BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT 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: Saeed Alneyadi.11

When answering the questions in this file, make a point to take a look at whether the most significant bit (remembering it can be in bit position 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.

It's best that you present all register values in hexadecimal rather than decimal. You will be able to understand what is happening more easily.

Monitor the RFLAGS. What instructions change RFLAGS, what instructions don't?

```
.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
movq                    $0x8877665544332211, %rax    # the value of %rax is: 0x400562 4195682

# Recall that -1 is represented as 0xff, 0xffff, etc. depending upon the size of the value
movb                    $-1, %al                     # the value of %rax is: 0x8877665544332211   -8613303245
movw                    $-1, %ax                     # the value of %rax is: 0x88776655443322ff   -8613303245
movl                    $-1, %eax                    # the value of %rax is: 0x887766554433ffff   -8613303245
movq                    $-1, %rax                    # the value of %rax is: 0xffffffff   4294967295

movl                    $-1, %eax                    # the value of %rax is: 0xffffffffffffffff   -1
cltq                                                 # the value of %rax is: 0xffffffff   4294967295

movl                    $0x7fffffff, %eax            # the value of %rax is: 0xffffffffffffffff   -1
cltq                                                 # the value of %rax is: 0x7fffffff   2147483647
movl                    $0x8fffffff, %eax            # the value of %rax is: 0x7fffffff   2147483647
cltq                                                 # the value of %rax is: 0x8fffffff   2415919103
                                                     # What is the difference between the values 0x7fffffff and 0x8fffffff
                                                     # what do you think the cltq instruction does?

movq                    $0x8877665544332211, %rax    # the value of %rax is: 0xffffffff8fffffff   -1879048193
                                                     # the value of %rdx *before* movb $0xAA, %dl executes is:

                        # Note the value of the 8-byte register values vs the 1, 2, or 4-byte register values
                        # How does each size instruction suffix affect the 8-byte register?  Don't write answers here; you'll need this info later.
movb                    $0xAA, %dl                   # the value of %rdx is: 0x7fffffffde88   140737488346760 => 0x7fffffffdeaa   140737488346794
movb                    %dl, %al                     # the value of %rax is: 0x8877665544332211   -8613303245
movsbw                  %dl, %ax                     # the value of %rax is: 0x88776655443322aa   -8613303245
movzbw                  %dl, %ax                     # the value of %rax is: 0x887766554433ffaa   -8613303245

movq                    $0x8877665544332211, %rax    # the value of %rax is: 0x88776655443300aa   -8613303245
movb                    %dl, %al                     # the value of %rax is: 0x8877665544332211   -8613303245
movsbl                  %dl, %eax                    # the value of %rax is: 0x88776655443322aa   -8613303245
movzbl                  %dl, %eax                    # the value of %rax is: 0xffffffaa   4294967210

movq                    $0x8877665544332211, %rax    # the value of %rax is: 0xaa   170
movb                    %dl, %al                     # the value of %rax is: 0x8877665544332211   -8613303245
movsbq                  %dl, %rax                    # the value of %rax is: 0x88776655443322aa   -8613303245
movzbq                  %dl, %rax                    # the value of %rax is: 0xffffffffffffffaa   -86 => 0xaa   170

movq                    $0x8877665544332211, %rax    # the value of %rax is: 0xaa   170
                                                     # the value of %rdx *before* movb $0x55, %dl executes is: 0x7fffffffdeaa   140737488346794
movb                    $0x55, %dl                   # the value of %rdx is: 0x7fffffffde55   140737488346709
movb                    %dl, %al                     # the value of %rax is: 0x8877665544332255   -8613303245
movsbw                  %dl, %ax                     # the value of %rax is: 0x8877665544330055   -8613303245
```

```
movzbw          %dl, %ax            # the value of %rax is: 0x8877665544330055    -8613303245

movq            $0x8877665544332211, %rax   # the value of %rax is: 0x8877665544330055    -8613303245
movb            %dl, %al            # the value of %rax is: 0x8877665544332211    -8613303245
movsbl          %dl, %eax           # the value of %rax is: 0x8877665544332255    -8613303245
movzbl          %dl, %eax           # the value of %rax is: 0x55    85

movq            $0x8877665544332211, %rax   # the value of %rax is: 0x55    85
movb            %dl, %al            # the value of %rax is: 0x8877665544332211    -8613303245
movsbq          %dl, %rax           # the value of %rax is: 0x8877665544332255    -8613303245
movzbq          %dl, %rax           # the value of %rax is: 0x55    85

#movq           $0x8877665544332211, %rax
#pushb          %al
#movq           $0, %rax
#               popb        %al

movq            $0x8877665544332211, %rax   # the value of %rax is: 0x55    85              the value of %rsp is: 0x7fffffffdd90
pushw           %ax                 # the value of %rsp is: 0x7fffffffdd8e
                                    # the difference between the two values of %rsp is: 0x000000000002    2
movq            $0, %rax            # the value of %rax is: 0x0    0
popw            %ax                 # the value of %rax is: 0x2211    8721        How did the value of %rsp change?

