CS 240
Laboratory 9 Assignment
X86 Disassembly and Reverse Engineering

Examine the C code on the left and the corresponding X86 code on the right for the function **test_prime**

Answer the questions below assuming that ***test_prime*** is called with **num = 7**

C code for function **test_prime**

```
int test_prime(long num) {

  for (long i =2; i <= num/2;++i)  {
    if (num % i == 0) {
       return  0;
    }
  }
  return  1;
}
```

Dump of X86 code produced by **gdb** for function **test_prime**

NOTE: the <+xx> on each line represents an offset from the starting address of the function.

```
0x0000000000400474 <+0>:     push   %rbp
0x0000000000400475 <+1>:     mov    %rsp,%rbp
0x0000000000400478 <+4>:     mov    %rdi,-0x18(%rbp)
0x000000000040047c <+8>:     movq   $0x2,-0x8(%rbp)
0x0000000000400484 <+16>:    jmp    0x4004a9 <test_prime+45>
0x0000000000400485 <+17>:    mov    -0x18(%rbp),%rax
0x0000000000400486 <+18>:    movq   $0x0,%rdx
0x0000000000400491 <+22>:    idivq  -0x8(%rbp)
0x0000000000400495 <+26>:    mov    %rdx,%rax
0x0000000000400498 <+29>:    test   %rax,%rax
0x000000000040049b <+32>:    jne    0x4004a9 <test_prime+40>
0x000000000040049d <+34>:    mov    $0x0,%eax
0x00000000004004a2 <+38>:    jmp    0x4004c6 <test_prime+72>
0x00000000004004a4 <+40>:    addq   $0x1,-0x8(%rbp)
0x00000000004004a9 <+45>:    mov    -0x18(%rbp),%rax
0x00000000004004b8 <+58>:    sar    %rax
0x00000000004004bb <+61>:    cmp    -0x8(%rbp),%rax
0x00000000004004bf <+65>:    jge    0x400485 <test_prime+17>
0x00000000004004c1 <+68>:    mov    $0x1,%eax
0x00000000004004c6 <+72>:    pop    %rbp
0x00000000004004c7 <+73>:    retq
```

1. What is the starting address of **test_prime** in memory?

2. What register is the argument stored in when the assembler code begins execution?

3. Circle and label the statements (there are two) that set the return value for the function.

4. Circle and label the X86 statements that test the condition in the **for** loop. Describe how num/2 is calculated in this code:

5. Circle and label the X86 statements that implement testing the conditional for the *if* statement in the body of the loop. Look up the *idivq* X86 instruction, and explain how the *num%2* is accomplished with the given code: