CSCI 320 – Matrix Multiplication

Kevin Gutierrez

Objective

The objective of this lab is to demonstrate 3x3 matrix multiplication in CISC and RISC

Equipment Used

EASy 68K simulator

RISC-V Assembler https://venus.kvakil.me/

Procedure

First I came up with the equations for each element of the product of two 3x3 matrices, A and B. The equations were as follows:

Row 0:

AB[0,0] = (A0 * B0) + (A1 * B3) + (A2 * B6)

AB[0,1] = (A0 * B1) + (A1 * B4) + (A2 * B7)

AB[0,2] = (A0 * B2) + (A1 * B5) + (A2 * B8)

Row 1:

AB[1,0] = (A3 * B0) + (A4 * B3) + (A5 * B6)

AB[1,1] = (A3 * B1) + (A4 * B4) + (A5 * B7)

AB[1,2] = (A3 * B2) + (A4 * B5) + (A5 * B8)

Row 2:

AB[2,0] = (A6 * B0) + (A7 * B3) + (A8 * B6)

AB[2,1] = (A6 * B1) + (A7 * B4) + (A8 * B7)

AB[2,2] = (A6 * B2) + (A7 * B5) + (A8 * B8)

Such that for a matrix in the $K \in \mathbb{R}^n$ vector space, element [1,1] would be represented as K0, and [1,3] would be represented as K2.

Using these equations, I first loaded the memory addresses into the appropriate address registers, loaded the matrixes using addressing with offsets for each element. Then I evaluated each matrix index and loaded the index into the resultant matrix AB address, starting at 0x1040.

New Operations Learned

Representation of a matrix in $\mathbb{R}^n$ as a flattened matrix, such that no columns exist, and at the end of a row, the next row begins as the previous row number plus 1.

The following RISC-V instructions/items

1. JAL
2. SB
3. LI
4. MUL
5. ADD
6. SW
7. The syntax of register access and memory offsets during register access

Program Description

The program first loads memory locations, offset by 0x20 bytes, for each flattened matrix.

Then the program branches to the populate the A and B 3x3 matrices. This is done precisely using the register indirect with offset addressing mode, and offsetting each element by increments of 0x1 each time.

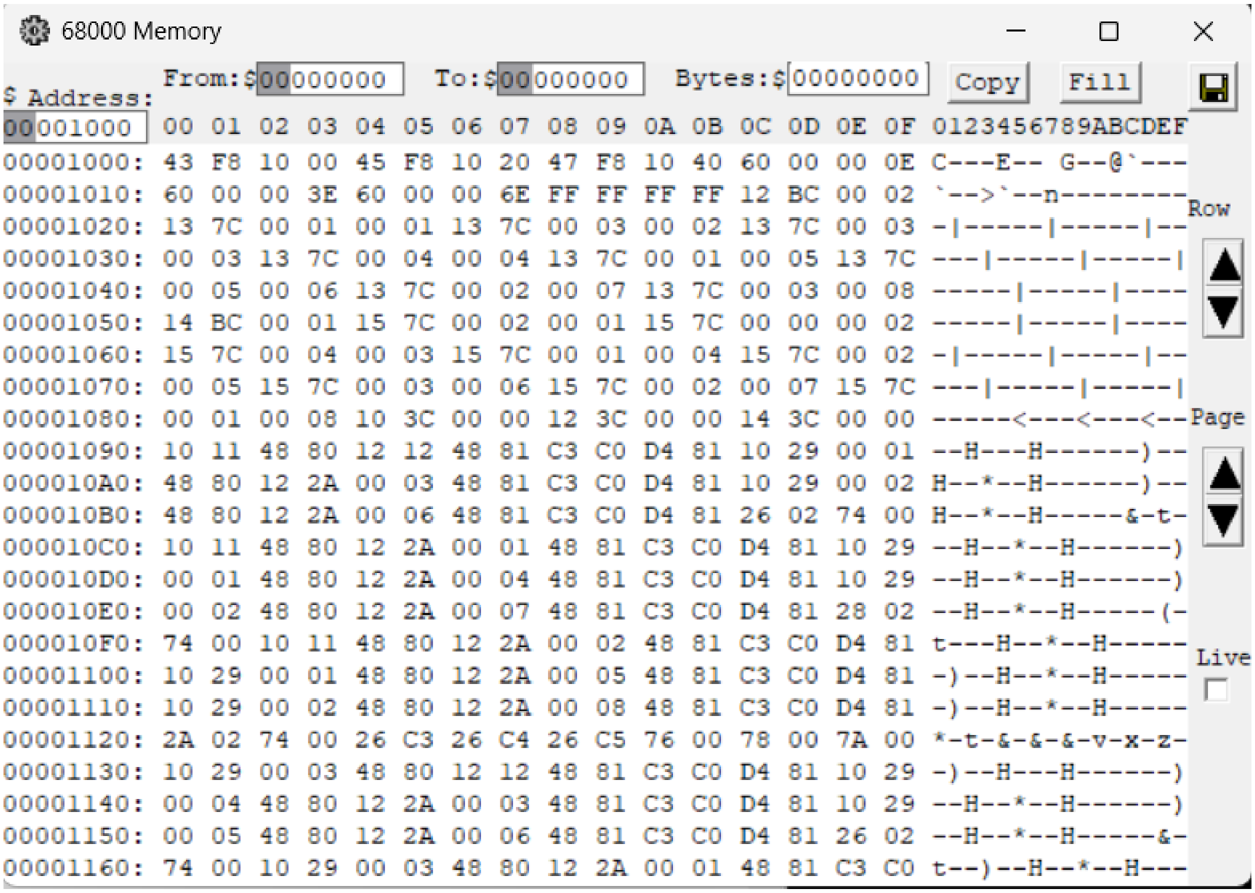
Finally, the AB resultant matrix is evaluated using the above formulas.