movq            $0x8877665544332211, %rax   # the value of %rax is: 0x8877665544332211    -8613303245        the value of %rsp is: 0x7fffffffdd90
pushw           %ax                 # the value of %rsp is: 0x7fffffffdd8e
                                    # the difference between the two values of %rsp is: 0x000000000002    2
movq            $-1, %rax           # the value of %rax is: 0xffffffffffffffff    -1
popw            %ax                 # the value of %rax is: 0xffffffffffffffff    -1 How did the value of %rsp change?


#movq           $0x8877665544332211, %rax
#pushl          %eax
#movq           $0, %rax
#popl           %eax

movq            $0x8877665544332211, %rax   # the value of %rax is: 0xffffffffffff2211    -56815      the value of %rsp is: 0x7fffffffdd90
pushq           %rax                # the value of %rsp is: 0x7fffffffdd88
                                    # the difference between the two values of %rsp is: 0x000000000008    8
movq            $0, %rax            # the value of %rax is: 0x8877665544332211    -8613303245
popq            %rax                # the value of %rax is: 0x0    0              How did the value of %rsp change?

                                    # what rflags are set?

movq            $0x500, %rax        # the value of %rax is: 0x8877665544332211    -8613303245
movq            $0x123, %rcx        # the value of %rcx is: 0x4004f0    4195568
# 0x123 - 0x500
subq            %rax, %rcx          # the value of %rax is: 0x500    1280
                                    # the value of %rcx is: 0x123    291

                                    # what rflags are set?

movq            $0x500, %rax        # the value of %rax is: 0x500    1280
movq            $0x123, %rcx        # the value of %rcx is: 0xfffffffffffffc23    -989
# 0x500 - 0x123
subq            %rcx, %rax          # the value of %rax is: 0x500    1280
                                    # what rflags are set?

movq            $0x500, %rax        # the value of %rax is: 0x3dd    989
movq            $0x500, %rcx        # the value of %rcx is: 0x123    291 => 0x500    1280
# 0x500 - 0x500
subq            %rcx, %rax          # the value of %rax is: 0x500    1280
                                    # what rflags are set?

movb            $0xff, %al          # the value of %rax is: 0x0    0
# 0xff +=1 (1 byte)
incb            %al                 # the value of %rax is: 0xff    255             what rflags are set?

movb            $0xff, %al          # the value of %rax is: 0x0    0
# 0xff +=1 (4 bytes)
incl            %eax                # the value of %rax is: 0xff    255             what rflags are set?

movq            $-1, %rax           # the value of %rax is: 0x100    256
```

```
# 0xff +=1 (8 bytes)
incq    %rax                        # the value of %rax is: 0xffffffffffffffff    -1    what rflags are set?

movq    $0x8877665544332211, %rax   # the value of %rax is: 0x0    0
movq    $0x8877665544332211, %rcx   # the value of %rax is: 0x8877665544332211    -8613303245    what rflags are set?
addq    %rcx, %rax                  # the value of %rax is: 0x8877665544332211    -8613303245    what rflags are set?

