CSci 2021, Fall 2018
Homework Assignment II: Solution Set
Due: Monday, October 15th, at beginning of lecture

Submit this assignment on paper at **the beginning** of your lecture section. We strongly recommend that you type and print out your solutions. Please label your assignment with your name, UMN email address, and the time of your recitation section (10:10, 11:15, 12:20, or 1:25).

Problem 1
Consider the following C code for a function with a `for` loop:

```
int count_up_to(long n){
        long x = 1;
        for (long i = 0; i < n; i++){
                if (x == i) {
                        x = x * 5;
                }
                else {
                        x = x / 5;
                }
        }
        return x;
}
```

Based on the C code above, fill in the blanks below in its corresponding assembly source code. You may only use the assembly-language register names `rax`, `rcx`, `rdi` or `rsi`. You may wish to review section 3.5.5 of the textbook for some relevant instructions.

```
count_up_to:
    movq  $1, %rax
    movq  $0, %rcx
.L1:
    cmpq  %__rdi__, %_rcx___
    jge   .L4
.L2:
    cmpq  %rax, %rcx
    __jne__   .L3
    imulq $5, %rax
    addq  $1, %rcx
    jmp   ____.L1____
.L3:
    cqto
    movq  $_5__, %_rsi__
    _idivq__ %rsi
    addq  $1, %rcx
    jmp   __.L1__
.L4:
    ret
```

Problem 2

Consider the table below, which shows the initial contents of some registers and memory locations:

Initial Values			
Registers	Values	Memory	Values
rax	10	0x2FF8	38
rdx	40	0x3000	190
rcx	20	0x3008	3
rbx	0x3008	0x3010	68

a. Fill in Table 1 showing the results if the following machine code is run from the initial state:

```
movq  $0,   %rax
movq  $100, %rdx
addq  %rcx, %rax
imulq %rax, %rdx
```

Table 1			
Registers	Values	Memory	Values
rax	20	0x2FF8	38
rdx	2000	0x3000	190
rcx	20	0x3008	3
rbx	0x3008	0x3010	68

b. Fill in Table 2 showing the results if instead the following machine code is run from the initial state: The solution of the first table is for address addition that goes from high to low. This is how that CPU actually interprets positive offsets for the stack and thus is the correct solution for this problem. The second table was not counted against students, but the first table's soltions should be followed in the future.

```
leaq  8(%rbx), %rax
movq  $150,    (%rax)
movq  $1,      %rdx
addq  %rcx,    (%rbx, %rdx, 8)
imulq %rdx,    %rcx
```

Table 2: low to high WAS NOT COUNTED WRONG FOR GRADING			
Registers	Values	Memory	Values
rax	0x3010	0x2FF8	38
rdx	1	0x3000	190
rcx	20	0x3008	3
rbx	0x3008	0x3010	170

Table 2: high to low CORRECT SOLUTION			
Registers	Values	Memory	Values
rax	0x3000	0x2FF8	38
rdx	1	0x3000	170
rcx	20	0x3008	3
rbx	0x3008	0x3010	68

Problem 3

This is the assembly associated with the function: `long function_A(long n)`:

```
function_A:
            cmpq        $1, %rdi
            jle         .L5
            movl        $1, %edx
            movl        $1, %eax
            jmp         .L3
.L4:
            imulq       %rdx, %rax
            addq        $1, %rdx
.L3:
            cmpq        %rdi, %rdx
            jle         .L4
            rep ret
.L5:
            movl        $1, %eax
            ret
```

A. Write C code that corresponds to the assembly given above. Give the variables meaningful names, not the names of registers.

B. Explain in a sentence or two what this function does.

Problem 3 Solution

A.

```
long function_A(long n){
  if (n < 2){
    return 1;
  }
  long product = 1;
  for (long i = 1; i <= n; i++){
    product *= i;
  }
  return product;
}
```

B. This function computes n!.

Problem 4: (Conditional moves and jumps.)

