Instructions:

This is a closed book, closed notes quiz. No calculators, mobile phones, or computers are allowed. You have 120 minutes to complete the exam. There are a total of 8 questions for a total of 100 points. This exam has 11 pages. Make sure you have all 11 pages before starting the exam.

There should be enough space on the exam for your answers. If you need more space, I have blank sheets of paper for you to use. If you use extra paper, clearly mark on the question where it is continued so that I can find it.

Good Luck!
1. (10 points)

Recall the **leaq** instruction calculates the address of the first operand and stores that address in the destination. Given the register values in the table below, indicate the result that will be stored in `%r11` for each of the instructions. If the instruction is not a valid instruction, put N/A for the result.

Give your answers in hexadecimal.

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%rcx</code></td>
<td>0x4</td>
</tr>
<tr>
<td><code>%rdx</code></td>
<td>0x8</td>
</tr>
<tr>
<td><code>%r10</code></td>
<td>0xbc00</td>
</tr>
<tr>
<td><code>%r8</code></td>
<td>0xa400</td>
</tr>
<tr>
<td><code>%rdi</code></td>
<td>0x2</td>
</tr>
<tr>
<td><code>%rax</code></td>
<td>0xbd00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result stored in <code>%r11</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>leaq (, %rdx, 8), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq (%rcx), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq 0x40(, %rdi, 4), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq 0x38(%r8, %rdi), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq (%r10, %rdx, 7), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq 0x20(%r10, %rdx, 2), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq (%rax, %rdi, 1), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq 0x34(%r8), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq (%r10, %rdx), %r11</td>
<td>N/A</td>
</tr>
<tr>
<td>leaq 0xcd7, %r11</td>
<td>N/A</td>
</tr>
</tbody>
</table>
2. (20 points)

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1420</td>
<td>0xDD7D</td>
</tr>
<tr>
<td>0x1428</td>
<td>0x38A4</td>
</tr>
<tr>
<td>0x1430</td>
<td>0xF705</td>
</tr>
<tr>
<td>0x1438</td>
<td>0x544C</td>
</tr>
<tr>
<td>0x1440</td>
<td>0xEC2A</td>
</tr>
<tr>
<td>0x1448</td>
<td>0xF197</td>
</tr>
<tr>
<td>0x1450</td>
<td>0xE574</td>
</tr>
<tr>
<td>0x1458</td>
<td>0x4A96</td>
</tr>
<tr>
<td>0x1460</td>
<td>0x45BE</td>
</tr>
<tr>
<td>0x1468</td>
<td>0xCA8F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0x1000</td>
</tr>
<tr>
<td>%rbx</td>
<td>0x1400</td>
</tr>
<tr>
<td>%rcx</td>
<td>0x400</td>
</tr>
<tr>
<td>%rdx</td>
<td>0x440</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x20</td>
</tr>
<tr>
<td>%rdi</td>
<td>0x60</td>
</tr>
<tr>
<td>%rsp</td>
<td>0x1440</td>
</tr>
<tr>
<td>%rbp</td>
<td>0x2</td>
</tr>
<tr>
<td>%r9</td>
<td>0x4</td>
</tr>
<tr>
<td>%r10</td>
<td>0x8</td>
</tr>
</tbody>
</table>

Fill in the following table showing the effects of the following instructions in terms of both the register or address of the memory location that will be updated and the resulting value. The destination column will either be a register name or a memory address and the value column will be the value held in that register or memory location. Assume each instruction is independent of the others (i.e., the registers and memory locations have the above values at the start of each instruction). You can assume all the memory locations hold 8 byte long integers. If you calculate a memory address that is not shown above, put N/A in the Destination and Value columns. Write all numerical values in hexadecimal.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>popq %rax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addq (%rbx, %rdi), %rax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addq %rax, (%rbx, %rdi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sarq $8, (%rbx, %rsi)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xorq %rdx, %rdx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>salq $4, %rdx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subq %rsi, %rdx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notq %rbp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>incq 0x1000(%rdx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushq %rdx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Starting with C code of the following form:

```c
long func3(long x) {
    long y;
    if (________________) {
        return 0;
    }
    y = ______________;
    return y + func3(y);
}
```

GCC generates the following assembly code:

```
func3:
    movq %rdi, %rax
    testq %rdi, %rdi
    jne .L7
    ret
.L7:
    pushq %rbx
    sarq %rax
    movq %rax, %rbx
    movq %rax, %rdi
    call func3
    addq %rbx, %rax
    popq %rbx
    ret
```

What value does `func3` store in the callee-saved register `%rbx`? ________________________________

Fill in the missing expressions in the above C code.
4. (10 points)

Recall that floating point numbers are represented in the form $V = (-1)^s \times M \times 2^E$ where $M$ is encoded in `frac`, and $E$ is encoded in `exp`. Assume we have the following tiny 10-bit floating point representation where bits 0-4 represent the fraction field, bits 5-8 represents the exponent field, and bit 9 represents the sign bit, as shown below.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>exp</td>
<td>frac</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write down the floating point representation of the following binary number with the representation system described above.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101101110</td>
<td></td>
</tr>
</tbody>
</table>

Write down the binary representation of the following floating point number with the representation described above.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.125</td>
</tr>
</tbody>
</table>
5. (10 points)

Consider the following structure declaration:

```c
struct prob {
    char a;
    int b;
    char c;
    long d;
};
```

The following function (with some expressions omitted) operates on this structure. Recall, to access a field from a pointer to a structure, you use the arrow (->) notation.

```c
void question5(struct prob *sp, char x, int y, long z) {
    sp->a = ___________;  
    sp->b = ___________;  
    sp->c = ___________;  
    sp->d = ___________;  
}
```

What are the offsets (in bytes from the start of the structure) of the following fields in the above structure:

a: ______________

b: ______________

c: ______________

d: ______________

How many total bytes does the structure require? ________________________

The compiler generates the assembly code below. Using this code, fill in the missing expressions in the question5 function above. The last page of this quiz shows all the registers for the X86-64 system.

```assembly
question5:
  leal -1(%rsi), %eax
  movb %al, (%rdi)
  movl %edx, 4(%rdi)
  leal (%rsi,%rsi,2), %esi
  movb %sil, 8(%rdi)
  addq $42, %rcx
  movq %rcx, 16(%rdi)
  ret
```
6. (10 points)

Shown below is the assembly code for the function \texttt{prob6()}, which calls \texttt{func6()}. Right before the call to \texttt{func6()}, \texttt{%rip}, \texttt{%rsp}, and stack memory have the values shown below.

\begin{verbatim}
0000000000400497 <func6>:
  400497:   48 01 f7                add    %rsi,%rdi
  40049a:   48 01 fa                add    %rdi,%rdx
  40049d:   48 01 d1                add    %rdx,%rcx
  4004a0:   4c 01 c1                add    %r8,%rcx
  4004a3:   4c 01 c9                add    %r9,%rcx
  4004a6:   48 89 c8                mov    %rcx,%rax
  4004a9:   48 03 44 24 08          add    0x8(%rsp),%rax
  4004ae:   c3                      retq

00000000004004af <prob6>:
  4004af:   6a 07                   pushq  $0x7
  4004b1:   41 b9 06 00 00 00       mov    $0x6,%r9d
  4004b7:   41 b8 05 00 00 00       mov    $0x5,%r8d
  4004bd:   b9 04 00 00 00          mov    $0x4,%ecx
  4004c2:   ba 03 00 00 00          mov    $0x3,%edx
  4004c7:   be 02 00 00 00          mov    $0x2,%esi
  4004cc:   bf 01 00 00 00          mov    $0x1,%edi
->4004d1:   e8 c1 ff ff ff          callq  400497 <func6>
  4004d6:   48 83 c4 08          add    $0x8,%rsp
  4004da:   c3                      retq
\end{verbatim}

Below, show the values of \texttt{%rip}, \texttt{%rsp}, and the value in memory that \texttt{%rsp} points to, right at the start of the call to \texttt{func6()} but before any of the code in \texttt{func6()} is executed.

\begin{verbatim}

\begin{tabular}{|c|c|}
\hline
Register & Value \\
\hline
%rip & 0x4004d1 \\
%rsp & 0x7fffffffdfde8 \\
\hline
\end{tabular}

0000000000400497 <func6>:

\begin{verbatim}
->400497:   48 01 f7                add    %rsi,%rdi
  40049a:   48 01 fa                add    %rdi,%rdx
  40049d:   48 01 d1                add    %rdx,%rcx
  4004a0:   4c 01 c1                add    %r8,%rcx
  4004a3:   4c 01 c9                add    %r9,%rcx
  4004a6:   48 89 c8                mov    %rcx,%rax
  4004a9:   48 03 44 24 08          add    0x8(%rsp),%rax
  4004ae:   c3                      retq
\end{verbatim}

\end{verbatim}
Show the values of %rip, %rsp and the value in memory that %rsp points to, right before the ret instruction of func6 is executed.

