Report on Assignment 1

ELL782: Computer Architecture

Master of Technology

IN

COMPUTER TECHNOLOGY

BY

MOHAMMAD YUSUF

2023EET2757

Submitted to-

Dr. Kaushik Saha

Department of Electrical Engineering

Indian Institute of Technology Delhi,

Delhi, 110016

2023-2024

Software Requirement Specifications

Specifications

RISC-V Venus Simulator embedded in VS Code

This add-on for Visual Studio Code incorporates the well-known Venus RISC-V simulator. Given that no further tools are required, it offers a stand-alone learning environment. It utilizes the regular VS Code debugging features while running RISC-V assembly code.

Device specifications

Installed RAM 16GB, Windows 10, 64 bit operating system, x64 based processor

Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz   2.59 GHz

Functionality

The software performs Gaussian elimination to reduce the given matrix to its row-echelon form.

It then determines the solutions to the system of linear equations, which may be unique, infinite, or nonexistent.

Execution Platform

The code is designed to run on a RISC-V processor.

Design Considerations

Assumptions and Dependencies

The code assumes that the input matrix is represented in row-major format.

It assumes that the input matrix is provided in memory at the address specified by the variable A.

The code does not have any external dependencies on other libraries or tools.

General Constraints

The software is designed to work with matrices of 5x6 (augmented matrix, nxm), with the number of equations specified by the variable n and the number of columns in the augmented matrix specified by the variable m.

The constraints of the RISC-V architecture, such as available registers and instruction set, may affect the design and implementation.

Goals and Guidelines

The primary goal is to perform Gaussian elimination efficiently to determine the solutions to the system of linear equations.

The code handle cases of unique solutions, infinite solutions, and no solutions.

Numerical stability and precision in floating-point calculations should be considered.

Memory usage only for A (5x6 matrix or 120 bytes), n (4 bytes), m (4 bytes) and array for storing the solutions (length 20 bytes).

Development Methods

The code is implemented in assembly language for the RISC-V architecture.

It utilizes floating-point operations for matrix manipulation. Instructions like flw, fsub.s, fmul.s, fdiv.s are used.

The code follows a row-wise Gaussian elimination approach to reduce the matrix to roe echelon form and then find the solutions.

Architectural Strategies

Strategies and Algorithms

The software employs a row-wise Gaussian elimination algorithm to transform the input matrix into its row-echelon form.

It uses floating-point arithmetic for division and elimination operations.

Software Architecture

High-Level Overview

The software operates on a given matrix A to perform Gaussian elimination.

It iteratively processes rows and columns to reduce the matrix to its row-echelon form.

Once the row-echelon form is achieved, it determines the solutions to the system of linear equations.

Detailed System Design

Gaussian Elimination Module

This module iteratively processes the rows and columns of the input matrix A to reduce it to row-echelon form.

It calculates ratios, performs division, and eliminates elements to achieve the desired form.

Inputs: Matrix A, size parameters n and m.

Outputs: Row-echelon form of matrix A.

pseudocode2.txt

```
Function GaussianElimination(matrix A, int n, int m):
    For i from 0 to n-1:
        pivot_row = i
        pivot = A[i][i]

        For j from i+1 to n-1:
            factor = A[j][i] / pivot
            For k from i to m-1:
                A[j][k] = A[j][k] - factor * A[i][k]

    Return A

```

Solution Determination Module

This module determines the solutions to the system of linear equations based on the row-echelon form of the matrix.

It checks for unique solutions, infinite solutions, or no solutions.

Inputs: Row-echelon form of matrix A, size parameters n and m.

psedocode.txt

```
Function DetermineSolutions(matrix A, int n, int m):
    unique_solutions = true
    infinite_solutions = false
    no_solutions = false

    For i from 0 to n-1:
        all_zeros = true
        for j from 0 to m-2:
            if A[i][j] != 0:
                all_zeros = false
                break

        if all_zeros and A[i][m-1] != 0:
            no_solutions = true
            Return "No solution exists"

        if all_zeros and A[i][m-1] == 0:
            infinite_solutions = true

    if infinite_solutions:
        Return "Infinitely many solutions exist"

    if unique_solutions:
        solutions = []
        For i from n-1 to 0:
            sum = 0
            For j from i+1 to m-2:
                sum = sum + A[i][j] * solutions[j]
            solutions[i] = (A[i][m-1] - sum) / A[i][i]
        Return "Unique Solutions:", solutions
```

Outputs: Information about the nature of solutions.

