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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xFF</td>
<td>%rax</td>
<td>0x100</td>
</tr>
<tr>
<td>0x108</td>
<td>0xAB</td>
<td>%rcx</td>
<td>0x1</td>
</tr>
<tr>
<td>0x110</td>
<td>0x13</td>
<td>%rdx</td>
<td>0x3</td>
</tr>
<tr>
<td>0x118</td>
<td>0x11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill in the following table showing the values for the indicated operands:

<table>
<thead>
<tr>
<th>Operand</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td></td>
</tr>
<tr>
<td>0x108</td>
<td></td>
</tr>
<tr>
<td>$0x108</td>
<td></td>
</tr>
<tr>
<td>(%rax)</td>
<td></td>
</tr>
<tr>
<td>8(%rax)</td>
<td></td>
</tr>
<tr>
<td>0xD(%rax,%rdx)</td>
<td></td>
</tr>
<tr>
<td>260(%rcx,%rdx)</td>
<td></td>
</tr>
<tr>
<td>0xFC(,%rcx,4)</td>
<td></td>
</tr>
<tr>
<td>(%rax,%rdx,8)</td>
<td></td>
</tr>
</tbody>
</table>

Fill in the following table showing the effects of the following instructions in terms of both the register or memory location that will be updated and the resulting value:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>addq %rcx, (%rax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subq %rdx, 8(%rax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imulq $16, (%rax,%rdx,8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>incq 16(%rax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decq %rcx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subq %rdx, %rax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Draw the memory layout of the following struct (starting at memory location 0 below) and give the total size of the structure (in other words, what does the `sizeof(struct question2)` return?).

```c
struct question2 {
    char a;
    int b;
    int *c;
    char d;
};
```

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
```

`sizeof(struct question2)` ________________

3. The following array `int array[25][7]` is stored at the address: 0x4000. What is the address of the integer at `array[8][3]`?

4. Suppose `%rax` has the value of 42, `%rcx` has the value 5, indicate the value that will be stored in register `%rdx` for each of the instructions:

```
leaq -5(%rax), %rdx
leaq 0x45, %rdx
leaq (%rax, %rcx), %rdx
leaq (%rax, %rcx, 2), %rdx
leaq 0x21(%rcx, %rcx, 1), %rdx
```
5. **Part A** Write the following function, `swapNybble()`, which takes an unsigned char as input and returns an unsigned char that has the lower 4-bits swapped with the upper 4 bits of the input. For example, `swapNybble(0xAB)` would return `0xBA`.

```c
unsigned char swapNybble(unsigned char c){
    // Returns an unsigned char with the upper 4 bits swapped with
    // the lower 4 bits.
    return c; // Placeholder for actual implementation.
}
```
Part B Shown below is the assembly code for a function `test()`, which calls `swapNybble(0xAB)`. Right before the call to `swapNybble()`, `%rip`, `%rsp`, and the stack have the values shown below.

