Do not turn the page until 12:30.

Instructions

- This exam contains 14 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 two pages (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 110 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>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
<th>F10</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>5</td>
<td>79</td>
</tr>
</tbody>
</table>
**Question M1: Number Representation  [8 pts]**

(A) Take the 32-bit numeral \(0xC0800000\). Circle the number representation below that has the most negative value for this numeral. [2 pt]

<table>
<thead>
<tr>
<th>Floating Point</th>
<th>Sign &amp; Magnitude</th>
<th>Two’s Complement</th>
<th>Unsigned</th>
</tr>
</thead>
</table>

(B) Let \(\text{float } f\) hold the value \(2^{20}\). What is the largest power of 2 that gets rounded off when added to \(f\)? Answer in exponential form, not just the exponent. [2 pt]

Traffic lights display three basic colors: red (R), yellow (Y), and green (G), so we can use them to encode base 3! We decide to use the encoding \(0\leftrightarrow R, 1\leftrightarrow Y, 2\leftrightarrow G\). For example, \(5 = 1\times3^1 + 2\times3^0\) would be encoded as YG. Assume each traffic light can only display one color at a time.

(C) What is the unsigned decimal value of the traffic lights displaying RGYY? [2 pt]

(D) If we have 9 bits of binary data that we want to store, how many traffic lights would it take to store that same data? [2 pt]

**Question M2: Design Question  [2 pts]**

(A) The machine code for x86-64 instructions are variable length. Name one advantage and one disadvantage of this design decision. [2 pt]

<table>
<thead>
<tr>
<th>Advantage:</th>
<th>Disadvantage:</th>
</tr>
</thead>
</table>
Question M3: Pointers & Memory [8 pts]

For this problem we are using a 64-bit x86-64 machine (little endian). Below is the count_nz function disassembly, showing where the code is stored in memory.

(A) What are the values (in hex) stored in each register shown after the following x86 instructions are executed? Use the appropriate bit widths. Hint: what is the value stored in %rsi? [4 pt]

<table>
<thead>
<tr>
<th>Register</th>
<th>Value (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>0x 0000 0000 0040 0544</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x FFFF FFFF FFFF FFFF</td>
</tr>
<tr>
<td>%eax</td>
<td>0x</td>
</tr>
<tr>
<td>%bx</td>
<td>0x</td>
</tr>
</tbody>
</table>

leal 2(%rdi, %rsi), %eax
movw (%rdi,%rsi,4), %bx

(B) Complete the C code below to fulfill the behaviors described in the inline comments using pointer arithmetic. Let char* charP = 0x400544. [4 pt]

```c
char v1 = *(charP + _____); // set v1 = 0xDB
int* v2 = (int*)((___________*)charP - 2); // set v2 = 0x400534
```
Question M4: Procedures & The Stack  [10 pts]

The function `count_sp` counts the number of spaces in a char array (this is the recursive version of the mystery function from the Midterm). The function and its disassembly are shown below:

```c
int count_sp(char* str) {
    if (*str)
        return (*str == ' ') + count_sp(str+1);
    return 0;
}
```

0000000000400536 <count_sp>:

(A) The right-most column/portion of the disassembly is first generated as the output of which of the following? Circle one. [1 pt]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Assembler</th>
<th>Linker</th>
<th>Loader</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(B) The left-most column of the disassembly was generated by which of the following? [1 pt]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Assembler</th>
<th>Linker</th>
<th>Loader</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(C) Why is %rbx being pushed onto the stack? What is %rbx being used for in this function? [2 pt]

<table>
<thead>
<tr>
<th>Why push:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Usage:</th>
</tr>
</thead>
</table>
SID: __________

(D) What is the return address to `count_sp` that gets stored on the stack? Answer in hex. [1 pt]

(E) Provide a call to `count_sp` that is *guaranteed* to cause a *segmentation fault*. [1 pt]

```
count_sp( ___________ );
```

(F) We call `count_sp(" ! ")`. Fill in the incomplete snapshot of the stack below (in hex) once this call to `count_sp` returns to `main`. For unknown words, write “garbage”. [4 pt]

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffffffdb68</td>
<td>&lt;ret addr to main&gt;</td>
</tr>
<tr>
<td>0x7fffffffdb60</td>
<td>&lt;original rbx&gt;</td>
</tr>
<tr>
<td>0x7fffffffdb58</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb50</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb48</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb40</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb38</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb30</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb28</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffdb20</td>
<td></td>
</tr>
</tbody>
</table>
```
Question M5: C & Assembly [8 pts]

Answer the questions below about the following x86-64 assembly function, which uses a struct:

```
mystery:
.L3:    testq  %rdi, %rdi     # Line 1
   je .L4       # Line 2
   cmpw %si, 0(%rdi)   # Line 3
   je .L5       # Line 4
   movq 8(%rdi), %rdi # Line 5
   jmp .L3       # Line 6
.L4:   movl  $0, %eax   # Line 7
   retq          # Line 8
.L5:   movl  $1, %eax  # Line 9
   retq          # Line 10
```

(A) What C variable type would %rsi be in the corresponding C program? [1 pt]

_______ rsi

(B) %rdi is a pointer to a struct that contains 2 fields. What is the width of the second field? [1 pt]

______ bytes

(C) Based on Line 5, give an intuitive name for the second field in the struct. [1 pt]


(D) Convert lines 1, 2, 7, and 8 into C code. Use variable names that correspond to the register names (e.g. al for the value in %al). [3 pt]

    if ( ___________ ) ___________ ;

(E) Describe at a high level what you think this function accomplishes (not line-by-line). [2 pt]


**Question F6: Caching [10 pts]**

We have 64 KiB of RAM and a 2-KiB L1 data cache that is 4-way set associative with 32-byte blocks and random replacement, write-back, and write allocate policies.

(A) Calculate the TIO address breakdown: [1.5 pt]

<table>
<thead>
<tr>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(B) How many management bits (bits other than the block data) are there in every line in the cache? [1 pt]

______ bits

(C) The code snippet below accesses an array of doubles. Assume i is stored in a register. Calculate the Miss Rate if the cache starts **cold**. [2.5 pt]

