Indian Institute of Technology Delhi

ELL 782: Computer Architecture

Assignment-1

Solve systems of linear equations using the Gaussian elimination method in RISC-V ISA

Name: *Ranjan Kumar Singha*
Entry No.: 2023EET2192
Course Instructor: Kaushik Saha

Contents

1 **Software Requirement Specification**

2 Design Considerations

3 Architectural Strategies

4 Software Architecture

5 Detailed System Design

6 Testing

1 Software Requirement Specification

Requirement	Description
Software title	Gaussian Elimination Implementation on RISC-V ISA
Version	1.0
Developed By	[Ranjan Kumar Singha/IIT Delhi]
Target Platform	RISC-V ISA architecture (e.g.- RV32IMAF)
Development Language	Assembly language for RISC-V (e.g.- RISC-V assembly)
Development Environment	VScode with RISC-V toolchain and Venus Simulator
Compiler/Assembler	RISC-V specific assembler
Execution Platform	RISC-V compatible processor or simulator
Required Hardware	RISC-V compatible hardware or simulator
Operating System	Platform-independent

2 Design Considerations

- **Input Matrix:** The input matrix is assumed to be in Row-Major format.
- **Cross-Platform Compatibility:** The software is designed to work seamlessly on various operating systems, ensuring platform independence.
- **Development Environment**: To execute the program, Visual Studio Code (VScode) is a prerequisite.
- **Required Extensions:** Essential VScode extensions, such as "RISC-V Support" and "RISC-V Venus simulator," are necessary for syntax highlighting and execution.
- **Data Representation:** Matrix elements must be in float format, as double format is not supported.
- **Reserved Registers:** Registers a0 (x10) and a1 (x11) are reserved for system calls and should not be used to avoid program failure.
- **Output Format:** The output for unique solutions should be presented in hexadecimal format.

3 Architectural Strategies

3.1 Algorithm Selection:

Gaussian Elimination: The core numerical method used for solving linear equations. This algorithm is chosen for its effectiveness in handling a wide range of equation systems.

3.2 Modular Design:

Modular Components: The system is designed with a modular structure, separating different functionalities into distinct modules. This enhances maintainability, reusability, and ease of testing.

Module Independence: Modules, such as matrix partial pivoting, elimination, back-substitution, matrix printing, and solution printing, are designed to be self-contained and independent.

3.3 Numerical Stability:

Partial Pivoting: Implemented within the Gaussian Elimination algorithm to improve numerical stability. It prevents division by small numbers and reduces the potential for computational errors.

3.4 Data Types and Sizes:

32-Bit Floating-Point: Float data type is used throughout the system in 32-bit floating-point format, adhering to the IEEE-754 standard.

Data Representation: The format comprises 1 bit for sign, 8 bits for the exponent, and 23 bits for the fraction. This choice ensures compatibility, precision, and efficient use of memory.

3.5 Error Handling:

Robust Error Handling: Robust error handling mechanisms are integrated to gracefully handle unexpected scenarios, ensuring the system's reliability and user-friendliness.

3.6 Performance Optimization:

Efficiency: The system is optimized for efficiency to minimize computational overhead and execution time, providing faster results for solving linear equations.

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 take 3 equations,

  $4x + 7y - 2z = 1$
  $9x + 5y + z = 2$
  $x - y + 5z = 9$

  Then augmented Matrix is

  $$\begin{array}{ccc|c} 4 & 7 & -2 & 1 \\ 9 & 5 & 1 & 2 \\ 1 & -1 & 5 & 9 \end{array}$$

- **Partial Pivoting for Consistent Output:**

  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.

- **Forward Elimination:**

  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.

  $$\begin{array}{ccccc} 9 & 5 & 1 & 2 & 0 \\ 0 & 6.4 & -2.1 & 0.7 & 0 \\ 0 & 0 & 3.5 & 8.5 & -0.5 \end{array}$$

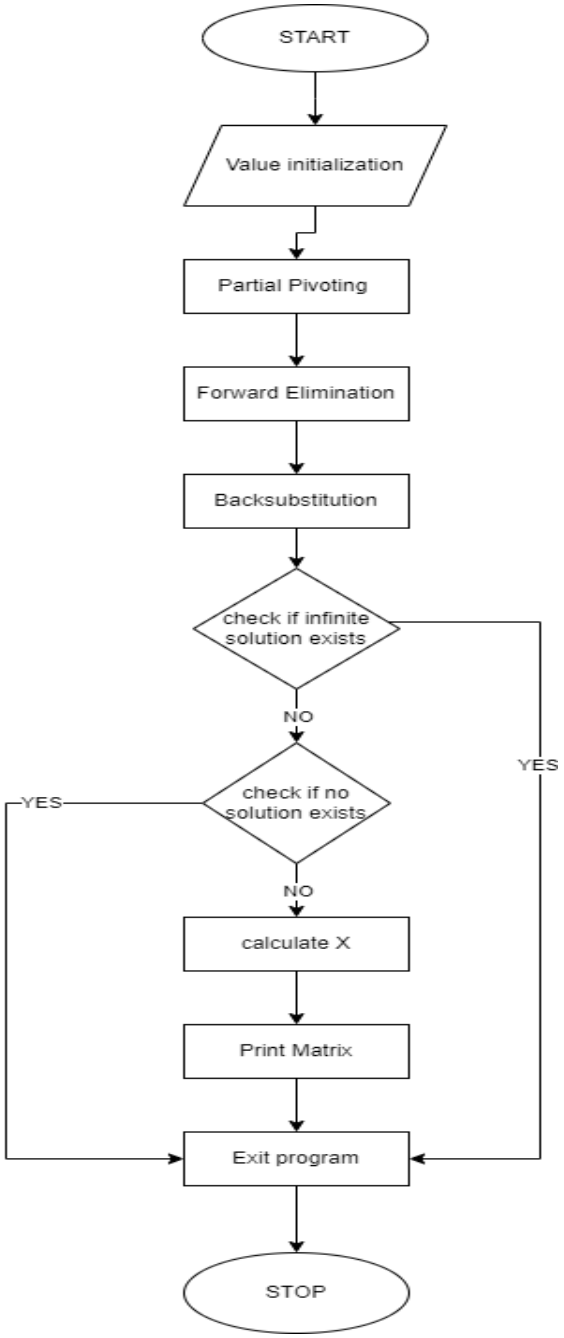
- **Back Substitution to find Elimination:**

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

- **Singular Matrices Condition:**
  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 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 consisting of distinct functions executed for every step of algorithm, functions are as follows:

5.1

A. 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 and solution array consist of calculated values of variables, also declared strings for different cases of solutions.

```
.data
mat:
    .float -6.0, 4.0, 6.0, -8.0, -3.0, -50.0
    .float -4.0, 3.0, 4.0, -4.0, -9.0, 11.0
    .float -3.0, -1.0, -1.0, -6.0, 2.0, -29.0
    .float -2.0, -4.0, 6.0, -5.0, 7.0, -53.0
    .float -7.0, -9.0, -10.0, 5.0, 9.0, 12.0

X:
    .float 0.0, 0.0, 0.0, 0.0, 0.0
```

B. Partial Pivoting:

In this partial pivoting process, we are comparing if the diagonal elements are less than the elements below them, if it happens then we swap the whole row elements.

```
# Comparison mat[i][i] <= mat[j][i]
fabs.s  f1, f1              # Calculate absolute value of diagonal element
fabs.s  f2, f2              # Calculate absolute value of element below diagonal

li x13, 0                   # x13: k = 0

mv x12, x0                  # x12 = 0
flt.s x12, f1, f2           # if f1 <= f2, set x12 to 1
bne x12, x0, swap_loop      # if x12 != 0 then branch to swap_loop

flt.s x12, f2, f1           # if f2 <= f1, set x12 to 1
bne x12, x0, no_swap        # if x12 != 0 then branch to no_swap
```

C. Gauss Elimination:

In this process we convert the augmented matrix into row-echelon form.

```
fmul.s f10, f7, f9              # f10: term * mat[i][k]
fsub.s f8, f8, f10              # f8 = f8 - f10

# Update value in matrix
fsw f8, 0(x19)

addi x18, x18, 1                # increment k
j elimination_loop
```

D. Back Substitution:

Through this process we get the values of the unknown one by one.

```
# Locating X[i]
mul x12, x8, x3                 # i * offset
add x12, x5, x12                # B.A + (i * offset)

# Storing into X[i]
fsw f11, 0(x12)                 # Updating the value in x[i] = mat[i][n-1]
flw f12, 0(x12)                 # Load X[i] into f12

addi x9, x8, 1                  # j = i + 1
j j_loop
```

E. Unique Solution:

```
print_unique_solution:
    bge x31, x7, exit_program  # if i >= n - 1, exit_program

    mv x30, x31                # Load i to x30
    mul x30, x30, x3           # i * offset
    add x30, x30, x5           # B.A. + (i * offset)

    addi a0, x0, 34            # Load a0 with a constant value (34)

    lw a1, 0(x30)              # Load the value at address x30 into a1
    ecall                      # Perform a system call (specific syscall action unclear)
```

F. Infinite numbers of solutions:

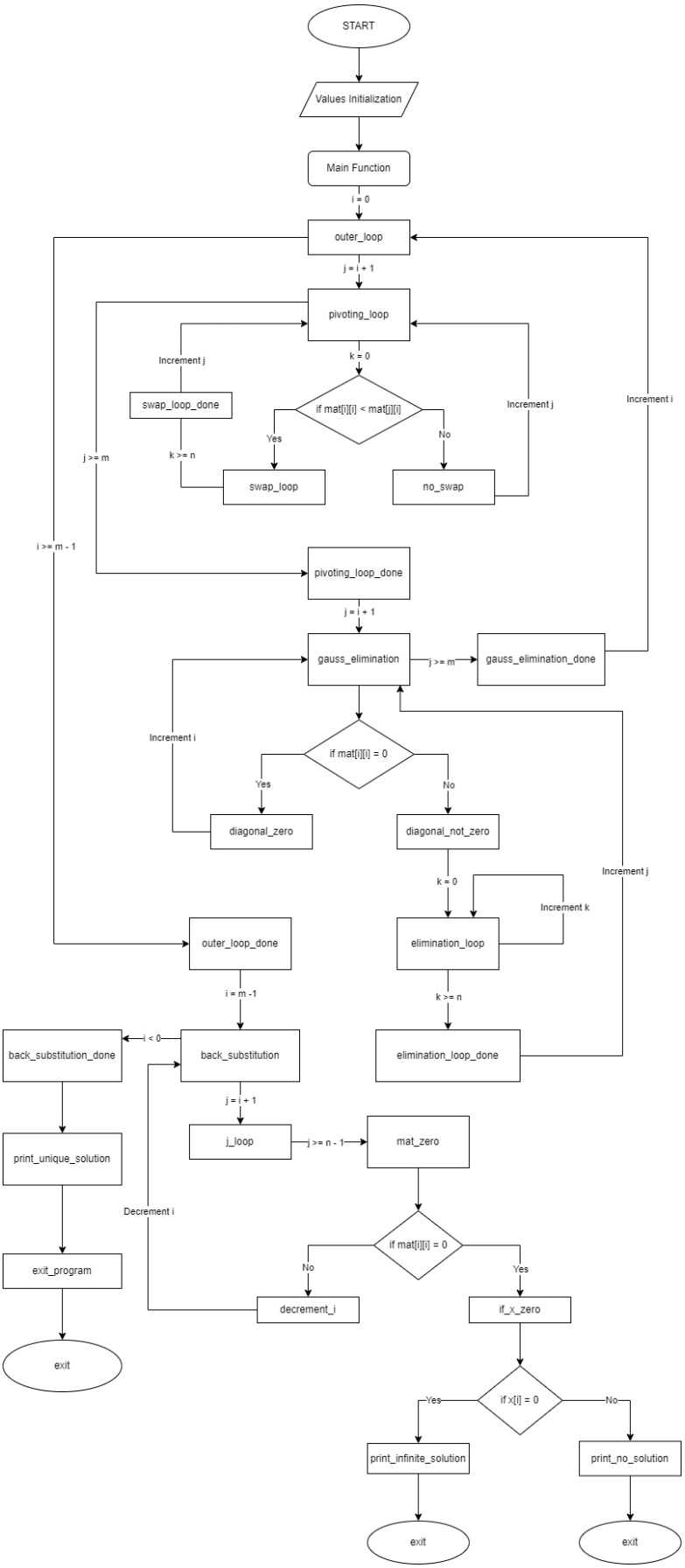
```
print_infinite_solution:
    addi a0,x0,4
    la a1, str_infinite_solution   # Load address of the string
    ecall                           # Print the string
    j exit_program
```

