ITERATIVE TOWER OF HANOI

-> The question is about tower of hanoi(n poles).it is an iterative solution. In tower of hanoi to solve it we have to m ke (2power n)-1 moves which might be from source to destination or source to auxiliary or from auxiliary pole to de tination pole(this is in the case of three poles).

-> In the soln, we make three arays namely source array ,auxiliary array, destination array.these three arrays are bein stored on working stack itself.The first value of array(aarray[0]) stores number of disks being present on the pole. th remanining aray contains the disk number in order from the bottom.

-> IN the given code we have many functions each serving a seperate purpose. we
have a iterator to iterate 2powern -1 times and it is being stored in %r15. the value (2powern)-1 is stored in %r14. fu ctionalities of various functions

1)f1:

```
xor %rax, %rax
movl $64, %ecx
rep stosb
ret
```

f1 ->it is used to make the arrays instantiate to '0'.
i.e arr[]={0};
Here, %rcx is known as counter register.
the 'rep' will repeat the operation as number of times as value stored in %rcx register
The word 'stosb' will do the operation movq rax, (rdx). it moves the value stored in %rax to memory address store in (%rdx).It fills only one byte at a time.

2)f2 -> It returns the top disk number on the pole. If the pole is empty then it return a large number.

```
f2:
movl (%rdi),%ecx
cmpl $0, %ecx
jz .peek_empty
dec %ecx
movq 4(%rdi, %rcx, 4), %rax
ret
.peek_empty:
movl $100000, %eax
ret
```

It returns a large value , since in tower of hanoi we will be moving only a disk with smaller number onto a d sk with a large number.Since if one pole is empty on give two poles, we will move to empty one.So to make it pos ible we aare giving a large number to empty pole.

3)f3:

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

f3 -> It stores the value of the top disk number into %rax register.
it removes the top disk from pole i.e it makes the top value to zero.it decreases the value of base of array(arr[0]) , ince the number of diska on the pole has been decreased.

4)f4:

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

f4 -> here %rdi contains the value of register to which the disk has to be moved.It takes in two values as argu ents one is the base pointer of pole to which disk has to be moved, and the number of the disk to be moved. To de ide to ehich pole it should be moved we will compare the outputs from F2.

5).greater branch:

```
mov %rdi, %rax
mov %rsi, %rdi
mov %rax, %rsi
```

for f4 we have to pass base address of to which index it has to moved .we will pass it by using this function. we w ll compare above this function and if %rsi has large value
then we will swap thefunctions here.

6)f5:

```
mov %rdi, %r9
call f2
mov %rax, %r10
mov %rsi, %rdi
call f2
mov %r9, %rdi
cmp %rax, %r10
jg .less_branch
```

f5:
In this funtion the shift operations happens as we pass only the arguments in f7 the operation will be done here , here we are storing the rdi in r9 because we are modifying the rdi after this so we are storing it after this storing we will call f2 and we will store the return value in r10 and after this we will store rsi to rdi becaause the rdi is always the first argument so we have to store the argument in rdi and again call the function f2 and after this the rdi is restored again by r9 and comparing the last returned value and r10, after this if the return value will less than r10 then we have swap the rdi and rsi, for shifting the top element otherwise if continues and calls f3 function after this i am restoring the rdi and rsi and call f4 after this again i will restore the rdi and rsi .

7)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
```

In this funtion the source array i.e first array is initialised like if the no of rings are 3 then the array is updated arr[]={1,2,3},like this it has to initialise the first array according the given no f rings.here rcx is like counter and as we are storing them 1,2,..n-1,n.so we will decrease the rcx by one and storing n the array this will be take care by the loop instruction. For this code we have to iterate about 2 power n -1 times so this 2 power n -1 in r14 and r15 is as i. As equivalent to for loop as for(int i=0;i<pow(2,n)).

8)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
```

it is loop function .here the iterator is %r14 and the max value is stored in %r15.Its is the main loop from which all functions are sourced. In it first we will take sourcea and destination arrays, source and auxiliary and at last auxiliar and destination register.
In this loop we will do the the main operations like shifting rings , at first the shift will happen between first and third array they are storing their base address to rdi and rsi as it passes as argument for f5 function and after returning from the f5 we will increase r15 by 1 like we will do shift between second and third, one and third arrays after every shift we will increase the r15 by 1 like that this will continue upto r15 = r14. Then the shift ends.

9)
F8:
this loop is called when the for loop gets over, In this loop it free the memory upto now what we have used . In this the base pointers above address is stored in rsp and returns that means it will pop all elements .after returning it goes to the where the

solve is called and exits the code.

10)
Explanation of printing :
As the given example is 3 rings
1. the first line is first array and the second line is auxilary array and the third line is:
destination array.
At first
3 2 1
-
-
1 moves to destination first
3 2
-
1
after 2 moves to the auxilary
3
2
1
after this 1 moves to auxilary otherwise the condition will fail if we move other
things the condition is the top element should be less than the next element
3
2 1
-
after this 3 moves to dest as dest should start with 3
-
2 1
3
after this 1 moves to first array to keep 2 on the 3 we should take the 1 out
1
2
3
after this 2 moves to dest
1
-
3 2
after this obviously 1 will move to dest
-
-
3 2 1.
-->-->-->Bugs found:

It is at two lines at 113 and 131
In 113 we should go to less_branch when it is less not greater
In 131 we should mov rsp to rbp other wise it cannot remember where the function
starts.
This the assembly code of a iterative solution of tower of hanoi.