Do not turn the page until 10:50.

Instructions

- This exam contains 5 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed one page (US letter, double-sided) of handwritten notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, and watches.
- You have 60 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read all questions first and start where you feel the most confident.
- Relax. You are here to learn.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>22</td>
<td>20</td>
<td>14</td>
<td>24</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>
Question 1: Number Representation  [22 pts]

Consider the signed char \( x = 0b\ 1010\ 1000 \).

(A) What is the value of \( x \)? You may answer as the sum of powers of 2. [2 pt]

(B) In theory (math), what is the difference in the values of \( \text{(unsigned char)}x \) and \( x \)? Your answer should be a decimal number. [2 pt]

(C) In C, what is the value of char \( y = \text{(unsigned char)}x - x \)? [2 pt]

(D) Which of the following expressions will result in a positive (non-negative, non-zero) result? (Circle one) [4 pt]

\[
x << 4 \quad x^0x81 \quad x | \sim x \quad \hat{\sim}(x >> 1)
\]

For the rest of this problem we are working with a new floating point datatype (flo) that follows the same conventions as IEEE 754 except using 8 bits split into the following fields:

| Sign (1) | Exponent (3) | Mantissa (4) |

(E) What is the value of the numeral from above \( 0b\ 1010\ 1000 \) in this representation? [4 pt]

(F) What is the encoding of the most negative real number that we can represent (\( \infty \) is not a real number) in this floating point scheme (binary)? [4 pt]

(G) What will occur if we cast \( \text{flo} f = \text{(flo)}x \) (i.e. try to represent the value stored in \( x \) as a \( \text{flo} \))? (Circle one) [4 pt]

Rounding    Underflow    Overflow    None of these
**Question 2:** Pointers & Memory  [20 pts]

For this problem we are using a 64-bit x86-64 machine (little endian). The current state of memory (values in hex) is shown below:

<table>
<thead>
<tr>
<th>Word Addr</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>AC</td>
<td>AB</td>
<td>03</td>
<td>01</td>
<td>BA</td>
<td>5E</td>
<td>BA</td>
<td>11</td>
</tr>
<tr>
<td>0x08</td>
<td>5E</td>
<td>00</td>
<td>68</td>
<td>0C</td>
<td>BE</td>
<td>A7</td>
<td>CE</td>
<td>FA</td>
</tr>
<tr>
<td>0x10</td>
<td>1D</td>
<td>B0</td>
<td>99</td>
<td>DE</td>
<td>AD</td>
<td>60</td>
<td>BB</td>
<td>40</td>
</tr>
<tr>
<td>0x18</td>
<td>14</td>
<td>1D</td>
<td>EC</td>
<td>AF</td>
<td>EE</td>
<td>FF</td>
<td>CO</td>
<td>70</td>
</tr>
<tr>
<td>0x20</td>
<td>BA</td>
<td>B0</td>
<td>41</td>
<td>20</td>
<td>80</td>
<td>AA</td>
<td>BE</td>
<td>EF</td>
</tr>
</tbody>
</table>

**char**  `charP = 0x12`

**int**  `intP = 0x8`

**long**  `longP = 0x30`

(A) Using the values shown above, complete the C code below to fulfill the behaviors described in the comments using pointer arithmetic.  [8 pt]

```c
char v1 = *(charP + _______);  // set v1 = 0xAF
int* v2 = &intP[_______];      // set v2 = 0x14
```

(B) What are the values (in hex) stored in each register shown after the following x86-64 instructions are executed? We are still using the state of memory shown above.  
*Remember to use the appropriate bit widths.*  [12 pt]

<table>
<thead>
<tr>
<th>Register</th>
<th>Data (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>0x 0000 0000 0000 0019</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x 0000 0000 0000 0003</td>
</tr>
<tr>
<td>%r9b</td>
<td>0x</td>
</tr>
<tr>
<td>%eax</td>
<td>0x</td>
</tr>
<tr>
<td>%r8</td>
<td>0x</td>
</tr>
</tbody>
</table>
Question 3: Design Questions [14 pts]

Answer the following questions in the boxes provided with a single sentence fragment. Please try to write as legibly as possible.

(A) Name the two issues with Sign and Magnitude that Two’s Complement fixed. [4 pt]

(B) Briefly describe an advantage of making addresses and registers both the width of a word. [4 pt]

(C) Briefly explain your answers to the following questions if we moved 1 bit from the mantissa field (now 22 bits) to the exponent field (now 9 bits) in floating point: [6 pt]

<table>
<thead>
<tr>
<th>Will the total number of representable floats (normalized + denormalized + special cases) change? Circle one: Yes No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The frequency of rounding will (circle one): Increase, Decrease, or Stay the same</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
</tr>
</tbody>
</table>
Question 4: C & Assembly [24 pts]

Answer the questions below about the following x86-64 assembly function:

```
mystery:
  movl $0, %eax               # Line 1
  .L2: cmpl %esi, %eax        # Line 2
  jge .L1                    # Line 3
  movslq %eax, %rdx          # Line 4
  leaq (%rdi,%rdx,2), %rcx   # Line 5
  movzwl (%rcx), %edx        # Line 6
  andl $1, %edx              # Line 7
  movw %dx, (%rcx)           # Line 8
  addl $1, %eax              # Line 9
  jmp .L2                    # Line 10
  .L1: retq                  # Line 11
```

(A) What **variable type** would %rdi be in the corresponding C program? [4 pt]

(B) **Briefly** describe why Line 4 is needed before Line 5. [4 pt]

(C) This function uses a **for** loop. Fill in the corresponding parts below, using register names as variable names. None should be blank. [8 pt]

```
for ( __________ ; __________ ; __________ )
```

(D) If we call this function with the value **3 as the second argument**, what value is returned? [4 pt]

(E) Describe at a high level what you think this function **accomplishes** (not line-by-line). [4 pt]
**Question 5: Procedures & The Stack  [20 pts]**

The recursive power function `power()` calculates `base^pow` and its x86-64 disassembly is shown below:

```c
int power(int base, unsigned int pow) {
    if (pow) {
        return base * power(base,pow-1);
    }
    return 1;
}
```

```
00000000004005a0 <power>:
    4005a0:  85 f6     testl %esi,%esi
    4005a2:  74 10     je  4005b4 <power+0x14>
    4005a4:  53        pushq %rbx
    4005a5:  89 fb     movl %edi,%ebx
    4005a7:  83 ee 01   subl $0x1,%esi
    4005aa:  e8 f1 ff ff ff    call 4005a0 <power>
    4005af:  0f af c3   imull %ebx,%eax
    4005b2:  eb 06     jmp 4005ba <power+0x1a>
    4005b4:  b8 01 00 00 00 movl $0x1,%eax
    4005b9:  c3        ret
    4005ba:  5b        popq %rbx
    4005bb:  c3        ret
```

(A) How much space (in bytes) does this function take up in our final executable? [2 pt]

(B) Circle one: The label `power` will show up in which table(s) in the object file? [4 pt]

   Symbol Table   Relocation Table   Both Tables   Neither Table

(C) Which register is being saved on the stack? [2 pt]
(D) What is the return address to `power` that gets stored on the stack? Answer in hex. [2 pt]

(E) Assume `main` calls `power(8,3)`. Fill in the snapshot of memory below the top of the stack in hex as this call to `power` returns to `main`. For unknown words, write “unknown”. [6 pt]

(F) Harry the Husky claims that we could have gotten away with not pushing a register onto the stack in `power`. Is our intrepid school’s mascot correct or not? Briefly explain. [4 pt]