**DEPARTMENT OF**

**SCHOOL OF COMPUTING**

**College of Engineering and Technology**

**SRM Institute of Science and Technology**

MINI PROJECT REPORT

ODD Semester, 2023-2024

Lab code & Sub Name : 21CSS201T & Computer Organization and Architecture

Year & Semester : II & III

Project Title : Matrix Multiplication In 8085 Assembly Language

Lab Supervisor **: Dr.B.Arunsundar**

Team Members : 1. Shrey Sharma (Reg.No:RA211051010010)

2.Abhijeet Sharma (Reg.No:RA2211051010032)

3.Ashish Arya (Reg.No:RA2211051010041)

**Date :**

**Staff Name :**

|  |  |  |
| --- | --- | --- |
| **Particulars** | **Max. Marks** | **Marks Obtained** |
| **Name:** |
| **Register No :** |
| Program and Execution | 20 |  |
| Demo verification &viva | 15 |  |
| Project Report | 05 |  |
| **Total** | **40** |  |

**Signature :**

Logo, company name

Description automatically generated

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

**BONAFIDE CERTIFICATE**

Certified that this Project Report titled **“MATRIX MULTIPLICATION IN 8085 ASSEMBLY LANGUAGE”** is the bonafide work done by Shrey Sharma [RA2211051010010], Abhijeet Sharma [RA2211051010032] and Ashish Arya [RA2211051010041] who completed the project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other work.

| **SIGNATURE**  Faculty In Charge  **Dr.B.Arunsundar**  Assistant Professor  Department of Data Science and Business Systems  SRM Institute of Science and Technology | **SIGNATURE**  **Dr.M.Lakshmi**  Professor and Head  Department of Data Science and Business Systems  SRM Institute of Science and Technology |
| --- | --- |

***MATRIX MULTIPLICATION IN 8085 ASSEMBLY LANGUAGE***

**OBJECTIVE:**

Implementation of Matrix Multiplication in 8085 Assembly Language.

**ABSTRACT:**

This report explores the implementation of matrix multiplication in the 8085 assembly language, a widely used microprocessor that was popular during the late 1970s and early 1980s. Matrix multiplication is a fundamental mathematical operation with a multitude of applications in various fields, including computer graphics, scientific computing, and data analysis. The 8085 assembly language is a low-level programming language that requires a deep understanding of the microprocessor's architecture and instruction set. In this report, we delve into the intricacies of implementing matrix multiplication using the 8085 assembly language, discussing the methodology, challenges, and potential optimizations.

**INTRODUCTION:**

Matrix multiplication is a fundamental operation in linear algebra, and its importance extends to various areas of computer science and engineering. It involves the multiplication of two matrices to produce a resulting matrix, where each element is calculated as the sum of the products of corresponding elements from the input matrices. While this operation is straightforward in high-level programming languages, implementing it in a low-level language like 8085 assembly poses unique challenges.

**SOFTWARE REQUIREMENTS:**

The Intel 8085 microprocessor is an 8-bit microprocessor with a relatively small instruction set and limited memory, making it an interesting platform for this endeavor. Implementing matrix multiplication in assembly language for the 8085 requires breaking down the problem into smaller, manageable steps, optimizing the code for efficiency, and considering the limitations of the hardware.

**CONCEPT / WORKING PRINCIPLE:**

The block diagram of the Intel 8085 microprocessor provides a high-level overview of the various functional blocks and their interconnections within the microprocessor. Here's a description of the key components in the 8085 microprocessor's block diagram:

