VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)
IBRAHIMBAGH, HYDERABAD – 500031
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
B.E. – V SEMESTER MINI PROJECT – II (2025-26)

# Course-based Project

# on

# Matrix Operations in ARM Assembly for Real-Time Image Processing using CPUlator [online ARM Compiler]

1. Saiteja (1602-23-735-175)
2. Sashank (1602-23-735-176)
3. Srinivas (1602-23-735-182)

# 1. Aim of the Project

To implement matrix operations in ARM assembly language and demonstrate how they serve as the computational foundation for tasks in real-time image processing such as filtering, enhancement, and feature extraction.

# 2. Objectives

1. To understand ARM instruction-level programming for matrix-based image processing.
2. To implement 3×3 matrix addition, subtraction, and multiplication used in basic image filtering.
3. To validate execution of matrix operations using an ARM simulator for real-time processing workloads.

# 3. Introduction

Matrix operations are fundamental to image processing, where every image is represented as a matrix of pixel intensities. Tasks such as edge detection, smoothing, sharpening, contrast enhancement, and feature extraction rely heavily on mathematical operations between pixel matrices and filter kernels.

Real-time embedded systems – including surveillance cameras, mobile photo processors, vehicle vision modules, and robotics – frequently use ARM-based processors for high-speed, low-power computation. Implementing matrix operations directly in ARM assembly allows us to understand how such systems perform operations like:

- Applying 3×3 convolution kernels
- Detecting edges using Sobel or Prewitt filters
- Smoothing images using blur or averaging matrices

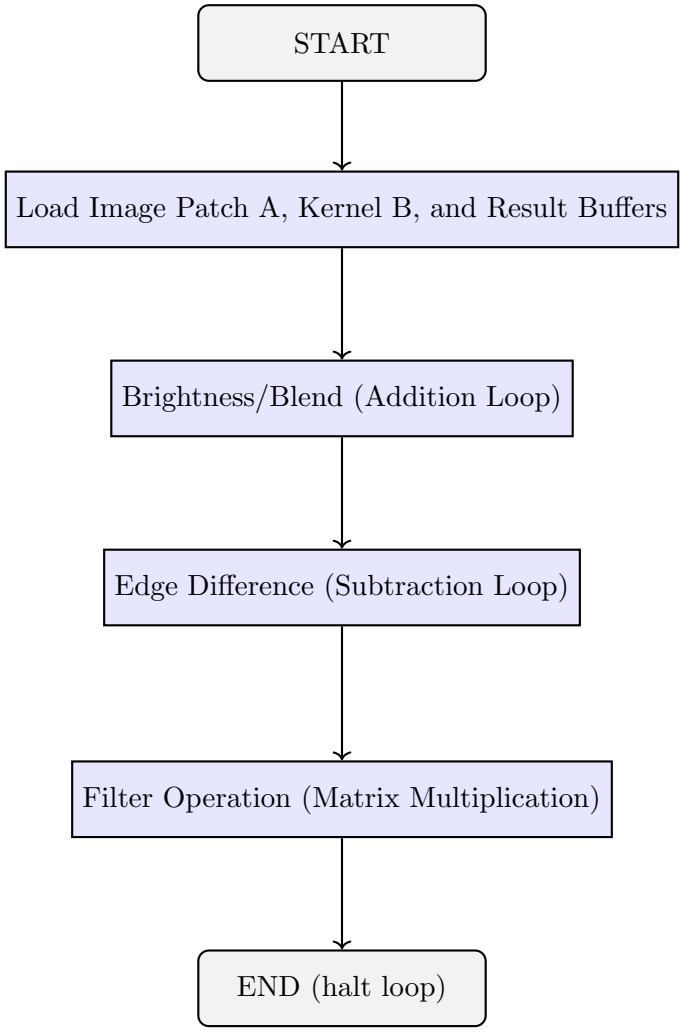
This project demonstrates core matrix operations (addition, subtraction, multiplication) in ARM assembly as the building blocks for image-filtering processes. Simulating the logic at assembly level provides deeper insight into the low-level execution mechanisms behind modern real-time imaging.

# 4. Methodology

The matrix operations in this project are treated as fundamental steps in applying 3×3 image filters. Each operation mirrors behaviors used in convolution and pixel enhancement.

