 6: Initializes the inner loop counter (cx) to 6, representing the number of columns to process within each row.
2. **Inner Loop (columns\_loop):**

* mov ax, 0: Clears the accumulator register (ax) to zero. This register will be used to accumulate the sum of the pixel values within the 3x3 window.

1. **Sum Calculation:**

* A series of add al, [si] instructions are used to add the values of the nine pixels within the current 3x3 window to the accumulator register (al). The si register is incremented appropriately to access the neighboring pixels in the input matrix.

1. **Average Calculation:**
   * mov bl, 9: Moves the value 9 into the bl register.
   * div bl: Divides the sum stored in the al register by 9, resulting in the average value. The quotient is stored in al, and the remainder in ah.
2. **Storing Result:**

* mov [di], al: Stores the calculated average value at the current location pointed to by the di register in the outdata matrix.

1. **Next Column:**

* inc di: Increments the di register to point to the next location in the output matrix.
* inc si: Increments the si register to point to the next pixel in the input matrix.
* loop columns\_loop: Decrements the inner loop counter (cx) and jumps back to the beginning of the inner loop if cx is not zero.

1. **Next Row:**

* add si, 2: Adjusts the si register to point to the beginning of the next row in the input matrix.

1. **Outer Loop Control:**

* pop cx: Restores the outer loop counter from the stack.
* loop rows\_loop: Decrements the outer loop counter (cx) and jumps back to the beginning of the outer loop if cx is not zero.

1. **Procedure End:**

* ret: Returns control to the calling procedure.

A screenshot of a computer program

Description automatically generated

Figure : Code to create a 3x3 filter

1. **Formatting and Printing Input/Output Data in Hexadecimal:**

* Write a function to convert the input and output data into hexadecimal format.
* Add 30 to values below A and 37 to those above A for proper representation.
  1. **Initialization:**
* lea si, outdata: Loads the effective address of the outdata array into the si register, which will be used as a pointer to access the array elements.
* mov cx, 6: Initializes the outer loop counter (cx) to 6, representing the number of rows to be printed.
  1. **Outer Loop (r\_loop):**
* push cx: Saves the current value of the outer loop counter on the stack to preserve it for later restoration.
* mov cx, 6: Initializes the inner loop counter (cx) to 6, representing the number of columns to be printed within each row.
  1. **Inner Loop (c\_loop):**
* mov dl, [si]: Loads the current data byte from the outdata array into the dl register.
* cmp dl, 9h: Compares the loaded value with 9h (hexadecimal 9).
* jge char1: If the value is greater than or equal to 9h, it means the hexadecimal digit is A-F. The code jumps to the char1 label.
* add dl, 30h: If the value is less than 9h, add 30h to convert it to its ASCII representation (0-9).
* jmp next1: Jump to the next1 label to continue processing.
* Char1: Add 37h to the value in dl to convert it to its ASCII representation (A-F).
* mov ah, 02h: Set the function code for the DOS interrupt to print a character.
* int 21h: Invoke the DOS interrupt to print the character in dl.
* inc si: Increment the si register to point to the next element in the outdata array.
* loop c\_loop: Decrement the inner loop counter (cx) and jump back to the beginning of the inner loop if cx is not zero.
  1. **Cursor Positioning:**
* mov dl, 10: Load the ASCII code for a newline character into dl.
* mov ah, 02h: Set the function code for the DOS interrupt.
* int 21h: Print a newline character to move the cursor to the next line.
* mov dl, 13: Load the ASCII code for a carriage return character into dl.
* mov ah, 02h: Set the function code for the DOS interrupt.
* int 21h: Print a carriage return character to move the cursor to the beginning of the line.
  1. **Outer Loop Control:**
* pop cx: Restore the outer loop counter from the stack.
* loop r\_loop: Decrement the outer loop counter (cx) and jump back to the beginning of the outer loop if cx is not zero.
  1. **Procedure End:**
  + ret: Return control to the calling procedure.

