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

2. %rax = %rsp + 4 * %rcx
lea only calculates the memory address and puts it inside %rax, mov puts the quad-byte located at that address into %rax

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. 0
b. 0
c. i < 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.

```
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.
0xD5

The C code of doSomething and mystery is:

```
int doSomething(int a) {
    if (a <= 0) return 0;
    for (int i = a; i > 0; --i);
    else return 2*a;
}

int mystery(int a) {
    if (doSomething(a) == 0x1AA) return 1;
    return 0;
}
```

So to print “left shark” the return value of mystery should be 1. Then the return value of doSomething should be 0x1AA. It’s easy to see that the input of mystery cannot be 0. Then the input should be 0x1AA / 2 = 0xD5

2. doSomething pushes %rbx to the stack at the beginning, and pops the value back into %rbx at the end.

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?

Because sometimes some of the fields in the struct is not aligned correctly, and padding is used to skip some bytes so that the fields are aligned.

2. Imagine that the table below represents 48 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								

2. Assuming sizeof(long) = 8, sizeof(int) = 4, sizeof(short) = 2, sizeof(char) = 1, sizeof(void*) = 8, then the alignment of babyShark_t is 8, that of mommaShark_t is also 8

2a. The offset of b.h[2] relative to the beginning of mommaShark_t is 0x14. Therefore, &b.h[2] = 0x4014