Introduction

At first I started researching how I would do the implementation of the normal n queens using the bitwise operations. I started writing a c++ code on visual studio code for the implementation of the n queens only. I found a code on the internet that opened up for me the general logic for doing the tree implementation along with the java script code. Three functions, the main, the fill, the solve. The main retrieves the n and calls fill. Fill stores the user inputs into the three arrays col, d1, and d2 with specific offsets.

MIPS Problem

In doing the mips I wanted to make the program as flexible as much as possible. I used the stack for recursion and to allow more dynamic data allocation. There is a label called fill which takes the user input which is either 0 or 2, if it is 0 then it would be stored in the three arrays col, d1, and d2 with their appropriate offsets. If the value was 0 then the value is stored normally, if it is 2 then the index is stored such that it would be used in solve. The same offsets as the offsets in c++. After filling the arrays I call the solve which goes through the values in the arrays with their appropriate offsets. When we go through the data in the three arrays I check if it is either 0 or 2, if it is 2 it goes to the checkLizard label which does the logic of adding the tree or not. Bitwise operations are done using and, or, nor, and not. When applying the tree the srl instruction is also used. **THERE ARE MORE SCREENSHOTS OF THE INPUTS, BELOW IS THE OUTPUT ONLY.**

```
234  store_one:
235  li $a2, 1
236  sw $a2, col($s0)
237  sw $a2, d1($s0)
238  sw $a2, d2($s0)
239  j store_diag    #then continue with store diag
240
241
242
243  #this fixes things before going on and checking for another solutions
244  return:
245  li $a1, 2
246  beq $t1, $a1, lizardLogic       #checking if t1 is equal to 2 then we will preform the lizard logic again
247  #this is the loop to retreive the data from the three arrays
248  #these data will be restored for the next solution
249  loop4:
250  lw $t1, col($t0)        #loading the value in col with offset of t0 in t1
251  #getting the copies of t6 and t0
252  move $t7, $t6
253  move $t8, $t0
254  #same logic as the fill and solve
255  sub $t6, $t6, $t0
256  add $t5, $t6, $t5
257  sub $t5, $t5, $t4
```

Line: 22 Column: 2 Show Line Numbers

Mars Messages Run I/O

Clear

```
0
0
Next Row: 0
0
0
0

0 1 2 1
0 0 0 0
0 2 1 0
1 0 0 0

2
-- program is finished running (dropped off bottom) --
```

```
10          firstrow: .asciiz "First Row: "
11          rows: .asciiz "Next Row: "
12  .text
13
14          #print the enter the number of n
15          li $v0, 4
16          la $a0, ask
17          syscall
18
19          #read from the user the n
20          li $v0, 5
21          syscall
22
23          #preparing before going to the function
24          move $t1, $v0   #n is stored in t1
25          move $t2, $zero   #the number of solutions is stored in t2
```

Line: 1 Column: 1 Show Line Numbers

Mars Messages Run I/O

Clear

```
0
0

0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 2 0 0 1
0 0 0 0 0 0 0 0
0 0 1 0 0 2 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0

92
-- program is finished running (dropped off bottom) --
```

RISC V Problem

In doing the risc part, I took the code of mips and pasted it in the rars simulator. I removed the z in asciiz and added in each string a \0 at the end. I removed all move instructions and made them add and added zero at the end. I also removed all $ signs. I replaced the syscall with ecall. There were some registers in mips used that were not in risc so I replaced the t7-t9 with s8-s10. Immediate are also carried out using the s11 register. I also replaced the v0 with a7 for system call. This made it work normally. **THERE ARE MORE SCREENSHOTS OF THE INPUTS, BELOW IS THE OUTPUT ONLY.**

```
13
14          #print the enter the number of n
15          li a7, 4
16          la a0, ask
17          ecall
18
19          #read from the user the n
20          li a7, 5
21          ecall
22
23          #preparing before going to the function
24          add t1, a7, zero        #n is stored in t1
25          add t2, zero, zero   #the number of solutions is stored in t2
26          add t0, zero, zero  #the r
27
28
29          #storing them in the stack
30          li s11, 12
31          sub sp, sp, s11
```

Line: 11 Column: 28 Show Line Numbers

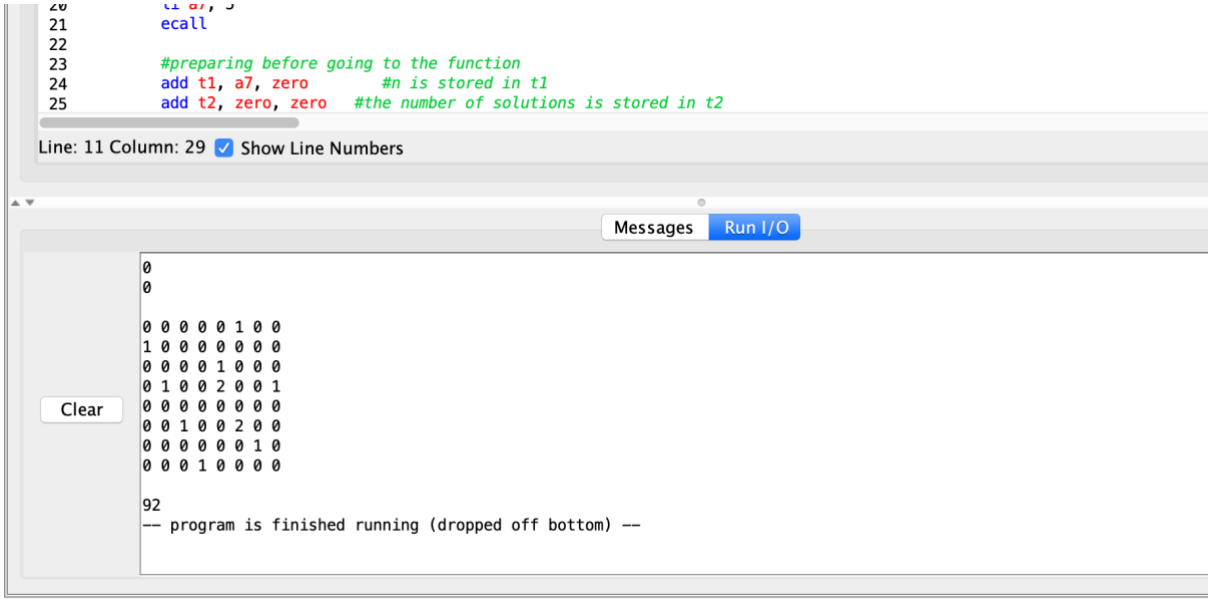
Messages Run I/O

Clear

```
Next Row: 0
2
0
0
Next Row: 0
0
0
0

0 1 2 1
0 0 0 0
0 2 1 0
1 0 0 0

2
-- program is finished running (dropped off bottom) --
```



X86-64 Problem

In this code I was doing something very similar to the mips. However, I did the code based on an array already written such that I parse the data from it instead of user input data. I perform a semi bitwise operation, in which I use the same operations in the mips and risc such as and, or, not, and nor in the 1's and the 2's. I have the solve subroutine that does the operation of solving the board with bitwise operations. I also do the 2 implementation which is the tree using the same idea in mips as well. I performed the and between the col, d1, and d2 with their offsets and then not each one of them and then oring all of them. After that I not all the result and check if it is zero or not. If it is zero then I can put a tree here. The value of it being one or zero is dependent on the checkLizard label that does most of the logic to determing whether or not it shall be 0 or 1 to satisfy the logic. The logic is done by literally anding the results and then noting it. I take the position which is the col or with d1 & d2. This position if it contains a one in the subsequent column then it will be a one, otherwise it shall be 0. I assumed that the user to make this work will have to change the value of n and the values in the array to get their desired output. **It is important to know that I overwrite the content of the 'user input' to be the result, because I was not able to convert it to another array, however, it made the code a lot shorter.**