Testing

Testing Method

For this part we use 2 online resources,

1. Hex to Decimal converter (URL = https://gregstoll.dyndns.org/~gregstoll/floattohex/)

2. 5x5 system of linear equations solver (URL = https://math.cowpi.com/systemsolver/5x5.html and https://matrix.reshish.com/gaussSolution.php)

To verify the result generated by my written RISC-V based code.

Outputs

Example 1: (Unique Solutions)

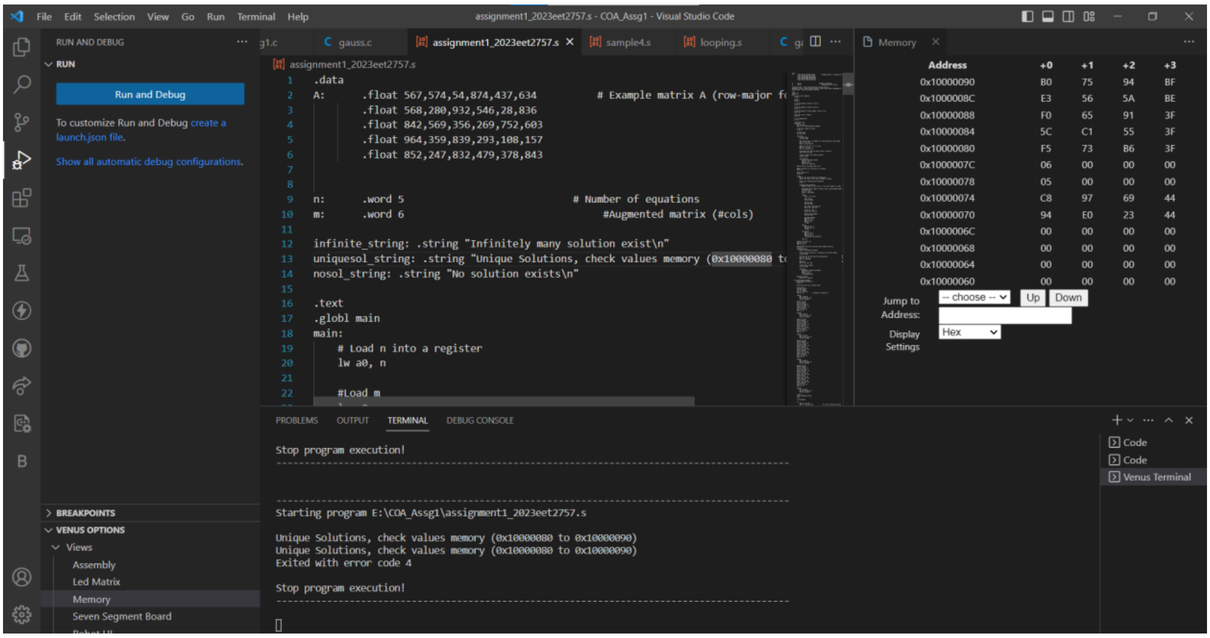
[A:B] = [567, 574, 54, 874, 437, 634]

[568, 280, 932, 546, 28, 836]

[842, 569, 356, 269, 752, 603]

[964, 359, 839, 293, 108, 157]

[852, 247, 832, 479, 378, 843]



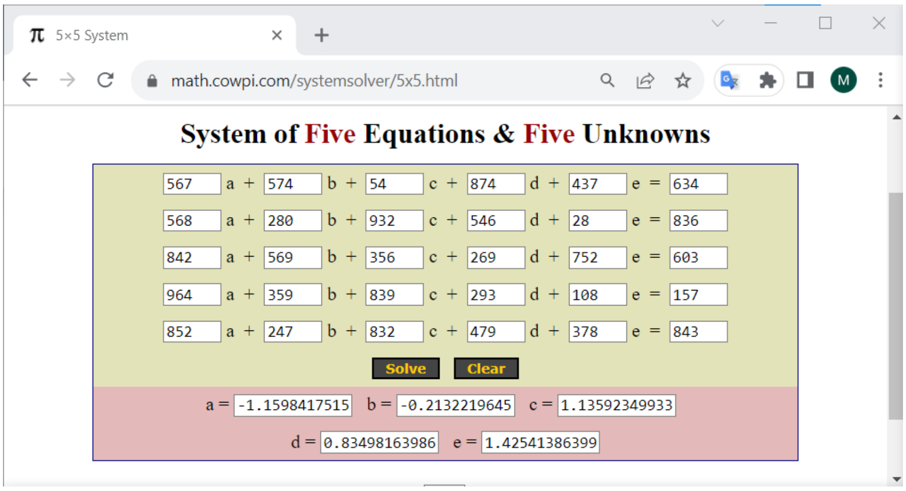
Note – Venus Simulator stores data in memory in **little endian form**.

