# CSCI 592
# LAB ASSIGNMENT – 8

Written by
DINESH SEVETI

Date: 04-05-2025

**OBJECTIVE**

The objective of this project is to implement matrix multiplication of two 3x3 matrices in both EASy68K and RISC-V assembly languages without utilizing loops. The multiplication will be performed explicitly for each element of the resulting matrix.

**TECHNOLOGY USED**

- EASy68K: An assembler and emulator for the Motorola 68000 microprocessor and simulator for running and debugging Assembly Code
- Venus: A RISC-V Simulator for running and debugging RISC-V assembly code.

**PROCEDURE**

- In this project, we began by defining two 3x3 matrices, A and B, with predefined values, along with a result matrix C where the product of A and B would be stored.
- The next step was to manually compute the elements of matrix C by performing element-wise multiplication of the corresponding row from matrix A and column from matrix B.
- Each element of C was calculated as the sum of the products of the corresponding elements in A and B. For instance, C[0][0] was computed as A[0][0]*B[0][0] + A[0][1]*B[1][0] + A[0][2]*B[2][0], and the same approach was followed for all elements of matrix C.
- The EASy68K assembly program was then written to implement the matrix multiplication by directly accessing each matrix element from memory and storing the results in matrix C after each calculation.
- Similarly, the RISC-V assembly code was written with direct addressing for the matrix elements, manually performing the calculations for each element of the resulting matrix and storing them in the appropriate locations.
- After coding, both programs were tested and verified using their respective simulators to ensure that the outputs matched the expected results.
- Once the matrix multiplication was complete, the programs gracefully exited using a system call or exit instruction.

**OPERATIONS**

- The first operation in this project involves defining two 3x3 matrices, A and B, with predefined values. A third 3x3 matrix, C, is also defined to store the results of the matrix multiplication.
- Next, for each element of matrix C, the sum of the element-wise multiplication of the corresponding row from matrix A and column from matrix B is manually computed. This involves calculating the dot product of the rows of matrix A with the columns of matrix B for each element of matrix C.
- In both the EASy68K and RISC-V implementations, the elements of matrices A and B are directly accessed using their memory addresses. These elements are then loaded into registers for the computation.
- After computing the products and summing them for each element of matrix C, the results are stored in the corresponding location in the result matrix C.

- The assembly code for both EASy68K and RISC-V is then executed in their respective simulators to perform the matrix multiplication. Each instruction in the code performs a specific operation, such as loading matrix elements into registers, performing multiplication, adding the results, and storing the final values in memory.
- Finally, after performing the matrix multiplication and storing the results in matrix C, the program exits gracefully using a system call or exit instruction.

## ALGORITHM

- The algorithm begins by defining two 3x3 matrices, A and B, with predefined values. A third 3x3 matrix, C, is also defined to store the product of matrices A and B.
- For each element of matrix C, denoted as C[i][j], where both i and j range from 0 to 2, the result element is initialized to 0. The dot product of the i-th row of matrix A and the j-th column of matrix B is then calculated. Specifically, the dot product for each element C[i][j] is computed as: C[i][j] = A[i][0] * B[0][j] + A[i][1] * B[1][j] + A[i][2] * B[2][j].
- After the computation, the result is stored in the corresponding position in matrix C. This process is repeated for all elements of matrix C, manually calculating each value through element-wise multiplication.
- The algorithm relies on directly accessing the elements of matrices A and B using memory addresses, and after all the elements of matrix C have been calculated and stored, the program terminates with a system call or exit instruction. This approach ensures matrix multiplication is performed step by step without using loops or higher-level constructs.

