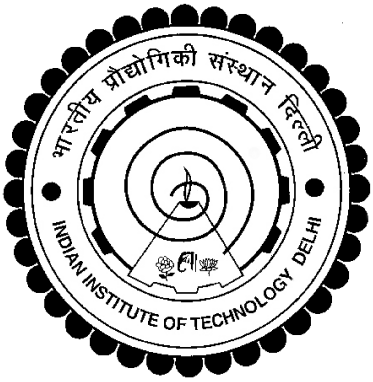


Indian Institute of Technology Delhi

ELL 782: Computer Architecture

Assignment On

Solve a set of linear equations using the Gaussian elimination method In RISC-V

Name:	Bitthal Patel
Entry No.:	2023EET2184
Course Instructor:	Kaushik Saha

1 Software Requirement Specification

Specification	**Details**
Software Name	Linear equations solver In RISC-V
Developed By	Bitthal Patel/IIT Delhi
Development Language	RISC-V Assembly
Target Platform	RISC-V architecture

Table 1: Software Specification Details

Functionality	**Details**
Purpose	solve systems of linear equations using Gaussian elimination.
Input	Augmented Matrix.
Output	No solution,Infinte Soluntion and Solution vector (x) in case of Unique solution
Precision	upto 2^{23} (single-precision floating-point).
Error Handling	Detect and handle singular matrices.

Table 2: Functional Details

Execution Platform	**Details**
Processor Type	RISC-V architecture (e.g., RV32IMAF, RV64IMAF)
Instruction Set	RISC-V ISA (possibly with extensions)
Memory Requirements	Dependent on the matrix size and precision level.

Table 3: Platform Details

2 Design Considerations

- Assumed input matrix in Row-Major form.
- Works on any operating system.
- Visual studio code is required for execution.
- Extensions like RISC-V Support and RISC-V Venus simulator are needed for syntax highlighting and execution.
- Matrix elements should be in float format, double format is not supported.

- Registers a0(x10), and a1(x11) are not taken in use as they are reserved for system calls, using them results in program failure.
- Output for the unique solution is in Hex format.

3 Architectural Strategies

3.1 Gaussian Elimination Algorithm:

- Utilize the Gaussian elimination algorithm as the numerical method for solving linear equations.
- Used partial pivoting to improve numerical stability and prevent division by small numbers.

3.2 Modular Design:

- Divided the software into modular components to enhance maintainability and reusability
- Modules include matrix partial pivoting, elimination, back-substitution, print matrix, and print solution.

3.3 Data Types and Sizes:

- Float data is used in 32-bit floating point format.
- 23 bits are used for fraction, 8 bits for exponent and 1 bit for sign

4 Software Architecture

4.1 Modules:

- **Take Equations as Augmented Matrix:**

  Given a system of linear equations, represent it as an augmented matrix by combining the coefficients of variables and constants.
  Example: let equations as

  2x + 3y - z = 1
  4x + 7y + z = 2
  x + y + 2z = 3

  Then augmented matrix is

$$\begin{bmatrix} 2 & 3 & -1 & | & 1 \\ 4 & 7 & 1 & | & 3 \\ 1 & 1 & 2 & | & 3 \end{bmatrix}$$

- **Partial Pivoting:**

  Before performing any row operations, choose the pivot element in the current column as the largest absolute value among the remaining elements in that column. Swap rows if necessary to place the pivot element in the current row. The pivot element is chosen to minimize division by small numbers, which helps improve numerical stability.

- **Elimination Step:**

  For each row below the current row, subtract a multiple of the current row from that row to make the element below the pivot element zero. The multiple is chosen such that the element below the pivot becomes zero.
  After completing the elimination steps, the augmented matrix should be in row echelon form, where the leading coefficient (pivot) in each row is to the right of the leading coefficient in the row above it, and each column below a pivot contains all zeros.

$$\left[\begin{array}{ccc|c} 2 & 3 & -1 & 1 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 9/2 & 5/2 \end{array}\right]$$