1. **Data Initialization:** Two 3×3 matrices represent small image patches or filter kernels. Additional memory regions store the processed output.
2. **Matrix Addition:** Useful in brightness enhancement, image blending, and combining feature maps. Implemented as 9 sequential additions through register-indexed addressing.
3. **Matrix Subtraction:** Common in edge detection (difference-based filters) and motion detection. Performed by subtracting pixel-wise values of two matrices.
4. **Matrix Multiplication:** Represents convolution-like operations where row and column elements accumulate to form a filtered output pixel. A triple-nested loop computes dot products similar to applying a 3×3 kernel to an image block.
5. **Result Storage:** Outputs of each operation—useful for filtering, contrast enhancement, or feature extraction—are stored in dedicated memory blocks.
6. **Program Termination:** Execution halts, allowing the simulation environment to display processed matrix results.

### Flow Diagram




## 5. Hardware / Software Implementation

### Software Environment

All simulations were performed using the online ARMv7 environment:

`https://cpulator.01xz.net/?sys=arm`

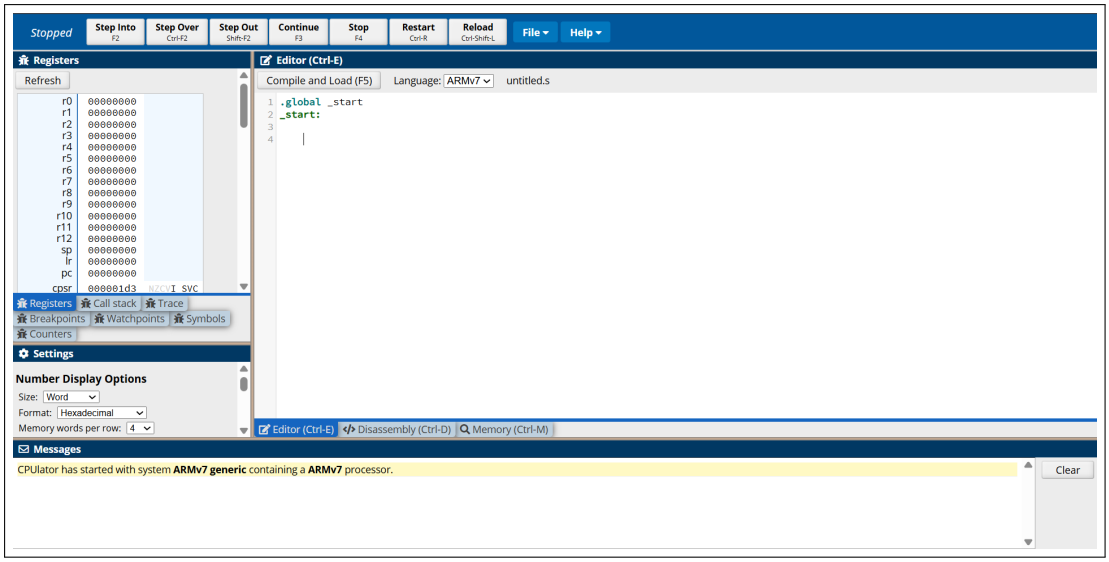



Figure 1: Online ARM Simulator Interface

## Procedure

1. Enter the ARM assembly code in the simulator.
2. Assemble and execute the program.
3. Observe the register updates for each matrix operation.
4. Analyze the memory region storing the image-processed outputs.
5. Compare results with theoretical image filter computations.

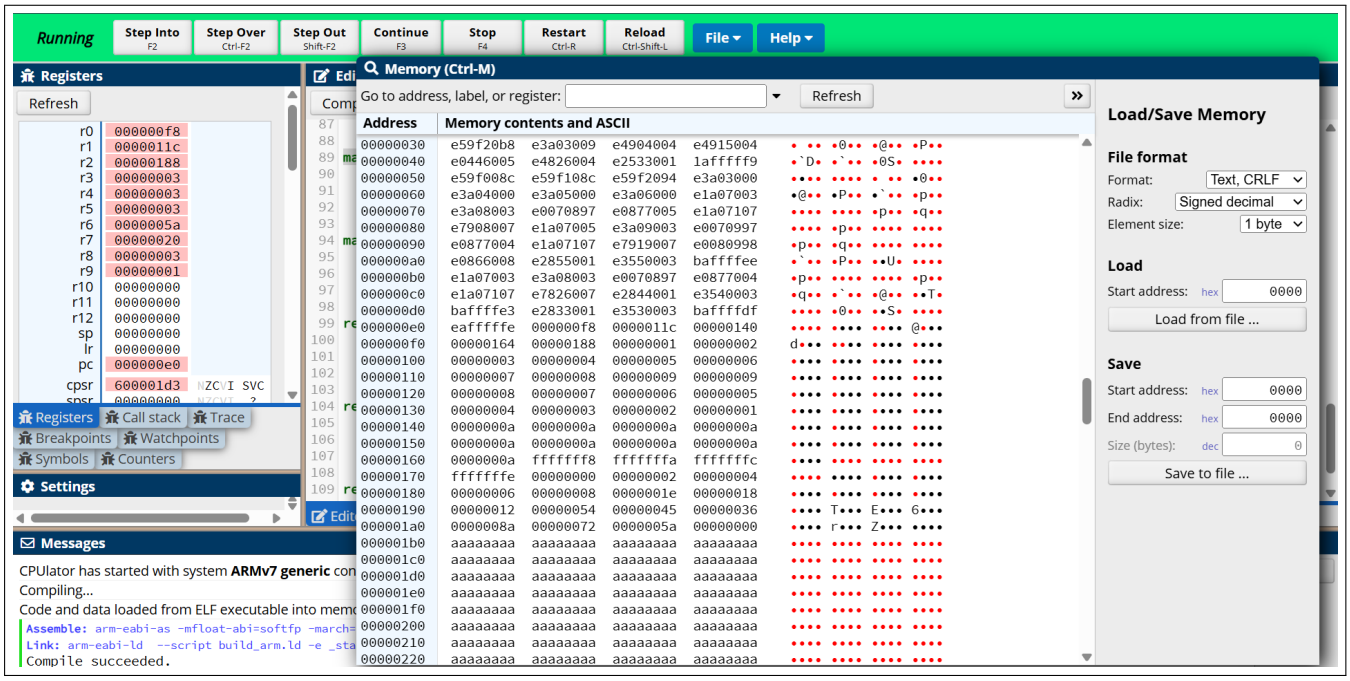



Figure 2: Memory Output After Matrix-Based Image Processing

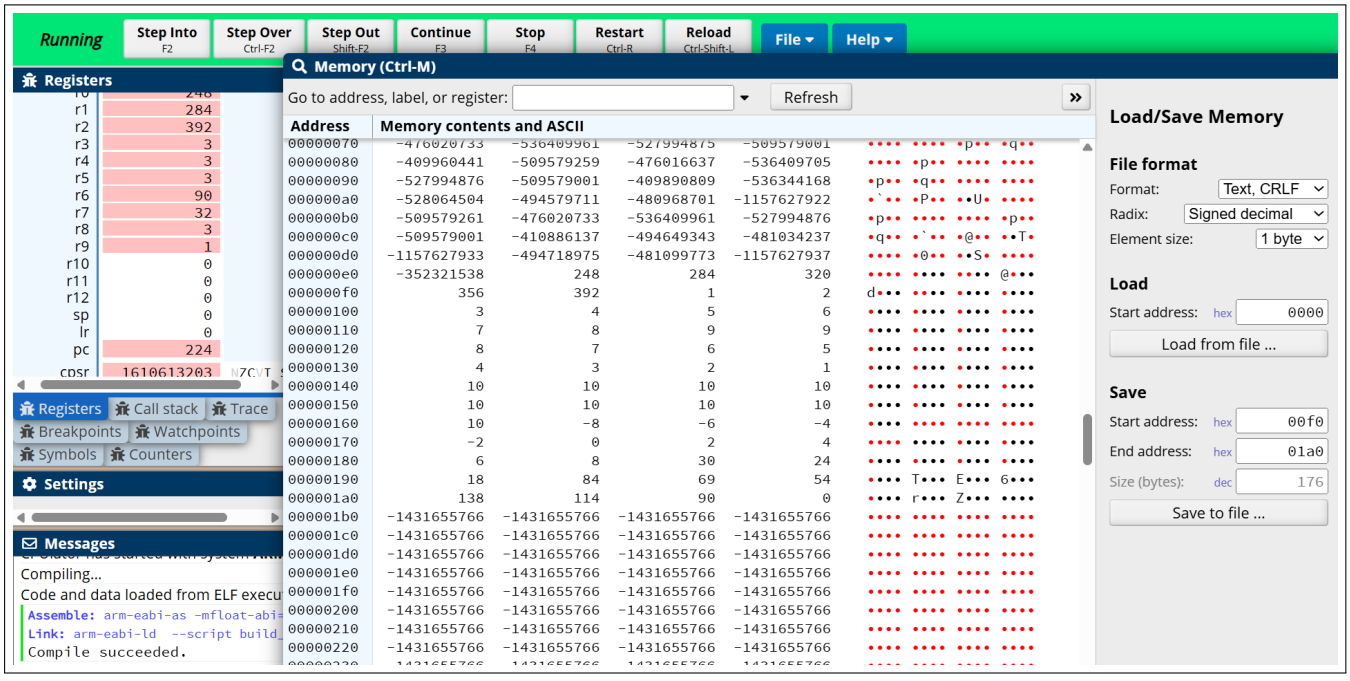



Figure 3: Signed Decimal Interpretation of Processed Matrix

# 6. Results and Discussion

The matrices used in this project represent a small 3×3 grayscale image patch and a 3×3 processing kernel:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

These can represent, for example:

- A small image region
- A sharpening filter, blur kernel, or gradient extractor

The ARM program produced:

## Addition (Brightness or Blending)

$$\begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

## Subtraction (Edge Difference)

$$\begin{bmatrix} -8 & -6 & -4 \\ -2 & 0 & 2 \\ 4 & 6 & 8 \end{bmatrix}$$

## Multiplication (Filter Application)

$$\begin{bmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{bmatrix}$$

These results correspond to common operations in image-processing applications such as contrast adjustment, gradient-based edge detection, and convolution-like filtering.

# 7. Appendix: ARM Assembly Code

Listing 1: ARM Assembly Code for Matrix Operations

```
    .text
    .global _start

_start:
    // ===== MATRIX ADDITION =====
    LDR r0, =matrixA
    LDR r1, =matrixB
    LDR r2, =resultAdd
    MOV r3, #9

add_loop:
    LDR r4, [r0], #4
    LDR r5, [r1], #4
    ADD r6, r4, r5
    STR r6, [r2], #4
    SUBS r3, r3, #1
    BNE add_loop

    // ===== MATRIX SUBTRACTION =====
    LDR r0, =matrixA
    LDR r1, =matrixB
