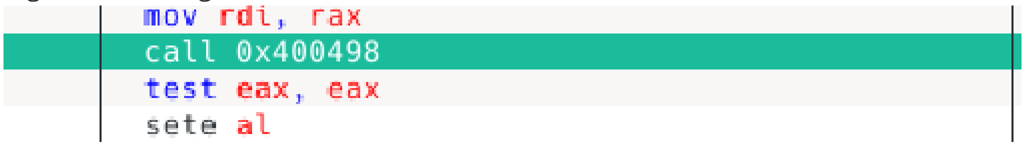
Additional Assignment2

(1) a high-level description of what you did

Right out of the gate, in EDB, I found there was a function call:

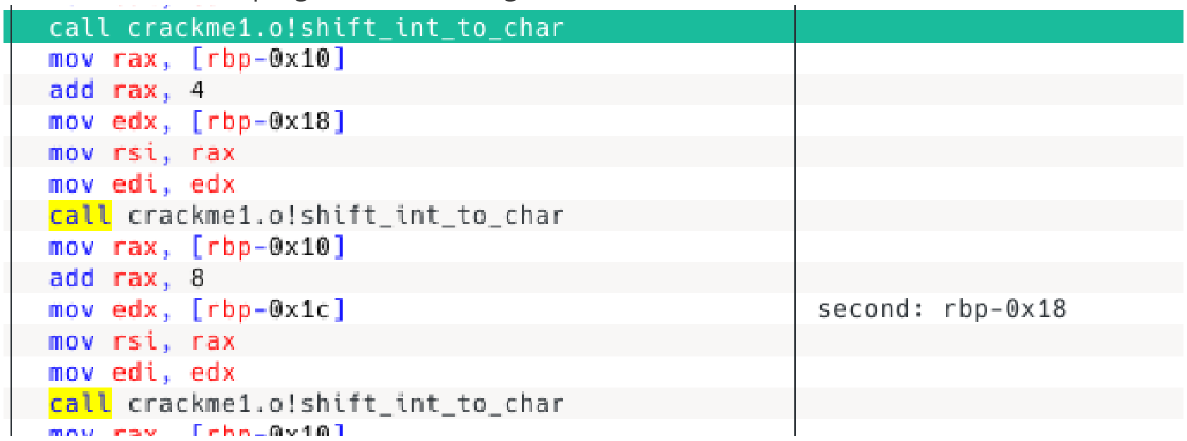


```
mov rdi, rax
call 0x400498
test eax, eax
sete al
```

around the end of the program. Looking at it in a debugger, I figured it was probably a `strcmp`. So in order to make the return value 0(meaning the input string matches the correct string), I tried to make it compare the correct string to itself. But for some reason, it does not return 0 even though I tried both `*rdi = *rsi` and `*rsi = *rdi` to make the two arguments the same.

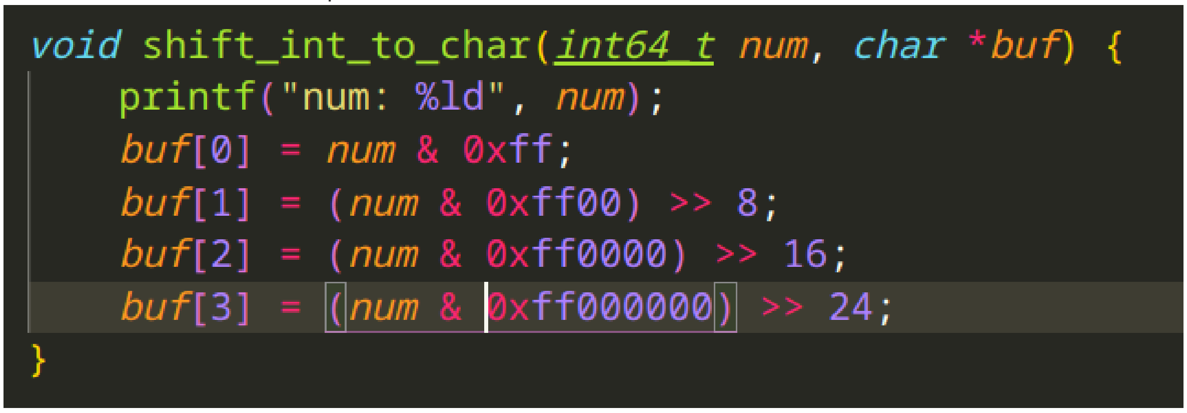
So my first attempt to force the program into the success branch failed.

So I ran the program with a debugger again and found that the function `shift_int_to_char` was called 3 times in the program, each writing 4 characters into a buffer of size 15.



```
call crackme1.o!shift_int_to_char
mov rax, [rbp-0x10]
add rax, 4
mov edx, [rbp-0x18]
mov rsi, rax
mov edi, edx
call crackme1.o!shift_int_to_char
mov rax, [rbp-0x10]
add rax, 8
mov edx, [rbp-0x1c]                         second: rbp-0x18
mov rsi, rax
mov edi, edx
call crackme1.o!shift_int_to_char
mov rax, [rbp-0x10]
```

Upon further inspection, I found that the function `shift_int_to_char` converts a 4 byte number into 4 characters. So I reimplemented the function as shown below:

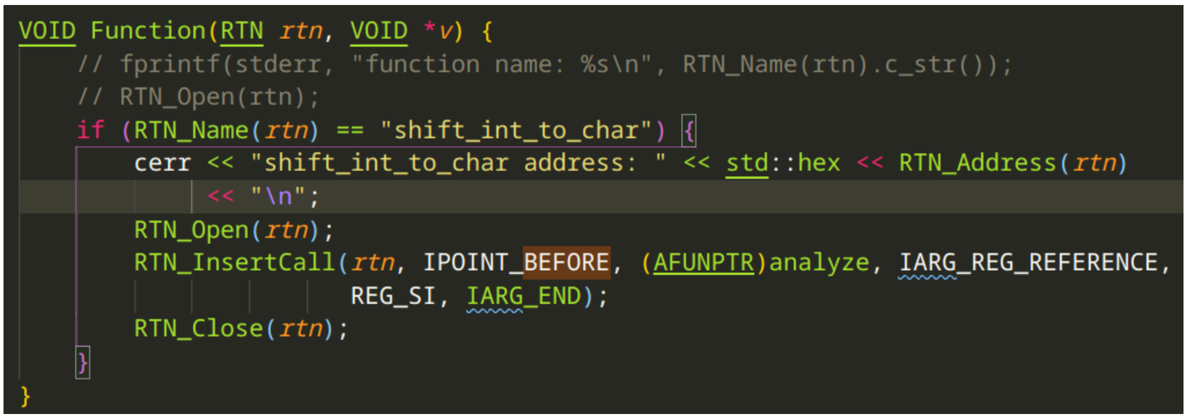


```c
void shift_int_to_char(int64_t num, char *buf) {
    printf("num: %ld", num);
    buf[0] = num & 0xff;
    buf[1] = (num & 0xff00) >> 8;
    buf[2] = (num & 0xff0000) >> 16;
    buf[3] = (num & 0xff000000) >> 24;
}
```

.

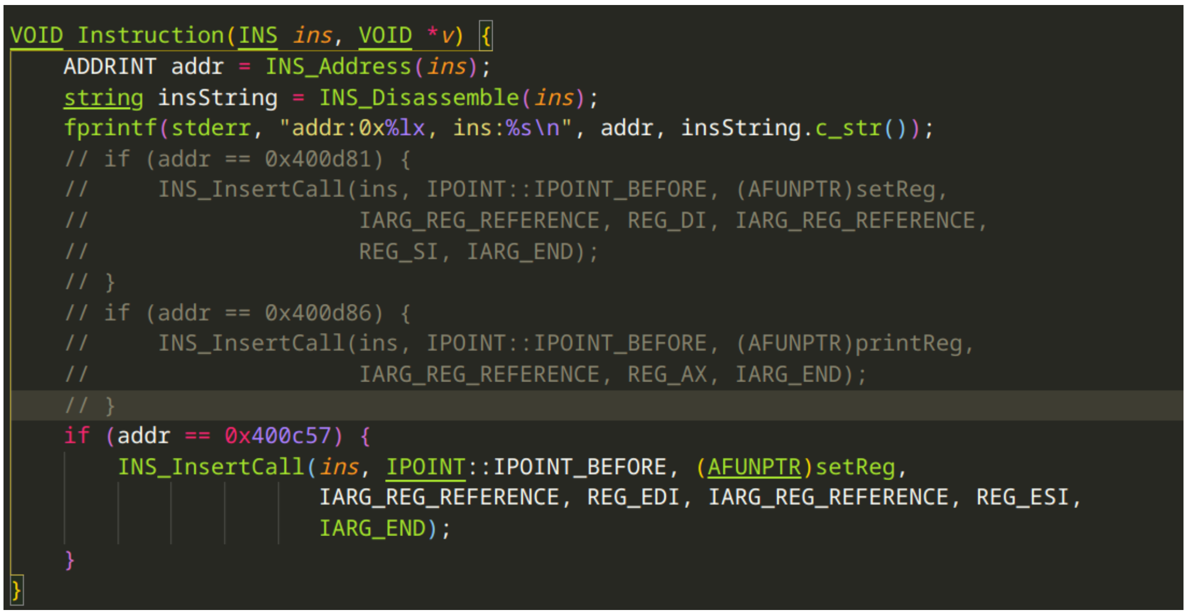
To get the correct input string, what I needed to do next was locate the calls to `shift_int_to_char` and get the first argument value, and feed it into my implementation to get the result string.

In order to locate the function calls, I found the address of `shift_int_to_char` with the instrumentation below:



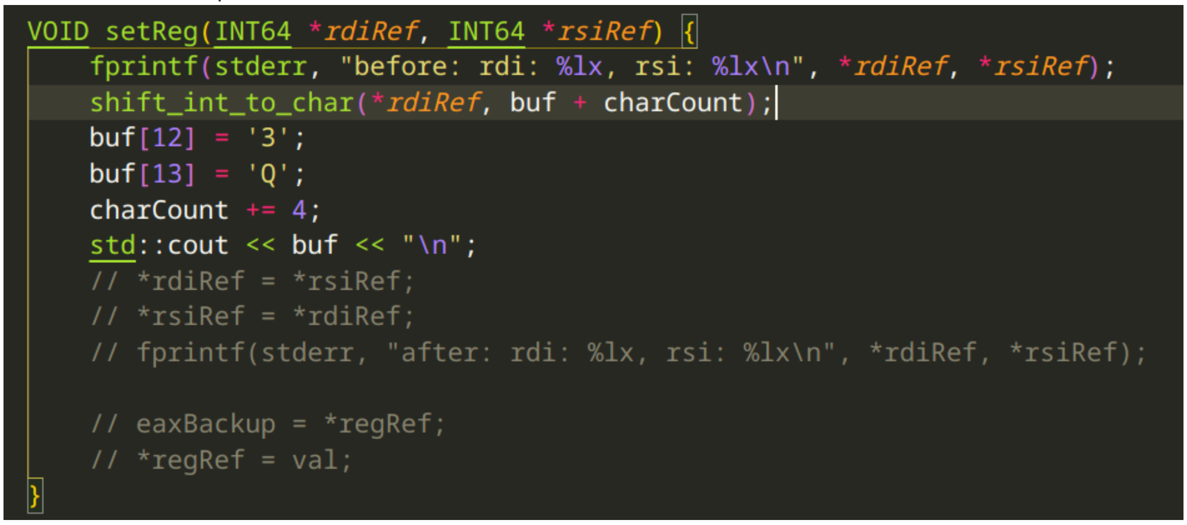
```
VOID Function(RTN rtn, VOID *v) {
    // fprintf(stderr, "function name: %s\n", RTN_Name(rtn).c_str());
    // RTN_Open(rtn);
    if (RTN_Name(rtn) == "shift_int_to_char") {
        cerr << "shift_int_to_char address: " << std::hex << RTN_Address(rtn)
             << "\n";
        RTN_Open(rtn);
        RTN_InsertCall(rtn, IPOINT_BEFORE, (AFUNPTR)analyze, IARG_REG_REFERENCE,
                       REG_SI, IARG_END);
        RTN_Close(rtn);
    }
}
```