A. The following assembly code implements a simple function whose definition has three different cases:

```
three_cases:
        xorq    %rax, %rax
        movq    $-1, %rdx
        testq   %rdi, %rdi
        setne   %al
        cmovs   %rdx, %rax
        ret
```

Rewrite the function so that it computes the same result, but uses conditional jump instructions jXX and labels, and does not use conditional move or set-condition (cmovXX or setXX) instructions.

B. (Based on the textbook problem 3.61.) The following C function loads a 64-bit value from a pointer, but if the pointer is null it just returns the value -1, so it never dereferences a null pointer.

```
long safe_load(long *p) {
    if (p)
        return *p;
    else
        return -1;
}
```

At first this might seem like a kind of choice that cannot be implemented using a conditional move, since it would be bad to always dereference the pointer, and then return either the loaded value or -1 based on whether the pointer was null. The function will crash at the time it tries to load from a null pointer, even if the loaded value is later not used. However, we could give the function a different structure to avoid this problem and still use a conditional move. First, to get the core idea, show how to rewrite the function in C without any conditional side-effects. Specifically, write a function that has the same behavior, but does not use an `if` statement or other control flow. You should use the `? :` ternary operator, but only in a case where all the arguments to the `? :` are simple variables. (Hint: the function will still have to do a dereference, but the dereference will have to be unconditional. In other words, the function needs to dereference *something* every time in executes.)

Now, write an assembly-language version based on your new C version, which uses only conditional move instructions, and no conditional jump instructions or labels.

Problem 4 Solution

```
a. (10pts)
three_choices: # gcc 8.2 -Og:
        testq   %rdi, %rdi
        js      .L3
        jle     .L4
        movl    $1, %eax
        ret
.L3:
        movq    $-1, %rax
        ret
.L4:
        movl    $0, %eax
        ret
```

```
b. (24 pts)

long safe_load(long *p) {
    long m1 = -1;
    long *safep = &m1;
    int is_null = !p;
    long *p2 = is_null ? safep : p;
    return *p2;
}


safe_load:
        subq    $16, %rsp
        movq    $-1, 8(%rsp)
        leaq    8(%rsp), %rax
        testq   %rdi, %rdi
        cmove   %rax, %rdi
        movq    (%rdi), %rax
        addq    $16, %rsp
        ret
```

Problem 5

The following C code defines a simple linked list data structure.

```
struct Linked_List {
        int val;
        struct Linked_List *next;
};
```

Next is a function in assembly with the given function definition:

```
int link_func_A(struct Linked_List *s, int v);
```

The assembly for the function was produced with GCC.

```
link_func_A:
        jmp         .L2
.L4:
        cmpl        %esi, 0(%rdi)
        je          .L5
        movq        8(%rdi), %rdi
.L2:
        testq       %rdi, %rdi
        jne         .L4
        movl        $0, %eax
        ret
.L5:
        movl        $1, %eax
        ret
```

(Hint: The value of the struct is accessed by accessing the register with `0(%rcx)`, or at a zero offset of the register. To access the next pointer you will need an offset of 8, so the corresponding access would look like `8(%rcx)`.)

A. Write the C code that corresponds to the assembly. Make sure to not use register names for variables. (Hint: You should use a while loop. Can you find the loop condition?)

B. Explain in a sentence or two what this function does.

Problem 5 Solution

A. The problem checks if the pointer is NULL by testing against itself bitwise (`testq %rdi, %rdi`). The moving of pointers is done by moving the address the register is pointing at with offsets of 0s and 8s. An offset of 0 gets the value, and an offset of 8 gets the next pointer `next`.

```
int link_func_A(struct Linked_List *s, int v){
  while (s != NULL){

    if (s->val == v){
      return 1;
    }
    s = s->next;
  }
  return 0;
}
```

B. The function searches for the value `v` in a linked list and returns 1 if it is found and 0 if it not found.