SOFTWARE DESIGN DOCUMENT

for

Gaussian Elimination on RV32IMF

Prepared by: Mitesh Pandey (2023EET2473)

Submitted to: Prof. Kaushik Saha
Lecturer
IIT Delhi

September 3, 2023

Contents

1 Software Requirement Specifications

1.1 Specifications, Functionality and Execution Platform

RISC V Unpriviledged ISA		
Base	**Version**	**Status**
RV32I	2.1	Ratified
Extension	**Version**	**Status**
M	2.0	Ratified
F	2.2	Ratified
Execution platform	RISC-V Venus Simulator	

2 Design Considerations

2.1 Assumptions and Dependencies

- The program is being executed in RISC-V Venus Simulator embedded in VS Code.
- RISC-V Venus Simulator embedded in VS Code is assumed to be supporting 32-bit floating point instructions.
- The augmented matrix consisting of the system of equations and its solution is generated separately using a Python script and then fed to the RISC V program in the data segment.
- The final solution is displayed in the terminal as a sequence of hexadecimal numbers which is left on the user's end to verify for correctness.

2.2 General Constraints

- 32-bit floating point instructions are used for multiplication and division tasks. It may pose a problem if the values of the coefficients are near zero.
- The lines of code have been increased to make the code readable to avoid problems while debugging as Gaussian elimination involves nested loops.

2.3 Goals and Guidelines

- A full implementation of Gaussian elimination on RISC-V would require more code and error handling. To serve that, code is written to make it as modular as possible using informative labels.
- Each branch/module is responsible for handling specific statements of the main C program making it easier to locate the error location in case of debugging.
- The total program size is approximately 1600 bytes considering the 'print' procedure for the augmented matrix and the solution.
- The modular approach also helped in managing the registers effectively, as it didn't pose a problem of 'inadequate registers'.

2.4 Development Methods

- A full C code implementation was made in order to ease the conversion process to RISC V Assembly code.
- The C code was compiled to assembly using *RISCV-rv32-gc* compiler and the pattern of the code conversion was studied thoroughly before final implementation to get an idea of how the compiler does it.
- Agile development methodology has been implemented widely during the program development cycle. Smaller portions were written, and tested and then the new code was introduced.

3 Architectural Strategies

3.1 Breakdown of If block

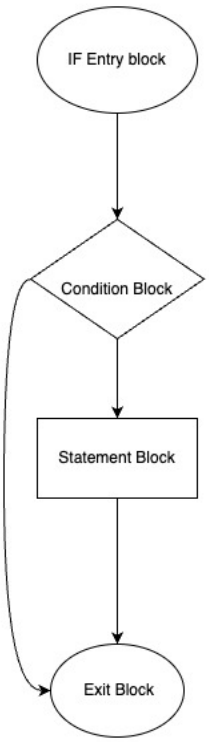

Figure 3.1: If-Block implementation strategy

if-Block is broken down into sub-blocks so as to make the implementation part clear, reduce the use of more registers, and make the structure modular enough to ease the debugging process.

3.2 Breakdown of For Block

for-loop is also broken into sub-blocks for similar reasons.

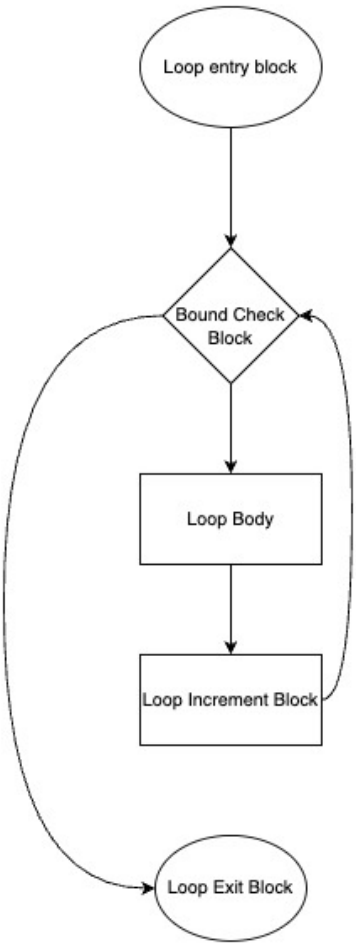

Figure 3.2: FOR-Block implementation strategy

3.3 Bundling of variables with the shared base address

Variables used in the code such as indexes i,j,k, row_number, and col_number are bundled into a single variable label. Upon loading that variable all the variables can be fetched simply by varying offset with the base address.

```
.data
constants: .word 0 0 0 5 6 # i j k row_number col_number
la a0,constants
lw a1,0(a0) # a1 = i
lw a2,8(a0) # a1 = k
lw a3,16(a0) #a3 = col_number
```

