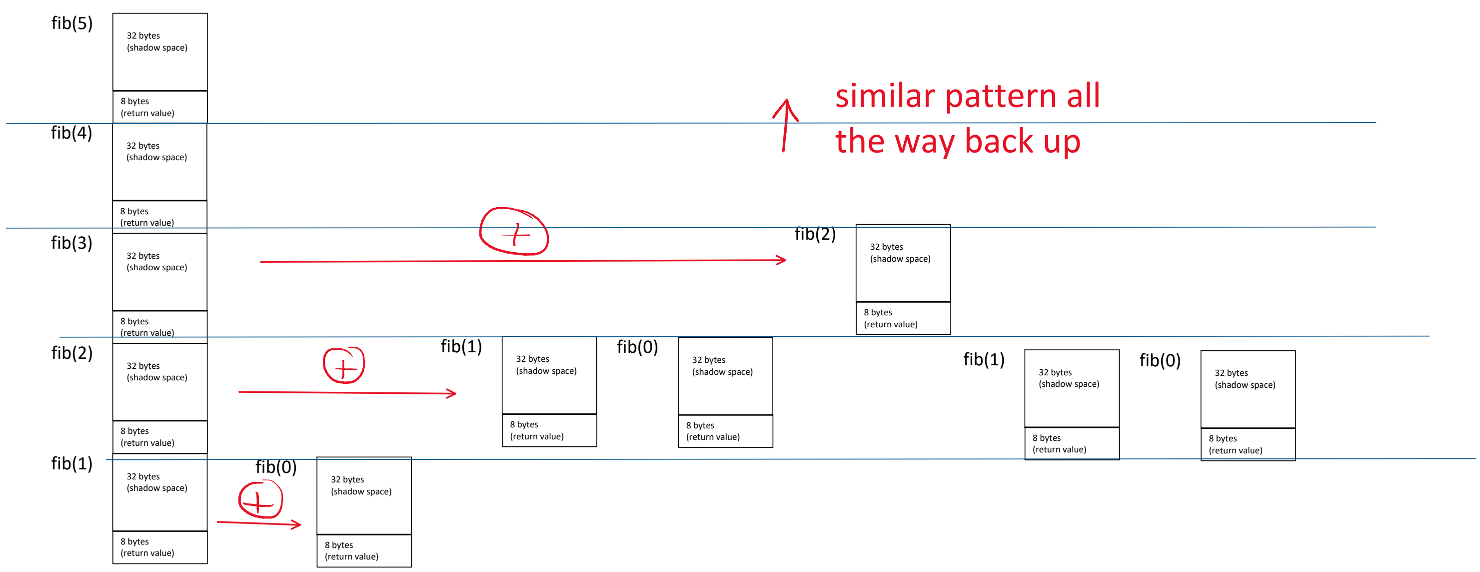
Assignment 2

Sunday, November 8, 2020 1:18 PM

For the Fibonacci Question, I wrote two versions of the function.

fibX64_old: The first version translates the pseudo-code exactly from the Assignment. So, for fib(5), we get **fib(4)** and **fib(3)** through recursion, add our results together together and get our result. In this case, to get **fib(4)**, we have to get fib(3) and fib(2), to get fib(3) we have to get fib(2) and fib(1), and so on. So we get our result for **fib(4)**, but now we also have to get **fib(3)** again, starting from scratch, calling the function recursively all the way to the bottom again.

The first time I call the fib(n-1) function, I treat it as a non-leaf, in that it creates shadow space for the recursive call to fib(n-1). Every calll of fib(X) has its own shadow space of 32 bytes. This means that, when we call **fib(4)**, we have to make space for fib(3), which makes space for fib(2) (*which also makes space for fib(1) for the addition*), which makes space for fib(1) (*which also makes space for fib(0) for the addition*). In terms of space used, the stack will look something like this:



Original Algorithm

```
long long fibonacci_recursion(long long fin)
{
  if(fin <= 0)
    return fin;
  else if (fin == 1)
    return 1;
  else
    return fibonacci_recursion(fin-1) + fibonacci_recursion(fin-2);
}
```

The shadow spaces to the right mean that we are reusing these areas in memory. Since we are reusing memory and the stack space used by fib(n-1) will never be greater than fib(n), we reach max stack depth early on in the program, as we are working down from fib(5) to fib(1) (on the left hand side).

So for version 1, if we have fib(n), then stack space will be n*(32+8). The time complexity of the program is O(*2^n*)

Second version of the program is just called **fibX64** in the .asm file included. This version uses the same pseudocode algorithm included in the assignment, but I have changed the order a little bit

```
long long fibonacci_recursion(long long fin)
{
    if(fin <= 0)
        return fin
    else
        return fib_rec(fin);
}
long long fib_rec(long long fin) {
    if(fin == 1)
        return 1
    else
        return fib_rec(fin-1) + fib_rec(fin-2)
}
```

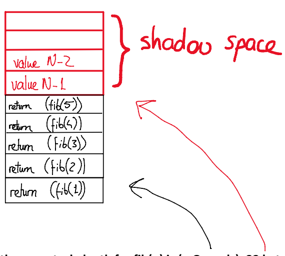
So I just split the function in two, with a guard function and a recursive function.

For this version, I'm not entirely sure if I am bending the rules for x64 calling conventions, which is why I included both versions for my submission. The goal was to have every recursive call of *fib_rec()* to use the exact same shadow space (32-bytes only). Every recursive call of fib_rec() would push the return address on stack, which meant I had to make a variable in memory called **lvlRecursion** which counts how many levels of recursion we are in, and tells us how many QWORDs in stack we have to skip in order to get to get to that shared shadow space. The reason I'm not sure if this is valid to x64 calling conventions is that fib_rec() should be considered a non-leaf function, which should create shadow space for every function calls it will make (in this case, a recursive call), but I am skipping that step here.

The reason I wanted shared shadow space is, in the algorithm we have
`fib_rec(fin-1) + fib_rec(fin-2)`
But this issue with this is that we get *fib_rec(fin-1)*, but in order to get this result we'll have to get *fib_rec(fin-2), fib_rec(fin-3),* etc. Once we have this result, we then have to get *fib_rec(fin-2)* again for addition, but we already calculated this last time within *fib_rec(fin-1)*. That means that the call for + *fib_rec(fin-2)* is a waste of time, if we can get a way of retaining the value for *fib_rec(fin-2)* from the first call.

So, in the shadow space, I have 2 QWORDs, valueNminus1 and valueNminus2. This retains the last two values calculated at all times. When we're finished the calculating, we have to switch around these values, so that valueNminus2 = valueNminus1, and valueNminus1 = Sum. RAX also contains the sum when we return from the function call. So through the entire recursive process, we only have to maintain valueNminus1 and valueNminus2, as well as lvlRecursion to know how many QWORDs in stack we have to skip over (we're skipping over all the recursive return addresses) to get to valueNminus1 and valueNminus2.

So for fib_rec(5), the max depth of our stack will look like this:



So the max stack depth for fib(n) is (n Qwords)+32 bytes.

This version of the program also drops the time complexity of the program down from O(*2^n*) to just O(n) without chaging the algorithm, as we'll only have to make n recursive calls in total.