The same operations were done in the RISC-V implementation

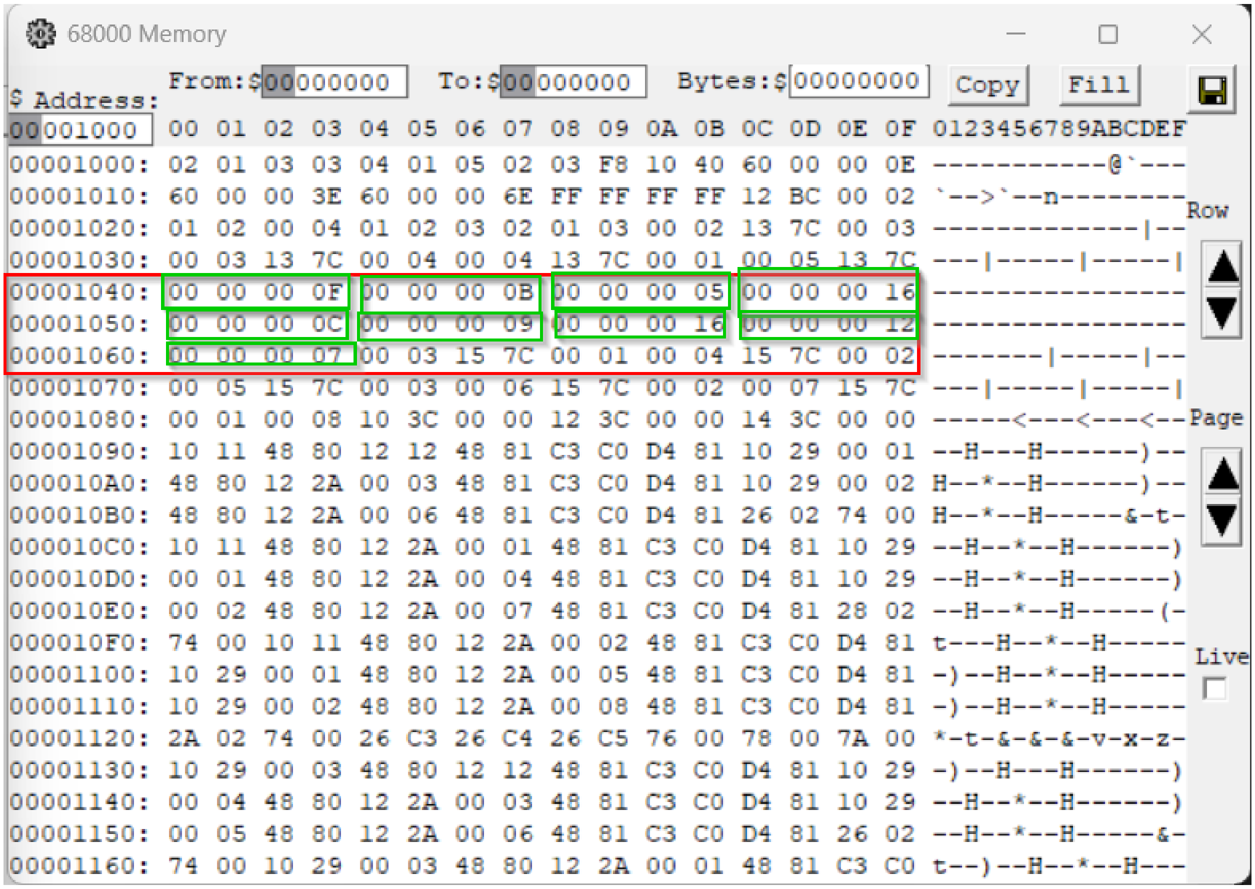
Conclusion

This was a phenomenal introduction to nxn matrix multiplication. The only item that I would have liked to have changed would have been to generalize the formula to be able to evaluate the product of two matrices A, B, such that $\{A,B \in \mathbb{R}^n \mid n \in Z+\}$

Memory before running (CISC)



Memory after running (CISC)



