EHB326E – INTRODUCTION TO EMBEDDED SYSTEMS
HW2

Instructor: Prof. Dr. Müştak Erhan Yalçın
Assistant: Alp Eren Kıyak

Kamil Eren Ezen
040210021

Problem 1551 Definition:

The problem I have to solve for my software part is: You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise). You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

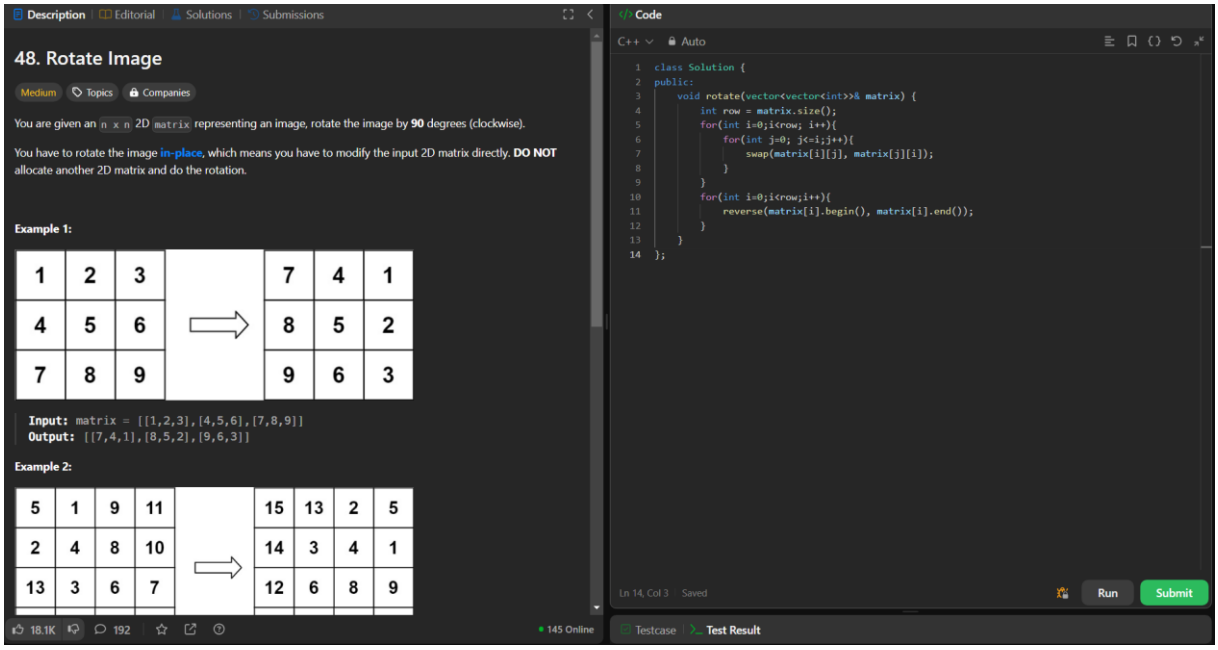


Figure 1: Leetcode problem and the solution

Approach of my solution is to take the transpose of the matrix first then swapping the columns. Tricky part was to convert this code to assembly-style.

Assembly Solution

Part 1: Creating the matrix

I wanted provide the inputs through directly in code itself. So, in the first part of the code, I created the matrix in scratchpad sequentially using LOAD and WRMEM instructions.

```
; Constants
LOAD sF, 03         ; Matrix size nxn (n = 3)

START:
    ; Row 1: [1, 2, 3]
    LOAD s0, 01
    WRMEM s0, 00
    LOAD s0, 02
    WRMEM s0, 01
    LOAD s0, 03
    WRMEM s0, 02

    ; Row 2: [4, 5, 6]
    LOAD s0, 04
    WRMEM s0, 03
    LOAD s0, 05
    WRMEM s0, 04
    LOAD s0, 06
    WRMEM s0, 05

    ; Row 3: [7, 8, 9]
    LOAD s0, 07
    WRMEM s0, 06
    LOAD s0, 08
    WRMEM s0, 07
    LOAD s0, 09
    WRMEM s0, 08
```

Part 2: Taking Transpose of the Matrix

Then took the transpose of the matrix by swapping elements above and below the main diagonal.

