CS107 November 4th, 2026

Autumn 2026

CS107 Quiz 5 Solution

This is a closed book, closed note, closed computer quiz. You have 10 minutes to complete it.

SUNet ID (**`@stanford.edu`**): _____

First and Last Name: _____

I accept the letter and spirit of Stanford's Honor Code.

[signed] _____

Do not turn over until instructed to do so.

Problem 1: Code Generation and Reverse Engineering [8 points, 2 points per blank]

Given the **gcc**-generated x86, determine what each missing line of C code must be.

Recall that two-operand instructions like **add**, **and**, and **or** store results in the second operand. And remember that the most general memory operand, **D(B,I,S)**, refers to the address **B + I·S + D**, where **B** and **I** are registers, **S** is 1, 2, 4, or 8, and **D** is a constant. Any of the four pieces may be omitted.

When writing your answers, use the lines below the code boxes, not the ones within them.

a. Recall that **%rdi** and **%rsi** store the first two parameters, which here are **arr** and **k**. Assume the first **mov** instruction maps to **(i)** and the next two map to **(ii)**.

C Code:

```
void
egg(long arr[], long k) {
  ____________________; (i)
  ____________________; (ii)
}
```

x86:

```
egg:
   mov %rsi, (%rdi)
   mov 8(%rdi), %rax
   add %rax, (%rdi,%rsi,8)
   ret
```

i. **arr[0] = k**

ii. **arr[k] += arr[1]**

b. Assume the **lea** is generated from **(i)**, and the others are generated from **(ii)**. And remember that **%rax** stores the return value.

C Code:

```
unsigned long
ham(unsigned long x,
    unsigned long y) {
  x *= _____________; (i)
  return ___________; (ii)
}
```

x86:

```
ham:
   lea (%rdi,%rdi,8), %rax
   and $0xfff0, %rax
   or  %rsi, %rax
   ret
```

i. **9**

ii. **(0xfff0 & x) | y**

Problem 2: `void *`s in x86 [2 points]

Consider the C function on the left, which accepts a generic address alongside two offsets and returns a generic address computed using all three. Note that **%rdx** stores the third parameter.

```
void *salty(void *base, long m, long n) {
   return (char *)((int *) base + m) + n;
}
```

```
salty:
   lea (%rdx,%rsi,____), %rax
   add %rdi, %rax
   ret
```

Inspect the corresponding assembly on the right, which is complete, save for the missing third piece of **lea**'s first argument. Which of 1, 2, 4, or 8 must that be? Explain why, but be brief.

%rsi backs the parameter **m**, which, because it's being added to an **int ***, must be scaled by **sizeof(int) == 4**.