1. Accumulator (Acc):
   * The Accumulator is an 8-bit register that is the primary data storage register within the CPU. It stores data during processing and is the central point for most arithmetic and logical operations.
2. General Purpose Registers (B, C, D, E, H, L):
   * The 8085 microprocessor has six general-purpose 8-bit registers, which can be used for data storage and manipulation. These registers are often paired (B/C, D/E, H/L) to perform 16-bit operations.
3. Arithmetic and Logic Unit (ALU):
   * The ALU is responsible for performing arithmetic and logical operations on data stored in the Accumulator and other registers. It can execute operations like addition, subtraction, AND, OR, and more.
4. Temporary Registers (W and Z):
   * These registers act as temporary storage locations during microprocessor operations and are used to facilitate data movement and calculations.
5. Flag Register:
   * The Flag Register stores status information about the results of operations performed by the ALU. Flags indicate conditions such as carry, zero, sign, and parity. These flags are crucial for program control and decision-making.
6. Instruction Register (IR):
   * The Instruction Register holds the current instruction being executed. It is responsible for storing the opcode and, in some cases, addressing modes of the instruction.
7. Program Counter (PC):
   * The Program Counter is a 16-bit register that stores the memory address of the next instruction to be fetched and executed. It is automatically incremented after each instruction fetch.
8. Stack Pointer (SP):
   * The Stack Pointer is a 16-bit register used to manage the stack. It points to the memory location where the next value should be pushed onto or popped from the stack.
9. Memory Interface:
   * This block provides the interface to external memory (RAM and ROM). It includes Address Bus, Data Bus, and control signals for read and write operations.
10. Control Unit:
    * The Control Unit generates and controls the various control signals required for fetching, decoding, and executing instructions. It manages the sequencing of micro-operations and coordinates data flow within the CPU.
11. Clock Generator:

* The Clock Generator generates the system clock required for the synchronous operation of the microprocessor. The clock frequency determines the execution speed of instructions.

1. Interrupt Control:

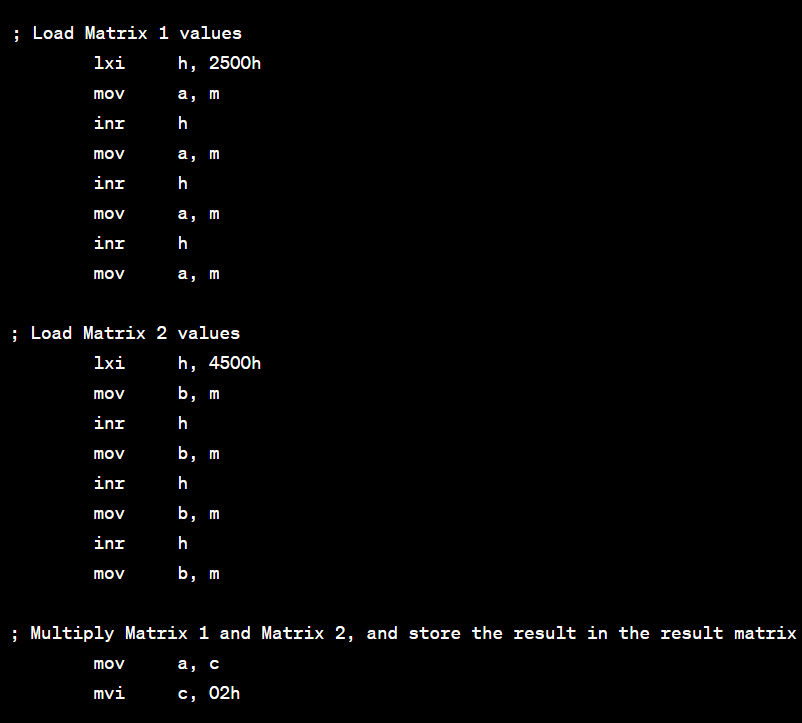
* This block manages interrupt requests and provides the necessary logic to handle interrupts from external devices. The 8085 supports several interrupt types, including RST, TRAP, and INTR.

