Tower of Hanoi

What is the code?

The code given is to print each step of solving the problem of Tower of Hanoi.

Bugs Found:

1) **Line no. 131**

```
mov %rsp,%rbp  # Bug 1: The current stack pointer needs to be moved to Base pointer to access.
```

The base pointer is updated now. So stops the segmentation fault that was given by the code before.

2) **Line no. 113**

```
jl .less_branch # Bug 2: If r10 is less than rax then the source and destination need to be swapped.
```

It should be less than not greater than.
Because the switch should happen from the pole with the disk of lower Disk size to higher size.

Function and loops Definitions :

I)_start:

```
_start:
 movl $3, %edi
 cmpl $1, (%rsp)
 jle .solve
 mov 16(%rsp), %rax
 movsbl (%rax), %edi
 subl $'0', %edi
```

Puts the default argument in %edi. Checks if an argument was passed to the function. If Passed changes it to that. Jumps or falls to .solve that calls the solve function.

II)solve:

```
solve:
 push %rdi
 mov %rsp,%rbp  # Bug 1: The current stack pointer needs to be moved to Base pointer to access.
```

```
 sub $64, %rsp
mov %rsp, %rdi
call f1

sub $64, %rsp
mov %rsp, %rdi
call f1

sub $64, %rsp
mov %rsp, %rdi
call f1

lea -64(%rbp), %rax
mov (%rbp),%rcx
mov %rcx, (%rax)
add $4, %rax
```

Pushes the first argument. The bug is fixed. That is the current stack pointer is moved to %rbp.

Allocate 64 bytes to first array. This array represents the source pole. Array[0] holds the number of disks in the pole.
After solve calls f1. *f1 Initialises everything in the array to zero.*

The function does the same the same for the auxiliary pole and destination pole arrays.

So -64(%rbp) is the base address of source array,
-128(%rbp) is the base address of auxiliary array,
-192(%rbp) is the base address of destination array.

III) .init_s:
Now the function prepares to fall through init_s . init_s will loop through n times. Each time, it will add the discs. For example if it is 3. It will put it is 3,3,2,1 ie, Array[0] is the no. of disks. The later numbers are the disks on the pole.

Now the function stores 2^N-1 as %r14. That is the number of steps that are required in tower of hanoi problem. %r15 has zero it is the looping variable.

```
.init_s:
 mov %rcx, (%rax)
 add $4, %rax
 loop .init_s
```

```
call pt7

mov (%rbp), %cl
mov $1, %r14
shl %cl, %r14
dec %r14
xor %r15,%r15
```

IV) f7: The loop

```
f7:
 lea -64(%rbp), %rdi
 lea -192(%rbp), %rsi
 call f5

 inc %r15
 cmp %r14, %r15
 jge f8

 lea -64(%rbp), %rdi
 lea -128(%rbp), %rsi
 call f5

 inc %r15
 cmp %r14, %r15
 jge f8

 lea -192(%rbp), %rdi
 lea -128(%rbp), %rsi
 call f5

 inc %r15
 cmp %r14, %r15
 jge f8
 jmp f7
```

In the loop the f5 is called three times each time with the following arguments and moves

Valid move from S to D
Valid move from S to A

Valid move from A to D
And repeats this 2^N-1 times.

Each time a move is done the counter %r15 is increased and is compared to 2^N-1 and when they become equal you remove everything from stack and return and exit the program.

V)f5: The Move Function

```
f5:
 mov %rdi, %r9
 call f2
 mov %rax, %r10
 mov %rsi, %rdi
 call f2
 mov %r9, %rdi
 cmp %rax, %r10
 jl .less_branch # Bug 2: If r10 is less than rax then the source and
destination need to be swapped.
.greater_branch:
 mov %rdi, %rax
 mov %rsi, %rdi
 mov %rax, %rsi
.less_branch:
 call f3
 push %rdi
 push %rsi
 mov %rsi, %rdi
 mov %rax, %rsi
 call f4
 pop %rsi
 pop %rdi
 jmp pt7
```

f5 is a move function which takes the from pole and to pole as arguments. It calls f2 function which returns the top of the disk value of the array. The function compares the top values and swaps it in the .greater branch if required before calling the f3 and f4 functions which does the actual swap.

VI) f2: The top()

It returns the value of the top element of array. If it is empty it returns 100000. Ie so that the smaller or valid is chosen.

```
f2:
 movl (%rdi),%ecx
```

```
 cmpl $0, %ecx
 jz .peek_empty
 dec %ecx
 movq 4(%rdi, %rcx, 4), %rax
 ret
```

VII) f3: The pop()

```
f3:
 movl (%rdi),%ebx
 dec %ebx
 movl 4(%rdi, %rbx, 4), %eax
 movl $0, 4(%rdi, %rbx, 4)
 mov %ebx, (%rdi)
 ret
```

F3 is like a pop(). It deletes the last element of the array and returns it. It also reduces the size of the array. Ie the number of disks in the array.

VIII)f4: The push()

```
f4:
 movl (%rdi),%ecx
 mov %rsi, 4(%rdi, %rcx, 4)
 inc %ecx
 mov %ecx, (%rdi)
 ret
```

f4 is like the push() which takes two arguments, the array and the value to be pushed. It pushes the value at the end of the array ie, add a disk. It also increases the size of the array or pole by 1. It does not return anything.

Explanation of Output:
Each three lines are the output of one valid move. Total there will 2^N-1 moves.

In the given case the
1 3 2 1
2
3
Is the initial state

5 3 2
6
7 1

You move 1 from Source to Destination.

9 3
10 2
11 1

You move 2 from source to Auxiliary

13 3
14 2 1
15

You move 1 from Destination to Auxiliary

17
18 2 1
19 3

You move 3 from Source to Auxiliary

21 1
22 2
23 3

You move 1 from Auxiliary to source

25 1
26
27 3 2

You move 2 from Auxiliary to Destination

29
30
31 3 2 1

You move 1 from Source to destination

The puzzle is over and you have moved everything from source to destination.