Code Listing (CISC)

```
*-----------------------------------------------------------
* Title      : 3x3 Matrix Multiplication
* Written by : Kevin Gutierrez
* Date       : 3/31/25
* Description: 3x3 matrix multiplication program
*-----------------------------------------------------------

    ORG    $1000
START:

    ;A1 -> Matrix A
    LEA.L $00001000, A1

    ;A2 -> Matrix B
    LEA.L $00001020, A2

    ;A3 -> A * B result
    LEA.L $00001040, A3


    BRA POPULATE_A

    BRA POPULATE_B

    BRA CALCULATE_A_B_PRODUCT
```

```
    SIMHALT

POPULATE_A:
    ; [row 1]
    MOVE.B #$2,  (A1)
    MOVE.B #$1,  1(A1)
    MOVE.B #$3,  2(A1)

    ; [row 2]
    MOVE.B #$3,  3(A1)
    MOVE.B #$4,  4(A1)
    MOVE.B #$1,  5(A1)

    ; [row 3]
    MOVE.B #$5,  6(A1)
    MOVE.B #$2,  7(A1)
    MOVE.B #$3,  8(A1)

POPULATE_B:
    ; [row 1]
    MOVE.B #$1,   (A2)
    MOVE.B #$2,  1(A2)
    MOVE.B #$0,  2(A2)

    ; [row 2]
    MOVE.B #$4,  3(A2)
    MOVE.B #$1,  4(A2)
    MOVE.B #$2,  5(A2)

    ; [row 3]
    MOVE.B #$3,  6(A2)
    MOVE.B #$2,  7(A2)
    MOVE.B #$1,  8(A2)


CALCULATE_A_B_PRODUCT:
    ;D0 working data register
    ;D1 working data register
    ;D2 working sum register
    MOVE.B #0,  D0
    MOVE.B #0,  D1
    MOVE.B #0,  D2

    ;D3 AB(k,0) value
    ;D4 AB(k,1) value
    ;D5 AB(k,2) value

    ;Reset D3,D4,D5 after each row has been evaluated

    ;==================================== AB[0,0]
============================================================================
================================================

    ; Compute AB[0,0] = [(A0B0)+(A1B3)+(A2B6)]
```

```
    ;Evaluate A0B0, Add to sum
;BEGIN BLOCK                                ;O[0,0]
    MOVE.B (A1), D0 ; -> move A0 to D0
    EXT.W D0        ; -> extend D0 from byte to word size
    MOVE.B (A2), D1 ; -> move B0 to D1
    EXT.W D1        ; -> extend D1 from byte to word size

    MULS.W D0, D1  ; -> multiply D0 by D1, store in D1. D1 should now be -1
    ADD.L D1, D2 ; D2 now contains I0*K0
;END BLOCK

;NOTE - All subsequent Evalute Ii*Kj blocks are repititions of the above
block

    ;Evaluate A1B3, Add to sum
    MOVE.B 1(A1), D0
    EXT.W D0
    MOVE.B 3(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A2B6, Add to sum
    MOVE.B 2(A1), D0
    EXT.W D0
    MOVE.B 6(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[0,0] into D3
    MOVE.L D2, D3

    ; Reset sum register
    MOVE.L #0, D2

    ;==================================== AB[0,1]
=============================================================================
=================================================

    ; Compute AB[0,1] = [(A0B1)+(A1B4)+(A2B7)]
    ;Evaluate A0B1, Add to sum
    MOVE.B (A1), D0
    EXT.W D0
    MOVE.B 1(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A1B4, Add to sum
    MOVE.B 1(A1), D0
    EXT.W D0
    MOVE.B 4(A2), D1
```

```
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A2B7, Add to sum
    MOVE.B 2(A1), D0
    EXT.W D0
    MOVE.B 7(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[0,1] into D4
    MOVE.L D2, D4

    ; Reset sum register
    MOVE.L #0, D2


    ;==================================== AB[0,2]
=============================================================================
=================================================

    ; Compute AB[0,1] = [(A0B2)+(A1B5)+(A2B8)]
    ;Evaluate A0B2, Add to sum
    MOVE.B (A1), D0
    EXT.W D0
    MOVE.B 2(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A1B5, Add to sum
    MOVE.B 1(A1), D0
    EXT.W D0
    MOVE.B 5(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A2B8, Add to sum
    MOVE.B 2(A1), D0
    EXT.W D0
    MOVE.B 8(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[0,2] into D5
    MOVE.L D2, D5
```

```
    ; Reset sum register
    MOVE.L #0, D2

    ;==================================== LOAD FIRST ROW INTO RESULTANT
MATRIX REGISTER
=============================================================================
===========

    MOVE.L D3, (A3)+
    MOVE.L D4, (A3)+
    MOVE.L D5, (A3)+

    ;Reset element registers
    MOVE.L #0, D3
    MOVE.L #0, D4
    MOVE.L #0, D5

    ;===================================== AB[1,0]
=============================================================================
=================================================

    ; Compute AB[1,0] = [(A3B0)+(A4B3)+(A5B6)]
    ;Evaluate A3B0, Add to sum
    MOVE.B 3(A1), D0
    EXT.W D0
    MOVE.B (A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A4B3, Add to sum
    MOVE.B 4(A1), D0
    EXT.W D0
    MOVE.B 3(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A5B6, Add to sum
    MOVE.B 5(A1), D0
    EXT.W D0
    MOVE.B 6(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[0,0] into D3
    MOVE.L D2, D3

    ; Reset sum register
    MOVE.L #0, D2
```

```
    ;==================================== AB[1,1]
=============================================================================
=================================================
    ; Compute AB[1,1] = [(A3B1)+(A4B4)+(A5B7)]

    ;Evaluate A3B1, Add to sum
    MOVE.B 3(A1), D0
    EXT.W D0
    MOVE.B 1(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A4B4, Add to sum
    MOVE.B 4(A1), D0
    EXT.W D0
    MOVE.B 4(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A5B7, Add to sum
    MOVE.B 5(A1), D0
    EXT.W D0
    MOVE.B 7(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[1,1] into D4
    MOVE.L D2, D4

    ; Reset sum register
    MOVE.L #0, D2

;==================================== AB[1,2]
=============================================================================
=================================================
    ; Compute AB[1,2] = [(A3B2)+(A4B5)+(A5B8)]

    ;Evaluate A3B2, Add to sum
    MOVE.B 3(A1), D0
    EXT.W D0
    MOVE.B 2(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A4B5, Add to sum
    MOVE.B 4(A1), D0
    EXT.W D0
```

```
    MOVE.B 5(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A5B8, Add to sum
    MOVE.B 5(A1), D0
    EXT.W D0
    MOVE.B 8(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[1,2] into D5
    MOVE.L D2, D5

    ; Reset sum register
    MOVE.L #0, D2

;==================================== LOAD SECOND ROW INTO RESULTANT MATRIX
REGISTER
============================================================================
===========

    MOVE.L D3, (A3)+
    MOVE.L D4, (A3)+
    MOVE.L D5, (A3)+

    ;Reset element registers
    MOVE.L #0, D3
    MOVE.L #0, D4
    MOVE.L #0, D5

;==================================== AB[2,0]
============================================================================
================================================
    ; Compute AB[2,0] = [(A6B0)+(A7B3)+(A8B6)]

    ;Evaluate A6B0, Add to sum
    MOVE.B 6(A1), D0
    EXT.W D0
    MOVE.B 0(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A7B3, Add to sum
    MOVE.B 7(A1), D0
    EXT.W D0
    MOVE.B 3(A2), D1
    EXT.W D1

    MULS.W D0, D1
```

```
    ADD.L D1, D2

    ;Evaluate A8B6, Add to sum
    MOVE.B 8(A1), D0
    EXT.W D0
    MOVE.B 6(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[2,0] into D3
    MOVE.L D2, D3

    ; Reset sum register
    MOVE.L #0, D2

;==================================== AB[2,1]
============================================================================
================================================
    ; Compute AB[2,1] = [(A6B1)+(A7B4)+(A8B7)]

    ;Evaluate A6B1, Add to sum
    MOVE.B 6(A1), D0
    EXT.W D0
    MOVE.B 1(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A7B4, Add to sum
    MOVE.B 7(A1), D0
    EXT.W D0
    MOVE.B 4(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A8B7, Add to sum
    MOVE.B 8(A1), D0
    EXT.W D0
    MOVE.B 7(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place OB[2,1] into D4
    MOVE.L D2, D4

    ; Reset sum register
    MOVE.L #0, D2
```

```
;==================================== AB[2,2]
================================================================================
=================================================
    ; Compute AB[2,2] = [(A6B2)+(A7B5)+(A8B8)]

    ;Evaluate A6B2, Add to sum
    MOVE.B 6(A1), D0
    EXT.W D0
    MOVE.B 2(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2


    ;Evaluate A7B5, Add to sum
    MOVE.B 7(A1), D0
    EXT.W D0
    MOVE.B 5(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Evaluate A8B8, Add to sum
    MOVE.B 8(A1), D0
    EXT.W D0
    MOVE.B 8(A2), D1
    EXT.W D1

    MULS.W D0, D1
    ADD.L D1, D2

    ;Finally, place AB[2,2] into D5
    MOVE.L D2, D5

    ; Reset sum register
    MOVE.L #0, D2

;==================================== LOAD THIRD ROW INTO RESULTANT MATRIX
REGISTER
================================================================================
===========

    MOVE.L D3, (A3)+
    MOVE.L D4, (A3)+
    MOVE.L D5, (A3)+


    END    START
```

Code Listing - RISC Implementation

```
.text

_boot:
    # Load addresses for matrices
    li x1, 0x1000          # x1 -> Matrix A (Base Address)
    li x2, 0x1020          # x2 -> Matrix B (Base Address)
    li x3, 0x1040          # x3 -> Matrix Result (Base Address)

    # Branch to POPULATE Matrix A
    jal x0, POPULATE_A

    # Branch to POPULATE Matrix B
    jal x0, POPULATE_B

    # Branch to Calculite A * B Product
    jal x0, CALCULiTE_A_B_PRODUCT

    # End execution


POPULATE_A:
    # Matrix A [row 1]
    li t0, 2
    sb t0, 0(x1)          # A[0][0]
    li t0, 1
    sb t0, 1(x1)          # A[0][1]
    li t0, 3
    sb t0, 2(x1)          # A[0][2]

    # Matrix A [row 2]
    li t0, 3
    sb t0, 3(x1)          # A[1][0]
    li t0, 4
    sb t0, 4(x1)          # A[1][1]
    li t0, 1
    sb t0, 5(x1)          # A[1][2]

    # Matrix A [row 3]
    li t0, 5
    sb t0, 6(x1)          # A[2][0]
    li t0, 2
    sb t0, 7(x1)          # A[2][1]
    li t0, 3
    sb t0, 8(x1)          # A[2][2]


POPULATE_B:
    # Matrix B [row 1]
    li t0, 1
    sb t0, 0(x2)          # B[0][0]
    li t0, 2
    sb t0, 1(x2)          # B[0][1]
```