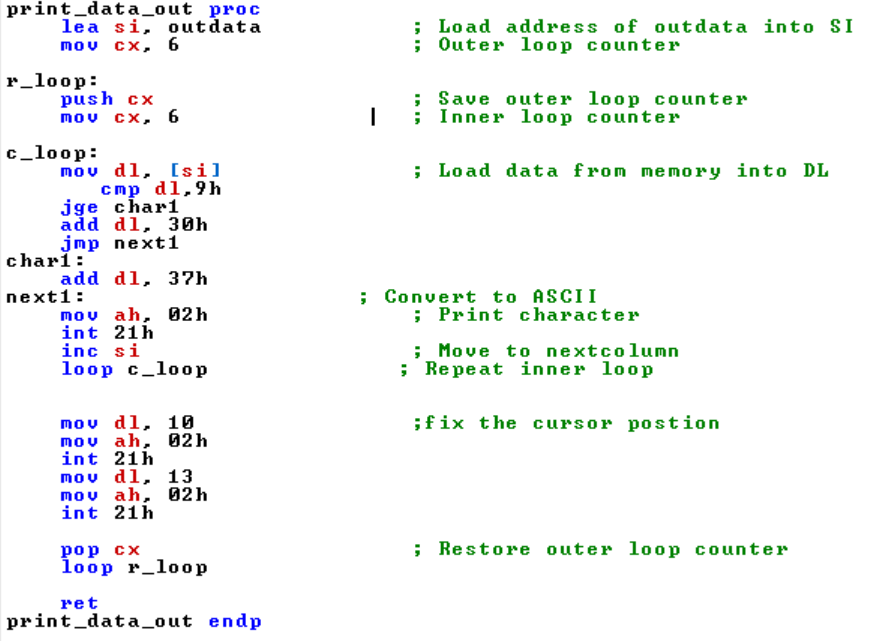


Figure : Procedure to print output in correct form

A screenshot of a computer program

Description automatically generated Similar process is done for the print\_data\_in:

Figure : Procedure to print the input matrix in the correct format.

# Main Code Workflow:

# Initialize the DS and ES segments.

# Call the predefined procedures in sequence:

# Display introductory text.

# Print the input data (indata).

# Perform the convolution using the moving average filter.

# Display the output text.

# Print the resulting outdata.

# 

Figure : Code of the main segment calling all the procedures.

# Output

**A screen shot of a computer

Description automatically generated**

Figure : Output of the program.

# 4 Flowchart

**A diagram with text and images

Description automatically generated with medium confidence**

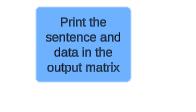
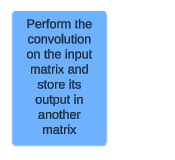
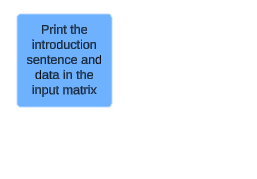
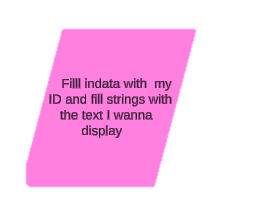


Figure : Flow chart describing the flow of the program .

# 5 Clock Cycle Calculation

Selected Procedure:

calc\_avg proc

lea si,indata 8cc

lea di,outdata 8cc

mov cx,6 4cc

rows\_loop:

push cx 11cc

mov cx,6 4cc

columns\_loop:

mov ax, 0 4cc

add al, [si] 9+EA=9+5=14cc

add al, [si+1] 9+EA=9+9=18cc

add al, [si+2] 9+EA=9+9=18cc

add al, [si+8] 9+EA=9+9=18cc

add al, [si+9] 9+EA=9+9=18cc

add al, [si+10] 9+EA=9+9=18cc

add al, [si+16 ] 9+EA=9+9=18cc

add al, [si+17] 9+EA=9+9=18cc

add al, [si+18] 9+EA=9+9=18cc

mov bl, 9 4cc

div bl 90cc

mov [di], al 16+EA=16+5=21cc

inc di 2cc

inc si 2cc

loop columns\_loop taken 17 cc not taken 5cc

add si,2 4cc

pop cx 8cc

loop rows\_loop taken 17 cc not taken 5cc

ret intrasegment 8cc

calc\_avg endp

Inner loop :

* Taken 5 times

5\*(4+14+8\*18+4+90+21+2+2+17)=5\*298=1490

* Not taken once

(4+14+8\*18+4+90+21+2+2+5)=286

* All inner\_loop 1776

Outer loop:

* Taken 5 times

5\*(11+4+1776+4+8+17)=9100

* Not taken once

(11+4+1776+4+8+5)=1808

* All outer\_loop 10908

Full code:

* (8+8+4+8+10908)=10936 cc

Timing:

Clock Frequency = 5 MHz

Time per Clock = 1/(5×10^6)

Time for Procedure = Total Cycles × Time per Clock

=10936\* 1/(5×10^6)=0.0021872

# 6 Conclusion

This project successfully constructed a 2D moving average filter on the 8086 microprocessor, demonstrating the application of assembly language in image processing. The 3x3 filter applied to the 8x8 matrix resulted in a 6x6 output, demonstrating the filter's smoothing effect. The timing analysis of the technique also supplied information about the microprocessor's performance. Overall, the study demonstrated the practical use of low-level programming in computational tasks.

# 7 Resources:

1. The Art of Assembly Language by Randall Hyde
2. *Assembly Language for x86 Processors* by Kip R. Irvine
3. *Course material from the Microprocessor course in Prince*