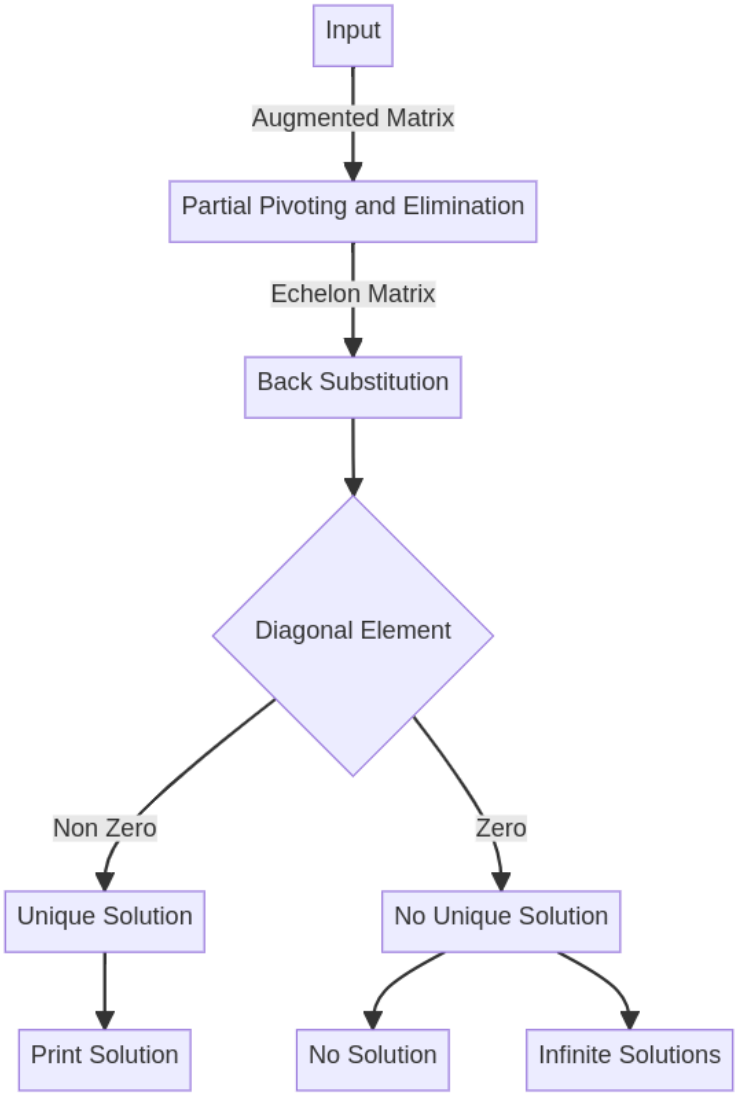
- **Back Substitution:**

  Starting from the last row and working upwards, solve for each variable one by one. Substitute the values of already solved variables into the equations to find the values of the remaining variables.

- **Handling Singular Matrices:**
  Gaussian elimination with partial pivoting can help detect and handle singular matrices (matrices with no Unique solution) during the elimination process if a diagonal element is zero, it indicates that the system is singular and has no unique solution, and if the value of x corresponding to the diagonal element is zero, then Infinite solution exists otherwise No solution.

- **Solution:**
  The result of the back substitution will give you the solution to the system of linear equations.

4.2 Block Diagram:



5 Detailed System Design:

Using VScode for implementing the code and Venus Simulator in VScode supports RV32IMAF syntax, F means this supports Floating point instructions and follows IEEE-754 32bit format that is 1 bit is allocated as the sign bit, the next 8 bits are allocated as the exponent field, and the last 23 bits are the fractional parts of the normalized number, Hence giving 2^{23} precision for fraction bits.
Implementation consists of different functions executed for every step of algorithm. Functions are as follows:

5.1 Declaration and Initialization:

The Augmented Matrix is of size 5x6 as there are five variables and taken in 1-D Array as Row Major Order ,in float and solution array consist of calculated values

of variables, also declared strings for different cases of solutions.

```
arr:  .float -6.00, 4.00, 6.00, -8.00,-3.00,-5.35,
-4.00, 3.00, 4.00, -4.00,-9.00,11.25,
-3.00, -1.00, -1.00,-6.00,2.00,-29.34,
-2.00, -4.00, 6.00,-5.00,7.00, -53.68,
-7.00, -9.00, -10.00,5.00,9.00,12.36
sol: .float 0.0, 0.0, 0.0, 0.0, 0.0

no_solution_msg: .asciiz "No solution exists.\n"
solu: .asciiz "x= \n"
unique_solution_msg: .asciiz "Unique solution exists.\n"
Infinite_solution_msg: .asciiz "Infinite solution exists.\n"
```

