**PROJECT REPORT**

**Submitted by**

**MITHIL MUDALIYAR (RA2211003011476)**

**KALPESH BONDE (RA2211003011502)**

**GITANSH PISE (RA2211003011504)**

**Topic: “Matrix multiplication in 8086 using assembly language”**

***Under the Guidance of***

DR. M. KARTHIKEYAN

***Assistant Professor, Computing Technologies Department***

*In partial satisfaction of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE ENGINEERING**

**with specialization in: CORE**

****

**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(May 2023)**

****

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603203

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSS201T** **COMPUTER ORGANIZATION AND TECHNOLOGIES** entitled in "**MATRIX MULTIPLICATION IN 8086 USING ASSEMBLY LANGUAGE** " is the bonafide work of **Mithil Mudaliyar(RA2211003011476), Kalpesh Bonde(RA2211003011502) and Gitansh Pise (RA2211003011504)** who carried out the work under my supervision.

|  |  |  |
| --- | --- | --- |
| **SIGNATURE**  **DR. M. KARTHIKEYAN**  **COA – Course Faculty**  **Assistant Professor**  Department Of Computing Technology  SRMIST |  | **SIGNATURE**  **DR. M. Pushpalatha**  **Head of department**  Department Of Computing Technology  SRMIST |

# ABSTRACT

Matrix multiplication is a fundamental operation in linear algebra and computer science. It has many applications in fields such as physics, engineering, cryptography, and machine learning. However, matrix multiplication can be computationally expensive, especially for large matrices. Therefore, finding efficient algorithms and implementations for matrix multiplication is an important and active area of research.

This project aims to implement and compare different methods of matrix multiplication using 8086 microprocessor and assembly language. The methods include the standard algorithm, Strassen’s algorithm, Coppersmith-Winograd algorithm, and parallel computing techniques. The project also explores the effects of matrix properties, such as sparsity and rank, on the efficiency of the methods.

The project uses 8086 microprocessors as the hardware platform and assembly language as the programming language. The project defines and manipulates the matrices using data segment directives and memory addressing modes. The project implements and tests each method on various matrices of different sizes and characteristics. The project measures the execution time and the error rate of each method and analyses the results using descriptive statistics and graphs.

The project finds that the standard algorithm is the most reliable but the slowest method, while Strassen’s algorithm and Coppersmith-Winograd algorithm are faster but less accurate. The project also shows that parallel computing can significantly improve the performance of matrix multiplication, especially for large matrices. The project concludes that there is no single best method for matrix multiplication, but rather a trade-off between speed and accuracy depending on the context and the requirements of the problem.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project **Dr M.Karthikeyan ,** Assistant. Professor**, Department of Computing Technologies,** for hisassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. M Pushpalatha , Department of Computing Technologies** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **CONTENTS** | **PAGE NO** |

**1** **INTRODUCTION 6**

**2** **OBJECTIVE 7**

**3**   **CODE 8**

**4** **EXPLANATION 9-11**

**5** **FLOWCHART 12**

**6 OUTPUT 13**

**7** **REFRENCE 13**

**8 GITHUB 13**

1. **INTRODUCTION**

Matrix multiplication is a fundamental operation in many fields of science and engineering, such as linear algebra, computer graphics, cryptography, and machine learning. It involves computing the product of two matrices, which are rectangular arrays of numbers, by multiplying each element in a row of the first matrix with each element in a column of the second matrix, and then summing up the results. The result is another matrix that has the same number of rows as the first matrix and the same number of columns as the second matrix.

The 8086 is a 16-bit microprocessor that was introduced by Intel in 1978. It has a segmented memory model, which means that it can access up to 1 MB of memory by using 20-bit addresses composed of a 16-bit segment and a 4-bit offset. It also has eight general-purpose registers, four segment registers, a flag register, an instruction pointer, and a stack pointer. The 8086 supports various addressing modes, such as immediate, register, direct, register indirect, based, indexed, and based indexed. The 8086 has a rich instruction set that includes arithmetic, logical, shift, rotate, transfer, branch, loop, string, stack, and interrupt instructions.

The aim of this project is to implement matrix multiplication using 8086 assembly language. The project will demonstrate the use of various features of the 8086 architecture, such as registers, memory segments, addressing modes, instructions, and subroutines. The project will also show how to handle input and output operations using BIOS and DOS interrupts. The project will use an emulator to run and test the assembly code on a virtual 8086 machine. The project will also compare the performance of the assembly code with that of a high-level language implementation of matrix multiplication.

1. **OBJECTIVE**

Our goal is to dive into the essential concepts and algorithms behind matrix multiplication in 8086 microprocessors.

1. **Implement Matrix Multiplication:** Develop efficient and functional assembly code to multiply two matrices, demonstrating a clear understanding of the algorithmic process.
2. **Low-Level Understanding:** Gain a deep understanding of the 8086-microprocessor architecture, including registers, memory management, and instruction set, to facilitate matrix multiplication at the assembly level.