Address	+0	+1	+2	+3
0x10000030	00	80	52	44
0x1000002C	00	00	51	44
0x10000028	00	00	E0	41
0x10000024	00	80	08	44
0x10000020	00	00	69	44
0x1000001C	00	00	8C	43
0x10000018	00	00	0E	44
0x10000014	00	80	1E	44
0x10000010	00	80	DA	43
0x1000000C	00	80	5A	44
0x10000008	00	00	58	42
0x10000004	00	80	0F	44
0x10000000	00	C0	0D	44

matrix A in row major form.

Lets check the sanity of the output.

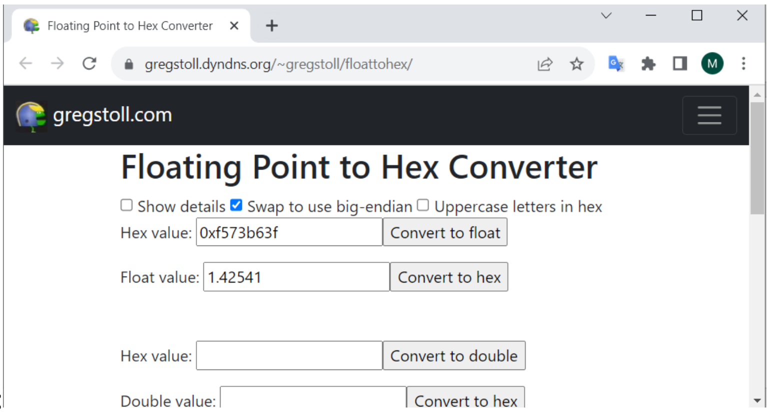
Actual solutions:



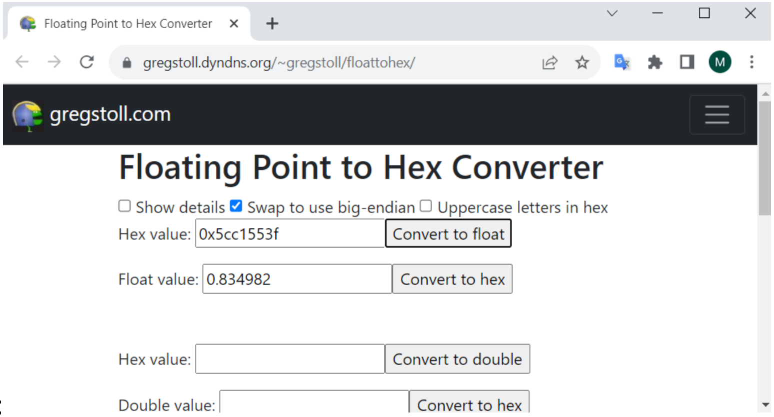
Verification

Note- For the sake of simplicity, I have stored the solutions (if they exist) in memory locations from (0x10000080 to 0x10000090)

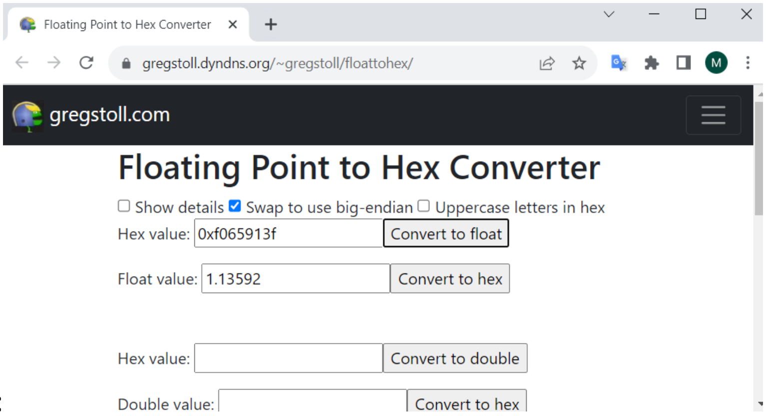
Address	+0	+1	+2	+3
0x10000090	B0	75	94	BF
0x1000008C	E3	56	5A	BE
0x10000088	F0	65	91	3F
0x10000084	5C	C1	55	3F
0x10000080	F5	73	B6	3F