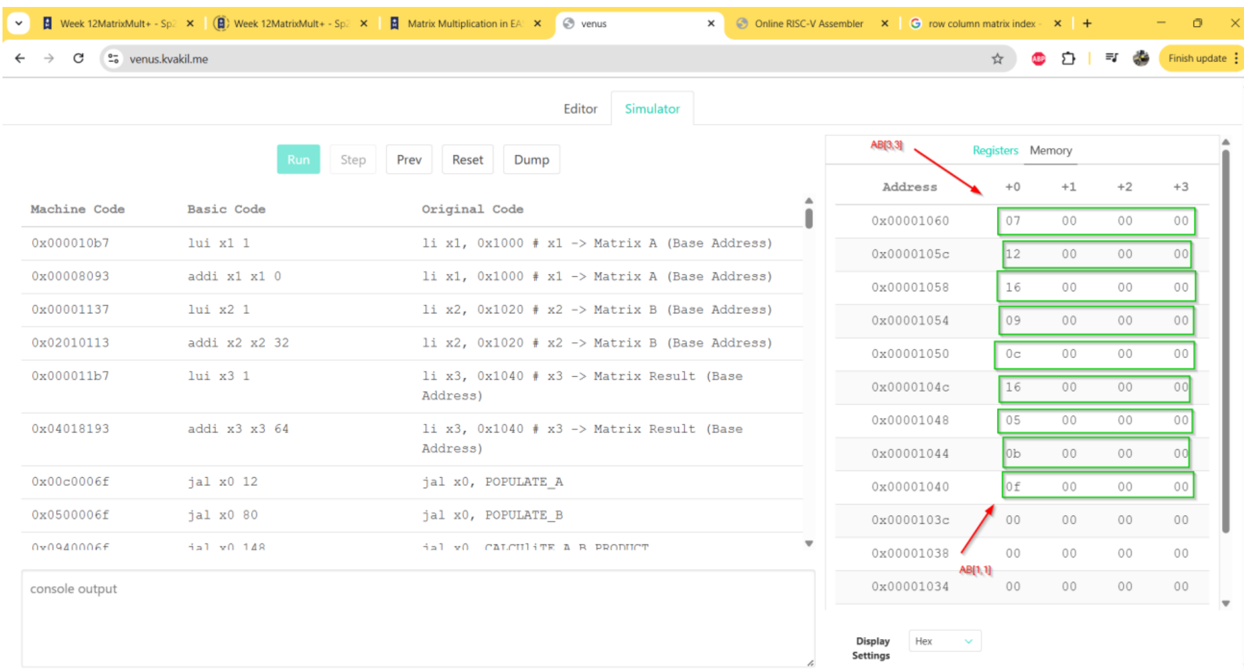
```
    li t0, 0
    sb t0, 2(x2)           # B[0][2]

    # Matrix B [row 2]
    li t0, 4
    sb t0, 3(x2)           # B[1][0]
    li t0, 1
    sb t0, 4(x2)           # B[1][1]
    li t0, 2
    sb t0, 5(x2)           # B[1][2]

    # Matrix B [row 3]
    li t0, 3
    sb t0, 6(x2)           # B[2][0]
    li t0, 2
    sb t0, 7(x2)           # B[2][1]
    li t0, 1
    sb t0, 8(x2)           # B[2][2]


CALCULiTE_A_B_PRODUCT:
    # Registers: t0, t1, t2 for working; t3, t4, t5 for results
    # Compute AB[0][0], AB[0][1], AB[0][2] (Row 0)

    li t2, 0               # Reset sum register
    lb t0, 0(x1)           # Load A00
    lb t1, 0(x2)           # Load B00
    mul t0, t0, t1         # A00 * B00
    add t2, t2, t0         # Sum += A00 * B00
    lb t0, 1(x1)
    lb t1, 3(x2)
    mul t0, t0, t1
    add t2, t2, t0
    lb t0, 2(x1)
    lb t1, 6(x2)
    mul t0, t0, t1
    add t2, t2, t0
    sw t2, 0(x3)           # Store AB[0][0]

    li t2, 0               # Reset sum register
    lb t0, 0(x1)           # Load A00
    lb t1, 1(x2)           # Load B01
    mul t0, t0, t1         # A00 * B01
    add t2, t2, t0
    lb t0, 1(x1)
    lb t1, 4(x2)
    mul t0, t0, t1
    add t2, t2, t0
    lb t0, 2(x1)
    lb t1, 7(x2)
    mul t0, t0, t1
    add t2, t2, t0
```

```
sw t2, 4(x3)          # Store AB[0][1]

li t2, 0              # Reset sum register
lb t0, 0(x1)          # Load A00
lb t1, 2(x2)          # Load B02
mul t0, t0, t1        # A00 * B02
add t2, t2, t0
lb t0, 1(x1)
lb t1, 5(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 2(x1)
lb t1, 8(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 8(x3)          # Store AB[0][2]

# Compute AB[1][0], AB[1][1], AB[1][2] (Row 1)

li t2, 0              # Reset sum register
lb t0, 3(x1)          # Load A10
lb t1, 0(x2)          # Load B00
mul t0, t0, t1        # A10 * B00
add t2, t2, t0
lb t0, 4(x1)
lb t1, 3(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 5(x1)
lb t1, 6(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 12(x3)         # Store AB[1][0]