movq    $0x8877665544332211, %rax   # the value of %rax is: 0x10eeccaa88664422    12201375818
andq    $0x1, %rax                  # the value of %rax is: 0x8877665544332211    -8613303245

movq    $0x8877665544332211, %rax   # the value of %rax is: 0x1    1    explain why the values for AND/OR/XOR are
andq    %rax, %rax                  # the value of %rax is: 0x8877665544332211    -8613303245    what they are
orq     %rax, %rax                  # the value of %rax is: 0x8877665544332211    -8613303245
xorq    %rax, %rax                  # the value of %rax is: 0x8877665544332211    -8613303245

movq    $0x8877665544332211, %rax   # the value of %rax is: 0x0    0
andw    $0x3300, %ax                # the value of %rax is: 0x8877665544332211    -8613303245 explain the value in the 8 byte register vs
                                    #the value in the 2 byte register

salq    $4, %rax                    # the value of %rax is: 0x8877665544332200    -8613303245    Why?

movq    $0xff0000001f000000, %rax   # the value of %rax is: 0x8776655443322000    -8685643418
                                    # to help you understand what's happening in this part of the code, write the value in %rax in binary
                                    # on a piece of scratch paper for the remaining instructions in this file
                                    # and watch the bits move as each shift instruction occurs.
                                    # You should notice how each of the 1-, 2-, 4-, and 8-byte shift instructions works

                                    # within the 8-byte register.
sall    $1, %eax                    # the value of %rax is: 0xff0000001f000000    -7205759351 do these shift instructions do what you
                                    # expected?
sall    $1, %eax                    # the value of %rax is: 0x3e000000    1040187392
sall    $1, %eax                    # the value of %rax is: 0x7c000000    2080374784
sall    $1, %eax                    # the value of %rax is: 0xf8000000    4160749568
sall    $1, %eax                    # the value of %rax is: 0xf0000000    4026531840

movq    $0xff000000ff000000, %rax   # the value of %rax is: 0xe0000000    3758096384
salq    $1, %rax                    # the value of %rax is: 0xff000000ff000000    -7205758975
salq    $1, %rax                    # the value of %rax is: 0xfe000001fe000000    -1441151795
salq    $1, %rax                    # the value of %rax is: 0xfc000003fc000000    -2882303590
salq    $1, %rax                    # the value of %rax is: 0xf8000007f8000000    -5764607180
salq    $1, %rax                    # the value of %rax is: 0xf000000ff0000000    -1152921436

movq    $0xff000000000000ff, %rax   # the value of %rax is: 0xe000001fe0000000    -2305842872
sarq    $1, %rax                    # the value of %rax is: 0xff000000000000ff    -7205759403
sarq    $1, %rax                    # the value of %rax is: 0xff8000000000007f    -3602879701
sarq    $1, %rax                    # the value of %rax is: 0xffc000000000003f    -1801439850
sarq    $1, %rax                    # the value of %rax is: 0xffe000000000001f    -9007199254
sarq    $1, %rax                    # the value of %rax is: 0xfff000000000000f    -4503599627

movq    $0xff000000000000ff, %rax   # the value of %rax is: 0xfff8000000000007    -2251799813
shrq    $1, %rax                    # the value of %rax is: 0xff000000000000ff    -7205759403
shrq    $1, %rax                    # the value of %rax is: 0x7f8000000000007f    91873432398
shrq    $1, %rax                    # the value of %rax is: 0x3fc000000000003f    45936716199
shrq    $1, %rax                    # the value of %rax is: 0x1fe000000000001f    22968358099
shrq    $1, %rax                    # the value of %rax is: 0xff000000000000f    11484179049

movq    $0xff000000000000ff, %rax   # the value of %rax is: 0x7f8000000000007    57420895248
sarw    $1, %ax                     # the value of %rax is: 0xff000000000000ff    -7205759403
sarw    $1, %ax                     # the value of %rax is: 0xff000000000007f    -7205759403
sarw    $1, %ax                     # the value of %rax is: 0xff000000000003f    -7205759403
sarw    $1, %ax                     # the value of %rax is: 0xff000000000001f    -7205759403
sarw    $1, %ax                     # the value of %rax is: 0xff000000000000f    -7205759403

movq    $0xff000000000000ff, %rax   # the value of %rax is: 0xff00000000000007    -7205759403
shrw    $1, %ax                     # the value of %rax is: 0xff000000000000ff    -7205759403
shrw    $1, %ax                     # the value of %rax is: 0xff000000000007f    -7205759403
shrw    $1, %ax                     # the value of %rax is: 0xff000000000003f    -7205759403
shrw    $1, %ax                     # the value of %rax is: 0xff000000000001f    -7205759403
shrw    $1, %ax                     # the value of %rax is: 0xff000000000000f    -7205759403

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.