Having found the address(0x400c57), now I could insert some calls(setReg) before the function runs:



```
VOID Instruction(INS ins, VOID *v) {
    ADDRINT addr = INS_Address(ins);
    string insString = INS_Disassemble(ins);
    fprintf(stderr, "addr:0x%lx, ins:%s\n", addr, insString.c_str());
    // if (addr == 0x400d81) {
    //     INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)setReg,
    //                    IARG_REG_REFERENCE, REG_DI, IARG_REG_REFERENCE,
    //                    REG_SI, IARG_END);
    // }
    // if (addr == 0x400d86) {
    //     INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)printReg,
    //                    IARG_REG_REFERENCE, REG_AX, IARG_END);
    // }
    if (addr == 0x400c57) {
        INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)setReg,
                       IARG_REG_REFERENCE, REG_EDI, IARG_REG_REFERENCE, REG_ESI,
                       IARG_END);
    }
}
```

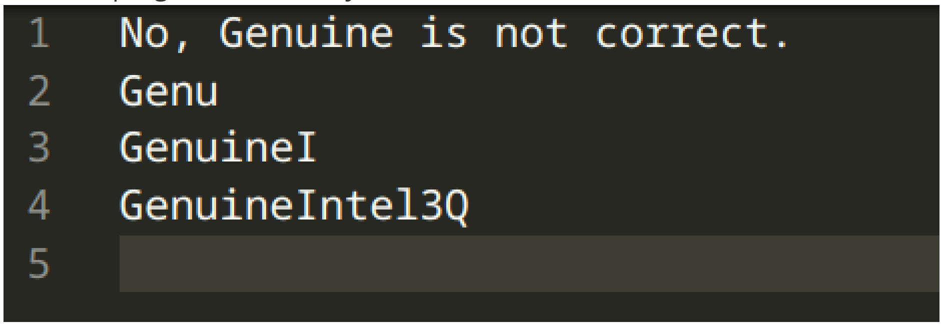
In `setReg`, I run my own `shift_int_to_char` to convert the number to chars and save it in a buffer. Since the last characters are set to 3Q in both crackme1 and crackme2, I set the last chars to 3Q here. I also print the buffer whenever this function is called:



```
VOID setReg(INT64 *rdiRef, INT64 *rsiRef) {
    fprintf(stderr, "before: rdi: %lx, rsi: %lx\n", *rdiRef, *rsiRef);
    shift_int_to_char(*rdiRef, buf + charCount);
    buf[12] = '3';
    buf[13] = 'Q';
    charCount += 4;
    std::cout << buf << "\n";
    // *rdiRef = *rsiRef;
    // *rsiRef = *rdiRef;
    // fprintf(stderr, "after: rdi: %lx, rsi: %lx\n", *rdiRef, *rsiRef);

    // eaxBackup = *regRef;
    // *regRef = val;
}
```

Now the program will literally tell me the answer itself!



```
No, Genuine is not correct.
Genu
GenuineI
GenuineIntel3Q
```