TRANSLATION TO TARGET CODE

Your task is to convert this code into assembly code.

a)

```
#include <vector>
float matrix_vector_mult(const std::vector<std::vector<float>>& matrix, const
std::vector<float>& vector) {
    if (matrix.size() != 4 || matrix[0].size() != 4 || vector.size() != 4) return 0.0f;
    float result[4] = {0};
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
    }
    return result[0] + result[1] + result[2] + result[3];
}
```

b)

```
import numpy as np
def compute_exp_threshold(arr):
    result = np.zeros_like(arr)
    for i in range(len(arr)):
        result[i] = min(np.exp(arr[i]), 100.0)
    return result
```

Memory Management

a) Analyze memory management (stack, heap, static)

```
def factorial_with_cache(n, cache=None):
    if cache is None:
        cache = [-1] * (n + 1)  # Initialize cache with -1
    if n < 0:
        return 0
    if n <= 1:
        return 1
    if cache[n] != -1:
        return cache[n]
    cache[n] = n * factorial_with_cache(n - 1, cache)

    return cache[n]
```

b) Analyze memory management (stack, heap, static)

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
def tree_depth(node):
    if node is None:
        return 0
    left_depth = tree_depth(node.left)
    right_depth = tree_depth(node.right)
    return max(left_depth, right_depth) + 1
```

Three Address Code , Control Flow Graph, Quadruples and Tuples

a)

```
#include <vector>
float matrix_vector_mult(const std::vector<std::vector<float>>& matrix, const
std::vector<float>& vector) {
    if (matrix.size() != 4 || matrix[0].size() != 4 || vector.size() != 4) return 0.0f;
    float result[4] = {0};
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
    }
    return result[0] + result[1] + result[2] + result[3];
}
```

b)

```
import numpy as np
def compute_exp_threshold(arr):
    result = np.zeros_like(arr)
    for i in range(len(arr)):
        result[i] = min(np.exp(arr[i]), 100.0)
    return result
```

DAG .. IDENTIFY IMPORTANT INSTRUCTIONS FROM THE GIVEN CODE = AND THEN GENERATE DIRECTED ACYCLIC GRAPHS

A)
```
#include <vector>

void transpose_matrix(std::vector<std::vector<float>>& matrix, int n, int row = 0) {
    if (row >= n) return; // Base case: stop when row exceeds matrix size
    for (int j = row + 1; j < n; j++) {
        // Swap matrix[row][j] with matrix[j][row]
        float temp = matrix[row][j];
        matrix[row][j] = matrix[j][row];
        Matrix
```

B)
```
#include <vector>

float determinant(std::vector<std::vector<float>>& matrix, int n) {
    if (n <= 0) return 0.0f;
    if (n == 1) return matrix[0][0];
    if (n == 2) return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];

    float det = 0.0f;
    for (int j = 0; j < n; j++) {
        std::vector<std::vector<float>> submatrix(n - 1, std::vector<float>(n - 1));
        for (int i = 1; i < n; i++) {
            for (int k = 0, col = 0; k < n; k++) {
                if (k == j) continue;
                submatrix[i - 1][col++] = matrix[i][k];
            }
        }
        det += (j % 2 == 0 ? 1 : -1) * matrix[0][j] * determinant(submatrix, n - 1);
    }
    return det;
}
```