   ANSWER: During the program, the %rax was changing so many times. However, in most cases, only two hexadecimal values changes by using the commands that only changes 8-bits from the register. Just a reminder, movq changes 64-bits of the register, movl changes 32-bit, movw changes 16-bits, and movb changes 8-bits. The thing that I was expecting to change the decimal value of the register. From the comments, you can that these decimal values didn't change.

2. What did you observe happens when the **cltq** instruction is executed? Did it matter what value is in **%eax**? What is the difference between 0x7fffffff and 0x8fffffff ? Does **cltq** have any operands?

   ANSWER: The value of %rax changes from 0x8fffffff 2415919103 to 0xffffffff8fffffff -1879048193. The cltq command did an extension of %ax from %eax to %rax. The value of %eax matters since it consist the half of the %rax register. So, without it, we will not able to get the new value of %rax registar. The difference between 0x7fffffff and 0x8fffffff is 0x10000000. The cltq doesn't have any operand.

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 is the difference between the value 0xAA and the value 0x55? 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).

   ANSWER: The movsXX move zero-extend from first data type to the second data type where it fills the remiaing bytes with zero. The movzXX move sign-extend from first data type to the second data type where it fills the remiaing bytes with the significant value. The difference between 0xAA and 0x55 is 0x55. The source and destination registers have the same type as the XX values. Using these commands, the data changes that happened is the same as I expected. The register %rax received, in most cases, just two hexadecimal values with the change of the reminaing values based on what command used. If it movsXX, it will be filled with significant values while movzXX will fill with zero values.

4. Write a paragraph that describes what you observed as you watched different push/pop instructions execute. What values are put on the stack based on the suffix used? (Use the instructions further down in this question to see stack values.) 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. ☺ It will be well worth your time to do so as the semester continues.

   ANSWER: The suffix used in the instructions of push/pop specific which part of the data should be pushed or poped. And it's also specify how much %rsp change. The %rsp change by the size of the data

type represented with suffix. So, If the suffix is ‘q’, the whole register will be pushed to the stacked. If the suffic is ‘l’, it will push half of the register. If the suffix is ‘w’, it will push quarter of the register. If the suffix is ‘b’, it will push 8-bit of the register. After the push command, the register will be reset to 0.

5. What did you observe happen to the condition code values as instructions that process within the ALU executed? What instructions caused changes? What instructions within this program did not cause condition codes to change? When changes occurred, were the changes what you expected? Why or why not?

   ANSWER: The condition code values changes as instructions implemented. Each instruction code measure a specific condition of the instruction implement.

6. There were some instructions that performed bitwise AND/OR/XOR data manipulation. What did you observe as the suffix changed? Is it consistent with respect to what you learned about these bitwise instructions in class?

   ANSWER: As the suffix change, the instruction changed from AND to OR, and from OR to XOR. The value of %rax didn’t change after the execution of AND and OR which makes sense. However, the execution of XOR set %rax to 0x0 which also makes sense. This is consistent with respect to what you learned about these bitwise instructions in class.

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? Is it consistent with respect to what you learned about these bitwise instructions in class?

   ANSWER: The size of the data stored in the register changed because using a suffix for data type different from the one stored in the register. The program truncates the value of the register then do the shift. Based on what I learn, it is not consistent since shift should cover all the bits, not just part of the register.

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?

   ANSWER: The %rip changes as the program move from one line to another. The register doesn’t change with the same amount. The amount changed differ from two adjacent lines to others.

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 **M** and **N** in the Lab 5 Description; include your answers to those questions here. Based upon your experiences with this exercise, what can you conclude with respect to push/pop instructions when used with the q, l, w, and b suffixes?

   ANSWER: I’ve received 4 error messages saying that there is invalid instruction suffix of push and pop commands. Each code section used one push and one pop command.
   QUESTION M ANSWER: The same error messages appear but instead of four, just two error messages.
   QUESTION N ANSWER: The same error messages appear but instead of four, just two error messages. When using the push/pop command, the command suffix should match with the operand suffix or it will result an error.

10. Any other comments about what you observed?

ANSWER: No, I think the pervious answers description is almost everything I observed.