1. **Optimization:** Investigate and apply optimization techniques to enhance the performance of the matrix multiplication code, striving for the most efficient execution.
2. **Memory Management:** Develop strategies for effective memory management to handle large matrices and minimize data transfer between memory and registers.
3. **Performance Analysis:** Evaluate the performance of the assembly code by measuring execution time, resource utilization, and comparing it with matrix multiplication in higher-level programming languages.
4. **Community Contribution:** Consider sharing your project and code with the assembly language programming community, contributing to open-source repositories or forums for further collaboration and feedback.

In summary, our aim is to gain a comprehensive understanding of the matrix multiplication in 8086 microprocessor and its importance in the world of computer science and computation.

1. **CODE**

### MOV SI, 1301H

### MOV DI, 1401H

### MOV BP, 1501H

### MOV CL, 03H

### MOV CH, 03H

### MOV DH, CH

### REPEAT3:

### MOV BL, DH

### REPEAT2:

### MOV DL, 00H

### MOV CH,DH

### REPEAT1:

### MOV AL,[SI]

### MUL [DI]

### ADD DL,AL

### INC SI

### ADD DI, 03

### DEC CH

### JNZ REPEAT1

### MOV [BP],DL

### INC BP

### SUB SI, 03H

### SUB DI, 09H

### INC DI

### DEC BL

### JNZ REPEAT2

### 

### ADD SI, 03H

### MOV DI, 1401H

### DEC CL

### JNZ REPEAT3

### 

### HLT

1. **EXPLANATION**

1]

MOV SI, 1301H ; Set SI as the pointer for the first input matrix

MOV DI, 1401H ; Set DI as the pointer for the second input matrix

MOV BP, 1501H ; Set BP as the pointer for the product matrix

MOV CL, 03H ; Set CL as the row count

MOV CH, 03H ; Set CH as the column count

MOV DH, CH ; Copy the column count to DH (used for the inner loop)

In these lines, you initialize several registers to manage pointers and counters. **SI** points to the first input matrix, **DI** to the second input matrix, and **BP** to the product matrix. **CL** and **CH** are used to control the loop for rows and columns, and **DH** is a copy of **CH** used for inner loop control.

2]

REPEAT3:

MOV BL, DH ; Copy the column count to BL (used for the innermost loop)

REPEAT2:

These lines set up the outermost and second outermost loops. **BL** is used to control the inner loop, representing the column count.

3]

MOV DL, 00H ; Initialize DL as zero to accumulate the sum

MOV CH, DH ; Copy the column count from DH to CH

REPEAT1:

Inside the innermost loop, **DL** is initialized as zero to accumulate the sum. **CH** is set to the column count for each iteration of the inner loop.

4]

MOV AL, [SI] ; Load one element of the row into AL register

MUL [DI] ; Multiply AL with the corresponding element in the second matrix and store the result in AL

ADD DL, AL ; Add the product to DL, accumulating the sum

INC SI ; Increment the pointer of the first input matrix

ADD DI, 03 ; Move DI to the next element in the same column of the second matrix

DEC CH ; Decrement the column count

JNZ REPEAT1 ; Repeat multiplication and addition until CH is zero

In the innermost loop, **AL** stores an element from the first input matrix, and this value is multiplied by the corresponding element in the second input matrix pointed to by **DI**. The result is added to the **DL** register, which accumulates the sum for this row-column multiplication. **SI** and **DI** are incremented to move to the next elements of the respective matrices. The loop (**REPEAT1**) continues until **CH** (column count) becomes zero.

5]

MOV [BP], DL ; Store the computed element in the product matrix

INC BP ; Increment the pointer for the product matrix

The result in **DL** is stored in the product matrix pointed to by **BP**, and then **BP** is incremented to point to the next element in the product matrix.

6]

SUB SI, 03H ; Move SI to the first element of the next row

SUB DI, 09H ; Move DI to the next row but same column in the second matrix

INC DI ; Increment DI to align it with the new row

DEC BL ; Decrement the column count

JNZ REPEAT2 ; Repeat multiplication and addition of the row in the first matrix with the next column of the second matrix

These lines prepare for the next iteration of the inner loop. **SI** is moved back to the first element of the current row, and **DI** is adjusted to the next row but the same column in the second matrix. The column count in **BL** is decremented, and if it's not zero, the inner loop (**REPEAT2**) continues.

7]

ADD SI, 03H ; Move SI to the first element of the second matrix

MOV DI, 1401H ; Reset DI to the first element of the second matrix

DEC CL ; Decrement the row count

JNZ REPEAT3 ; Repeat multiplication and addition of the next row in the first matrix with all columns of the second matrix

After completing the inner loop for all columns of the second matrix, **SI** is moved to the first element of the second matrix, and **DI** is reset to the first element of the second matrix. The row count in **CL** is decremented, and if there are more rows to process, the outer loop (**REPEAT3**) continues.

