ID ______________________________ Name ______________________________

Assume all operations are based on 64-bit instruction operated on the Intel CPU and represent a number in a hexadecimal form unless specified.

1. For each of the following sentences, you are to indicate whether or not the expression always yields "**true**" or "**false**". If you write no answer you will get **0** points. If you write the right answer, you will get **+2** points. You write the wrong answer, you will get **-1** points. <u>Justify your answer if you choose false</u>.

   a) Instructions are executed in an ascending order of their addresses.

   b) Speculative execution in x86-64 always benefits instructions in terms of performance.

   c) x86-64 honors monotonicity in computations of numbers.

   d) The virtual address is represented in 64 bits where upper 16 bits must be set to zero.

   e) Addresses used in programs should be chosen with care so that two addresses in different programs do not overlap each other.

   f) Commonly used instructions tend to be smaller bytes in length than less common instructions.

   g) A machine code views the memory as simply a large byte-addressable array.

   h) Instructions which generate less than 8 bytes in the destination register will leave the remaining bytes unchanged.

   i) Up to six arguments between functions can be passed via registers. Therefore, in x86-64 a function cannot pass the number of arguments more than six.

   j) Both $577 and 577 are treated as an immediate in the operand.

   k) The size of register in the operand must match the size designated by the last character of the instruction.

   l) Local variables are often kept in registers rather than stored in memory location. Hence, a pointer local variable can be in the register.

   m) When performing a cast involving both a size change and a change of "sign" in C, the operation in the assembly should change the sign first.

   n) The stack grows downward such that the top element of the stack has the lowest address. The stack pointer `%rsp` holds the value of the top stack element.

   o) The load effect address instruction `leaq` can be used to generate pointers for memory reference. In addition, it can be used to compactly describe common arithmetic operations. In the second case, the destination operand must be a register.

   p) `0x8DE0E24` is a valid virtual address for a pointer.

q) `0x8DE0E24` is a `valid` virtual address for `int`-typed and `long`-typed integers.

r) `cmp`, `test` and `lea` instructions set condition codes and destination registers.

```
int t1 = random(); double d1 = (double)t1;
int t2 = random(); double d2 = (double)t2;
int t3 = random(); double d3 = (double)t3;
float f2 = (float)d2;
double d4 = d2 – d1;
double d5 = (double) (t2 – t1);
double d6 = (d1 + d2) + d3; double d7 = (d1 * d2) * d3;
double d8 = d1 + (d2 + d3); double d9 = d1 * (d2 * d3);
d10 = d1 / d1; d11 = d2 / d2;
```

Do not include special values of the floating point.

s) **`f2`** and **`d2`** are the same value.

t) **`d4`** and **`d5`** are the same value.

u) **`d6`** and **`d8`** are the same value.

v) **`d7`** and **`d9`** are the same value.

w) **`d10`** and **`d11`** are the same value.

2. Answer to the following question with a short answer.

```
xorq %rcx, %rcx
```

a) Show different instructions of the same operation as the above as many as you can.

```
typedef __int128 int128_t;                      // two’s complement
void store_uprod (int128_t *dest, int64_t x, int64_t y) {
    *dest = x * (int128_t) y;
}
```

b) In the function `store_uprod` the product of `x` and `y` is `0x1234567890abcdef` stored in memory at `0xFD88932E0`. Show values of `dest`, `*dest`, `*dest+1` and `*(dest+1)`.

$(a < 0 == b < 0)$ && $(t < 0\ != a < 0)$ where $t = a + b$

c) The overflow register (OF) is set according to the above C expression. Interpret the above expression.

d) What is the primary usage for instruction `test`.

```
sarq %cl, %rax
```

e) If values on register `%rax` and `%rcx` are respectively `0xED78F3E9` and `0x11`, interpret the above instruction. What is the values on the destination register after execution of the instruction.

3. Assume the following values are stored at the indicated memory addresses and registers.

Addresses	Value	Register	Value
0x200	0xBB	%rax	0x200
0x208	0xBC	%rbx	0x208
0x210	0x24	%rcx	0x1
0x218	0x21	%rdx	0x10B

Fill in the following table.

Operand	Value	Operand	Value	Operand	Value
(%rbx)		8(%rbx)		7(%rax, %rcx)	
0x1FC(, %rcx, 4)		(%rbx, %rcx, 8)			

Instruction	Destination	Value
addq %rcx, (%rax)		
subq %rdx, 8(%rax)		
incq 16(%rax)		
imulq $16, (%rax, %cdx, 8)		

4. Bit manipulation. Function `float_twice` should return a bit-level equivalent of expression `2×f` for floating point argument `f`. Both the argument and result are passed as unsigned integers, but they are to be interpreted as the bit-level representation of single-precision floating point values. When the argument is NaN, return that argument.

Fill the following four blanks in the code below.

```
unsigned float_twice(unsigned uf) {
  unsigned sign = uf>>31;
  unsigned exp = uf>>23 & 0xFF;
  unsigned frac = uf & 0x7FFFFF;
  if ((a)________) {
      frac = 2*frac;
      if ((b)________) {
         frac = frac & 0x7FFFFF;
         exp = 1;
      }
  } else if ((c)_________) {
      exp++;
      if ((d)________) {
         frac = 0;
      }
  }
  return (sign << 31) | (exp << 23) | frac;
}
```

5. For each of the following lines of assembly language determine the appropriate instruction suffice based on the operand.

```
mov__ %eax, (%rsp)
mov__ %dx, (%rax)
mov__ (%rax), %dx
```

6. Consider the following assembly code

```
loop:
  movl   %esi, %ecx
  movl   $1, %edx
  movl   $0, %eax
  jmp    .L2
.L3:
  movq   %rdi, %r8
  andq   %rdx, %r8
  orq    %r8, %rax
  salq   %cl, %rdx
.L2:
  testq  %rdx, %rdx
  jne    .L3
  rep, ret
```

The preceding code was generated by compiling C code that had the following overall form:

```
long loop (long x, long n)
{
   long result = ___________;
   long mask;
   for (mask = ____________; mask _____________; mask = ___________) {
      result |= ______________;
   }
   return result;
}
```

Fill in the missing parts of the C code

a) Which registers hold program values `x`, `n` `result` and `mask`?

b) What are the initial values of `result` and `mask`?

c) What is the `test` condition for `mask`?

d) How dose `mask` get updated?

e) How does `result` get updated?

f) Fill in all the missing parts of the above C code.