Figure 1

5.2 Partial Pivoting:

- Initializing the loop variables i,jand k.
- Now, checking the Pivoting condition inside the loop, there's an if condition that checks whether the absolute value of the diagonal element **'a[i][i]'** (where i is the current row index) is smaller than the absolute value of the corresponding element in the row below it **'(a[k][i])'**
- If the condition is met (i.e., if the diagonal element is smaller in magnitude than the element below it), a row swap is performed. This means that the current row **'i'** is swapped with the row **'k'**. The swap is achieved by iterating over all columns **(j)** and exchanging the values of the elements in these two rows.
- This effectively reorders the rows of the coefficient matrix to ensure that the element with the largest absolute value in the current column is moved to the diagonal position **(a[i][i])**, which can help improve the numerical stability of the Gaussian elimination process.
- Partial pivoting is a technique used to minimize potential issues with division by small values during the elimination step of Gaussian elimination, making the algorithm more accurate when solving systems of linear equations.

5.3 Begin Gauss Elimination:

- Inside the loop, check if the diagonal element **'a[i][i]'** (where 'i' is the current row index) is equal to zero. If it is zero, the loop breaks, as dividing by zero is undefined in mathematical operations.
- If the diagonal element is nonzero, the code proceeds to calculate the coefficient **'term'**. This coefficient is determined by dividing the element in the current row **'k'** and the current column **'i' ('a[k][i]')** by the diagonal element in the same column and row **('a[i][i]')**. This calculation represents the factor by which the current row needs to be scaled before subtracting it from the row above to eliminate the variable **'a[k][i]'**.

- After calculating the coefficient **'term'**, there's another loop with the variable **'j'** that iterates over all columns in the matrix (n). For each column, it subtracts the scaled row **'i'** (multiplied by term) from the row **'k'**. This effectively eliminates the variable **'a[k][i]'** from row **'k'**.

- Finally, rows are modified to achieve an upper triangular form of the coefficient matrix.

5.4 Begin Back-substitution:

- For each row **'i'**, the code initializes the value of the variable **'x[i]'** with the right-hand side constant, which is stored in the last column of the coefficient matrix **'a' '(a[i][n - 1])'**. This step effectively initializes the solution for the current variable.

- Next, there's an inner loop with the variable **'j'** that iterates over the columns to the right of the diagonal element **'(i + 1 to n - 1)'**. In this loop, the code updates the solution variable **'x[i]'** by subtracting the products of the coefficients **'a[i][j]'** and the corresponding solution variables **'x[j]'**. This step accounts for the contributions of the variables to the right of the current one.

- Then checking for singular matrices which checks if **'x[i]'** is equal to zero. If it is, it performs further checks, and if not, then it calls **Print** function.

- If **'x[i]'** is zero and the diagonal element **'a[i][i]'** is also zero, it prints **"Infinite solution exists"**. This situation implies that there are infinitely many solutions to the system.

- If **'x[i]'** is nonzero and the diagonal element **'a[i][i]'** is zero, it prints **"No solution exists"**. This means there is no unique solution to the system.

5.5 Output Solution:

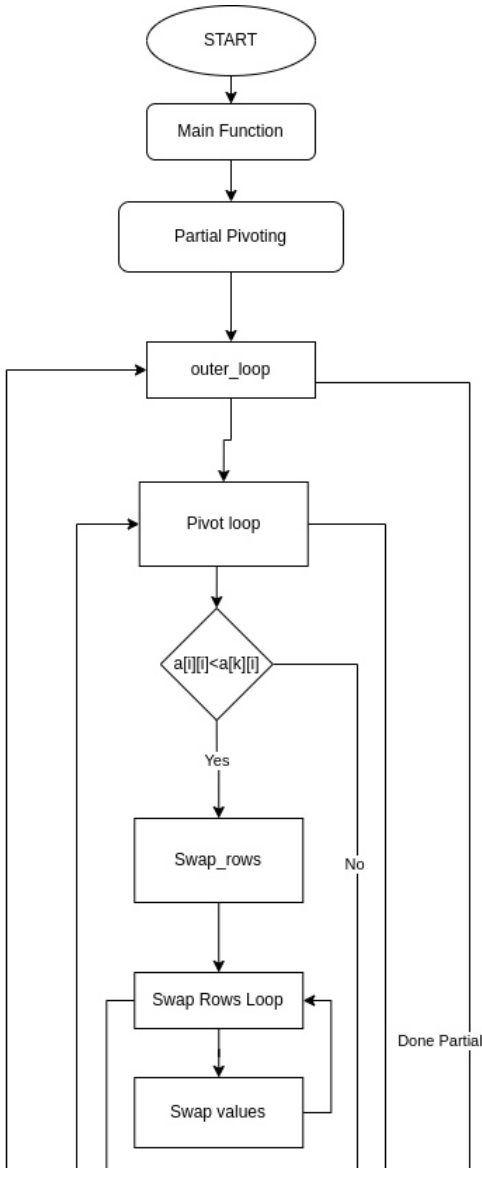
Now, then, for the case of Unique Solution, we print the values stored in the solution array. The output will be in Hex as it supports Hex printing.