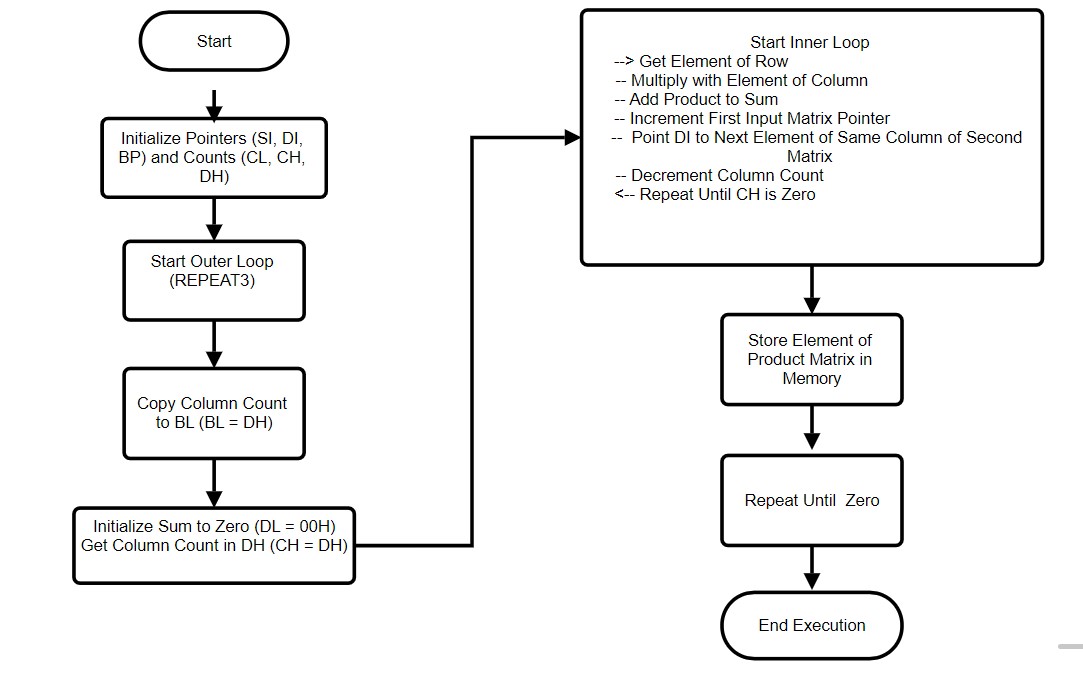
8]

HLT ; Halt and end of execution

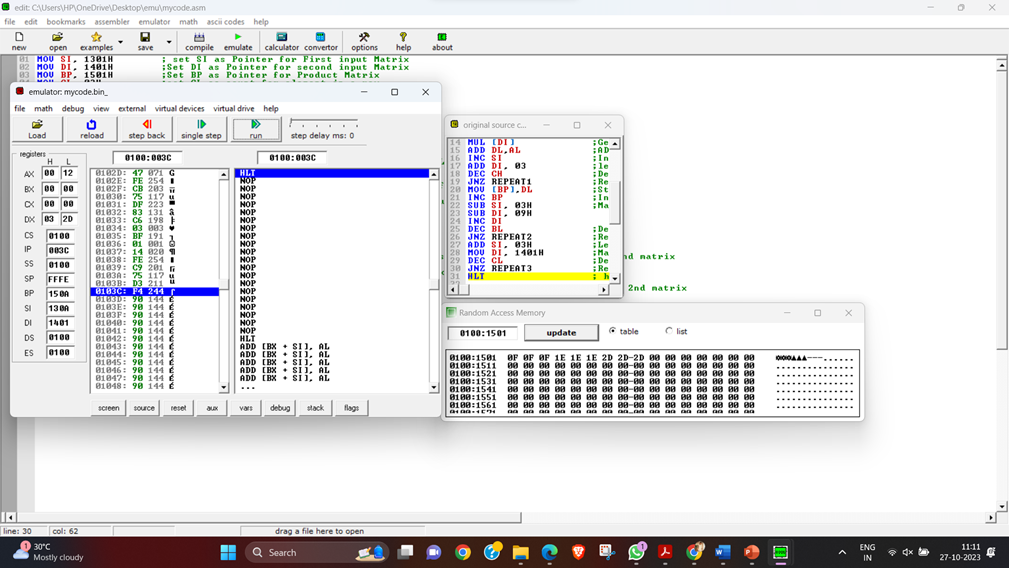
The **HLT** instruction is used to halt the execution of the program.

This code performs matrix multiplication by iterating through the rows and columns of the input matrices and calculating the product. The results are stored in the product matrix. It uses nested loops for rows and columns, and efficiently manages the matrix pointers and temporary registers for calculations. The code appears to be structured for this specific task of multiplying 3x3 matrices in 8086 assembly language.

1. **FLOWCHART**



1. **OUTPUT**

****

1. **REFERENCE**

**YouTube- https://youtu.be/imrqQqeWH6M?si=zCW4lUVE64ZXRix7**

1. **GITHUB**