Q4.

This code refers to the famous problem 'TOWERS OF HANOI', and it uses an iterative approach. It describes the various steps in moving all discs from one rod to another, while ensuring that their order isn't disturbed (Disc with the smaller size cannot be placed below a disc with a larger size). Hence, we have 3 rods for this purpose, namely: Source, Auxiliary, Destination. Hence, we take the help of auxiliary to move all the discs from source to destination, ensuring that the order isn't disturbed.

CODE:

_start:
In this function, we initialize the number of discs we are dealing with and storing it in register %rdi, here. In this case, number 3 is stored. Otherwise, we do double-dereferencing of the particular position of stack pointer, and store the corresponding argument in%rdi.Here , we call the function 'solve'.

solve:
In this function,we initialize all the three rods we are going to work with. We initialize them to be empty and allocate 64 bytes of memory for each rod. (We take help of f1 to do this). Also, we first store the current size of a rod in the bottom of a rod, (hence we initially store 3,that has been passed as an argument to this function (here) at the base of source).

f1:
In this function, we initialize the three rods to be empty, by inserting the value zero , for every 8 bytes, in the 64 bytes allocated for a single row.
The value zero at a particular position means that nothing is present there.

. init_s:
We set the values of the source, by using a loop.
After we inserted the size of the rod (total number of discs which we are about to hold), we insert the largest number 3, above the size, followed by 2 and 1,one placed on top of another.
We also initialize the looping variable and the limit value for the loop in registers %r15 (initialized with 0) and %r14 (has the value (2 power n)-1, i.e, 2^3-1=7, in this case).

f7:
This function acts as the main loop, in which we process every iteration. As a first step to achieve our target, we move the source and destination as first and second arguments respectively, and call function f5 (in which we move the required discs, based on the order of arguments). After doing so, we check for the looping condition (i<limit), and terminate the process if it is not met. Otherwise, we repeat the process for moving required discs from source to auxiliary, and then finally from auxiliary to destination (by checking the sizes of each rod before performing such an operation). After one such complete iteration, if the looping condition is met, we continue to the next iteration.

f5:
We first pass source as an argument to function f2, (which returns the top of argument if the rod's size (number of discs in it) is non-zero, else returns a very large number). We do this even for the second argument which is passed to f5 from f7 (destination). We compare both the values of the top elements of both these rods, and take help of another function f3 so as to pop the top whose value is smaller . (We do this after ensuring that the order of arguments is based on the their top values, out of which one will be popped; by swapping if necessary). We execute the disc movement operation by taking help of another function f4, by passing arguments in the correct order.

f2:
In this function, we just check if the number of discs in the rod passed as an argument equals zero. If yes, we return a very large number, indicating that nothing can be moved from this rod, and instead,disc from another rod should be moved here. If not, we return the top (peek) to the caller.

f3:
It operates similar to pop function.
This function takes two arguments, and decreases the size of the first argument (as we are moving one disc from that to the second argument) , and inserting zero in the place where the disc was initially present (indicating that there is nothing in that place now). Also, we return the top value of the first argument.

f4:
We take the value which has been returned by the function f3, which is the first argument for this function, (top value of rod, whose top disc's size is less than the top disc size of the rod, passed as second argument), and we insert it at the top of the rod, on which we were supposed to place our disc, which is passed as a second argument while calling this function.We also increase the size of this destination rod to which we moved our disc, by 1.

f8:
We move the stack pointer, to 8 units above the initially stored base pointer, so as to pop all the data which we have used thus far. In other words, we are deallocating the space in the stack, once the main loop (f7) terminates.

We exit the program, once all the iterations are executed.

BUGS:
1) Inside function f5 (line 112), when we compare %rax and %r10, the buggy code had the step : jg .less_branch. But it should be replaced by : jl .less_branch, so as to ensure that the arguments %rdi, %rsi are passed to the function f3 in such a way that the top element of the first argument is smaller than the top element of second argument (to execute a valid movement of discs).

2) Inside the solve function (line 130) , we should first store the current stack pointer %rsp inside %rbp (after pushing %rdi) so as to execute the next instructions, without accessing any garbage/invalid memory locations.