li t2, 0              # Reset sum register
lb t0, 3(x1)          # Load A10
lb t1, 1(x2)          # Load B01
mul t0, t0, t1        # A10 * B01
add t2, t2, t0
lb t0, 4(x1)
lb t1, 4(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 5(x1)
lb t1, 7(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 16(x3)         # Store AB[1][1]

li t2, 0              # Reset sum register
lb t0, 3(x1)          # Load A10
lb t1, 2(x2)          # Load B02
```

```
mul t0, t0, t1        # A10 * B02
add t2, t2, t0
lb t0, 4(x1)
lb t1, 5(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 5(x1)
lb t1, 8(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 20(x3)         # Store AB[1][2]

# Compute AB[2][0], AB[2][1], AB[2][2] (Row 2)

li t2, 0              # Reset sum register
lb t0, 6(x1)          # Load A20
lb t1, 0(x2)          # Load B00
mul t0, t0, t1        # A20 * B00
add t2, t2, t0
lb t0, 7(x1)
lb t1, 3(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 8(x1)
lb t1, 6(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 24(x3)         # Store AB[2][0]

li t2, 0              # Reset sum register
lb t0, 6(x1)          # Load A20
lb t1, 1(x2)          # Load B01
mul t0, t0, t1        # A20 * B01
add t2, t2, t0
lb t0, 7(x1)
lb t1, 4(x2)
mul t0, t0, t1
add t2, t2, t0
lb t0, 8(x1)
lb t1, 7(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 28(x3)         # Store AB[2][1]

li t2, 0              # Reset sum register
lb t0, 6(x1)          # Load A20
lb t1, 2(x2)          # Load B02
mul t0, t0, t1        # A20 * B02
add t2, t2, t0
lb t0, 7(x1)
lb t1, 5(x2)
mul t0, t0, t1
```

```
add t2, t2, t0
lb t0, 8(x1)
lb t1, 8(x2)
mul t0, t0, t1
add t2, t2, t0
sw t2, 32(x3)          # Store AB[2][2]

ret
```

RISC-V Simulator Output



Registers Memory

zero	0x00000000
ra (x1)	0x00001000
sp (x2)	0x00001020
gp (x3)	0x00001040
tp (x4)	0x00000000
t0 (x5)	0x00000003
t1 (x6)	0x00000001
t2 (x7)	0x00000007
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x00000000
a1 (x11)	0x00000000

Display Settings Hex