G. No Solution:

```
print_no_solution:
    addi a0,x0,4
    la a1, str_no_solution    # Load address of the string    # Load mat[i][n-1]: 0(x25) into f11

    # Locating X[i]
    mul x12, x8, x3
    ecall                     # Print the string
    j exit_program
```

5.2 Flow Chart of RISC-V ISA implementation:


START
Values Initialization
Main Function
i = 0
outer_loop
j = i + 1
pivoting_loop
k = 0
Increment j
swap_loop_done
if mat[i][i] < mat[j][i]
Increment j
Increment i
Yes
No
k >= n
j >= m
swap_loop
no_swap
i >= m - 1
pivoting_loop_done
j = i + 1
gauss_elimination
j >= m
gauss_elimination_done
Increment i
if mat[i][i] = 0
Yes
No
diagonal_zero
diagonal_not_zero
Increment j
k = 0
Increment k
outer_loop_done
elimination_loop
i = m -1
k >= n
back_substitution_done
i < 0
back_substitution
elimination_loop_done
j = i + 1
print_unique_solution
j_loop
j >= n - 1
mat_zero
Decrement i
if mat[i][i] = 0
exit_program
No
Yes
decrement_i
if_x_zero
exit
if x[i] = 0
Yes
No
print_infinite_solution
print_no_solution
exit
exit

6 Testing:

- The inputs that have been used for testing:

```
.data
mat:
    .float -6.0, 4.0, 6.0, -8.0, -3.0, -50.0
    .float -4.0, 3.0, 4.0, -4.0, -9.0, 11.0
    .float -3.0, -1.0, -1.0, -6.0, 2.0, -29.0
    .float -2.0, -4.0, 6.0, -5.0, 7.0, -53.0
    .float -7.0, -9.0, -10.0, 5.0, 9.0, 12.0

X:
    .float 0.0, 0.0, 0.0, 0.0, 0.0
```

- Actual Output obtained:

```
0x40000001
0xC0A00002
0xC0000002
0x403FFFFE
0xC0C00002
```

- Converting the Hexadecimal to float numbers:

```
Hexadecimal: 0x40000001 -> Float: 2.000000
Hexadecimal: 0xC0A00002 -> Float: -5.000001
Hexadecimal: 0xC0000002 -> Float: -2.000000
Hexadecimal: 0x403FFFFE -> Float: 3.000000
Hexadecimal: 0xC0C00002 -> Float: -6.000001
```

- Expected Correct Output (from python programme):

```
Solution:
x1 = 2.000000000000013
x2 = -5.000000000000044
x3 = -2.000000000000000
x4 = 2.9999999999999982
x5 = -6.000000000000027
```

- Error Percentage calculated (from python programme):

```
Mean Absolute Percentage Error (MAPE): 7.333333332878068e-06%
```