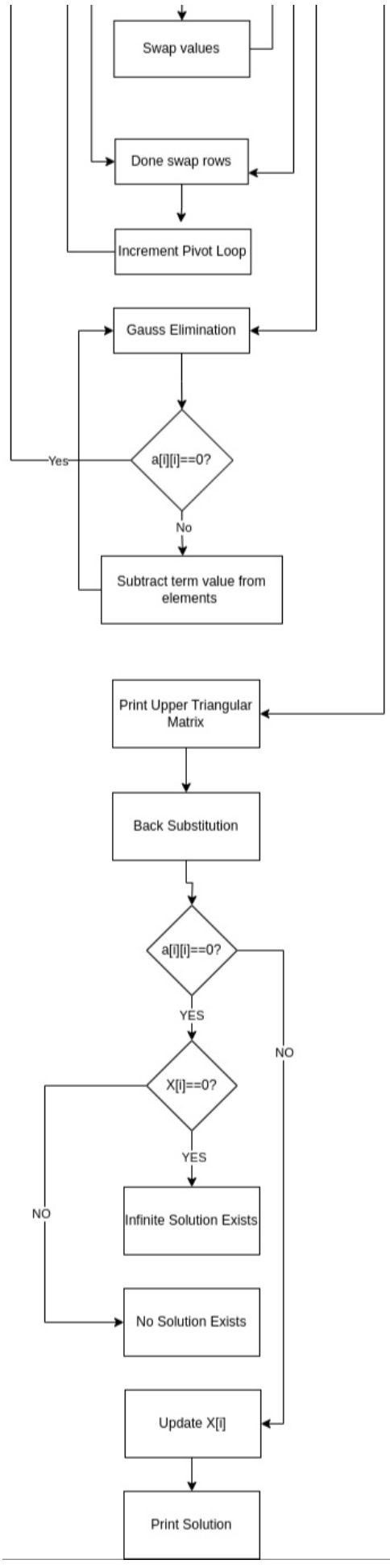
```
Upper Triangular Matrix :

0xC0E00000      0xC1100000      0xC1200000      0x40A00000      0x41100000      0x41400000
0x00000000      0x413B6DB7      0x41692492      0xC1449249      0xC12B6DB7      0xC2712492
0x00000000      0x00000000      0x412A2576      0xC0FDA895      0x4047CE0B      0xC27F1F38
0x00000000      0x00000000      0x00000000      0xC0AB1524      0x3F55B98C      0xC1A862A4
0x00000000      0x00000000      0x00000000      0x00000000      0xC0CB7BC5      0x42189CD5

X[0]=0x40000001
X[1]=0xC0A00002
X[2]=0xC0000002
X[3]=0x403FFFFE
X[4]=0xC0C00002
Exited with error code 0
Stop program execution!
```

5.6 Flow Chart:





6 Testing:

Used C++program of Gaussian Elimination using pivoting for cross-verifying the RISC-V Hex output with the C++program output. **Test Augumentend Matrix:**

$$A = \begin{bmatrix} -6 & 4 & 6 & -8 & -3 & -5.352671 \\ -4 & 3 & 4 & -4 & -9 & 11.252345 \\ -3 & -1 & -1 & -6 & 2 & -29.341213 \\ -2 & -4 & 6 & -5 & 7 & -53.686889 \\ -7 & -9 & -10 & 5 & 9 & 12.366598 \end{bmatrix}$$

Outputs:

Expected Correct Output	**RISC-V Output**
X[1]= -4.008226	X-RISC-V[1]= -4.008225
X[2]= 5.335697	X-RISC-V[2]= 5.335696
X[3]= -0.937791	X-RISC-V[3]= -0.937793
X[4]= 5.915885	X-RISC-V[4]= 5.915884
X[5]= -0.736339	X-RISC-V[5]= -0.736340

Mean Error=0.000001