```c
#define ARRAY_SIZE 256

double data[ARRAY_SIZE]; // &data = 0x1000 (physical addr)

for (i = 0; i < ARRAY_SIZE; i += 1)
    data[i] /= 100;
```

(D) For each of the proposed (independent) changes, write **IN** for “increased”, **NC** for “no change”, or **DE** for “decreased” to indicate the effect on the Miss Rate for the code above: [4 pt]

Use float instead  ____  Half the cache size  ____

Split the loop body into:  ____  No-write allocate  ____

  data[i] /= 10;
  data[i] /= 10;

(E) Assume it takes 100 ns to get a block of data from main memory. If our L1 data cache has a hit time of 2 ns and a miss rate of 3%, what is the average memory access time (AMAT)? [1 pt]

______ ns
Question F7: Processes  [9 pts]

(A) The following function prints out four numbers. In the following blanks, list three possible outcomes: [3 pt]

```c
void concurrent(void) {
    int x = 3, status;
    if (fork()) {
        if (fork() == 0) {
            x += 2;
            printf("%d", x);
        } else {
            wait(&status);
            wait(&status);
            x -= 2;
        }
    }
    printf("%d", x);
    exit(0);
}
```

(1) ______________  (2) ______________  (3) ______________

(B) For the following examples of exception causes, write “N” for intentional or “U” for unintentional from the perspective of the user process. [2 pt]

<table>
<thead>
<tr>
<th>System call</th>
<th>Hardware failure</th>
<th>Segmentation fault</th>
<th>Mouse clicked</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(C) Briefly define a zombie process. Name a process that can reap a zombie process. [2 pt]

Zombie process:

Reaping process:

(D) In the following blanks, write “Y” for yes or “N” for no if the following need to be updated when `execv` is run on a process. [2 pt]

Page table __________ PTBR __________ Stack __________ Code __________
Question F8: Virtual Memory  [10 pts]

Our system has the following setup:
- 20-bit virtual addresses and 64 KiB of RAM with 256-B pages
- A 4-entry TLB that is fully associative with LRU replacement
- A PTE contains bits for valid (V), dirty (D), read (R), write (W), and execute (X)

(A) Compute the following values: [4 pt]

<table>
<thead>
<tr>
<th>Page offset width</th>
<th># of physical pages</th>
<th># of virtual pages</th>
<th>TLBI width</th>
</tr>
</thead>
</table>

(B) Briefly explain why we make physical memory write-back and fully-associative. [2 pt]

Write-back:

Fully-associative:

(C) The TLB is in the state shown when the following code is executed. The code eventually causes a protection fault. What are the values of the variables when the fault occurs? [4 pt]

```c
long *p = 0x7F080;
for (int i = 0; 1; i++) {
    *p += 1;
    p += 4;
}
```

<table>
<thead>
<tr>
<th>TLBT</th>
<th>PPN</th>
<th>Valid</th>
<th>R</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7F0</td>
<td>0xC3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x7F2</td>
<td>0x3D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x004</td>
<td>0xF4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0x7F1</td>
<td>0x42</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

p = 0x____________
i = ______
**Question F9: Memory Allocation [9 pts]**

(A) In a free list, what is a **footer** used for? Be specific. Why did we not need to use one in allocated blocks in Lab 5? [2 pt]

<table>
<thead>
<tr>
<th>Footer:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Lab 5:</th>
</tr>
</thead>
</table>

(B) We are designing a dynamic memory allocator for a **64-bit computer** with **4-byte boundary tags** and **alignment size of 4 bytes**. Assume a footer is always used. Answer the following questions: [4 pt]

- Maximum tags we can fit into the header (ignoring size): _______ tags
- Minimum block size if we implement an **explicit** free list: _______ bytes
- Maximum block size (leave as expression in powers of 2): _______ bytes

(C) Consider the C code shown here. Assume that the malloc call succeeds and foo is stored in memory (not just in a register). Fill in the following blanks with “>” or “<” to compare the values returned by the following expressions just before return 0. [3 pt]

```c
#include <stdlib.h>
int ZERO = 0;
char* str = "cse351";

int main(int argc, char *argv[]) {
    int *foo = malloc(8);
    free(foo);
    return 0;
}
```

```c
&foo ___ &ZERO
&str ___ ZERO
&main ___ str
```
**Question F10: C and Java [5 pts]**

For this question, use the following Java object definition and C struct definition. Assume addresses are all 64-bits.

```java
public class RentalJ {
    String addr;
    short rooms;
    float rent;
    int[] zip;

    public void info() {
        System.out.println("Rental at "+addr);
    }
}

public class Apt extends RentalJ {
    int roommates;
    public int occupants() {
        return roommates+1;
    }
}
```

```c
struct RentalC {
    char* addr;
    short rooms;
    float rent;
    int zip[5];
};
```

(A) How much memory, in bytes, does an instance of `struct RentalC` use? How many of those bytes are *internal* fragmentation and *external* fragmentation? [3 pt]

<table>
<thead>
<tr>
<th><code>sizeof(struct RentalC)</code></th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 bytes</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

(B) How much longer, in bytes, are the following for `Apt` than for `RentalJ`? Assume the Java instance fields are aligned to 4 bytes. [2 pt]

Instance:

Virtual method table (vtable):
This page purposely left blank