Outer Loop (`Transpose_Loop_Outer`):

- This loop iterates through the rows of the matrix using the index i.

Inner Loop (`Transpose_Loop_Inner`):

- This loop iterates through the columns of the matrix using the index j.
    - Address Calculation:
        - `addr1 = i * n + j`: Calculates the address of the element at row `i` and column `j`.
        - `addr2 = j * n + i`: Calculates the address of the element at row `j` and column `i`.
    - Value Swap:
        - Values at these addresses are swapped if `j > i`. This ensures that swaps only occur for the upper triangle of the matrix.

```
    ; Transpose of the matrix
    LOAD s1, 00       ; i = 0
Transpose_Loop_Outer:
    LOAD s2, 00       ; j = 0
Transpose_Loop_Inner:

    ; Calculation of addresses: addr1 = i*n + j, addr2 = j*n + i
    LOAD s3, s1       ; addr1 = i * n
    SL0 s3            ; Multiply i by 2 (shift left)
    ADD s3, s1        ; addr1 = i * 3 (for n=3)
    ADD s3, s2        ; addr1 += j

    LOAD s4, s2       ; addr2 = j * n
    SL0 s4            ; Multiply j by 2
    ADD s4, s2        ; addr2 = j * 3
    ADD s4, s1        ; addr2 += i

    ; Read values from Scratchpad
    RDMEM s5, (s3)      ; s5 = matrix[i][j]
    RDMEM s6, (s4)      ; s6 = matrix[j][i]

    ; Swap values if j > i
    COMP s2, s1
    JUMP NC, Skip_Swap ; Skip swap if j <= i

    ; Perform the swap
    WRMEM s6, (s3)      ; matrix[i][j] = matrix[j][i]
    WRMEM s5, (s4)      ; matrix[j][i] = matrix[i][j]

Skip_Swap:
    ; Increment j
    ADD s2, 01
    COMP s2, sF       ; Check if j < n
    JUMP C, Transpose_Loop_Inner

    ; Increment i
    ADD s1, 01
    COMP s1, sF       ; Check if i < n
    JUMP C, Transpose_Loop_Outer
```

Part 3: Column Swapping

After taking the transpose of the matrix we need to swap the left-most columns with right-most columns, then progress inward. If n is odd for nxn matrix then the middle column stays put.

Outer Loop (`Column_Swap_Outer`):

- Iterates through the columns, swapping the `col_left` (starting at 0) with `col_right` (starting at `n-1`).

Inner Loop (`Column_Swap_Inner`):

- For each row (`i`), the addresses of the elements in the left and right columns are calculated:
  - `addr_left = i * n + col_left`: The address of the element in the left column.
  - `addr_right = i * n + col_right`: The address of the element in the right column.
- The values at these addresses are swapped.
- After completing the swaps for a row, the `col_left` index is incremented, and the `col_right` index is decremented.

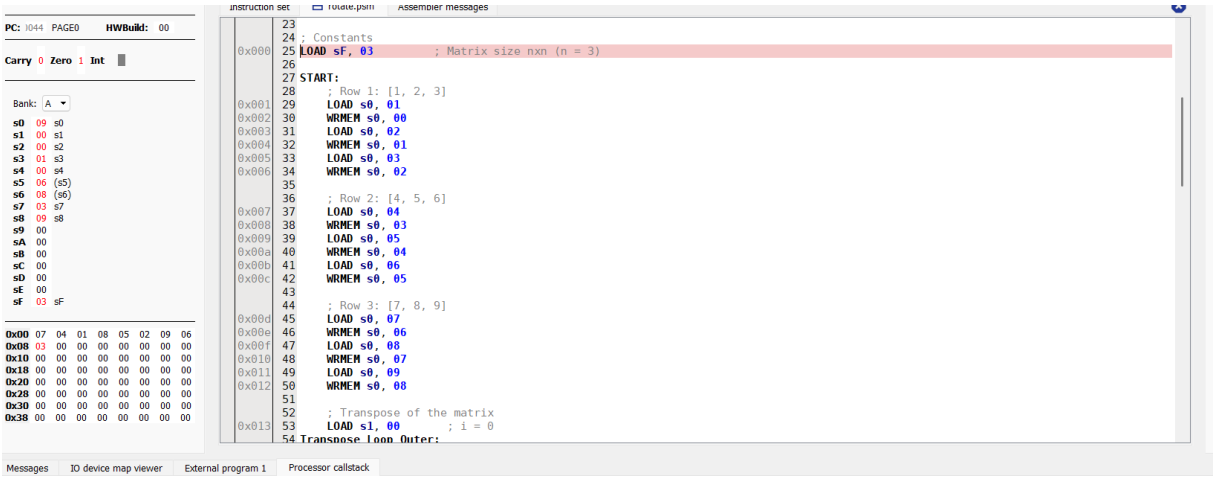
Process until the `col_left` index is no longer less than the `col_right` index, which means they met in the middle.

```
    ; Column Swapping
    LOAD s1, 00       ; i = 0 (row index)
    LOAD s2, 00       ; col_left = 0
    LOAD s3, sF       ; col_right = n - 1
    SUB s3, 01
Column_Swap_Outer:
    LOAD s4, s1       ; j = 0 (row iterator for column swapping)
Column_Swap_Inner:
    ; Adress calculation for left and right columns
    LOAD s5, s1       ; addr_left = i * n + col_left
    SL0 s5
    ADD s5, s1
    ADD s5, s2

    LOAD s6, s1       ; addr_right = i * n + col_right
    SL0 s6
    ADD s6, s1
    ADD s6, s3

    ; Swap values in left and right columns
    RDMEM s7, (s5)      ; s7 = matrix[i][col_left]
    RDMEM s8, (s6)      ; s8 = matrix[i][col_right]
    WRMEM s8, (s5)      ; matrix[i][col_left] = matrix[i][col_right]
    WRMEM s7, (s6)      ; matrix[i][col_right] = matrix[i][col_left]

    ; Increment row index
    ADD s1, 01
    COMP s1, sF       ; Check if row index < n
    JUMP C, Column_Swap_Inner

    ; Reset row index and move to next column pair
    LOAD s1, 00
    ADD s2, 01        ; col_left++
    SUB s3, 01        ; col_right--
    COMP s2, s3       ; Check if col_left < col_right
    JUMP C, Column_Swap_Outer
```

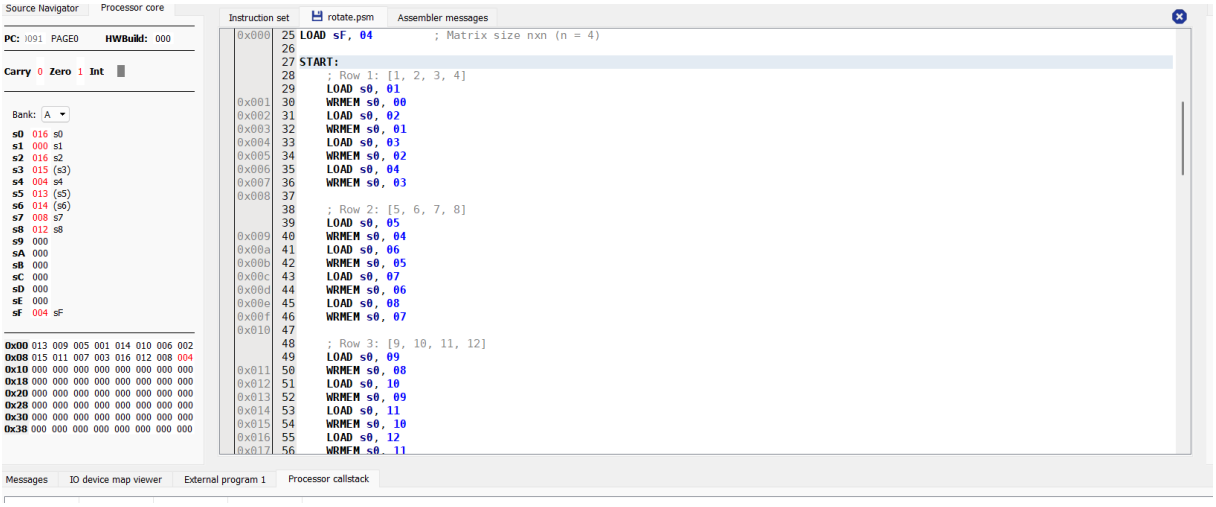
After this operation the result is the matrix rotated 90 degrees clockwise.

Different Scenarios

n=3



n=4



n=5