BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I HAVE PERFORMED ALL OF THE WORK TO CREATE THIS FILE AND/OR DETERMINE THE ANSWERS FOUND WITHIN THIS FILE MYSELF WITH NO ASSISTANCE FROM ANY PERSON (OTHER THAN THE INSTRUCTOR OR GRADERS OF THIS COURSE) AND I HAVE STRICTLY ADHERED TO THE TENURES OF THE OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.

THIS IS THE README FILE FOR LAB 5.

Name: Sam Espanioly

When answering the questions in this file, make a point to take a look at whether the most significant bit (remembering it can be bit 7, 15, 31 or 63 depending upon what size value we are working with) to see if the results you see change based on whether it is a 0 or a 1.

```
. file "lab5.s"
.globl main
    .type    main, @function

.text
main:
    pushq %rbp                              #stack housekeeping
    movq %rsp, %rbp

Label1:
                                            #as you go through this program note the changes to %rip starts with 0x555555555168 then 1a4
    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
    movb    $-1, %al                        # the value of %rax is:         0x88776655443322ff
    movw    $-1, %ax                        # the value of %rax is:         0x887766554433ffff
    movl    $-1, %eax                       # the value of %rax is:         0xffffff
    movq    $-1, %rax                       # the value of %rax is:         0xffffffffffff
    movl    $-1, %eax                       # the value of %rax is:         0xffffff
    cltq                                    # the value of %rax is:         0xffffffffffff
    movl    $0x7fffffff, %eax               # the value of %rax is:         0x7fffff
    cltq                                    # the value of %rax is:         0x7fffff
    movl    $0x8fffffff, %eax               # the value of %rax is:         0x8fffff
    cltq                                    # the value of %rax is:         0xffffff8fffff
                                            # what do you think the cltq instruction does?
                                            Extends the sign value or the registers' value
    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211

                                            # the value of %rdx *before* movb $0xAA, %dl executes is: 0x7fffffffe028
    movb    $0xAA, %dl                      # the value of %rdx is:         0x7fffffffe0aa
    movb    %dl, %al                        # the value of %rax is:         0x88776655443322aa
    movsbw    %dl, %ax                      # the value of %rax is:         0x887766554433ffaa
    movzbw    %dl, %ax                      # the value of %rax is:         0x88776655443300aa

    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
    movb     %dl, %al                       # the value of %rax is:         0x88776655443322aa
    movsbl    %dl, %eax                     # the value of %rax is:         0xffffffaa
    movzbl    %dl, %eax                     # the value of %rax is:         0xaa

    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
    movb     %dl, %al                       # the value of %rax is:         0x88776655443322aa
    movsbq    %dl, %rax                     # the value of %rax is:         0xffffffffffffffaa
    movzbq    %dl, %rax                     # the value of %rax is:         0xaa

    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
                                            # the value of %rdx *before* movb $0x55, %dl executes is: 0x7fffffffe0aa
    movb    $0x55, %dl                      # the value of %rdx is:         0x7fffffffe055
    movb    %dl, %al                        # the value of %rax is:         0x8877665544332255
    movsbw    %dl, %ax                      # the value of %rax is:         0x8877665544330055
    movzbw    %dl, %ax                      # the value of %rax is:         0x8877665544330055

    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
    movb     %dl, %al                       # the value of %rax is:         0x8877665544332255
    movsbl    %dl, %eax                     # the value of %rax is:         0x55
    movzbl    %dl, %eax                     # the value of %rax is:         0x55

    movq    $0x8877665544332211, %rax       # the value of %rax is:         0x8877665544332211
    movb     %dl, %al                       # the value of %rax is:         0x8877665544332255
    movsbq    %dl, %rax                     # the value of %rax is:         0x55
    movzbq    %dl, %rax                     # the value of %rax is:         0x55

                                            # answer questions below when included
```

```
                                    # in executable
#   movq   $0x8877665544332211, %rax  # the value of %rax is:          0x8877665544332211
#   pushb   %al
#   movq   $0, %rax
#   popb   %al                      # the value of %rax is:          0x11

    movq   $0x8877665544332211, %rax  # the value of %rax is: 0x8877665544332211   the value of %rsp is:0x7fffffffdf00
    pushw   %ax                     # the value of %rsp is:0x7fffffffdefe
                                    # the difference between the two values of %rsp is: f00 – efe = 002
    movq   $0, %rax                 # the value of %rax is:0x0
    popw   %ax                      # the value of %rax is:0x2211   How did the value of %rsp change?it went back to 0x7fffffffdf00

    movq   $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211   the value of %rsp is:0x7fffffffdf00
    pushw   %ax                     # the value of %rsp is:0x7fffffffdefe
                                    # the difference between the two values of %rsp is: 2
    movq   $-1, %rax                # the value of %rax is:0xffffffffffffffff
    popw   %ax                      # the value of %rax is:0xffffffffffff2211   How did the value of %rsp change? it went back to 0x7fffffffdf00

                                    # answer questions below when included
                                    # in executable
#   movq   $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211
#   pushl   %eax
#   movq   $0, %rax
#   popl   %eax                     # the value of %rax is:0x44332211

    movq   $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211   the value of %rsp is:0x7fffffffdf00
    pushq   %rax                    # the value of %rsp is:0x7fffffffdef8
                                    # the difference between the two values of %rsp is: 8
    movq   $0, %rax                 # the value of %rax is:0x0
    popq   %rax                     # the value of %rax is:0x8877665544332211   How did the value of %rsp change?0x7fffffffdf00

                                    # what eflags are set?

    movq   $0x500, %rax             # the value of %rax is:0x500
    movq   $0x123, %rcx             # the value of %rcx is:0x123
    # 0x123 - 0x500
    subq   %rax, %rcx               # the value of %rax is:0x500
                                    # the value of %rcx is:0xfffffffffffffc23

                                    # what eflags are set?[ CF SF IF ]

    movq    $0x500, %rax            # the value of %rax is:0x500
    movq   $0x123, %rcx             # the value of %rcx is:0x123
    # 0x500 - 0x123
    subq   %rcx, %rax               # the value of %rax is:0x3dd
                                    # what eflags are set?[ PF AF IF ]

    movq   $0x500, %rax             # the value of %rax is:0x500
    movq   $0x500, %rcx             # the value of %rcx is:0x500
    # 0x500 - 0x500
    subq   %rcx, %rax               # the value of %rax is:0x0
                                    # what eflags are set?[ PF ZF IF ]

    movb   $0xff, %al               # the value of %rax is:0xff
    # 0xff +=1 (1 byte)
    incb   %al                      # the value of %rax is:0x0   what eflags are set?[ pf af zf if ]

    movb   $0xff, %al               # the value of %rax is:0xff
    # 0xff +=1 (4 bytes)
    incl   %eax                     # the value of %rax is:0x100   what eflags are set?[ PF AF IF ]

    movq   $-1, %rax                # the value of %rax is:0xffffffffffffffff
    # 0xff +=1 (8 bytes)
    incq   %rax                     # the value of %rax is:0x0   what eflags are set?[ PF AF ZF IF ]

    movq   $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211
    movq   $0x8877665544332211, %rcx  # the value of %rax is:0x8877665544332211   what eflags are set? [ PF AF ZF IF ]
    addq   %rcx, %rax               # the value of %rax is:0x10eeccaa88664422   what eflags are set?  [ CF PF IF OF ]

    movq    $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211
    andq   $0x1, %rax               # the value of %rax is:0x1

    movq    $0x8877665544332211, %rax  # the value of %rax is:0x8877665544332211   explain why the values for AND/OR/XOR are
    andq   %rax, %rax               # the value of %rax is:0x8877665544332211   what they are
```