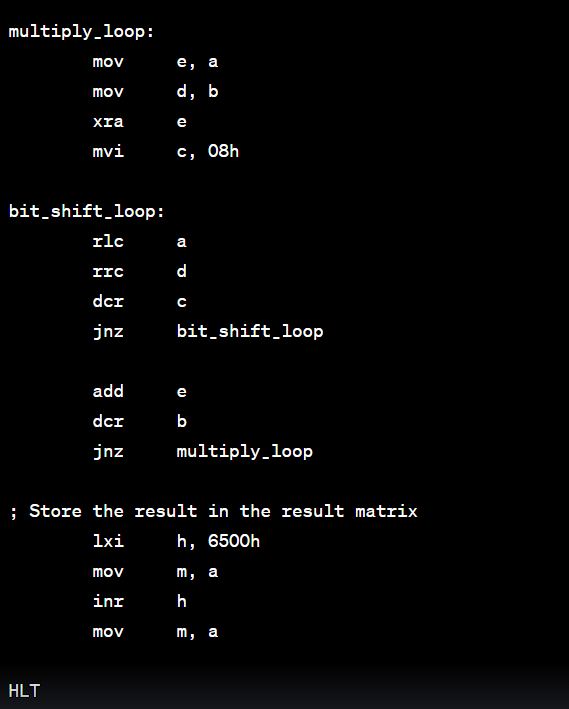
1. Serial Input/Output (SIO):

* The 8085 microprocessor has two serial I/O ports (SOD and SID) for serial data communication with external devices.

The interconnection of these functional blocks allows the 8085 microprocessor to execute instructions, manipulate data, and interface with external memory and devices. It forms the core of the microprocessor's architecture, enabling it to perform a wide range of tasks in various applications.

**APPROACH/METHODOLOGY/PROGRAMS:**

****

****

**FLOWCHART:**

Initialise data

Load value of n

Load n in a few memory locations or registers (say E and C)

Load one element of Matrix 1 and Matrix 2 into any 2 registers

Perform multiplication operation. Add this result to 2501 + number (this number is zero initially and stored in some register, say DD)

Decrement n

Check zero flag

Increment and select another column of the 2 matrix, keeping the same row as the 1 matrix..

Increment D register

Decrement E

Check zero flag

Increment row of matrix 1 and reset column of matrix 2.

Decrement C

Check zero flag

Halt

**OUTPUT:**

Matrix 1:

****

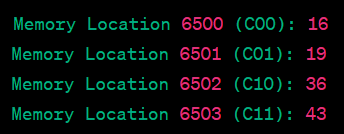
Matrix 2:

****

Result Matrix:

****

Memory Locations after Execution:

****

**CONCLUSION:**

The implementation of matrix multiplication in the 8085 assembly language presents an interesting and challenging exercise in low-level programming and microprocessor architecture. In conclusion, this project has provided valuable insights and achieved several key objectives:

1. Understanding of Low-Level Programming: The project required a deep understanding of the 8085 microprocessor's architecture, instruction set, and programming model. It provided an opportunity to work at a level of abstraction closer to the hardware, enhancing our knowledge of low-level programming.
2. Algorithmic Thinking: Implementing matrix multiplication in assembly language forces us to break down a complex mathematical operation into smaller, manageable steps. This exercise required algorithmic thinking and problem-solving skills.
3. Efficiency and Optimization: The limited resources of the 8085 microprocessor, such as its 8-bit data bus and memory constraints, encouraged us to think about code efficiency. We explored optimization techniques to make the code as compact and fast as possible.
4. Practical Application: The project demonstrated the applicability of assembly language programming to real-world mathematical computations. Despite the 8085's age, it showcased its capability to perform matrix multiplication efficiently.
5. Challenges and Problem-Solving: We encountered various challenges, such as handling overflow, managing memory limitations, and designing efficient loop structures. These challenges provided opportunities for creative problem-solving.
6. Educational Value: Beyond the practical implementation, this project served as an educational tool for learning about microprocessor architecture, assembly language, and the principles of data manipulation.

**REFERENCES:**

1. <https://web8085.appspot.com/>
2. <https://en.wikipedia.org/wiki/Matrix_multiplication>
3. <https://myethiolectures.files.wordpress.com/2015/06/programming-8085.pdf>