Problem 5: Recursive Power

In Problem 3, you implemented a program to compute "x to the power y" using iteration.

In this problem, you'll implement the **recursive** program that computes x raised to the power y.

Consider the following recursive Python program in which `main()` gets input from the user and then calls a recursive `power` function that computes "x to the power y" and returns it to `main` to be printed.

Your program only needs to handle nonnegative inputs.

```
def main():
    read x from user # this isn't real python syntax, but the
    read y from user # idea here is to get input from the user.
    print power(x, y)

def power(x, y):
    if y == 0:
        return 1
    else:
        return x * power(x, y-1)
```

Example run

Here is an example of the HMMM code running to compute `3**5 == 243`

Admittedly, it's not too exciting, but that's only because all of the excitement is inside the HMMM program itself!

```
Enter number: 3
Enter number: 5
243
```

Your task

Your job is to implement this recursive `power` function in the HMMM assembly language, as "faithfully" as possible.

By "faithful" we mean that you will have a section of HMMM code that corresponds to `main` and a section that corresponds to `power`. The `main` part will read in input `x` and `y` from the user and then jump to `power`. Your `power` code will, in the general case, call itself recursively to calculate `power(x, y-1)` and then multiply the return value of that recursion by `x` and return that value.

Notice that `power` does not print anything—it simply returns its value to `main` which does the printing. You'll need to use a stack as we described in class.

If you would like to start from the recursive factorial example we covered in class, you'll find it below for easy copy-and-paste. Be sure that you understand how this code works so that you can alter it appropriately!

```
# This is the recursive factorial from class:
00 read r1          # read input and put into r1.
01 setn r15 42      # 42 is the beginning of the stack, put that address in r15.
```

```
02 call r14 5      # begin function at line 5, but first put next address (03) into
r14.
03 jumpn 21        # Let's defer final output to line 21
04 nop             # no operation-but useful for squeezing in an extra input
05 jnez r1 8       # BEGINNING OF FACTORIAL FUNCTION! Check if r1 is non-zero. If it
is go to line 8 and do the real recursion!
06 setn r13 1      # otherwise, we are at the base case: load 1 into r13 and
07 jumpr r14       # return to where we were called from (address is in r14)
08 storer r1 r15   # place r1 onto the stack
09 addn r15 1      # increment stack pointer
10 storer r14 r15  # place r14 onto the stack
11 addn r15 1      # increment stack pointer
12 addn r1 -1      # change r1 to r1-1 in preparation for recursive call
13 call r14 5      # recursive call to factorial, which begins at line 5-but first
store next memory address in r14
14 addn r15 -1     # we're back from the recursive call! Restore goods off the
stack.
15 loadr r14 r15   # restoring r14 (return address) from stack
16 addn r15 -1     # decrement stack pointer
17 loadr r1 r15    # restoring r1 from stack
18 mul r13 r13 r1  # now for the multiplication
19 jumpr r14       # and return!
20 nop             # nothing
21 write r13       # write the final output
22 halt
```

Your code will only receive credit if it faithfully translates the Python pseudo-code for `power`. In particular, you need to use recursion.

Note! Factorial and power are **very** close to each other! You will only have to change a few lines (perhaps surprisingly few) in order to create your recursive power program if you start with the code above.

Extra hint! Consider making the first line

```
00 read r2
```

and then continuing with the factorial code This will allow the power to live in `r1`, which is convenient!

In addition, if you do this, the `nop` will help you avoid renumbering the whole thing!

As with each HMMM problem, do include a comment for each line of code, or almost every line, as shown above. Also, be sure to test your code thoroughly.

Congratulations! You've completed this week's HMMM problems.

If you'd like an even deeper challenge implementing recursion in HMMM, try the extra challenge problem, which uses *two* recursive calls in order to implement $F_n = F_{n-1} + F_{n-2}$