orq %rax, %rax # the value of %rax is:**0x8877665544332211**
xorq %rax, %rax # the value of %rax is:**0x0**

movq $0x8877665544332211, %rax # the value of %rax is:**0x8877665544332211**
andw $0x3300, %ax # the value of %rax is:**0x8877665544332200** explain the value in the 8 byte register vs
the value in the 2 byte register **the AND happens between %ax (0x2211) and 0x3300 = 0x2200**
%ax = 0010 00100 0001 0001 AND 0011 0011 0000 0000 = 0x3300
salq $4, %rax # the value of %rax is: **0x8776655443322000** Why? **Shifted to the left**

movq $0xff0000001f000000, %rax # the value of %rax is: **0xff0000001f000000** what do these 6 values look like in binary???
1111111100000000000000000000000011111111100000000000000000000000
sall $1, %eax # the value of %rax is:**0x3e000000**
00111110000000000000000000000000
sall $1, %eax # the value of %rax is:**0x7c000000**
01111100000000000000000000000000
sall $1, %eax # the value of %rax is:
11111000000000000000000000000000
sall $1, %eax # the value of %rax is:**0xf0000000**
11110000000000000000000000000000
sall $1, %eax # the value of %rax is:**0xe0000000**
11100000000000000000000000000000

movq $0xff000000ff000000, %rax # the value of %rax is:**0xff000000ff000000** what do these 6 values look like in binary???
1111111100000000000000000000000011111111100000000000000000000000
salq $1, %rax # the value of %rax is:**0xfe000001fe000000**
1111111000000000000000000000000111111111000000000000000000000000
salq $1, %rax # the value of %rax is:**0xfc000003fc000000**
1111110000000000000000000000001111111110000000000000000000000000
salq $1, %rax # the value of %rax is:**0xf8000007f8000000**
1111100000000000000000000000011111111100000000000000000000000000
salq $1, %rax # the value of %rax is:**0xf000000ff0000000**
1111000000000000000000000000111111111000000000000000000000000000
salq $1, %rax # the value of %rax is:**0xe000001fe0000000**
1110000000000000000000000001111111110000000000000000000000000000

movq $0xff000000000000ff, %rax # the value of %rax is:**0xff000000000000ff** what do these 6 values look like in binary???
1111111100000000000000000000000000000000000000000000000011111111
sarq $1, %rax # the value of %rax is:**0xff8000000000007f**
1111111110000000000000000000000000000000000000000000000001111111
sarq $1, %rax # the value of %rax is:
1111111111000000000000000000000000000000000000000000000000111111
sarq $1, %rax # the value of %rax is:
1111111111100000000000000000000000000000000000000000000000011111
sarq $1, %rax # the value of %rax is:
1111111111110000000000000000000000000000000000000000000000001111
sarq $1, %rax # the value of %rax is:
1111111111111000000000000000000000000000000000000000000000000111

movq $0xff000000000000ff, %rax # the value of %rax is:**0xff000000000000ff** what do these 6 values look like in binary???
1111111100000000000000000000000000000000000000000000000011111111
shrq $1, %rax # the value of %rax is:**0x7f8000000000007f**
0111111110000000000000000000000000000000000000000000000001111111
shrq $1, %rax # the value of %rax is:
0011111111000000000000000000000000000000000000000000000000111111
shrq $1, %rax # the value of %rax is:
0001111111100000000000000000000000000000000000000000000000011111
shrq $1, %rax # the value of %rax is:
0000111111111000000000000000000000000000000000000000000000001111
shrq $1, %rax # the value of %rax is:
0000011111111100000000000000000000000000000000000000000000000111

movq $0xff000000000000ff, %rax # the value of %rax is: what do these 6 values look like in binary???
1111111100000000000000000000000000000000000000000000000011111111
sarw $1, %ax # the value of %rax is:
1111111100000000000000000000000000000000000000000000000001111111
sarw $1, %ax # the value of %rax is:
1111111110000000000000000000000000000000000000000000000000111111
sarw $1, %ax # the value of %rax is:
1111111111000000000000000000000000000000000000000000000000011111
sarw $1, %ax # the value of %rax is:
1111111111100000000000000000000000000000000000000000000000001111
sarw $1, %ax # the value of %rax is:
1111111111110000000000000000000000000000000000000000000000000111

```
    movq    $0xff000000000000ff, %rax               # the value of %rax is:      what do these 6 values look like in binary???
            1111111100000000000000000000000000000000000000000000000011111111
    shrw    $1, %ax                                 # the value of %rax is:
            1111111100000000000000000000000000000000000000000000000001111111
    shrw    $1, %ax                                 # the value of %rax is:
            1111111100000000000000000000000000000000000000000000000000111111
    shrw    $1, %ax                                 # the value of %rax is:
            1111111100000000000000000000000000000000000000000000000000011111
    shrw    $1, %ax                                 # the value of %rax is:
            1111111100000000000000000000000000000000000000000000000000001111
    shrw    $1, %ax                                 # the value of %rax is:
            1111111100000000000000000000000000000000000000000000000000000111


    leave                                           #post function stack cleanup
    ret

.size   main, .-main
```

1. Write a paragraph that describes what you observed happen to the value in register **%rax** as you watched **mov**X (where X is 'q', 'l', 'w', and 'b') instructions executed. Describe what data changes occur (and, perhaps, what data changes you expected to occur that didn't). Make a point to address what happens when moving less than 8 bytes of data to a register.

The value would be all 8 bytes with movq and 4 bytes with movl 2 bytes with movw and 1 byte with movb basically when moving a long word to a register it changes the first 4 bytes and same goes for word it changes the first 2 bytes and byte changes the first byte.

2. What did you observe happens when the **cltq** instruction is executed? Did it matter what value is in **%eax**? Does **cltq** have any operands?

Cltq is convert long to quad which extends a long to a quad by extending the most significant bit. It does not have an immediate operands but it is affective to the operands on the previous line.

3. Write a paragraph that describes what you saw with respect to what happens as you use the **movs**XX and **movz**XX instructions with different sizes of registers. What do you observe with respect to the source and destination registers used in each instruction? Is there a relationship between them and the XX values? Describe what data changes occur (and, perhaps, what data changes you expected to occur that didn't).

MovsX and MovzX replace bytes but the difference between the both of them is that Movs is sign extended and Movz is zero extended. When the register sizes are different it replaces the small value from the larger value same as Mov except it extends it depending on which one you pick.

4. Write a paragraph that describes what you observed as you watched different push/pop instructions execute. What values were actually put on the stack? How did the value in %rsp change? Use the command **help x** from the command line in gdb. This will give you the format of the **x** instruction that allows you to see what is in specific addresses in memory. Note that a **word** means 2 bytes in x86-64, but it means 4 bytes when using the **x** command in gdb. To print 2 byte values with x, you must specify **h** for halfword. If you wish to use an address located in a register as an address to print from using **x**, use **$** rather than **%** to designate the register. For example, if you wanted to print, in hexadecimal format, 1 2-byte value that is located in memory starting at the address located in register **rsp**, then you could use **x/1xh $rsp**. If you wanted to print, in hexadecimal format, 1 8-byte value that is located in memory starting at the address located in register **rsp**, then you could use **x/1xg $rsp**. You might want to play with this command a little. ☺

the value of rsp and rip does change depending on what is pushed or popped when there's something pushed the value of one of those registers becomes smaller and when it is popped it goes back to what it was before the push command.

5. What did you observe happened to the condition code values as instructions that process within the ALU executed? What instructions caused changes? Were the changes what you expected? Why or why not?

The instructs did change whenever any line of code would execute all the MovXX commands changed the second operand after the call and cltq also changed the value if the most significant bit was 1 but did not change anything when it was 0. Push and Pop changed the values of many registers at the same time.

6. There were some instructions that caused bitwise AND/OR/XOR data manipulation. What did you observe?

The and or xor bitwise was working on the binary level which was nice to decode and understand.

7. There were some instructions that executed left or right bit shifting. What did you observe with respect to the register data? Did the size of the data being shifted change the result in the register? How?

Sometimes the shift to the right would depending on the call you make some functions would extend the msb and some would shift zeros instead and some would keep all the ones on the left and shift 0 after all the ones to the right. Shift to the left was not as fun but it had the same idea; the msb would get dumped unless commanded not to.

8. What did you observe happening to the value in register **%rip** over the course the program? Did it always change by the same amount as each instruction executed?

It changed depending on how much info was pushed in and it would equally change back to what it was before the push and therefore after the pop instruction would be called, the value it would change by is the same in reverse order.

9. What did you observe when you took the comments away from the two different instruction sets and tried to reassemble the program? There were questions in item **L** and **M** in the Lab 5 Description; include your answers to those questions here.

The code did bug out but I did not understand why because the instructions that were called made sense to me but maybe the fact that the registers were of different sizes caused the code to bug.

10. Any other comments about what you observed?

Fun project I loved going in depth about the CPU and registers values and how things get actually processed on a registers level.