4 Software Architecture

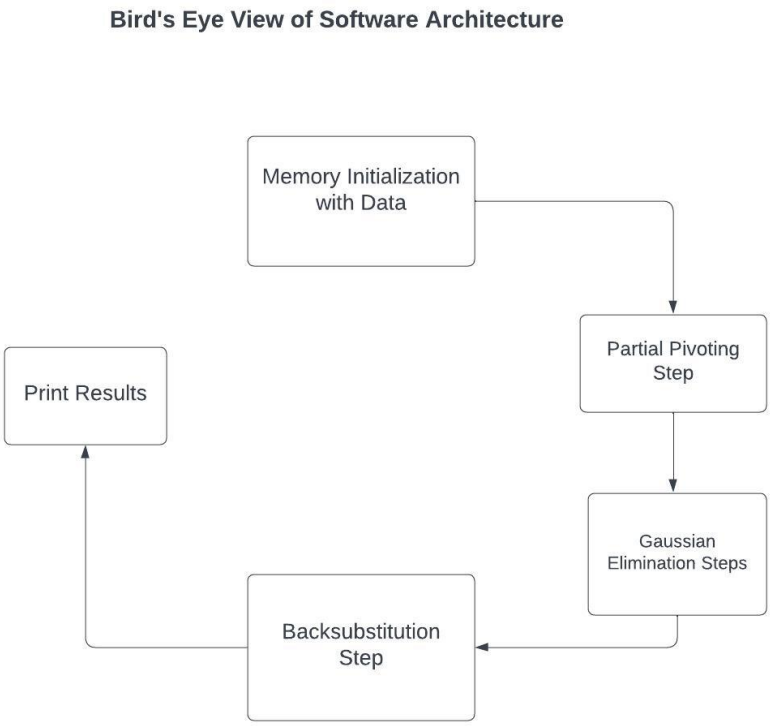

Figure 4.1: Interaction of different modules of program

5 Detailed System Design

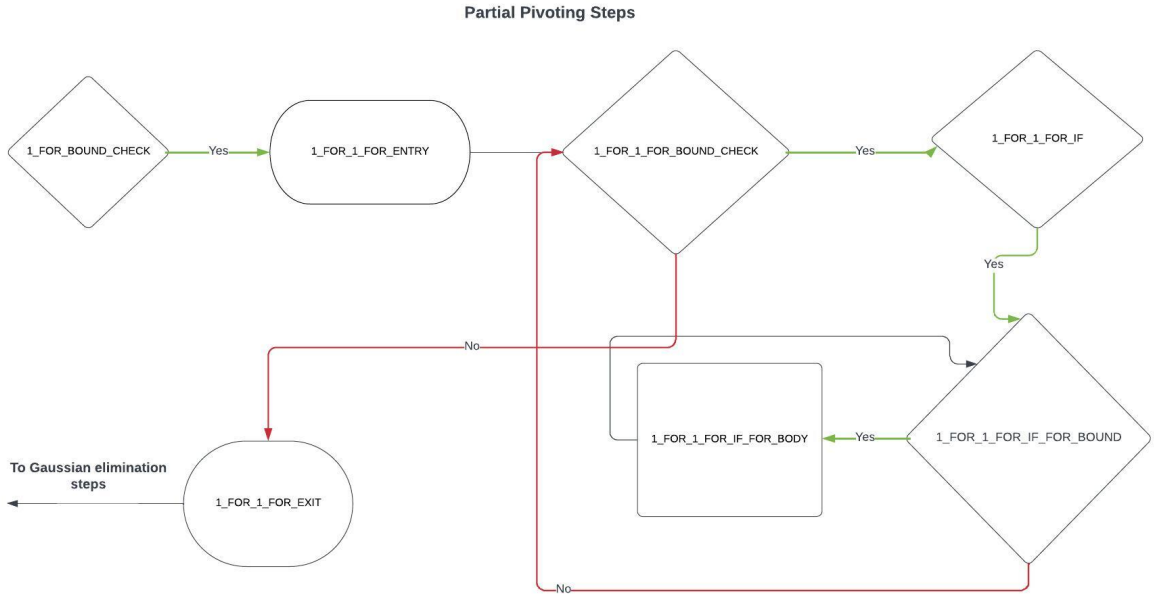

Figure 5.1: Different blocks of row swapping stage

where Naming conventions of the block mean as such :

- 1_FOR_1_FOR_IF_FOR_BODY :

```
for{ //First For loop of the program named as 1_FOR
    for{ // named as 1_FOR_1_FOR
        if( condition){ //named as 1_FOR_1_FOR_IF
            for{ //1_FOR_1_FOR_IF_FOR_BOUND
                !!! THIS IS THE SECTION REFERRED TO !!!
            }
        }
    }
}
for{ //Second For loop of the program named as 2-FOR

}
```