## CODE LISTING

## EASy68K IMPLEMENTATION

```
 ORG $1000

 MOVE.L  #matrix1,A0    ; Load address of matrix1 into A0
 MOVE.L  #matrix2,A1    ; Load address of matrix2 into A1
 MOVE.L  #result,A2     ; Load address of result matrix into A2
; Calculate result[0][0]
 MOVE.L  0(A0),D0       ; D0 = matrix1[0][0]
 MOVE.L  0(A1),D1       ; D1 = matrix2[0][0]
 MULS    D1,D0          ; D0 *= D1
 MOVE.L  4(A0),D2       ; D2 = matrix1[0][1]
 MOVE.L  12(A1),D3      ; D3 = matrix2[1][0]
 MULS    D3,D2
 ADD.L   D2,D0
 MOVE.L  8(A0),D4
 MOVE.L  24(A1),D5
 MULS    D5,D4
 ADD.L   D4,D0
 MOVE.L  D0,0(A2)
; Calculate result[0][1]
 MOVE.L  0(A0),D0
 MOVE.L  4(A1),D1
```

```
MULS    D1,D0
MOVE.L  4(A0),D2
MOVE.L  16(A1),D3
MULS    D3,D2
ADD.L   D2,D0
MOVE.L  8(A0),D4
MOVE.L  28(A1),D5
MULS    D5,D4
ADD.L   D4,D0
MOVE.L  D0,4(A2)
; Calculate result[0][2]
MOVE.L  0(A0),D0
MOVE.L  8(A1),D1
MULS    D1,D0
MOVE.L  4(A0),D2
MOVE.L  20(A1),D3
MULS    D3,D2
ADD.L   D2,D0

MOVE.L  8(A0),D4
MOVE.L  32(A1),D5
MULS    D5,D4
ADD.L   D4,D0
MOVE.L  D0,8(A2)
;Calculate result[1][0]
MOVE.L  12(A0),D0
MOVE.L  0(A1),D1
MULS    D1,D0
MOVE.L  16(A0),D2
MOVE.L  12(A1),D3
MULS    D3,D2
ADD.L   D2,D0
MOVE.L  20(A0),D4
MOVE.L  24(A1),D5
MULS    D5,D4
ADD.L   D4,D0
MOVE.L  D0,12(A2)
;Calculate result[1][1]
MOVE.L  12(A0),D0
MOVE.L  4(A1),D1
MULS    D1,D0
MOVE.L  16(A0),D2
MOVE.L  16(A1),D3
MULS    D3,D2
ADD.L   D2,D0
MOVE.L  20(A0),D4
MOVE.L  28(A1),D5
MULS    D5,D4
ADD.L   D4,D0
MOVE.L  D0,16(A2)
;Calculate result[1][2]
MOVE.L  12(A0),D0
MOVE.L  8(A1),D1
MULS    D1,D0
MOVE.L  16(A0),D2
MOVE.L  20(A1),D3
MULS    D3,D2
ADD.L   D2,D0
MOVE.L  20(A0),D4
MOVE.L  32(A1),D5
MULS    D5,D4
ADD.L   D4,D0
MOVE.L  D0,20(A2)
;Calculate result[2][0]
MOVE.L  24(A0),D0
```

```
 MOVE.L  0(A1),D1
 MULS    D1,D0
 MOVE.L  28(A0),D2
 MOVE.L  12(A1),D3
 MULS    D3,D2
 ADD.L   D2,D0
 MOVE.L  32(A0),D4
 MOVE.L  24(A1),D5
 MULS    D5,D4
 ADD.L   D4,D0
 MOVE.L  D0,24(A2)
 ;Calculate result[2][1]
 MOVE.L  24(A0),D0
 MOVE.L  4(A1),D1
 MULS    D1,D0
 MOVE.L  28(A0),D2
 MOVE.L  16(A1),D3
 MULS    D3,D2
 ADD.L   D2,D0

 MOVE.L  32(A0),D4
 MOVE.L  28(A1),D5
 MULS    D5,D4
 ADD.L   D4,D0
 MOVE.L  D0,28(A2)
 ;Calculate result[2][2]
 MOVE.L  24(A0),D0
 MOVE.L  8(A1),D1
 MULS    D1,D0
 MOVE.L  28(A0),D2
 MOVE.L  20(A1),D3
 MULS    D3,D2
 ADD.L   D2,D0
 MOVE.L  32(A0),D4
 MOVE.L  32(A1),D5
 MULS    D5,D4
 ADD.L   D4,D0
 MOVE.L  D0,32(A2)
 TRAP    #15   ; Exit
matrix1:
 DC.L    2, 1, 3
 DC.L    3, 4, 1
 DC.L    5, 2, 3
matrix2:
 DC.L    1, 2, 0
 DC.L    4, 1, 2
 DC.L    3, 2, 1
result:
 DS.L    9     ; Allocate space for result matrix
 SIMHALT       ; halt simulator
 END $1000     ; last line of source
```