Floating Point to Hex Converter
gregstoll.dyndns.org/~gregstoll/floattohex/
gregstoll.com
Floating Point to Hex Converter
Show details
Swap to use big-endian
Uppercase letters in hex
Hex value: 0xf573b63f
Convert to float
Float value: 1.42541
Convert to hex
Hex value:
Convert to double
Double value:
Convert to hex

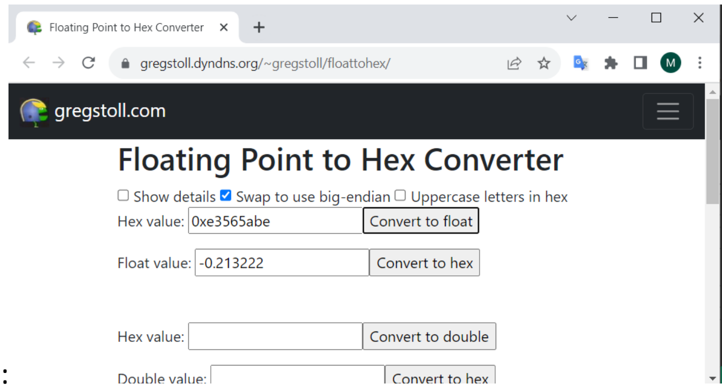
e:


Floating Point to Hex Converter
gregstoll.dyndns.org/~gregstoll/floattohex/
gregstoll.com
Floating Point to Hex Converter
Show details
Swap to use big-endian
Uppercase letters in hex
Hex value: 0x5cc1553f
Convert to float
Float value: 0.834982
Convert to hex
Hex value:
Convert to double
Double value:
Convert to hex

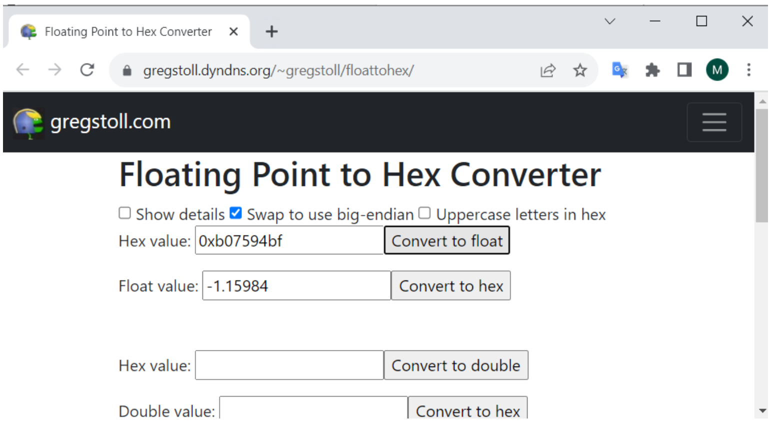
d:


Floating Point to Hex Converter
gregstoll.dyndns.org/~gregstoll/floattohex/
gregstoll.com
Floating Point to Hex Converter
Show details
Swap to use big-endian
Uppercase letters in hex
Hex value: 0xf065913f
Convert to float
Float value: 1.13592
Convert to hex
Hex value:
Convert to double
Double value:
Convert to hex

c:


Floating Point to Hex Converter
gregstoll.dyndns.org/~gregstoll/floattohex/
gregstoll.com
Floating Point to Hex Converter
Show details
Swap to use big-endian
Uppercase letters in hex
Hex value: 0xe3565abe
Convert to float
Float value: -0.213222
Convert to hex
Hex value:
Convert to double
Double value:
Convert to hex

b:

a: 

Error %:

