Computer Architecture

Software Requirement Specifications

Software Requirement Specifications (SRS) for Gaussian Elimination using RISC-V Instruction set Architecture

Requirement	Requirement Description	Functionality
RISC V simulator for assembly language code	Microsoft VS-CODE-based locally installed RISC-V simulator for assembly language code development, execution, and debugging	RISC-V Venus Simulator for execution of assembly language code
Input Matrix A and B	Input matrix is hardcoded inside the RISCV assembly language code stored in row major format	Takes two matrices A and B as input as augmented matrix and solve the set of linear equations
Solve for X	The software shall perform Gaussian elimination on the input augmented matrix.	Gaussian elimination algorithm to find the value of X which satisfies the equation AX=B where, X=x1,x2,x3,x4,x5.
Row reduced echelon form	Row reduction performed on the matrix coefficients till the augmented matrix is transformed into a upper triangular matrix	Upper triangular matrix in which all the elements below the diagonal are zero

Requirement	**Requirement Description**	**Functionality**
Partial pivoting (Algorithm Optimization)	Partial pivoting is a technique used to re-order the rows of a matrix It ensures that the largest absolute values in the current column are moved to the diagonal position matrix[i][i].	Partial pivoting improves numerical stability and reduces round-off errors. It can help prevent division by very small numbers and reduce the accumulation of errors in numerical computations.
Singular Case	When the determinant of the matrix is equal to zero. There exist two special cases i.e. there does not exist a unique solution for the system of linear equations	There is total three cases possible while solving a set of linear equations. a. Unique Solution b. No Solution c. Infinitely many solution
Unique Solution	There exists only one solution	X which satisfies the equation: AX=B where, X=x1,x2,x3,x4,x5.
No Solution	Singular matrix i.e., determinant of matrix is zero	The system no solution which can be consistent or inconsistent
Infinitely many Solution	Singular matrix i.e., determinant of matrix is zero	The system has infinitely many solutions.

Requirement	Requirement Description	Functionality
Error percentage	Testing of the assembly code by comparing the value of X with that of C++ program	Error reporting between RISCV Assembly and C++ program
Floating point registers(32 bit)	IEEE 754 32 bit floating point representation	Floating-point support in RISC-V ISA

Design Considerations

- The requirement for solving the set of linear equations is that we have 5 equations with 5 unknowns such that AX=B where A is 5*5 matrix and X is a 5*1 matrix. Our input is A and B and we have to solve for X.
- We have to solve the equation of below type:

  a11x1 + a12x2 + a13x3 + a14x4 + a15x5 = b1

  a21x1 + a22x2 + a23x3 + a24x4 + a25x5 = b2

  a31x1 + a32x2 + a33x3 + a34x4 + a35x5 = b3

  a41x1 + a42x2 + a43x3 + a44x4 + a45x5 = b4

  a51x1 + a52x2 + a53x3 + a54x4 + a55x5 = b5

  Solving for X = x1, x2, x3, x4, x5 that satisfy this system of equations

  • method reduces the system to an upper triangular matrix from which the unknowns are derived by the use of backward substitution method.

  • Assuming a1 ≠ 0, x is eliminated from the second equation by subtracting (a2/a1) times the first equation from the second equation and so on,eliminates x from third equation by subtracting (a3/a1) times the first equation from the third equation.

- The elimination procedure is continued until only one unknown remains in the last equation. After its value is determined, the procedure is stopped. Now, Gauss Elimination in C uses back substitution to get the values of X.

- The IEEE 754 floating-point format to be used is single precision where the format of exponent and mantissa fields are as follows:

Bit	Name	Description
31	Sign (S)	Sign bit (0 for positive, 1 for negative)
30-23	Exponent (E)	Exponent bits (8 bits)
22-0	Mantissa (M)	Mantissa bits (23 bits)

Exponent (E) uses 8 bits and includes a bias of 127, which means the actual exponent value is calculated as E - 127. Mantissa (M) uses 23 bits for representing the fractional part of the number, normalized to have a leading 1 bit.