## RISC-V IMPLEMENTATION

```
   .data
matrixA: .word 2, 1, 3          # First row of matrix A
        .word 3, 4, 1           # Second row of matrix A
        .word 5, 2, 3           # Third row of matrix A
matrixB: .word 1, 2, 0          # First row of matrix B
        .word 4, 1, 2           # Second row of matrix B
        .word 3, 2, 1           # Third row of matrix B
matrixC: .space 36              # Space for 3x3 result matrix C (9 words)
   .text
   .globl _start
```

```
_start:
    # Load addresses of matrix A, B, and C
    la      t0, matrixA         # Load address of matrix A
    la      t1, matrixB         # Load address of matrix B
    la      t2, matrixC         # Load address of result matrix C
    # Calculate C[0][0]
    lw      t3, 0(t0)           # A[0][0]
    lw      t4, 0(t1)           # B[0][0]
    mul     t3, t3, t4          # t3 = A[0][0] * B[0][0]
    lw      t5, 4(t0)           # A[0][1]
    lw      t6, 12(t1)          # B[1][0]
    mul     t5, t5, t6          # t5 = A[0][1] * B[1][0]
    add     t3, t3, t5          # t3 = t3 + t5
    lw      t5, 8(t0)           # A[0][2]
    lw      t6, 24(t1)          # B[2][0]
    mul     t5, t5, t6          # t5 = A[0][2] * B[2][0]
    add     t3, t3, t5          # t3 = t3 + t5

    sw      t3, 0(t2)           # Store result in C[0][0]
    # Calculate C[0][1]
    lw      t3, 0(t0)           # A[0][0]
    lw      t4, 4(t1)           # B[0][1]
    mul     t3, t3, t4          # t3 = A[0][0] * B[0][1]
    lw      t5, 4(t0)           # A[0][1]
    lw      t6, 16(t1)          # B[1][1]
    mul     t5, t5, t6          # t5 = A[0][1] * B[1][1]
    add     t3, t3, t5          # t3 = t3 + t5
    lw      t5, 8(t0)           # A[0][2]
    lw      t6, 28(t1)          # B[2][1]
    mul     t5, t5, t6          # t5 = A[0][2] * B[2][1]
    add     t3, t3, t5          # t3 = t3 + t5
    sw      t3, 4(t2)           # Store result in C[0][1]
    # Calculate C[0][2]
    lw      t3, 0(t0)           # A[0][0]
    lw      t4, 8(t1)           # B[0][2]
    mul     t3, t3, t4          # t3 = A[0][0] * B[0][2]
    lw      t5, 4(t0)           # A[0][1]
    lw      t6, 20(t1)          # B[1][2]
    mul     t5, t5, t6          # t5 = A[0][1] * B[1][2]
    add     t3, t3, t5          # t3 = t3 + t5
    lw      t5, 8(t0)           # A[0][2]
    lw      t6, 32(t1)          # B[2][2]
    mul     t5, t5, t6          # t5 = A[0][2] * B[2][2]
    add     t3, t3, t5          # t3 = t3 + t5
    sw      t3, 8(t2)           # Store result in C[0][2]
    # Calculate C[1][0]
    lw      t3, 12(t0)          # A[1][0]
    lw      t4, 0(t1)           # B[0][0]
    mul     t3, t3, t4          # t3 = A[1][0] * B[0][0]
    lw      t5, 16(t0)          # A[1][1]
    lw      t6, 12(t1)          # B[1][0]
    mul     t5, t5, t6          # t5 = A[1][1] * B[1][0]
    add     t3, t3, t5          # t3 = t3 + t5
    lw      t5, 20(t0)          # A[1][2]
    lw      t6, 24(t1)          # B[2][0]
