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:

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.

.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:

# Recall that -1 is represented as 0xff, 0xffff, etc. depending upon the size of the value

movb $-1, %al # the value of %rax is:

movw $-1, %ax # the value of %rax is:

movl $-1, %eax # the value of %rax is:

movq $-1, %rax # the value of %rax is:

movl $-1, %eax # the value of %rax is:

cltq # the value of %rax is:

movl $0x7fffffff, %eax # the value of %rax is:

cltq # the value of %rax is:

movl $0x8fffffff, %eax # the value of %rax is:

cltq # the value of %rax is:

# 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:

# 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:

movb %dl, %al # the value of %rax is:

movsbw %dl, %ax # the value of %rax is:

movzbw %dl, %ax # the value of %rax is:

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

movb %dl, %al # the value of %rax is:

movsbl %dl, %eax # the value of %rax is:

movzbl %dl, %eax # the value of %rax is:

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

movb %dl, %al # the value of %rax is:

movsbq %dl, %rax # the value of %rax is:

movzbq %dl, %rax # the value of %rax is:

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

# the value of %rdx \*before\* movb $0x55, %dl executes is:

movb $0x55, %dl # the value of %rdx is:

movb %dl, %al # the value of %rax is:

movsbw %dl, %ax # the value of %rax is:

movzbw %dl, %ax # the value of %rax is:

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

movb %dl, %al # the value of %rax is:

movsbl %dl, %eax # the value of %rax is:

movzbl %dl, %eax # the value of %rax is:

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

movb %dl, %al # the value of %rax is:

movsbq %dl, %rax # the value of %rax is:

movzbq %dl, %rax # the value of %rax is:

#movq $0x8877665544332211, %rax

#pushb %al

#movq $0, %rax

# popb %al

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

pushw %ax # the value of %rsp is:

# the difference between the two values of %rsp is:

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

popw %ax # the value of %rax is: How did the value of %rsp change?

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

pushw %ax # the value of %rsp is:

# the difference between the two values of %rsp is:

movq $-1, %rax # the value of %rax is:

popw %ax # the value of %rax is: 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: the value of %rsp is:

pushq %rax # the value of %rsp is:

# the difference between the two values of %rsp is:

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

popq %rax # the value of %rax is: How did the value of %rsp change?

# what rflags are set?

movq $0x500, %rax # the value of %rax is:

movq $0x123, %rcx # the value of %rcx is:

# 0x123 - 0x500

subq %rax, %rcx # the value of %rax is:

# the value of %rcx is:

# what rflags are set?

movq $0x500, %rax # the value of %rax is:

movq $0x123, %rcx # the value of %rcx is:

# 0x500 - 0x123

subq %rcx, %rax # the value of %rax is:

# what rflags are set?

movq $0x500, %rax # the value of %rax is:

movq $0x500, %rcx # the value of %rcx is:

# 0x500 - 0x500

subq %rcx, %rax # the value of %rax is:

# what rflags are set?

movb $0xff, %al # the value of %rax is:

# 0xff +=1 (1 byte)

incb %al # the value of %rax is: what rflags are set?

movb $0xff, %al # the value of %rax is:

# 0xff +=1 (4 bytes)

incl %eax # the value of %rax is: what rflags are set?

movq $-1, %rax # the value of %rax is:

# 0xff +=1 (8 bytes)

incq %rax # the value of %rax is: what rflags are set?

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

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

addq %rcx, %rax # the value of %rax is: what rflags are set?

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

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

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

andq %rax, %rax # the value of %rax is: what they are

orq %rax, %rax # the value of %rax is:

xorq %rax, %rax # the value of %rax is:

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

andw $0x3300, %ax # the value of %rax is: explain the value in the 8 byte register vs

#the value in the 2 byte register

salq $4, %rax # the value of %rax is: Why?

movq $0xff0000001f000000, %rax # the value of %rax is:

# 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: do these shift instructions do what you expected?

sall $1, %eax # the value of %rax is:

sall $1, %eax # the value of %rax is:

sall $1, %eax # the value of %rax is:

sall $1, %eax # the value of %rax is:

movq $0xff000000ff000000, %rax # the value of %rax is:

salq $1, %rax # the value of %rax is:

salq $1, %rax # the value of %rax is:

salq $1, %rax # the value of %rax is:

salq $1, %rax # the value of %rax is:

salq $1, %rax # the value of %rax is:

movq $0xff000000000000ff, %rax # the value of %rax is:

sarq $1, %rax # the value of %rax is:

sarq $1, %rax # the value of %rax is:

sarq $1, %rax # the value of %rax is:

sarq $1, %rax # the value of %rax is:

sarq $1, %rax # the value of %rax is:

movq $0xff000000000000ff, %rax # the value of %rax is:

shrq $1, %rax # the value of %rax is:

shrq $1, %rax # the value of %rax is:

shrq $1, %rax # the value of %rax is:

shrq $1, %rax # the value of %rax is:

shrq $1, %rax # the value of %rax is:

movq $0xff000000000000ff, %rax # the value of %rax is:

sarw $1, %ax # the value of %rax is:

sarw $1, %ax # the value of %rax is:

sarw $1, %ax # the value of %rax is:

sarw $1, %ax # the value of %rax is:

sarw $1, %ax # the value of %rax is:

movq $0xff000000000000ff, %rax # the value of %rax is:

shrw $1, %ax # the value of %rax is:

shrw $1, %ax # the value of %rax is:

shrw $1, %ax # the value of %rax is:

shrw $1, %ax # the value of %rax is:

shrw $1, %ax # the value of %rax is:

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.
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?
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).
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.

1. 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?
2. 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?
3. 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?
4. 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?
5. 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?
6. Any other comments about what you observed?