15-213: Introduction to Computer Systems
Written Assignment #2

This written homework covers machine programming concepts, procedure control flow, and structs.

Directions

Complete the question(s) on the following pages with single paragraph answers. These questions are not meant to be particularly long! Once you are done, submit this assignment on Canvas.

Below is an example question and answer.

Q: Please describe benefits of two's-complement signed integers versus other approaches.

A: Other representations of signed integers (ones-complement and sign-and-magnitude) have two representations of zero (+0 and −0), which makes testing for a zero result more difficult. Also, addition and subtraction of two's complement signed numbers are done exactly the same as addition and subtraction of unsigned numbers (with wraparound on overflow), which means a CPU can use the same hardware and machine instructions for both.

Grading

Assignments are graded via *peer review:*

1. Three other students will each provide short, constructive feedback on your assignment, and a score on a scale of 1 to 15. You will receive the maximum of the three scores.
1. You, in turn, will provide feedback and a score for three other students (not the same ones as in part 1). We will provide a *rubric*, a document describing what good answers to each question look like, to assist you. You receive five additional points for completing all of your peer reviews.

Due Date

This assignment is due on Wednesday, June 8 by 11:59pm Pittsburgh time (currently UTC−4). Remember to convert this time to the timezone you currently reside in.

Peer reviews will be assigned roughly 12 hours later, and are due a week after that.

Question 1

1. Suppose `%rdi` holds the value `0xA0A0` and `%rsi` holds the value `0x1F0F`. Provide the address calculation and the calculated address of the following:
   ```
   a. 0x8(%rdi)
   b. (%rdi, %rsi)
   c. (%rdi, $rsi, 2)
   d. 0x80(,%rdi, 2)
   ```
2. What does `lea (%rsp, %rcx, 4), %rax` put into `%rax`? How is this different from what `mov (%rsp, %rcx, 4), %rax` puts into `%rax`?

1a. %rdi + 0x8 = 0xA0A8
1b. $rdi + $rsi = 0xBFAF
1c. $rdi + 2 * $rsi = 0xDEBE
1d. 2 * $rdi + 0x80 = 0x141C0

2. mov will move the value stored in Src into %rax, where parentheses indicate pointer dereferencing, so what is stored at the memory address %rsp + 4 * %rcx. lea will move the calculated value into %rax, without dereferencing the value (remains as a pointer), so the calculated value of %rsp + 4 * %rcx.

Question 2

Consider the following x86-64 assembly function:

```
loopy:
        # a in %rdi, n in %esi
        mov         $0, %ecx
        mov         $0, %edx
        test        %esi, %esi
        jle         .L3
.L6
        movslq      %edx, %rax
        mov         (%rdi, %rax, 4), %eax
        cmp         %eax, %ecx
        cmovl       %eax, %ecx
        add         $1, %edx,,
        cmp         %ecx, %esi
        jg          .L6
.L3
        mov         %ecx, %eax
        ret
```

Fill in the blanks of the corresponding C code. Use the C variable names n, a, i, and x, not register names. Use array notation in showing accesses or updates to elements of a.

```
int loopy(int a[], int n)
{
        int i;
        int x = _(a)_;

        for (i =  _(b)_;  _(c)_;  _(d)_) {
            if (_(e)_)
                    _(f)_;
        }
        return x;
}
```

a. o
b. o
c. x < n
d. i++
e. x < a[i]
f. x = a[i]

Question 3

1. Wow Abi found a coolFunction! but unfortunately she can't find the code for the function mystery. All she has is this assembly snippet. What value should Abi call coolFunction with, in order to make it print out "left shark"? Please explain briefly why your answer works.

```c
int coolFunction(int n) {
    if (mystery(n) == 1) {
        printf("left shark\n");
    } else {
        printf("boo\n");
    }
    return 0;
}
```

doSomething:
```
0x0000000000400529:   sub        $0x10, %rsp
0x000000000040052a:   mov        %rbx, 8(%rsp)
0x000000000040052c:   mov        %edi, %ebx
0x000000000040052d:   test       %ebx, %ebx
0x000000000040052f:   jle        40053d <doSomething+0x14>
0x0000000000400531:   mov           %ebx, %edx
0x0000000000400533:   sub        $0x1, %edx
0x0000000000400536:   jne        400533 <doSomething+0xa>
0x0000000000400538:   lea        (%rbx, %rbx,1),%eax
0x000000000040053a:   mov        8(%rsp), %rbx
0x000000000040053b:   add        $0x10, %rsp
0x000000000040053c:   ret
0x000000000040053d:   mov        $0x0, %eax
0x000000000040053e:   mov        8(%rsp), %rbx
0x000000000040053f:   add        $0x10, %rsp
0x0000000000400541:   ret
```

mystery:
```
0x0000000000400542:   call       400529 <doSomething>
0x0000000000400547:   cmp        $0x1aa, %eax
0x000000000040055c:   sete       $al
0x000000000040055e:   movzbl     $al, %eax
0x0000000000400560:   ret
```

2. What does **doSomething** do in order to safely use the callee-save register, %rbx?

1. In order to return "left shark", we need **mystery** to return 1. **mystery** passes its argument to **doSomething,** and then returns 1 if the value returned by **doSomething** is equal to 426. **doSomething** takes its argument (in %rdi) and returns that value multiplied by 2 (saying that **doSomething** adds the value in %rdi to itself and returns that is also valid). Therefore, the correct input is **213**.
2. The assembly safely uses %rbx by saving it through the **mov %rbx, 8(%rsp)** command and then later restoring %rbx to its original value with **mov 8(%rsp), %rbx** before **doSomething** returns**.**

Question 4

```
typedef struct babyShark_t{
      char s;
      int h[3];
      long a;
      char *r;
      short k;
} babyShark_t;

typedef struct mommaShark_t{
      char a[3];
      babyShark_t b;
      int c;
} mommaShark_t;
```

1. We learned in lecture that a struct may need to have padding at the end. Why is this padding sometimes necessary?

Padding at the end of the struct is necessary to ensure each struct in an array of structs is properly aligned.

2. Imagine that the table below represents 64 bytes of memory and a mommaShark_t object begins at the beginning of the table (top left corner, addresses increase left to right and then top to bottom). In each cell, indicate the field of the mommaShark_t object that stretches across that byte. Be as specific as possible. Use x to indicate padding.
   a. If a mommaShark_t object begins at address 0x4000, what is the address of b.h[2] within that object?

mommaShark_t:

a[0]	a[1]	a[2]	x	x	x	x	x	b.s	x	x	x	b.h[0]	b.h[0]	b.h[0]	b.h[0]
b.h[1]	b.h[1]	b.h[1]	b.h[1]	b.h[2]	b.h[2]	b.h[2]	b.h[2]	b.a	b.a	b.a	b.a	b.a	b.a	b.a	b.a
b.r	b.r	b.r	b.r	b.r	b.r	b.r	b.r	b.k	b.k	x	x	x	x	x	x
c	c	c	c	x	x	x	x								

0x4000 + 20 / 0x4000 + 0x14 = 0x4014