    mul     t5, t5, t6          # t5 = A[1][2] * B[2][0]
    add     t3, t3, t5          # t3 = t3 + t5
    sw      t3, 12(t2)          # Store result in C[1][0]
    # Calculate C[1][1]
    lw      t3, 12(t0)          # A[1][0]
    lw      t4, 4(t1)           # B[0][1]
    mul     t3, t3, t4          # t3 = A[1][0] * B[0][1]
    lw      t5, 16(t0)          # A[1][1]
    lw      t6, 16(t1)          # B[1][1]
    mul     t5, t5, t6          # t5 = A[1][1] * B[1][1]
```

```
add     t3, t3, t5          # t3 = t3 + t5
lw      t5, 20(t0)          # A[1][2]
lw      t6, 28(t1)          # B[2][1]
mul     t5, t5, t6          # t5 = A[1][2] * B[2][1]
add     t3, t3, t5          # t3 = t3 + t5
sw      t3, 16(t2)          # Store result in C[1][1]
# Calculate C[1][2]
lw      t3, 12(t0)          # A[1][0]
lw      t4, 8(t1)           # B[0][2]
mul     t3, t3, t4          # t3 = A[1][0] * B[0][2]
lw      t5, 16(t0)          # A[1][1]
lw      t6, 20(t1)          # B[1][2]
mul     t5, t5, t6          # t5 = A[1][1] * B[1][2]
add     t3, t3, t5          # t3 = t3 + t5
lw      t5, 20(t0)          # A[1][2]
lw      t6, 32(t1)          # B[2][2]
mul     t5, t5, t6          # t5 = A[1][2] * B[2][2]
add     t3, t3, t5          # t3 = t3 + t5
sw      t3, 20(t2)          # Store result in C[1][2]
# Calculate C[2][0]
lw      t3, 24(t0)          # A[2][0]
lw      t4, 0(t1)           # B[0][0]
mul     t3, t3, t4          # t3 = A[2][0] * B[0][0]
lw      t5, 28(t0)          # A[2][1]
lw      t6, 12(t1)          # B[1][0]
mul     t5, t5, t6          # t5 = A[2][1] * B[1][0]
add     t3, t3, t5          # t3 = t3 + t5
lw      t5, 32(t0)          # A[2][2]
lw      t6, 24(t1)          # B[2][0]
mul     t5, t5, t6          # t5 = A[2][2] * B[2][0]
add     t3, t3, t5          # t3 = t3 + t5
sw      t3, 24(t2)          # Store result in C[2][0]
# Calculate C[2][1]
lw      t3, 24(t0)          # A[2][0]
lw      t4, 4(t1)           # B[0][1]
mul     t3, t3, t4          # t3 = A[2][0] * B[0][1]
lw      t5, 28(t0)          # A[2][1]
lw      t6, 16(t1)          # B[1][1]
mul     t5, t5, t6          # t5 = A[2][1] * B[1][1]
add     t3, t3, t5          # t3 = t3 + t5
lw      t5, 32(t0)          # A[2][2]
lw      t6, 28(t1)          # B[2][1]
mul     t5, t5, t6          # t5 = A[2][2] * B[2][1]
add     t3, t3, t5          # t3 = t3 + t5
sw      t3, 28(t2)          # Store result in C[2][1]
# Calculate C[2][2]
lw      t3, 24(t0)          # A[2][0]
lw      t4, 8(t1)           # B[0][2]
mul     t3, t3, t4          # t3 = A[2][0] * B[0][2]
lw      t5, 28(t0)          # A[2][1]
lw      t6, 20(t1)          # B[1][2]
mul     t5, t5, t6          # t5 = A[2][1] * B[1][2]
add     t3, t3, t5          # t3 = t3 + t5
lw      t5, 32(t0)          # A[2][2]
lw      t6, 32(t1)          # B[2][2]
mul     t5, t5, t6          # t5 = A[2][2] * B[2][2]
add     t3, t3, t5          # t3 = t3 + t5
sw      t3, 32(t2)          # Store result in C[2][2]
# Exit the program
li      a0, 10              # Exit system call
ecall
```

