Saahil Anup Karnik

06/11/2021

Final Report Submission

Introduction

This project is an extension of the existing matrix project done by my project batchmates. Since most matrix math features were already covered in the previous project I have focused on the following aspects of matrix: Transpose of a Matrix, Scalar Multiple of a Matrix, Null Matrix. Most of this research involved utilizing the text, *Assembly Language for x86 Processors* written by Kip Irvine (who also is responsible for the library mentioned above), to reinforce the implementation of algorithms via MASM.

Initially the user is provided with a choice by a prompted by userChoice byte to select operation for Transpose of Matrix, for Scalar multiple, for Null Matrix. The code is opened with standard assembly protocol for the execution of the above command. If the user enters the correct figure, then output is displayed on the screen. If the user enters incorrect input due to the errorChoice byte, an error message is flashed on the screen.

For all the matrix functions - Transpose of Matrix, Scalar multiple and Null Matrix, we are taking the reference of earlier program of initialization and display of matrix. First step is to initiate matrix which is already prepared by the code mov ebp and prompt the user for the size of matrix by moving matrixSize in loop. If we want to have the input number for matrix commands in decimal format, we use the call WriteDec. As iPos represents the row, it needs to be multiplied by the matrixSize\* TYPE matrixA to reference the correct row memory address. During the display of matrix, as loop counter decreases, iPos remains constant, therefore iPos represents the current row. Similarly, the matrix is displayed by mov ecx, matrixSize and mov eax, matrixSize.

Transpose Matrix

First the input value for matrix is taken and it is saved at offset address of Matrix A. The matrix transpose algorithm was implemented by using nested loops and the add or sub function on matching (i,j) indexes on matrix A.

Two loop of matrix size is used. The inner loop is for data exchange. The outer loop is for increasing pointer value. The indices are referenced by using an i-position value for the outer loop, and a j-position value for the inner loop. Each entry in the ith row can be accessed by the following math: jPos\*SIZE DWORD. Each row can be accessed by the following math: iPos \* SIZE DWORD \* SIZE A.

The count is of Matrix size. The data is saved to a new memory location possible 400 Bytes ahead. After the exchang, the DATA is displayed. The data is displayed from offset address of Matrix Pointer of Matrix A. The output is stored in Matrix C of the matching memory offset.

Scalar Matrix

User is provided option to choose for scalar matrix. All the pointer ebp are moved with offset address of Matrix A. The DATA is input to the Offset address of Matrix A. The DATA is saved and displayed.

Two loops are utilized. The inner loop is for exchange of DATA to accumulator. The scalar multiple is moved to register. The multiplication is performed, and DATA is saved to same pointer address means Matrix A is modified. When count ends the sequence shifts to display of modified Matrix A. Final display of matrix with scalar multiple is derived with the help of mul ebx mov tempValPos,eax command.

Null Matrix

User is provided option to choose for Null matrix. The null matrix calculation relies on using the XOR operation. The XOR operation sets the resultant bit to 1, if and only if the bits from the operands are different. If the bits from the operands are same (both 0/both 1), the resultant bit is cleared to 0. XORing an operand with itself changes the operand to 0. This is used to clear a register. Also, the CMP function checks whether each element of the matrix is 0 or not. If yes, the matrix is null. If not, the matrix is not null.

The Data pointer is initialized with offset address of Matrix A. The Matrix A is input and displayed. The counter of size equivalent to Matrix A is used. Two counter loops are used. The outer loop is for count up to Matrix A size. The inner loop is for comparison of DATA with zero. If DATA is NOT equal to zero a jump occurs and Not Null is displayed. Else if DATA is equal to zero then loop continues for further inspection.

Challenges

As most of the project was already written by my group, the main challenge I had in this project was to understand the logic behind the code written by my teammates. Furthermore, as I was writing this project alone, I was unable to derive inputs that can be derived during group discussions. Also, I am not very comfortable with the MASM language, so I had to put in efforts to relearn it.

Conclusion

Overall, the goal of the project was successful. The operations mentioned in the introduction were implemented correctly and give the user a real output. Because this is such a low-level language, the outputs of these operations could be used for solving real-world problems at markedly fast speeds for a single-threaded operation.