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: Yiming Cheng

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

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:0x00000000ffffffff

movq $-1, %rax # the value of %rax is:0xffffffffffffffff

movl $-1, %eax # the value of %rax is:0x00000000ffffffff

cltq # the value of %rax is:0xffffffffffffffff

movl $0x7fffffff, %eax # the value of %rax is:0x000000007fffffff

cltq # the value of %rax is:0x000000007fffffff

movl $0x8fffffff, %eax # the value of %rax is:0x000000008fffffff

cltq # the value of %rax is:0xffffffff8fffffff

# what do you think the cltq instruction does?

Add the leading 0 to the register,

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

# the value of %rdx \*before\* movb $0xAA, %dl executes is:0x00007fffffffdfb8

movb $0xAA, %dl # the value of %rdx is:0x00007fffffffdfaa

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:0x00000000ffffffaa

movzbl %dl, %eax # the value of %rax is:0x00000000000000aa

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:0x00000000000000aa

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

# the value of %rdx \*before\* movb $0x55, %dl executes is:0x00007fffffffdfaa

movb $0x55, %dl # the value of %rdx is:0x00007fffffffdf55

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:0x0000000000000055

movzbl %dl, %eax # the value of %rax is:0x0000000000000055

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:0x0000000000000055

movzbq %dl, %rax # the value of %rax is:0x0000000000000055

# answer questions below when included

# in executable

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

# pushb %al

# movq $0, %rax

# popb %al # the value of %rax is:

movq $0x8877665544332211, %rax # the value of %rax is:0x8877665544332211 the value of %rsp is:0x00007fffffffdec0

pushw %ax # the value of %rsp is:0x00007fffffffdebe

# the difference between the two values of %rsp is:2

movq $0, %rax # the value of %rax is:0x0000000000000000

popw %ax # the value of %rax is:0x0000000000002211 How did the value of %rsp change? 0x00007fffffffdec0

movq $0x8877665544332211, %rax # the value of %rax is:0x8877665544332211 the value of %rsp is:0x00007fffffffdf55

pushw %ax # the value of %rsp is:0x00007fffffffdebe

# the difference between the two values of %rsp is:97

movq $-1, %rax # the value of %rax is:0xffffffffffffffff

popw %ax # the value of %rax is:0xffffffffffff2211 How did the value of %rsp change? 0x00007fffffffdec0

# answer questions below when included

# in executable

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

# pushl %eax

# movq $0, %rax

# popl %eax # the value of %rax is:

movq $0x8877665544332211, %rax # the value of %rax is:0x8877665544332211 the value of %rsp is:0x00007fffffffdec0

pushq %rax # the value of %rsp is:0x00007fffffffdeb8

# the difference between the two values of %rsp is:8

movq $0, %rax # the value of %rax is:0x0000000000000000

popq %rax # the value of %rax is:0x8877665544332211 How did the value of %rsp change?0x00007fffffffdec0

# what eflags are set?PF ZF IF

movq $0x500, %rax # the value of %rax is:0x0000000000000500

movq $0x123, %rcx # the value of %rcx is:0x0000000000000123

# 0x123 - 0x500

subq %rax, %rcx # the value of %rax is:0x0000000000000500

# the value of %rcx is:0xfffffffffffffc23

# what eflags are set?CF SF IF

movq $0x500, %rax # the value of %rax is:0x0000000000000500

movq $0x123, %rcx # the value of %rcx is:0x0000000000000123

# 0x500 - 0x123

subq %rcx, %rax # the value of %rax is:0x00000000000003dd

# what eflags are set?PF AF IF

movq $0x500, %rax # the value of %rax is:0x0000000000000500

movq $0x500, %rcx # the value of %rcx is:0x0000000000000500

# 0x500 - 0x500

subq %rcx, %rax # the value of %rax is:0x0000000000000000

# what eflags are set?PF ZF IF

movb $0xff, %al # the value of %rax is:0x00000000000000ff

# 0xff +=1 (1 byte)

incb %al # the value of %rax is:0x0000000000000000 what eflags are set? PF AF ZF IF

movb $0xff, %al # the value of %rax is:0x00000000000000ff

# 0xff +=1 (4 bytes)

incl %eax # the value of %rax is:0x0000000000000100 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:0x0000000000000000 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 0F

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

andq $0x1, %rax # the value of %rax is:0x0000000000000001

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 andq is bitwise logical and ,orq is bitwise logical or, and norq is

xorq %rax, %rax # the value of %rax is:0x0000000000000000 bitwise logical nor.

