CSE 2421
Spring, 2020
Homework 3

Due Date: Wednesday, April 15, 11:30PM. Only submissions via Carmen will be accepted without prior approval. To receive credit, solutions must be clearly written or typed and appear in the same order as the questions below.

1. Consider the two 2-byte values: 0x8181 and 0xFFFF.
   a. If you interpret these values as unsigned values and add them together, what hex representation results if you represent the sum as a 2-byte value?
   0x8180
   b. If these two values were converted to **4-byte unsigned values**, what hex representation results? Use the format: 0xXXXXXXXX, where the digits are numbered 0(least significant hex digit) to 7(most significant hex digit).
   0x00008181
   0x0000FFFF
   c. If you represent these two values as 2 **4-byte unsigned values** and add them together, what hex representation results? Represent your answer as some hexadecimal number in the format: 0xXXXXXXXX, where the digits are numbered 0(least significant hex digit) to 7(most significant hex digit).
   0x00008181 + 0x0000FFFF = 0x00018180
   d. Sum the largest 2-byte unsigned value possible with itself and represent the result as a 4-byte unsigned value.
   0xFFFF + 0xFFFF = 0x0001FFFE
   e. Given the 4-byte results in (c) and (d), what hex value(s) are possible for hex digit in position 4 when the sum of any 2 ***possible*** 2-byte values are represented in 4-bytes?
   0 or 1
   f. Could the OF or CF flag ever be set by performing (a)? Why?
   OF will not be set. CF will be set since there is an unsigned overflow.
   g. Could the OF or CF flag ever be set by performing (c) or (d)? Why?
   Either OF or CF will not be set since there is no overflow.
   h. If you add the largest 2-byte unsigned value possible with itself 10 times (e.g. x+x+x+x+x+x+x+x+x+x) with the result in a 4-byte value, what happens to the value in hex digit position 4 as you repeatedly add? Can you overflow a 4-byte value doing this repeated sum 10 times? Would there be there be some number of times that the sum could be added that would overflow the 4-byte register?
   The largest 2-byte unsigned value is 0xFFFF, and the hex position 4 will be F since at the first addition F + F = E with a carry, then in position 4, F + F + 1 = F with a carry; let's take a second run, at position 5, E + F = D with a carry, then in position 4, F + F + 1 = F with a carry. Now we got a pattern and realized the position 4 did not changed, and at round 10, the position 5 should be 6 (F – 9 = 6), 7 + F = 6 with a carry, therefore, F + F + 1 will still be F at the tenth time.
   In 4-byte value representation, if we left shift the largest 2-byte unsigned value by 1 (multiply by 16), the value will be changed to 0x000FFFF0, which still can be stored in a 4-byte register, therefore, there is no overflow when we doing this repeated sum 10 times.
   The maximum value of a 4-byte register can be stored is 0xFFFFFFFF (4294967295), if we divide it by 0xFFFF (65535), we will get the maximum time of sum (without overflow) which is 65537.
   i. Would any of your answers above (items (a) through (h)) change if the two values were interpreted as signed values instead of unsigned values? If so, how?
   (b) 0x8181 in binary will be 1000000110000001, if we interpreted as a 4-byte value it will become to 0xFFFF8181, and 0xFFFF will be 0xFFFFFFFF.
   (c) 0xFFFF8181 + 0xFFFFFFFF = 0xFFFF8180
   (d) 0xFFFF + 0xFFFF = 0xFFFFFFFE for signed 4-byte value.
   (f) OF will be set since it is a signed overflow.
2. Given the following assembler instructions:
   movw $0x8181, %di
   movw $0xffff, %si
   a. What instructions would you need to implement question 1(a)?
   addw %di, %si
   b. What instructions would you need to implement question 1(b)?
   movzwl %di, %edi
   movzwl %si, %esi

c. What instructions would you need to implement question 1(c)?
addl %edi, %esi

d. What instructions would you need to implement question 1(d)?
addl %esi, %esi

3. Consider the two 2-byte values: 0x8181 and 0xFFFF.

a. If you interpret these values as **unsigned values** and multiply them together, what hex representation results if you represent the product as a 2-byte value?
0x7E7F

b. If you represent these two values as 2 **4-byte unsigned values** and multiply them together, what hex representation results? Represent your answer as some hexadecimal number in the format: 0xXXXXXXXX, where the digits are numbered 0(least significant hex digit) to 7(most significant hex digit).
0x00008181 * 0x0000FFFF = 0x81807E7F

c. Multiply the largest 2-byte unsigned value possible with itself and represent the result as a 4-byte unsigned value.
0xFFFF * 0xFFFF = 0xFFFFFFFE

d. Given the 4-byte results in (b) and (c), what do you observe about the size of the values you get?
They have the same size.

e. Could the OF or CF flag ever be set by performing (a)? Why?
OF will not set since the results fits the 4-byte register, CF will be set because in 2-byte register carry occurs.

f. Could the OF or CF flag ever be set by performing (c) or (d)? Why?
No. Since there is no overflow in 4-byte register.

g. Would any of your answers above (items (a) through (g)) change if the two values were interpreted as signed values instead of unsigned values? If so, how?
We just interpreted as signed values, not convert, therefore, I do not think the answer will change.

4. Given the following assembler instructions:
movw $0x8181, %ax
movw $0xffff, %si

a. What instructions would you need to implement question 3(a)?
mulw %si

b. What instructions would you need to implement question 3(b)?
mulw %si
shll $16, %esi
movzwl %ax, %eax
orl %esi, %eax

c. What instructions would you need to implement question 3(c)?
movw %si, %ax
mulw %si
shll $16, %esi
movzwl %ax, %eax
orl %esi, %eax

5. Given the following assembler instructions:
movw $0x8181, %ax
movw $0xffff, %si

d. What instructions would you need to implement question 3(a) if we interpret the values as signed?
imulw %si, %ax

e. What instructions would you need to implement question 3(b) if we interpret the values as signed?
movzwl %ax, %eax
movzwl %si, %esi
imull %esi, %eax

f. What instructions would you need to implement question 3(c) if we interpret the values as signed?
movw %ax, %dx
movzwl %ax, %eax
movzwl %dx, %edx
imull %edx, %eax