**PROJECT: TRICKY MATRIX PROBLEM**

* **Requirements and analysis of the problem**

1. Input is either a uint16\_t datatype or a uint64\_t.
2. uint16\_t is a 16-bit unsigned int which represents a 4x4 matrix. So, we can figure out that each nibble represents a row of the matrix where columns left to right of the matrix is viewed as right to left of a nibble (According to the demonstration in the question). The first(0th) row is the least significant nibble of the integer, the second(1th) row is the second least significant nibble, and so on till the most significant nibble((n-1)th).
3. Similarly, uint64\_t is a 64-bit unsigned int which represents an 8x8 matrix. Here, each byte (instead of nibble) represents a row and respectively column in a similar fashion as uint16\_t.
4. Our goal is to generate a mirror image representation and a counter clock-wise rotated representation of the input integer representing a matrix.

* **Formulating solution**

1. *Case 1: Producing mirror image of a matrix.*
2. Let us first consider the uint16\_t type input.
3. Let us consider an input, say, uint16\_t a = 0x124f (Given in the question).
4. Here the nibble ‘f’ represents the 0th row. 0th column is the 0th bit and the (n-1)th column is the 3rd bit of the nibble ‘f’. Hence, the 0th row will be 1111. Similarly, the next row is the nibble ‘4’ and the row will be 0010(right-to-left of 0100).
5. To get the mirror image we could reverse each nibble of the uint16\_t. Example: uint16\_t a = 0x124f is equal to binary 0001001001001111. After reversing each nibble, we get 1000010000101111, which is 0x842f.
6. So we need a module that will reverse each nibble of a uint16\_t type data.
7. Similarly, for uint64\_t datatype we can reverse each byte of the 64-bit word.
8. Below is the algorithm for reversing each nibble of uint16\_t as well as reversing each byte of uint64\_t datatype.
9. *Case 2: Producing rotated matrix pattern*
10. Note that counter clockwise rotation is equivalent to matrix transpose.
11. Let us first consider the uint16\_t type input.
12. Let us consider an input, say, uint16\_t a = 0x124f (Given in the question).
13. Binary of 0x124f = 0001 0010 0100 1111. After rotating a = 0x9531 whose binary representation is 1001 0101 0011 0001.
14. From the change in bit pattern we can observe the following:
    * Each bit of 1st nibble (right most) starting from left to right must be placed in the 0th bit position of each nibble from right to left.
    * Each bit of 2nd nibble starting from left to right must be placed in the 1th bit position of each nibble from right to left.
    * Similarly, for 3rd and 4th nibble.
15. We can perform similar operations for uint64\_t with only the difference in processing byte wise rather than nibble wise.
16. Below are the algorithms for generating anti-clockwise rotation for both uint16\_t and uint64\_t datatypes.

* **Algorithm**

1. Algorithms for generating mirror image.

mirror16(uint16\_t n): [Returns mirror image of uint16\_t]

i = 0

while i < 4:

// Get each nibble starting from right most

nibble = (num >> (i \* 4)) & 0x000f

// Reverse the extracted nibble

reverse\_the\_nibble(nibble)

// Push it to the resultant variable

result |= (nibble & 0x0f) << (i \* 4)

return result

mirror64(uint64\_t n): [Returns mirror image of uint64\_t]

i = 0

while i < 8:

// Get each byte starting from right most

byte = (num >> (i \* 8)) & 0x00ff

// Reverse the extracted byte

reverse\_the\_byte(byte)

// Push it to the resultant variable

result |= (byte & 0x00ff) << (i \* 8)

return result

1. Algorithms for generating counter clockwise rotated image

rotate16(uint16\_t n): [Returns counter clockwise rotated image of uint16\_t]

i = 0

bit\_pos\_mask = 0x01

while i < 4:

// Get each nibble starting from right most

nibble = (num >> (i \* 4)) & 0x000f

i = 3

while i >= 0:

if nibble & (1 << i):

ret\_val |= bit\_pos\_mask << ((3 - i) \* 4); // Set bit

else:

ret\_val &= ~(bit\_pos\_mask << ((3 - i) \* 4)); // Clear bit

i = i - 1

bit\_pos\_mask <<= 1

return result

rotate64(uint64\_t n): [Returns counter clockwise rotated image of uint64\_t]

i = 0

bit\_pos\_mask = 0x01

while i < 8:

// Get each byte starting from right most

byte = (num >> (i \* 8)) & 0x00ff

i = 7

while i >= 0:

if byte & (1 << i):

ret\_val |= bit\_pos\_mask << ((7 - i) \* 8); // Set bit

else:

ret\_val &= ~(bit\_pos\_mask << ((7 - i) \* 8)); // Clear bit

i = i - 1

bit\_pos\_mask <<= 1

return result

* **Implementation Details**

1. **Development Tools Used**
   * Arduino IDE
2. **Hardware Used for testing the code**
   * Arduino pro mini: ATMega328p
3. **Operating System**
   * Windows 10 x64

* **Coding Approach**

1. There are four functions for performing each tasks.
   1. uint16\_t mirror16(uint16\_t num): Returns a mirror image of uint16\_t data.
   2. uint64\_t mirror64(uint64\_t num): Returns a mirror image of uint64\_t data.
   3. uint16\_t rotate16(uint16\_t num): Returns counter-clockwise rotated image of uint16\_t data.
   4. Uint64\_t rotate64(uint64\_t num): Returns counter-clockwise rotated image of uint64\_t data.
2. There is a utility function called printHex64(uint64\_t val). This function is used to print a 64-bit integer/uint64\_t. There was no proper standard APIs available to print uint64\_t in hex so I had to create one.
3. The entire test code runs once in the void setup() function only.