    LDR r2, =resultSub
    MOV r3, #9

sub_loop:
    LDR r4, [r0], #4
    LDR r5, [r1], #4
    SUB r6, r4, r5
    STR r6, [r2], #4
    SUBS r3, r3, #1
    BNE sub_loop

    // ===== MATRIX MULTIPLICATION =====
    LDR r0, =matrixA
    LDR r1, =matrixB
    LDR r2, =resultMul
    MOV r3, #0
```

```
38
39 mul_outer:
40     MOV r4, #0
41
42 mul_middle:
43     MOV r5, #0
44     MOV r6, #0
45
46 mul_inner:
47     MOV r7, r3
48     MOV r8, #3
49     MUL r7, r7, r8
50     ADD r7, r7, r5
51     LSL r7, r7, #2
52     LDR r8, [r0, r7]
53
54     MOV r7, r5
55     MOV r9, #3
56     MUL r7, r7, r9
57     ADD r7, r7, r4
58     LSL r7, r7, #2
59     LDR r9, [r1, r7]
60
61     MUL r8, r8, r9
62     ADD r6, r6, r8
63
64     ADD r5, r5, #1
65     CMP r5, #3
66     BLT mul_inner
67
68     MOV r7, r3
69     MOV r8, #3
70     MUL r7, r7, r8
71     ADD r7, r7, r4
72     LSL r7, r7, #2
73     STR r6, [r2, r7]
74
75     ADD r4, r4, #1
76     CMP r4, #3
77     BLT mul_middle
78
```

```
79      ADD r3, r3, #1
80      CMP r3, #3
81      BLT mul_outer
82
83  end:
84      B end
85
86      .data
87      .align 2
88
89  matrixA:
90      .word 1,2,3
91      .word 4,5,6
92      .word 7,8,9
93
94  matrixB:
95      .word 9,8,7
96      .word 6,5,4
97      .word 3,2,1
98
99  resultAdd:
100     .word 0,0,0
101     .word 0,0,0
102     .word 0,0,0
103
104 resultSub:
105     .word 0,0,0
106     .word 0,0,0
107     .word 0,0,0
108
109 resultMul:
110     .word 0,0,0
111     .word 0,0,0
112     .word 0,0,0
```