```
00000000000000400594 <test>:
  400594:   48 83 ec 08           sub    $0x8,%rsp
  400598:   bf ab 00 00 00     mov $0xab,%edi
->  40059d:   e8 bb ff ff ff  callq  40055d <swapNybble>
  4005a2:   48 83 c4 08     add $0x8,%rsp
  4005a6:   c3                retq
```

Right before call to `swapNybble()`

<table>
<thead>
<tr>
<th>%rip</th>
<th>0x40059d</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td>0x7fffffffefe330</td>
</tr>
</tbody>
</table>

Stack

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffffffefe340</td>
<td>0x0</td>
</tr>
<tr>
<td>0x7fffffffefe338</td>
<td>0x4005b0</td>
</tr>
<tr>
<td>0x7fffffffefe330</td>
<td>0x0</td>
</tr>
<tr>
<td>0x7fffffffefe328</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Below, show the values of `%rip`, `%rsp`, and the stack right after the `callq` opcode jumps to the start of `swapNybble()` but before any of the code in `swapNybble()` is executed.

<table>
<thead>
<tr>
<th>%rip</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td></td>
</tr>
</tbody>
</table>

Stack

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffffffefe340</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffefe338</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffefe330</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffefe328</td>
<td></td>
</tr>
</tbody>
</table>

Show the values of of `%rip`, `%rsp`, and the stack right after the `callq` has finished but before the add instruction has executed.

After return from `swapNybble()` (Fill in the blanks below).
00000000000400594 <test>:
  400594: 48 83 ec 08           sub   $0x8, %rsp
  400598: bf ab 00 00 00        mov    $0xab, %edi
  40059d: e8 bb ff ff ff       callq  40055d <swapNybble>
->  4005a2: 48 83 c4 08        add    $0x8, %rsp
  4005a6: c3                      retq

%rip |
%rsp |

<table>
<thead>
<tr>
<th>Stack</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffffffe340</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffe338</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffe330</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffe328</td>
<td></td>
</tr>
</tbody>
</table>
6. The following C program and its assembly language version are shown below. Fill in the missing blanks in the assembly language output.

```c
void sumArray(long *a, long len, long *sum) {
    /* Sum an array with length len and store the answer in sum.
     * arguments:
     *     a -- an array of long integers to sum
     *     len -- the length of the array
     *     sum -- a pointer that holds the sum of the array
     */
    long i;
    long answer = 0;
    for (i = 0; i < len; i++) {
        answer += a[i];
    }
    *sum = answer;
}
```

```
000000000040055d <sumArray>:
 40055d:   b9 00 00 00 00          movq    $0x0,__________
 400562:   b8 00 00 00 00          movq    $0x0,%rax
 400567:   eb 08                   jmp     ______________
 400569:   48 03 0c c7             addq   (____________,%rax,8),%rcx
 40056d:   48 83 c0 01             addq   $0x1,%rax
 400571:   48 39 f0                cmpq   %rsi,%rax
 400574:   7c f3                   j_____ 400569 <sumArray+0xc>
 400576:   48 89 0a                movq   %rcx, ______________
 400579:   c3                      retq
```
7. A C function \texttt{loopy} and the assembly code it compiles to on a 64-bit Linux machine is shown below:

\begin{verbatim}
loopy:
  movl $0, %eax
  movl $0, %ecx
  jmp .L2
.L4:
  cmpq %rdx, (%rsi,%rcx,8)
  jle .L3
  addq $1, %rax
.L3:
  subq $1, %rdx
  addq $1, %rcx
.L2:
  cmpq %rdi, %rcx
  jl .L4
ret
\end{verbatim}

\begin{verbatim}
long loopy(long n, long *a, long value) {
  long i;
  long x = _____________;
  for(i = ______________; ______________; i++) {
    if (______________________) {
      x = _________________;
    }
  }
  _________________;
  _________________;
  return x;
}
\end{verbatim}

Based on the assembly code, fill in the blanks in the C source code.

Notes:
- You may only use the C variable names \texttt{n}, \texttt{a}, \texttt{i}, \texttt{value}, and \texttt{x}, not register names.
- Use array notation to show accesses or updates to elements of array \texttt{a}.
8. A C function `func` and the x86-64 assembly code it compiles to on Linux machine is shown below:

Part A (5 pts): Based on the assembly code, fill in the blanks in the C source code. Note, the only lines in the C code are what are shown above.

Notes:
- You may only use the C variable names `a`, `b`, `c`, `i`, numbers, and C expressions in the blanks.
- Use array notation to show accesses or updates to elements of array `a`.

Part B (1 pt): Describe in one short sentence what this function does. (This is a hint that if your C code doesn't do something that is easy to explain in English, you probably want to take another look at what you came up with).

Practice Problems in the book

The practice problems in sections 3.4 to 3.10 are also good ways to prepare for the quiz. I often draw inspiration from these problems for quiz questions, so it is likely you will see similar problems to these on the quiz.