% error for e = 0.000151011991%

% error for d = 1.66098568E-5%

% error for c = 0.000308060578%

% error for b = 4.31309892E-5%

% error for a = 0.000271078946%

<u>Example 2: (Infinite solution)</u>

[A:B]= 1x + 2y + 3z + 4w + 5v = 10

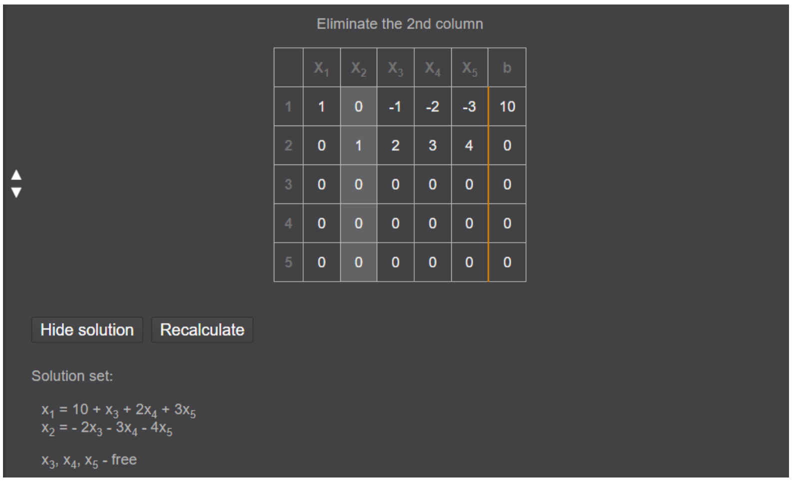
2x + 3y + 4z + 5w + 6v = 20

3x + 4y + 5z + 6w + 7v = 30

4x + 5y + 6z + 7w + 8v = 40

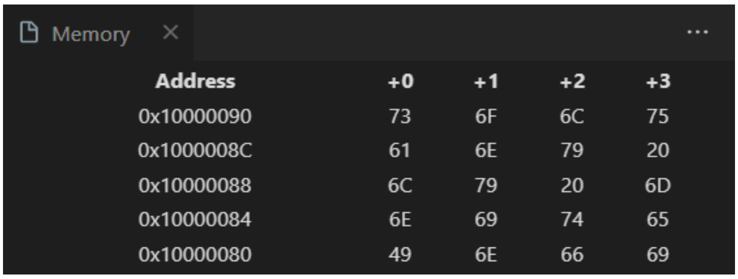
5x + 6y + 7z + 8w + 9v = 50

Actual:

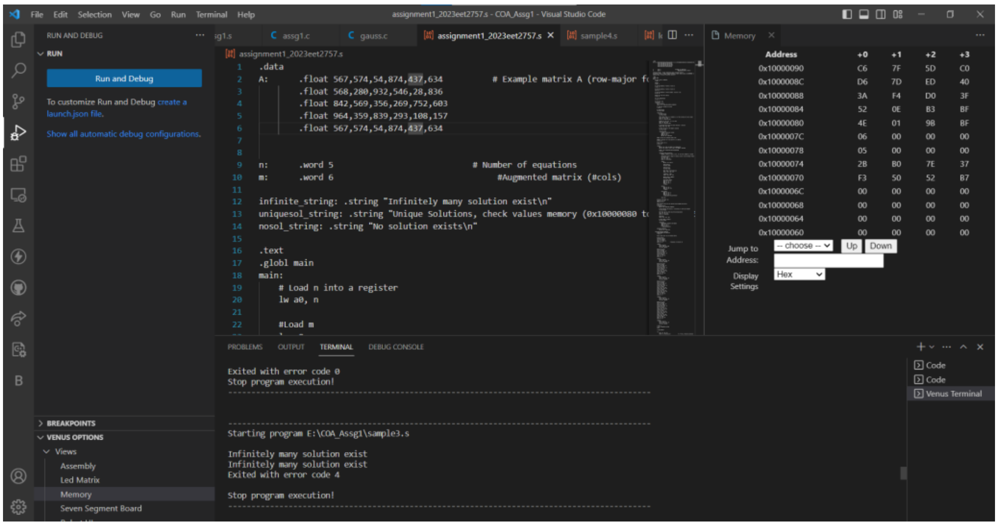


	X_1	X_2	X_3	X_4	X_5	b
1	1	0	-1	-2	-3	10
2	0	1	2	3	4	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

My output:



