CS 224 - Lab 8

100 Points

This assignment will develop our understanding of the Y86-64 instruction set. You will write simple Y86-64 programs to complete a series of tasks. You should use the online Y86-64 Simulator at https://boginw.github.io/js-y86-64/ to develop and test your answers.

What should you turn in?

Turn in a single zipped folder named `lab8.zip`. This zipped folder should contain exactly three files, named `1.txt`, `2.ys`, `3.ys`, and `4.ys`. These should have your solutions to each problem in this lab. Your solutions for questions 2-4 should only contain the code you inserted into the starter code, not any of the starter code itself.

Note: a .ys file is simply a .txt file (in plain text) with the file type changed. The easiest way to create this is using something like notepad or another text editor. Copy the assembly code from the online simulator into your text editor, then save it as a .ys file. You might need to click "All file types" or something similar to be able to do this" Or you can save it as a .txt file and then change the file type through your file explorer. Please let the TAs know if you have any questions.

1. **[18 points]** In this problem we will explore short sequences of one to three Y86-64 instructions for performing steps that show up frequently in more useful programs.

   (a) **[3 points]** Give a single instruction that will place the value 10 in register `%rax`.

   (b) **[3 points]** Provide a single instruction that will add the value in register `%rax` to the value in register `%rcx`, storing the result in register `%rcx`.

   (c) **[3 points]** Provide a sequence of instructions that will check to see if the value in register `%rax` is zero, and if it is, begin executing the instructions starting at address label `ifzero`.

   (d) **[3 points]** Provide a sequence of instructions that will swap the values in register `%rax` and register `%rbx`.

For the next two questions, consider the following portion of assembly code, which places two quad-word (64-bit) values in memory, and names the locations `a` and `b` respectively.

```
.align 8
a:
        .quad 0xaaaa
b:
        .quad 0x0000
```

(e) **[3 points]** Provide a sequence of instructions that will place the 64-bit value at memory location `a` into register `%rdx`

(f) **[3 points]** Provide a sequence of instructions that will place the 64-bit value `0xbbbb` into memory at location `b`.

2. **[25 points] Copying a block of memory**

   In this problem you will write a Y86-64 assembly program that copies a block of 64-bit values from one part of memory, labeled `src`, to another, labeled `dest`. You may assume that the two blocks of memory do not overlap. Your code should assume that the number of 64-bit values contained in each block is given by the value in register `%rax`.

   Starter code to help you get going is given below. This code puts the number of values to copy (3) into register `%rax` and also initializes and labels the `src` and `dest` blocks of values. You should test your program using this code skeleton. After your code runs, the `dest` block of memory should have the same contents as the `src` block of memory. You should also modify it to have blocks with a different number of values and ensure that it still copies them correctly.

   Your code should be inserted where the comment says `# YOUR CODE HERE`. This program should take roughly 12-17 assembly instructions to complete.

Memory Copy Starter Code:

```
irmovq $3,%rax

# YOUR CODE HERE

.align 8
# Source block
src:
    .quad 0xaaaa
    .quad 0xbbbb
    .quad 0xcccc

# Destination block
dest:
    .quad 0x1111
    .quad 0x2222
    .quad 0x3333
```

3. **[25 points] Summing elements of a linked list**

   In this problem you will an write a Y86-64 assembly program that sums the numbers stored in a linked list. Each element of the linked list consists of 128-bits/16 bytes of memory. The first 8 bytes contain a 64-bit value (the value that you should sum). The next 8 bytes contain the address of the next element in the linked list. If this is the last element of the linked list, then the second 8 bytes will contain the value 0. The address of the first element of the linked list will have the label `list`.

   Your program should finish with the sum of all the elements of the linked list in register `%rax`. You may assume that there is at least one element in the linked list.

   Starter code to help you get going is given below. It initializes the elements of the linked list and labels the firsts element `list`. The other labels are used internally in the linked list and should not be used by your program. In other words, your code should work for any modification to the linked list's internal labels. For this example linked list the correct sum is `0x10111`. You should test your program using this code. You should also modify the linked list and ensure that your program still sums the elements correctly.

   Your code should be inserted where the comment says `# YOUR CODE HERE`. This program should take roughly 8-13 assembly instructions to complete.

Linked List Starter Code:

```
# YOUR CODE HERE

# Sample linked list
.align 8
list:
ele1:
    .quad 0x01
    .quad ele2
ele2:
    .quad 0x10
    .quad ele3
ele3:
    .quad 0x100
    .quad ele4
elex:
    .quad 0x1000
    .quad ele1
ele4:
    .quad 0x10000
    .quad 0
```

4. **[32 points] Sorting numbers in a list with Bubble-sort**

   In this problem you will implement the Bubble-sort algorithm in Y86-64 assembly. This algorithm will sort a list of 64-bit numbers that are stored in memory. The address of the first number to sort will have the label `data`. You can assume that the number of numbers that you should sort will be in register `%rax`. You may assume that there are at least two numbers for you to sort.

   Starter code to help you get going is given below. It initializes a chunk of memory with a list of 10 numbers and gives the label `data` to the the address of the first number. It also places the number of numbers to sort (10) in register `%rax`. You should test your program using this code. If it works correctly the numbers should appear in the same memory locations, but now ordered from smallest to largest. You should also modify the list of numbers (both the number of numbers and their values and order) and ensure that your program still sorts the numbers correctly.

   Your code should be inserted where the comment says `# YOUR CODE HERE`. This program should take roughly 23-28 assembly instructions to complete.

   The Bubble-sort Algorithm

   Since you will be implementing the bubble-sort algorithm, we will give you a description of how this algorithm works.

   Given a list of n numbers, bubble-sort repeatedly passes through the list, "bubbling" the larger values up towards the end through the use of swaps. Two elements are swapped if they are out of order, i.e. if the larger one appears first in the list. If nothing is swapped on a pass through the list then the algorithm stops.

   Here is pseudo-code for the algorithm.

```
procedure bubble-sort(A : list of numbers)
    n = number of numbers in A
    repeat
        swapped = false
        for i := 0 to n - 2
            if  A[i] > A[i + 1] then
                swap A[i] with A[i + 1]
                swapped = true
            end if
        end for
    until not swapped
end procedure
```

   Disclaimer: The bubble-sort algorithm is *not* a good sorting algorithm. It is just a simple one to describe and implement. In the future when you need a sorting algorithm, we *do not* recommend bubble-sort!

Bubble-sort Starter Code:

```
irmovq $10,%rax

# YOUR CODE HERE

.align 8
data:
    .quad 0x05
    .quad 0x0a
    .quad 0x08
    .quad 0x02
    .quad 0x01
    .quad 0x03
    .quad 0x06
    .quad 0x04
    .quad 0x09
    .quad 0x07
```