CS 240
Laboratory 8 Assignment
Disassembly and Reverse Engineering

Analyze the X86 code for the C function ***test_prime*** for a specific input of **num = 7** and answer the questions that follow.

(in general, assume num > 1)

C function to test if a number is prime

```
int test_prime(long num) {

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

Dump of assembler code produce 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+53>
 0x0000000000400486 <+18>:    mov    -0x18(%rbp),%rax
 0x000000000040048a <+22>:    mov    %rax,%rdx
 0x000000000040048d <+25>:    sar    $0x3f,%rdx
 0x0000000000400491 <+29>:    idivq  -0x8(%rbp)
 0x0000000000400495 <+33>:    mov    %rdx,%rax
 0x0000000000400498 <+36>:    test   %rax,%rax
 0x000000000040049b <+39>:    jne    0x4004a4 <test_prime+48>
 0x000000000040049d <+41>:    mov    $0x0,%eax
 0x00000000004004a2 <+46>:    jmp    0x4004c6 <test_prime+82>
 0x00000000004004a4 <+48>:    addq   $0x1,-0x8(%rbp)
 0x00000000004004a9 <+53>:    mov    -0x18(%rbp),%rax
 0x00000000004004b8 <+64>:    sar    %rax
 0x00000000004004bb <+67>:    cmp    -0x8(%rbp),%rax
 0x00000000004004bf <+71>:    jge    0x400486 <test_prime+18>
 0x00000000004004c1 <+73>:    mov    $0x1,%eax
 0x00000000004004c6 <+78>:    leaveq
 0x00000000004004c7 <+79>:    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: