



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 $n \times n$ 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.

The screenshot shows the LeetCode interface for problem 48, "Rotate Image". The problem description states: "You are given an $n \times n$ 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."

Example 1:

1	2	3		7	4	1
4	5	6	→	8	5	2
7	8	9		9	6	3

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:

5	1	9	11		15	13	2	5
2	4	8	10	→	14	3	4	1
13	3	6	7		12	6	8	9

The C++ solution code is shown on the right:

```
1 class Solution {
2 public:
3     void rotate(vector<vector<int>>& matrix) {
4         int row = matrix.size();
5         for(int i=0; i<row; i++){
6             for(int j=0; j<i; j++){
7                 swap(matrix[i][j], matrix[j][i]);
8             }
9         }
10        for(int i=0; i<row; i++){
11            reverse(matrix[i].begin(), matrix[i].end());
12        }
13    }
14 }
```

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:
 - $\text{addr1} = i * n + j$: Calculates the address of the element at row i and column j .
 - $\text{addr2} = j * n + i$: 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 $n \times n$ 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:
; Address 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

PC: 0044 PAGE0 HWBuild: 00

Carry 0 Zero 1 Int

Bank: A

s0 09 s0
s1 00 s1
s2 00 s2
s3 01 s3
s4 00 s4
s5 06 (s5)
s6 08 (s6)
s7 03 s7
s8 09 s8
s9 00
sA 00
sB 00
sC 00
sD 00
sE 00
sF 03 sF

0x00 07 04 01 08 05 02 09 06
0x08 03 00 00 00 00 00 00 00
0x10 00 00 00 00 00 00 00 00
0x18 00 00 00 00 00 00 00 00
0x20 00 00 00 00 00 00 00 00
0x28 00 00 00 00 00 00 00 00
0x30 00 00 00 00 00 00 00 00
0x38 00 00 00 00 00 00 00 00

Instruction set rotate.psm Assembly messages

```
23  
24 ; Constants  
25 LOAD sF, 03 ; Matrix size nxn (n = 3)  
26  
27 START:  
28 ; Row 1: [1, 2, 3]  
29 LOAD s0, 01  
30 WRMEM s0, 00  
31 LOAD s0, 02  
32 WRMEM s0, 01  
33 LOAD s0, 03  
34 WRMEM s0, 02  
35  
36 ; Row 2: [4, 5, 6]  
37 LOAD s0, 04  
38 WRMEM s0, 03  
39 LOAD s0, 05  
40 WRMEM s0, 04  
41 LOAD s0, 06  
42 WRMEM s0, 05  
43  
44 ; Row 3: [7, 8, 9]  
45 LOAD s0, 07  
46 WRMEM s0, 06  
47 LOAD s0, 08  
48 WRMEM s0, 07  
49 LOAD s0, 09  
50 WRMEM s0, 08  
51  
52 ; Transpose of the matrix  
53 LOAD s1, 00 ; i = 0  
54 Transpose Loop Outer:
```

n=4

Source Navigator Processor core

PC: 0051 PAGE0 HWBuild: 000

Carry 0 Zero 1 Int

Bank: A

s0 016 s0
s1 000 s1
s2 016 s2
s3 015 (s3)
s4 004 s4
s5 013 (s5)
s6 014 (s6)
s7 008 s7
s8 012 s8
s9 000
sA 000
sB 000
sC 000
sD 000
sE 000
sF 004 sF

0x00 013 009 005 001 014 010 006 002
0x08 015 011 007 003 016 012 008 004
0x10 000 000 000 000 000 000 000 000
0x18 000 000 000 000 000 000 000 000
0x20 000 000 000 000 000 000 000 000
0x28 000 000 000 000 000 000 000 000
0x30 000 000 000 000 000 000 000 000
0x38 000 000 000 000 000 000 000 000

Instruction set rotate.psm Assembly messages

```
25 LOAD sF, 04 ; Matrix size nxn (n = 4)  
26  
27 START:  
28 ; Row 1: [1, 2, 3, 4]  
29 LOAD s0, 01  
30 WRMEM s0, 00  
31 LOAD s0, 02  
32 WRMEM s0, 01  
33 LOAD s0, 03  
34 WRMEM s0, 02  
35 LOAD s0, 04  
36 WRMEM s0, 03  
37  
38 ; Row 2: [5, 6, 7, 8]  
39 LOAD s0, 05  
40 WRMEM s0, 04  
41 LOAD s0, 06  
42 WRMEM s0, 05  
43 LOAD s0, 07  
44 WRMEM s0, 06  
45 LOAD s0, 08  
46 WRMEM s0, 07  
47  
48 ; Row 3: [9, 10, 11, 12]  
49 LOAD s0, 09  
50 WRMEM s0, 08  
51 LOAD s0, 10  
52 WRMEM s0, 09  
53 LOAD s0, 11  
54 WRMEM s0, 10  
55 LOAD s0, 12  
56 WRMEM s0, 11
```

n=5

Source Navigator Processor core

PC: 0114 PAGE0 HWBuild: 000

Carry 0 Zero 1 Int

Bank: A

s0 025 s0
s1 000 s1
s2 025 s2
s3 024 (s3)
s4 005 s4
s5 021 (s5)
s6 023 (s6)
s7 010 s7
s8 020 s8
s9 000
sA 000
sB 000
sC 000
sD 000
sE 000
sF 005 sF

0x00 021 016 011 006 001 022 017 012
0x08 007 002 023 018 013 008 003 024
0x10 019 014 009 004 025 020 015 010
0x18 005 000 000 000 000 000 000 000
0x20 000 000 000 000 000 000 000 000
0x28 000 000 000 000 000 000 000 000
0x30 000 000 000 000 000 000 000 000
0x38 000 000 000 000 000 000 000 000

Instruction set rotate.psm Assembly messages

```
25 LOAD sF, 05 ; Matrix size nxn (n = 5).  
26  
27 START:  
28 ; Row 1: [1, 2, 3, 4, 5]  
29 LOAD s0, 01  
30 WRMEM s0, 00  
31 LOAD s0, 02  
32 WRMEM s0, 01  
33 LOAD s0, 03  
34 WRMEM s0, 02  
35 LOAD s0, 04  
36 WRMEM s0, 03  
37 LOAD s0, 05  
38 WRMEM s0, 04  
39  
40 ; Row 2: [6, 7, 8, 9, 10]  
41 LOAD s0, 06  
42 WRMEM s0, 05  
43 LOAD s0, 07  
44 WRMEM s0, 06  
45 LOAD s0, 08  
46 WRMEM s0, 07  
47 LOAD s0, 09  
48 WRMEM s0, 08  
49 LOAD s0, 10  
50 WRMEM s0, 09  
51  
52 ; Row 3: [11, 12, 13, 14, 15]  
53 LOAD s0, 11  
54 WRMEM s0, 10  
55 LOAD s0, 12  
56 WRMEM s0, 11
```