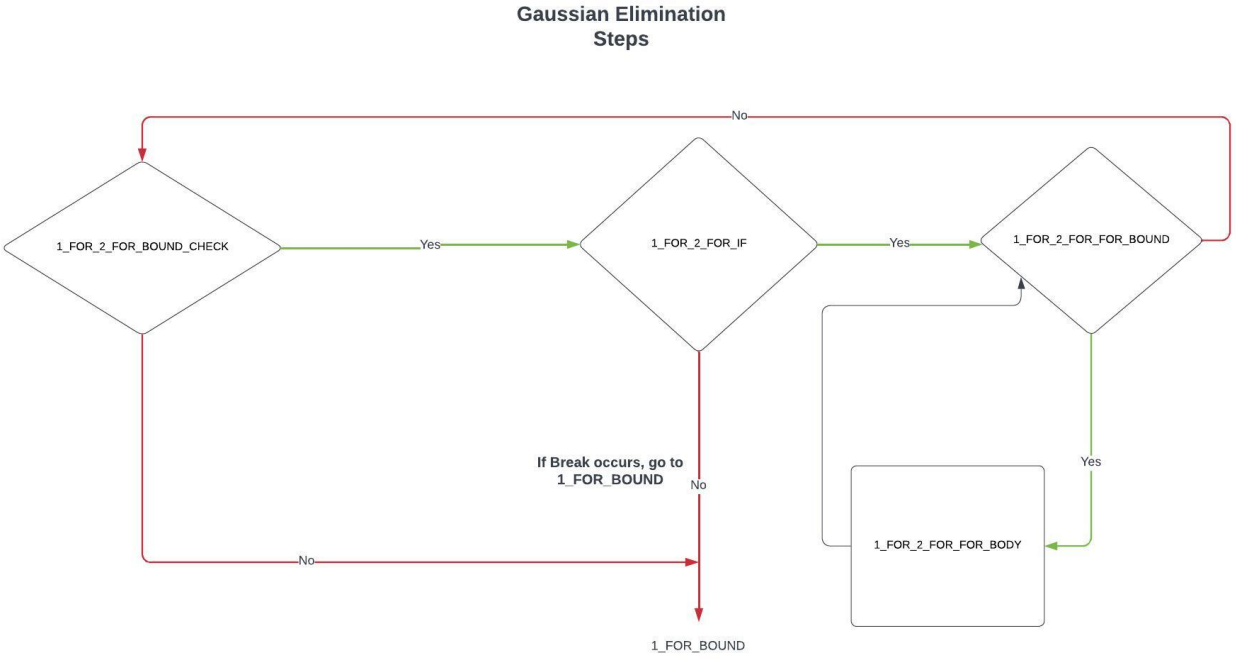

Figure 5.2: Different blocks of reduction step

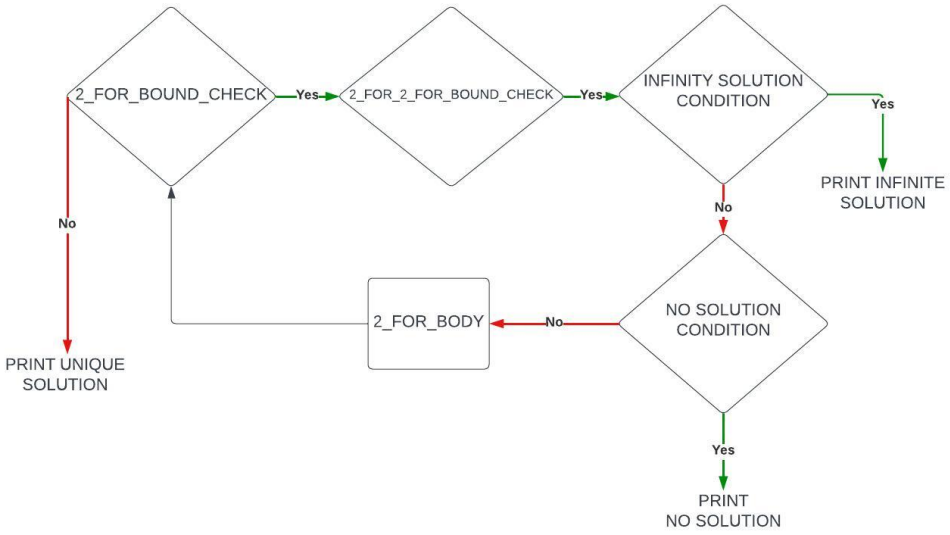

Figure 5.3: Different blocks of back substitution step

6 Testing

6.1 Methodology

- A Random system of linear equations and its solution was generated using the Python script.
- The inputs in the augmented matrix(5*6) in row-major form were given to the RISC V program.
- The output matrix(5*1) 'x' obtained in the hexadecimal form is then fed back to the Python script to compare the accuracy of the result.
- The error matrix and the average error were calculated.

6.2 Inputs, Outputs, and Comparisions

The first system of Linear Equations used for testing:

6.2.1 Unique solution case

$$
\begin{aligned}
-10a + 10b - 6c - 8d + 7e &= -7.8410879478257955 \\
4a + 9b - 7c + 7d + 1e &= 2.80035260405825 \\
-3a + 2b + 4c - 2d - 4e &= -0.39920033076525163 \\
-6a - 3b + 9c + 7d + 10e &= 12.6771619455123 \\
-2a - 6b + 1c - 7d + 4e &= -3.884903400648292
\end{aligned}
$$