**DESCRIPTION**

Matrix multiplication involves multiplying two matrices and summing the results of the element-wise multiplication of their corresponding rows and columns. In this project, matrix multiplication is performed manually for each element of the resulting matrix, avoiding the use of loops in the code. This project involves implementing matrix multiplication for two 3x3 matrices using assembly language, specifically in EASy68K and RISC-V, without utilizing loops. The process begins by defining two 3x3 matrices, A and B, with predefined values, and calculating their product to generate a third matrix, C. The multiplication is performed by calculating the dot product for each element of the result matrix C, where each element C[i][j] is derived from multiplying the corresponding row of matrix A with the corresponding column of matrix B, and summing the results. The assembly programs for both EASy68K and RISC-V are designed to directly access matrix elements from memory and perform the necessary multiplications and additions, storing the result in matrix C. Once all elements of matrix C are computed, the program gracefully exits. This project provides insight into low-level implementation of matrix operations, demonstrating how assembly language can handle memory operations, arithmetic calculations, and direct data manipulation without high-level abstractions.

**OBSERVATIONS**

**Matrix A:**

2 1 3
3 4 1
5 2 3

**Matrix B:**

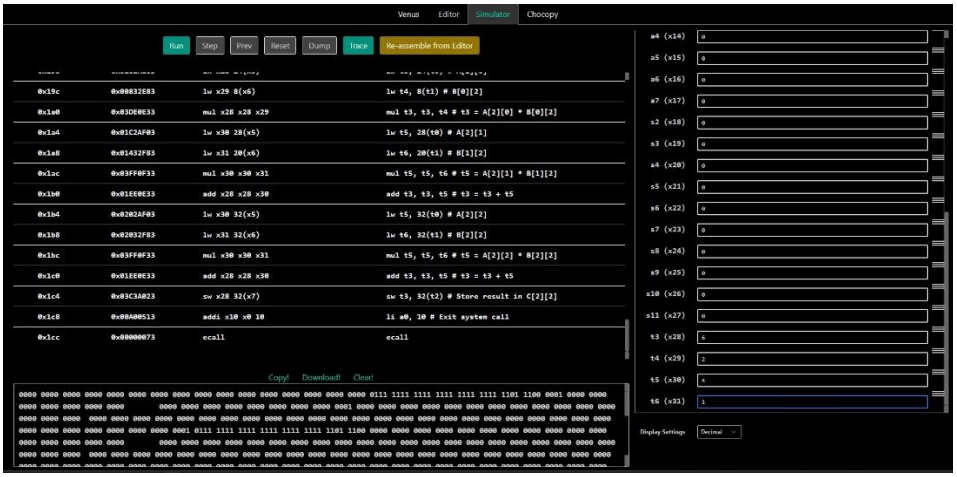
1 2 0
4 1 2
3 2 1

**Thus, the resulting matrix C is:**

15 11 5
22 12 9
22 18 7

During the implementation of matrix multiplication in both EASy68K and RISC-V, it was observed that without loops, each element of the result matrix had to be calculated manually, making the code longer and more repetitive but providing a deeper understanding of low-level operations. Direct memory addressing was crucial for accessing elements from matrices A and B and storing results in matrix C. The absence of loops made the code verbose less efficient compared to higher-level languages, while error handling and debugging required more manual checks. Overall, the project provided valuable insights into assembly programming, emphasizing memory manipulation and low-level computation.

## RESULTS


## CONCLUSIONS

In this project, we successfully implemented 3x3 matrix multiplication in both EASy68K and RISC-V assembly languages without using loops, manually computing each element of the resulting matrix through direct addressing and individual calculations. The process highlighted the challenges of working with low-level assembly, where code is more verbose and error-prone compared to higher-level languages. However, it also demonstrated the power and control assembly provides over hardware operations. Despite the complexity, both implementations achieved the desired result, showcasing the feasibility of performing such operations in assembly while emphasizing the trade-offs in efficiency and readability.