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

salq $4, %rax # the value of %rax is:0x8776655443322000 Why? Salq is to move the integer one bit left.

movq $0xff0000001f000000, %rax # the value of %rax is:0xff0000001f000000 what do these 6 values look like in binary???

sall $1, %eax # the value of %rax is:0x000000003e000000 111110000000000000000000000000

sall $1, %eax # the value of %rax is:0x000000007c000000 1111100000000000000000000000000

sall $1, %eax # the value of %rax is:0x00000000f8000000 1111100000000000000000000000000

sall $1, %eax # the value of %rax is:0x00000000f0000000 11110000000000000000000000000000

sall $1, %eax # the value of %rax is:0x00000000e0000000 11100000000000000000000000000000

movq $0xff000000ff000000, %rax # the value of %rax is:0xff000000ff000000 what do these 6 values look like in binary???

salq $1, %rax # the value of %rax is:0xfe000001fe000000 the bits move as each shift instruction appears

salq $1, %rax # the value of %rax is:0xfc000003fc000000

salq $1, %rax # the value of %rax is:0xf8000007f8000000

salq $1, %rax # the value of %rax is:0xf000000ff0000000

salq $1, %rax # the value of %rax is:0xe000001fe0000000

movq $0xff000000000000ff, %rax # the value of %rax is:0xff000000000000ff what do these 6 values look like in binary???

sarq $1, %rax # the value of %rax is:0xff8000000000007f arithmetic right shift for quadword

sarq $1, %rax # the value of %rax is:0xffc000000000003f

sarq $1, %rax # the value of %rax is:0xffe000000000001f

sarq $1, %rax # the value of %rax is:0xfff000000000000f

sarq $1, %rax # the value of %rax is:0xfff8000000000007

movq $0xff000000000000ff, %rax # the value of %rax is:0xff000000000000ff what do these 6 values look like in binary???

shrq $1, %rax # the value of %rax is:0x7f8000000000007f logical right shift for quadword

shrq $1, %rax # the value of %rax is:0x3fc000000000003f

shrq $1, %rax # the value of %rax is:0x1fe000000000001f

shrq $1, %rax # the value of %rax is:0xff0000000000000f

shrq $1, %rax # the value of %rax is:0x7f80000000000007

movq $0xff000000000000ff, %rax # the value of %rax is:0xff000000000000ff what do these 6 values look like in binary???

sarw $1, %ax # the value of %rax is:0xff0000000000007f shift arithmetic right

sarw $1, %ax # the value of %rax is:0xff0000000000003f

sarw $1, %ax # the value of %rax is:0xff0000000000001f

sarw $1, %ax # the value of %rax is:0xff0000000000000f

sarw $1, %ax # the value of %rax is:0xff00000000000007

movq $0xff000000000000ff, %rax # the value of %rax is:0xff000000000000ff what do these 6 values look like in binary???

shrw $1, %ax # the value of %rax is:0xff0000000000007f shift logical right

shrw $1, %ax # the value of %rax is:0xff0000000000003f

shrw $1, %ax # the value of %rax is:0xff0000000000001f

shrw $1, %ax # the value of %rax is:0xff0000000000000f

shrw $1, %ax # the value of %rax is:0xff0000000000007

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.

MovX could move a value from one place to another place. The X would be the size of the data. If the size is less than 8 bytes, the bigger data would still remain and the smaller amount of bytes would be changed by the X.

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

The cltq is used to change the long word to quad word. In this way, the 0x8fffffff would be negative. The cltq would change it into 0xffffffff8fffffff. Cltq would not have any operands.

1. 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).

The movsXX and movzXX would do the similar function which is move value from one place to another one. The movs would be used on the different sized registers like ax, eax. If the value is smaller than the register size. The values would be extended the sign of the value like 0 or f. Movz would just extend 0 at the front of the size. X could be s or w or l or q.

1. 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. 

Push could get the data from the register to store in the stack. The pop could give these data to the register. If the number is less than 8 numbers, the other bits would use 0 to finish. %rsp would increase the amount of data when it pushed into the register.

1. 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 condition code values would be changed when we use subtract, add, increment. The eflags would show the results. The subX, incX and addX would cause the change as I operate.

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

The instructions would change the value in the register due to the size. The And would do and the bitwise operation for the data in the registers. The or could do or operation, and the xor could do xor operation.

1. 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?

The first operand would be the number to shift, and the second operand would be the register which should shift its data. The size of the data would change the result in the register. Only the corresponding bits of data could be changed.

1. 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?

%rip would be changed when the instruction is operated. It would not change the same amount.

1. 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.

M: the program could not run due to the invalid instruction for push and pop

L: same errors. The reason could be the wrong size of the data.

1. Any other comments about what you observed?

The w or q or l or s would be important for the operations when we use the data. It could cause the different results

in the data of the registers.