**1. Problem 4.45**

A. Since in practice problem 4.7 it always returns 0, we know that when %rsp is pushed onto the stack, it still has its old value. So the code in this problem is not a correct representation because this would push the new value of %rsp onto the stack.

B. movq REG, 8(%rsp)

subq $8, %rsp

**2. Problem 4.46**

A. This code does not describe the sequence of events for popq. In practice problem 4.8, it always returned 0xabcd. If the code in this problem was correct it should return 0xabcd + 8.

B. addq $8, %rsp

leaq -0x08(%rsp), REG

**3.**

A. #define N 21 /\* plug something in here \*/

#define XSIZE (0x01<<N)

// #define XSIZE 2095029

// #define XSIZE 4192000

For this one, no matter if I compiled without options, with -m64, -m32, and with -Os, the result was about the same. N had to be between 20 and 21, usually very close to 21. The results were around 2095000. If you take this number and times 4, the number of bytes per int, it is close to 8388608, which is the size of the stack. After changing the size of the stack up to 16777216 bytes, N had to be between 21 and 22, close to 22. The size of the array maxed out around 4192000, which if you times by 4 is fairly close to the stack size. When the array got too big I got a segmentation fault error.

B. #define M 18 /\* plug something in here \*/

// #define MAXREC (0x01<<M)

// #define MAXREC 261882

#define MAXREC 523600

For this one, when I compiled with -m64 and with no options, M had to be just less than 18. The MAXREC was around 261800. If you times this by 32 it is close to the stack size of 8388608. When I compiled with -m32, M had to be between 17 and 18. MAXREC maxed out around 174500. If you times this by 48 it is close to the stack size of 8388608. When I compiled with -Os and when I doubled the stack size, M had to be just short of 19. MAXREC maxed out around 523500, which is double from the other cases. I also got segmentation errors when MAXREC got too large.