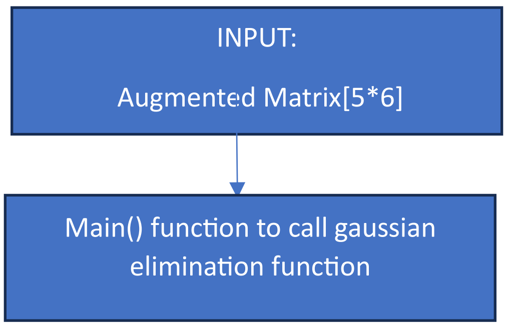
- The numerical algorithm used for solving linear equations with floating-point numbers is Gaussian elimination, also known as row reduction computational procedure, which converts the augmented matrix i.e., the coefficients of matrix A and B together into row reduced echelon form.
- The algorithm is quite efficient but incurs different cases when matrix is singular during computation. The cases are Unique solution (non -singular matrix), No solution and Infinite solution. We must consider all three solutions in our assembly code.
- Error is reported based on RISCV assembly output of matrix X and C++ program for the equation AX=B.
- We have optimized the assembly code for efficiency by taking into account pivoting. Partial pivoting is a technique used to re-order the rows of a matrix during Gaussian elimination to improve numerical stability and reduce round-off errors. It ensures that the largest absolute values in the current column are moved to the diagonal position (matrix[i][i]), which can help prevent division by very small numbers and reduce the accumulation of errors in numerical computations
- To achieve a balance between precision and performance, we have used floating point registers for storing input data, intermediate results, and the final solution. We have assumed that the IEEE 754 floating-point format provides sufficient precision and accuracy for solving linear equations, given the specified input data.
- The output matrix X =x1,x2,x3,x4,x5 is represented in hexadecimal format in both assembly code and C++ program.
- The assembly code is tested on comprehensive test cases to validate the correctness and accuracy of the algorithm implemented. The test cases include boundary cases, extreme values, and randomized inputs.
- You may assume that the chosen numerical algorithm i.e., gaussian elimination for solving linear equations behaves correctly and converges to a solution for the majority of input cases.
- The assembly code depends on the adherence of the hardware to the IEEE 754 standard for floating-point representation.

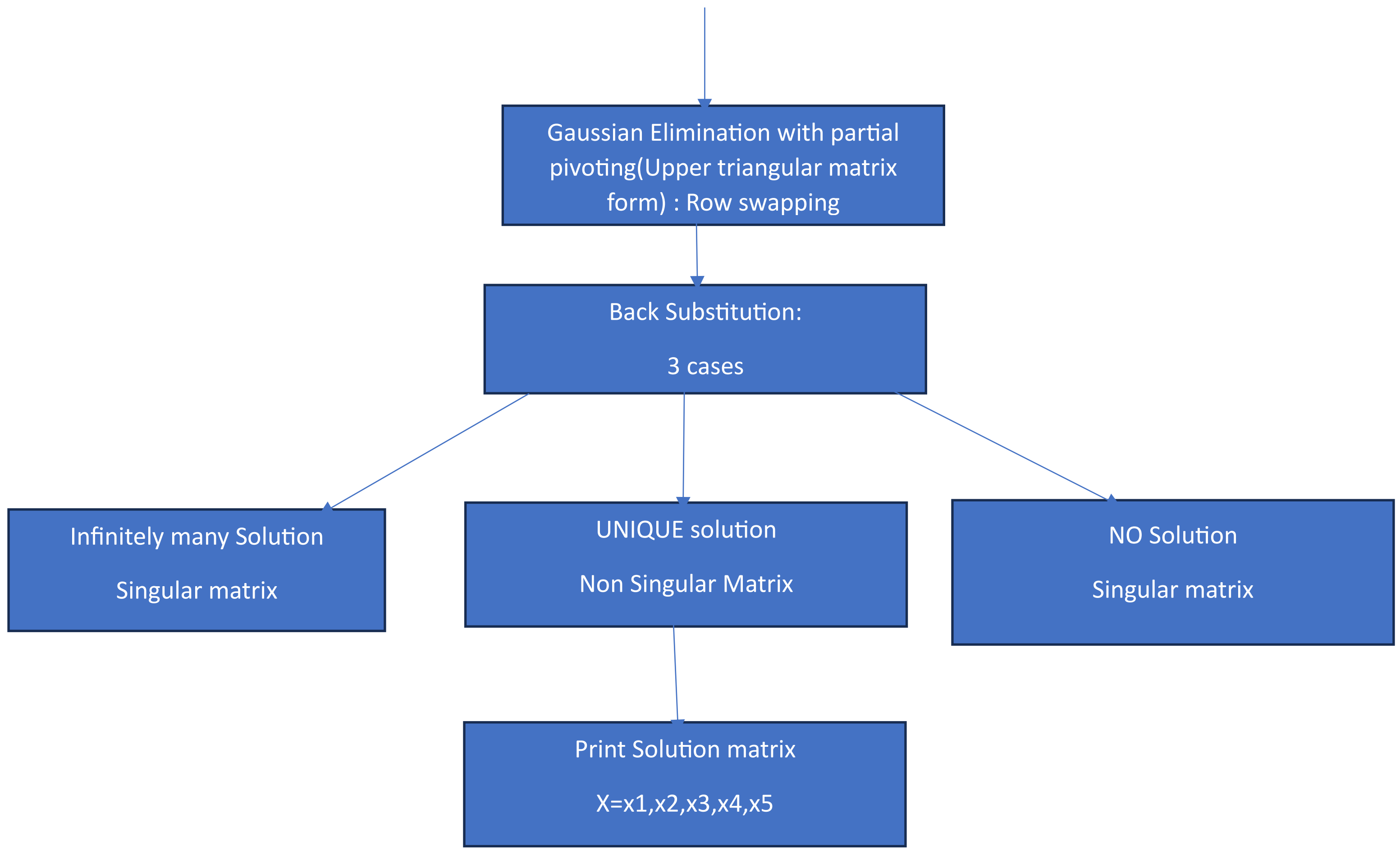
- RISC-V assembly, requires an assembler and a simulator to develop and test your code. We have used RISC-V Venus Simulator and gnu Assembler. We are using Visual Studio Code which is Integrated Development Environment (IDE) that supports RISC-V assembly, with appropriate extensions.
- RISC-V development is platform-independent, versatile and can run on various operating systems.
- Risc V has ‘32’- 32 bit registers.
- RISC-V is called “32-bit architecture” because it operates on 32- bit data.S0-s11 are used to store variables.T0-t6 to hold temporary results. A0-A7 are arguments i.e they are values passed to a function and values returned from a function call.
- RISC V is byte addressable.

Architectural Strategies

- The assembly code is divided into functions so that we don’t have to step over and only call the main function once to get the output on the output terminal. The code is divided into components, each responsible for specific task. This simplifies testing and enhances maintainability.
- We have floating point data representation for representing matrices, vectors, and temporary variables. Efficient data representation minimizes memory usage and maximizes computational efficiency. We have used minimum number of registers only limited to a0-a4, t0-t5, s0-s1 and loop counters and temp variables initialized globally for efficient usage of registers.
- Balancing precision with computational efficiency ensures accurate results without unnecessary overhead. We are using single precision for solving linear equations.
- We have an efficiently managed memory such that we are reusing registers to the optimize memory usage. Proper memory management reduces the risk of memory leaks and enhances overall performance.
- Applied optimization techniques specific to the RISC-V architecture, such as instruction reordering and loop unrolling which enhances code execution speed and efficiency.
- The assembly code includes comprehensive comments explaining the instructions for easy understanding and flow of program.

Software Architecture/ Detailed System Design:





The functionality of the system is divided into different modules:

- Input: Augmented matrix[A|B] with 5 rows and 6 columns. The coefficients are in floating point format.
- Main:We call the function gaussian elimination. Main() function is created to avoid step over the entire program and we get out output matrix X in few step overs.
- Gauss Elimination with partial pivoting: reduces the matrix to upper triangular form. Partial pivoting ensures largest absolute value in each column is in the diagonal position. We perform partial pivoting for each column and for each row below the current row, we check if the diagonal element is zero. We swap the rows, ensuring that the largest absolute value is moved to the diagonal position.
- Back substitution: We start with last row and move upwards having x5 equal to a constant. We subtract the known values of X (calculated in previous iterations) from the current equation to find the value of X[i]. Here, occurs two cases of no solution and infinite solution when matrix is Singular.
- Print matrix X: print the value of the binary representation of matrix[i][j] in hexadecimal format.

Pseudo Code

- Initialization:Initialize a matrix with coefficients and constants. Initialize an array to store the solution and initialize a variable to hold intermediate values.

```
mul t5 , a4 , a3 # t5 = 5*4 * k
add t5 , t5 , a2 # t5 = 5*4 * k + 4 * j
add t5,t5,t3 # effective address =( base + offset)
flw ft1 , 0(t5) # ft1 = matrix[k][j]
fsw ft1, 0(t4) # matrix[i][j] = ft1 = matrix[k][j]
fsw ft0, 0(t5) # matrix[k][j] = ft0
```

- Display Function: created to display the binary representation of matrix elements in hexadecimal format.

```
Display:
li a0,4 #ecall ID 4, which corresponds to print strings.
la a1,msg_x # address of s should be in a1
ecall
li a0,1 #Loads 1 into register a0, indicating that you want to print an integer.
mv a1,t1     #Loads the integer value at the address t5 into register a1.
ecall
```

- Back Substitution Function: created a function to find the solution to linear equations using back substituting the values of x. We start from the last equation and work upward. Calculate each variable (X) using known values of other variables.Here we also, check for cases where infinitely many solutions or no solutions exist.

```
flw ft0,0(t3)
flw ft1,0(a2)
slli a2,a1,2 # i * 4
fmul.s ft0,ft0,ft1 #matrix[i][j] * X[j]

add t4,t4,a2 # X + i * 4
flw ft1,0(t4) #
fsub.s ft1,ft1,ft0 # X[i] - matrix[i][j]*X[j]
fsw ft1,0(t4) # X[i] - matrix[i][j]*X[j]
```

- Swap Function: created a function to swap two rows in the matrix, used for partial pivoting.
- Partial Pivoting Function: created a function to perform partial pivoting during Gaussian elimination. We iterate through rows below the current row to find the largest value in the current column. Swap rows to place the largest value in the diagonal position. Eliminate elements below the diagonal to transform the matrix into upper triangular form.

```
#we compare ft1 and ft2 for partial pivoting and row swapping
operation
flt.s a3,ft1,ft2 # a3 = bool( ft1 < ft2)
xori a3,a3,1 # If a3 is T then result is F, else if a3 is F, then
result is T
bnez a3,label5 # branch if ft1 > ft2, i.e no swapping needed
# otherwise do row swapping
sw x0,4(a0) # initializing j = 0 for next loop
```

- Gaussian Elimination Function: we have base address of matrix and loop counters, we initialize the loop counters check the opposite case of high level code i.e branch if i >= N – 1. We initialize the loop counters and calculate the effective address by calculating offset and adding it to base address to access that element of 2D matrix. The matrix is stored in row major format.

```
• add t4 , t4 , a2 # t4 = 5*4 * i + 4 * j
• add t4, t4, t3 # effective address = (base + offset)
• flw ft0 , 0(t4) # ft0 = matrix[i][j]
• mul t5 , a4 , a3 # t5 = 5*4 * k
• add t5 , t5 , a2 # t5 = 5*4 * k + 4 * j
• add t5,t5,t3 # effective address =( base + offset)
```

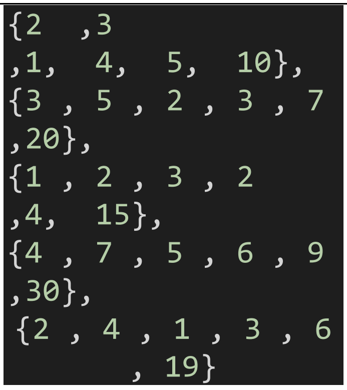
Testing:

C++ program used as a testing method to report error %.

Inputs	Expected Correct Output from C++	Actual Output Obtained	Avg Error %

`{-2, 4, 6, -9, 3, -50},`
`{-8, 6, -4, 4, -5, 11},`
`{-3, -5, 1, -7, 9, -29},`
`{2,3, -6, -5, 17, -3},`
`{-7, -9, -10, 5, 9, 12}`
	Solution to linear set of equations X=x1,x2,x3,x4,x5
	
	x[0] = 40a36860
	
	x[1] = c012dc84
	
	x[2] = c174e59f
		Unique Solution
		
		Solution for x1= 0x40A36862Solution for x2= 0xC012DC8F
		
		Solution for x3= 0xC174E5A8
			3.4

	x[3] = c11ad377
	
	x[4] = c109ed87
	
	PS C:\Users\anany>
		Solution for x4= 0xC11AD380
		
		Solution for x5= 0xC109ED8D	

 {2 ,3 ,1, 4, 5, 10},
{3 , 5 , 2 , 3 , 7 ,20},
{1 , 2 , 3 , 2 ,4, 15},
{4 , 7 , 5 , 6 , 9 ,30},
{2 , 4 , 1 , 3 , 6 , 19}
	Solution to linear set of equations X=x1,x2,x3,x4,x5
	
	x[0] = c10c0000
	
	x[1] = 40c40000
	
	x[2] = 3fc00000
	
	x[3] = bf400001
	
	x[4] = 40080000
		Unique Solution
		
		Solution for x1= 0xC10C0000
		
		Solution for x2= 0x40C40000
		
		Solution for x3= 0x3FC00000
		
		Solution for x4= 0xBF400000
		
		Solution for x5= 0x40080000
			0.2

{1,1,1,1,1,1}, {1,1,1,1,1,1}, {1,1,1,1,1,1}, {1,1,1,1,1,1}, {1,1,1,1,1,1}	Infinitely many solutions exist PS C:\Users\anany>	Singular matrix Infinitely many solution exists	