Expected output from Python solution:

$$x = [0.385801980.250567610.799842690.594670760.43823971$$

Output from RISC V program(Hex equivalents of floating point):

$$x = [0x3EC587CE, 0x3E804A5F, 0x3F4CC27A, 0x3F183C5A, 0x3EE060F3]$$

Error Matrix(expected output - output from RISC V):

$$error = [2.456728961e{-}07, 2.015665319e{-}07, 2.118831337e{-}07, 1.304219874e{-}07, 4.724146340e{-}08]$$

$$Average\_error = \frac{\sum error\_matrix}{5} = 1.6735720251848104e - 07$$

6.2.2 No Solution case

$$
\begin{aligned}
&5a - 8b - 3c + 7d - 5e = -171\\
&-2a + 7b - 8c + 1d - 10e = 56\\
&0a + 0b + 0c + 0d + 0e = 108\\
&9a + 9b - 8c + 4d - 10e = -35\\
&2a - 10b - 4c - 4d - 10e = -146
\end{aligned}
$$

Output from RISC V program: No solution exists!

6.2.3 Infinite solution case

$$
\begin{aligned}
&-10a + 8b + 10c + 8d + 5e = -49\\
&-7a - 1b - 4c + 10d + 1e = -77\\
&-10a - 9b - 8c + 8d + 3e = -61\\
&-7a - 1b - 4c + 10d + 1e = -77\\
&3a - 8b + 6c + 8d + 6e = 104
\end{aligned}
$$

Output from RISC V program: Infinite Solution exists!