Address	+0	+1	+2	+3
0x10000090	73	6F	6C	75
0x1000008C	61	6E	79	20
0x10000088	6C	79	20	6D
0x10000084	6E	69	74	65
0x10000080	49	6E	66	69



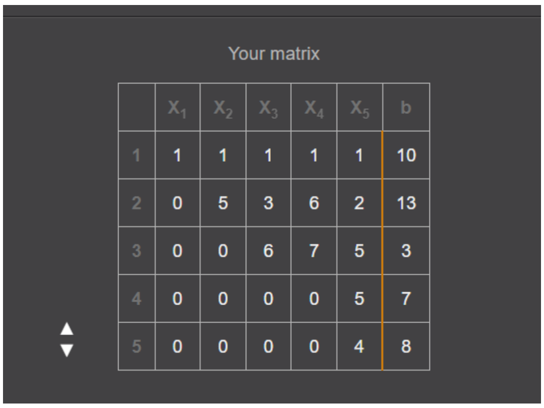
Note- It detects that the system of linear equations have infinite solutions in this case. It also produces some output.

It sometimes may not work, Reason: floating point operations like fdiv and fmul cause inaccuracy in precision.

Ex: 0x41100000 – 0x41100000 = 0x34800000 which is not correct, yet it still produces the result due to above reason

<u>Example 3: (No solution)</u>

Actual

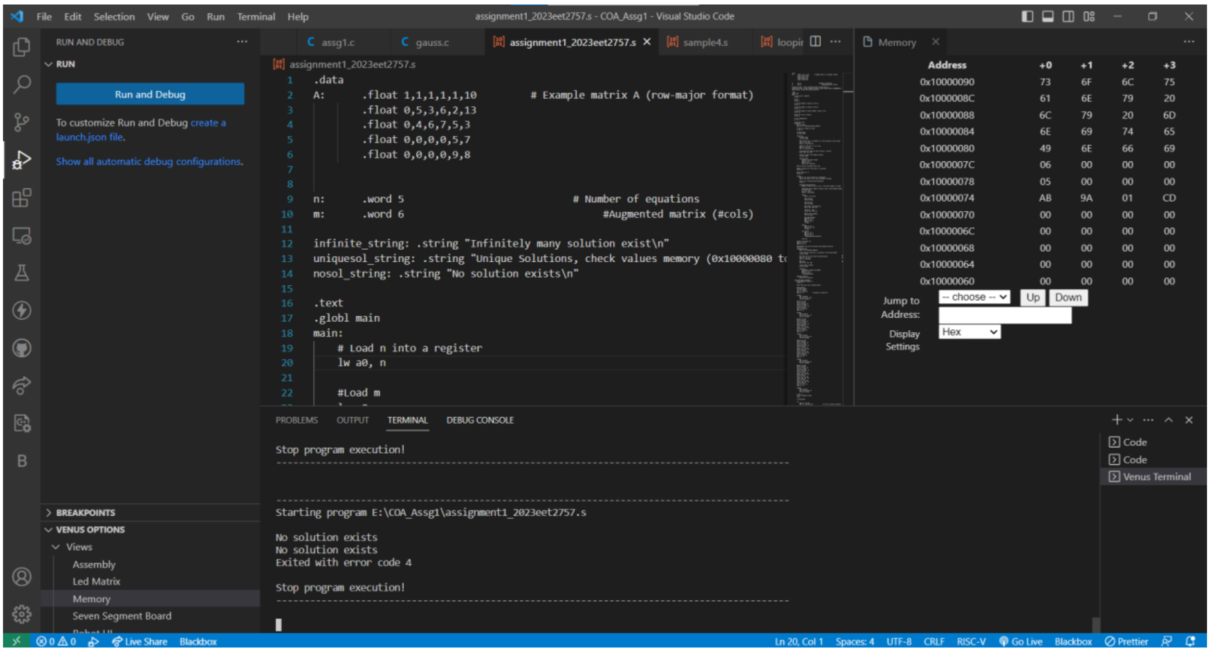


Your matrix

	X_1	X_2	X_3	X_4	X_5	b
1	1	1	1	1	1	10
2	0	5	3	6	2	13
3	0	0	6	7	5	3
4	0	0	0	0	5	7
5	0	0	0	0	4	8

The system is inconsistent (no solution)

My output:



Actual and My output are same. Hence the code can detect No solutions easily.