6.1100 Spring 2024 Miniquiz #4

Please submit your answers on Gradescope by March 7th, 2024, 11:59pm.

Name: 6.110 Staff

Email: 6.110-staff@mit.edu

1. x86 Introduction

Ben Bitdidle is trying to find his favorite number in a list of numbers. He has started writing a x86 assembly program to do this. He has completed most of his program:

```
find_magic_num(unsigned int*, int):
    pushq   %r15
    pushq   %r14
    pushq   %r13
    pushq   %r12
    pushq   __#1__
    movl    $-1, %ebx
    __#2__  %esi, %esi
    jle     .LBB2_5
    movq    %rdi, %r14
    movl    %esi, %r15d
    xorl    %r12d, %r12d
    xorl    %r13d, %r13d
.LBB2_3:
    addl    %r12d, %r13d
    movl    (%r14,%r12,4), __#3__
    callq   popcnt(unsigned int)
    ________#4________
    je      .LBB2_4
    incq    %r12
    cmpq    %r12, %r15
    jne     .LBB2_3
    jmp     .LBB2_5
.LBB2_4:
    movl    %r13d, %ebx
```

(Continued on the next page.)

```
.LBB2_5:
    movl    %ebx, %eax
    popq    %rbx
    ________#5________
    retq
```

But as you can see, some parts of his program are incomplete. He would like your help to complete it! The final program should be equivalent to the following C code:

```
int find_magic_num(unsigned int* y, int N) {
    int x = 0;
    for (int i = 0; i < N; i++) {
        x += i;
        if (popcnt(y[i]) == 4) {
            return x;
        }
    }
    return -1;
}
```

He wants his code to follow the x86 calling convention and was told that the `popcount()` also follows this convention. He also knows that **#1** should be a register, **#2** should be a x86 instruction name, **#3** should be a register, **#4** should be *one* x86 instruction, and **#5** should be *four* x86 instructions.

#1: %rbx

#2: testl

#3: %edi

#4: cmpl $4, %eax or cmpq $4, %rax

#5: popq %r12

popq %r13

popq %r14

popq %r1

2. Stacks and Addressing

Excited by the success of his magic numbers program, he wants to move on to more advanced programs that use the stack. He is writing a program that is equivalent to the following C function:

```
int histogram(unsigned int* x, int N) {
    int freq[100];
    for (int i = 0; i < N; i++) {
        if (x[i] > 98) {
            freq[99]++;
        } else {
            freq[x[i]]++;
        }
    }
    return 0;
}
```

This is what he has written so far:

```
histogram(unsigned int*, int):
    testl   %esi, %esi
    jle     .LBB3_4
    subq    $280, %rsp
    leaq    268(%rsp), %rax
    movl    %esi, %ecx
    xorl    %edx, %edx
.LBB3_2:
    movl    ____#1____, %esi
    cmpq    $99, %rsi
    leaq    ____#2____, %rsi
    cmovaeq %rax, %rsi
    incl    (%rsi)
    incq    %rdx
    cmpq    %rdx, %rcx
    jne     .LBB3_2
    addq    ____#3____, %rsp
.LBB3_4:
    xorl    %eax, %eax
    retq
```

(Questions are on the next page.)

a. Which instruction allocates space on the stack? (Use the line numbers)

Line 4 (subq $280, %rsp)

b. How is instruction on line 10 being used in the code? Which other instruction makes use of the change in the status register, and how?

cmovaeq performs a conditional move based on the status register. If the condition is satisfied (99 <= %rsi) then the address in %rax, which contains &freq[99], is moved into %rsi

c. What C code line(s) does assembly line 13 best correspond to?

Lines 5 and 7 (the increments to the freq array)

d. What should he put in place of **#1**?

(%rdi, $rdx, 4)

e. What should he put in place of **#2**? (Note: you can assume red zones)

-128(%rsp, %rsi, 4)

For people not familiar with red zones: they are something used in Linux and MacOS which guarantee that the 128 bytes above stack space are safe to use and will not be overwritten by the OS. For tail-functions that utilize ≤128 bytes of stack space, this saves the 2 instructions associated with reserving stack space.

f. What should he put in place of **#3**?

`$280`