0000000000400497 <func6>:
400497: 48 01 f7 add %rsi,%rdi
40049a: 48 01 fa add %rdi,%rdx
40049d: 48 01 d1 add %rdx,%rcx
4004a0: 4c 01 c1 add %r8,%rcx
4004a3: 4c 01 c9 add %r9,%rcx
4004a6: 48 89 c8 mov %rcx,%rax
4004a9: 48 03 44 24 08 add 0x8(%rsp),%rax
->4004ae: c3 retq

00000000004004af <prob6>:
4004af: 6a 07 pushq $0x7
4004b1: 41 b9 06 00 00 00 mov $0x6,%r9d
4004b7: 41 b8 05 00 00 00 mov $0x5,%r8d
4004bd: b9 04 00 00 00 mov $0x4,%ecx
4004c2: ba 03 00 00 00 mov $0x3,%edx
4004c7: be 02 00 00 00 mov $0x2,%esi
4004cc: bf 01 00 00 00 mov $0x1,%edi
4004d1: e8 c1 ff ff ff callq 400497 <func6>
4004d6: 48 83 c4 08 add $0x8,%rsp
4004da: c3 retq

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rip</td>
<td></td>
</tr>
<tr>
<td>%rsp</td>
<td></td>
</tr>
<tr>
<td>(%rsp)</td>
<td></td>
</tr>
</tbody>
</table>

How many arguments does func6 take? ________________
7. (15 points) Multiple Choice

Which of the following condition codes are used to determine if the instruction `jne` should take the jump or not? Circle all that apply.

A. Carry Flag
B. Zero Flag
C. Sign Flag
D. Overflow Flag

What is the program that converts a C program into assembly language? Circle one.

A. Assembler
B. Compiler
C. Objdump
D. Disassembler

If your compiler uses stack canaries to defend buffer overflow attacks, how can someone work around this? Circle all that apply.

A. Make the stack non-executable
B. ROP attacks
C. Code Injection attacks
D. There is no workaround for this defense

What line of assembly is equivalent to the following: `xorq %rax, %rax` Circle all that apply.

A. `orq %rax, %rax`
B. `xorl %eax, %eax`
C. `cmpq %rax, %rax`
D. `andq $0, %rax`

How much does Diane's Silk Dress Cost? Circle one.

A. 8 dollars
B. 9 dollars
C. 89 dollars
D. 224 dollars
You have the following array in C: int A[4][8]; which starts at address 0x10000. What is the address of the element at A[2][4]? Give your answer in hexadecimal.

Answer: _______________________

You have the following assembly code:

```
0000000000400714 <getString>:  
  400714:   sub    $0x18,%rsp
  400718:   lea    8(%rsp),%rdi
  40071d:   callq  400686 <gets>
  400722:   add    $0x18,%rsp
  400726:   retq
```

Unfortunately, gets (which works exactly how it was described in class) has a buffer overflow bug. What is the largest input string you can give to this function without modifying the return address of the caller? Pay particular attention about how the argument to gets is set up in the code above.

Answer: ______________________ bytes

You have the following C code and corresponding assembly. Fill in the missing part of the movq instruction below.

```
void main() {
    long array[MAX];
    for (long i = 0; i < MAX; i++) {
        array[i] = i;
    }
    // other code . . .
    return
}
```

main:
subq $56, %rsp
movl $0, %eax
jmp .L2
.L3:
    movq %rax, (%rsp,_____________________
    addq $1, %rax
.L2:
    cmpq $4, %rax
    jle .L3
    . . .
    addq $56, %rsp
    ret

What is the value of MAX? _______________________

Last question! You made it! Unless you are skipping around the quiz and doing questions out of order. In that case, keep going, you can do it!

Two instructions, cmp and test both have destination operands but neither of these instructions modify their destination operand. How are these instructions used?

Answer:
<table>
<thead>
<tr>
<th>X68-64 Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>%rax</strong></td>
</tr>
<tr>
<td><strong>%rbx</strong></td>
</tr>
<tr>
<td><strong>%rcx</strong></td>
</tr>
<tr>
<td><strong>%rdx</strong></td>
</tr>
<tr>
<td><strong>%rsi</strong></td>
</tr>
<tr>
<td><strong>%rdi</strong></td>
</tr>
<tr>
<td><strong>%rbp</strong></td>
</tr>
<tr>
<td><strong>%rsp</strong></td>
</tr>
<tr>
<td><strong>%r8</strong></td>
</tr>
<tr>
<td><strong>%r9</strong></td>
</tr>
<tr>
<td><strong>%r10</strong></td>
</tr>
<tr>
<td><strong>%r11</strong></td>
</tr>
<tr>
<td><strong>%r12</strong></td>
</tr>
<tr>
<td><strong>%r13</strong></td>
</tr>
<tr>
<td><strong>%r14</strong></td>
</tr>
<tr>
<td><strong>%r15</strong></td>
</tr>
</tbody>
</table>