1. Integers and Floats (7 points)

a. In the card game Schnapsen, 5 cards are used (Ace, Ten, King, Queen, and Jack) from 4 suits, so 20 cards in total. What are the minimum number of bits needed to represent a single card in a Schnapsen deck?

b. How many negative numbers can we represent if given 7 bits and using two’s complement?

Consider the following pseudocode (we’ve written out the bits instead of listing hex digits):

```plaintext
int a = 0b0100 0000 0000 0000 0000 0011 1100 0000
int b = (int)(float)a
int m = 0b0100 0000 0000 0000 0000 0011 0000 0000
int n = (int)(float)m
```

c. Circle one: True or False:

```
a == b
```

d. Circle one: True or False:

```
m == n
```

e. How many IEEE single precision floating point numbers are in the range [4, 6) (That is, how many floating point numbers are there where 4 <= x < 6?)
Au17 Final M3
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.

```
0000000000400536 <count_nz>:
  400536:  85 f6  testl %esi,%esi
  400538:  7e 1b  jle  400555 <count_nz+0x1f>
  40053a:  53    pushq %rbx
  40053b:  8b 1f  movl (%rdi),%ebx
  40053d:  83 ee 01  subl $0x1,%esi
  400540:  48 83 c7 04  addq $0x4,%rdi
  400544:  e8 ed ff ff ff  callq 400536 <count_nz>
  400549:  85 db  testl %ebx,%ebx
  40054b:  0f 95 c2  setne %dl
  40054e:  0f b6 d2  movzbl %dl,%edx
  400551:  01 d0  addl %edx,%eax
  400553:  eb 06  jmp 40055b <count_nz+0x25>
  400555:  b8 00 00 00 00  movl $0x0,%eax
  40055a:  c3    retq
  40055c:  5b    popq %rbx
  40055d:  c3    retq
```

(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>0x0000 0000 0040 0544</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x0000 0000 0040 0544</td>
</tr>
<tr>
<td>%eax</td>
<td>0x400534</td>
</tr>
<tr>
<td>%bx</td>
<td>0x400534</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]

```
char v1 = *(charP + _____);  // set v1 = 0xDB
int* v2 = (int*)((___________*)charP - 2);  // set v2 = 0x400534
```
Question 5: Procedures & The Stack  [24 pts]

The recursive function $\text{sum}_r()$ calculates the sum of the elements of an int array and its x86-64 disassembly is shown below:

```c
int sum_r(int *ar, unsigned int len) {
    if (!len) {
        return 0;
    } else
        return *ar + sum_r(ar+1, len-1);
}
```

```c
00000000000400507 <sum_r>:
400507:  41 53     pushq  %r12
400509:  85 f6     testl  %esi,%esi
40050b:  75 07     jne  400514 <sum_r+0xd>
40050d:  b8 00 00 00 00 movl $0x0,%eax
400512:  eb 12     jmp  400526 <sum_r+0x1f>
400514:  44 8b 1f     movl (%rdi),%r12d
400517:  83 ee 01     subl $0x1,%esi
40051a:  48 83 c7 04     addq $0x4,%rdi
40051e:  e8 e4 ff ff ff callq 400507 <sum_r>
400523:  44 01 d8     addl %r12d,%eax
400526:  41 5b     popq %r12
400528:  c3     retq
```

(A) The addresses shown in the disassembly are all part of which section of memory?  [2 pt]

(B) Disassembly (as shown here) is different from assembly (as would be found in an assembly file). Name two major differences:  [4 pt]
(C) What is the return address to sum_r that gets stored on the stack? Answer in hex. [2 pt]

0x

(D) What value is saved across each recursive call? Answer using a C expression. [2 pt]

(E) Assume main calls sum_r(ar, 3) with int ar[] = {3, 5, 1}. Fill in the snapshot of memory below the top of the stack in hex as this call to sum_r returns to main. For unknown words, write “0x unknown”. [6 pt]

(F) Assembly code sometimes uses relative addressing. The last 4 bytes of the callq instruction encode an integer (in little endian). This value represents the difference between which two addresses? Hint: both addresses are important to this callq. [4 pt]

<table>
<thead>
<tr>
<th>value (decimal):</th>
<th>address 1: 0x</th>
<th>address 2: 0x</th>
</tr>
</thead>
</table>

(G) What could we change in the assembly code of this function to reduce the amount of Stack memory used while keeping it recursive and functioning properly? [4 pt]
Wi17 Final Q1
1. C and Assembly (15 points)

Consider the following (partially blank) x86-64 assembly, (partially blank) C code, and memory listing. Addresses and values are 64-bit, and the machine is little-endian. All the values in memory are in hex, and the address of each cell is the sum of the row and column headers: for example, address 0x1019 contains the value 0x18.

Assembly code:
```assembly
foo:
    movl $0, ____
L1:    cmpq $0x0, %rdi
    je L2
    cmp _____, 0x1(%rdi)
    je _____
    mov 0x8(%rdi), %rdi
    jmp _____
L2:    ret
L3:    mov (%rdi), %eax
    jmp L2
```

C code:
```c
typedef struct person {
    char height;
    char age;
    struct person* next_person;
} person;

int foo(person* p) {
    int answer = _____;
    while (________) {
        if (p->age == 24){
            answer = p->_______;
            break;
        }
        p = ________________;
    }
    return answer;
}
```

Memory Listing
Bits not shown are 0.

<table>
<thead>
<tr>
<th></th>
<th>0x00</th>
<th>0x01</th>
<th>...</th>
<th>0x05</th>
<th>0x06</th>
<th>0x07</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>80</td>
<td>1B</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1008</td>
<td>80</td>
<td>1B</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1010</td>
<td>3F</td>
<td>18</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1018</td>
<td>3F</td>
<td>18</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1020</td>
<td>00</td>
<td>00</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1028</td>
<td>18</td>
<td>10</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1030</td>
<td>18</td>
<td>10</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1038</td>
<td>40</td>
<td>40</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1040</td>
<td>40</td>
<td>40</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x1048</td>
<td>00</td>
<td>00</td>
<td>...</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

(a) Given the code provided, fill in the blanks in the C and assembly code.
(b) Trace the execution of the call to `foo((person*) 0x1028)` in the table to the right. Show which instruction is executed in each step until `foo` returns. In each space, place the assembly instruction and the values of the appropriate registers after that instruction executes. You may leave those spots blank when the value does not change. You might not need all steps listed on the table.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>%rdi (hex)</th>
<th>%eax (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl</td>
<td>0x1028</td>
<td>0</td>
</tr>
<tr>
<td>cmpq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>je</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Briefly describe the value that `foo` returns and how it is computed. Use only variable names from the C version in your answer.
Question F5: Caching  [10 pts]

We have 16 KiB of RAM and two options for our cache. Both are two-way set associative with 256 B blocks, LRU replacement, and write-back policies. Cache A is size 1 KiB and Cache B is size 2 KiB.

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

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

(B) The code snippet below accesses an integer array. Calculate the Miss Rate for Cache A if it starts cold. [3 pt]

```c
#define LEAP 4
#define ARRAY_SIZE 512
int nums[ARRAY_SIZE];       // &nums = 0x0100 (physical addr)
for (i = 0; i < ARRAY_SIZE; i+=LEAP)
    nums[i] = i*i;
```

(C) For each of the proposed (independent) changes, write MM for “higher miss rate”, NC for “no change”, or MH for “higher hit rate” to indicate the effect on Cache A for the code above: [3.5 pt]

<table>
<thead>
<tr>
<th>Change</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct-mapped</td>
<td>MM</td>
</tr>
<tr>
<td>Double LEAP</td>
<td>MM</td>
</tr>
<tr>
<td>Increase block size</td>
<td>NC</td>
</tr>
<tr>
<td>Write-through policy</td>
<td>MM</td>
</tr>
</tbody>
</table>

(D) Assume it takes 200 ns to get a block of data from main memory. Assume Cache A has a hit time of 4 ns and a miss rate of 4% while Cache B, being larger, has a hit time of 6 ns. What is the worst miss rate Cache B can have in order to perform as well as Cache A? [2 pt]
Au17 Final F7
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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segmentation fault</th>
<th>Mouse clicked</th>
</tr>
</thead>
<tbody>
<tr>
<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 _______
Sp17 Final Q3
3. Virtual Memory (9 points)

Assume we have a virtual memory detailed as follows:

- 256 MiB Physical Address Space
- 4 GiB Virtual Address Space
- 1 KiB page size
- A TLB with 4 sets that is 8-way associative with LRU replacement

For the following questions it is fine to leave your answers as powers of 2.

a) How many bits will be used for:
   - Page offset? __________
   - Virtual Page Number (VPN)? __________
   - Physical Page Number (PPN)? __________
   - TLB index? __________
   - TLB tag? __________

b) How many entries in this page table?

c) We run the following code with an empty TLB. Calculate the TLB miss rate for data (ignore instruction fetches). Assume \( i \) and \( \text{sum} \) are stored in registers and \( \text{cool} \) is page-aligned.

```c
#define LEAP 8
int cool[512];
... // Some code that assigns values into the array cool
... // Now flush the TLB. Start counting TLB miss rate from here.
int sum;
for (int i = 0; i < 512; i += LEAP) {
  sum += cool[i];
}
```

**TLB Miss Rate:** (fine to leave you answer as a fraction) __________
**Au16 Final Q7**

**Question F7:** Virtual Memory  [10 pts]

Our system has the following setup:

- 24-bit virtual addresses and 512 KiB of RAM with 4 KiB pages
- A 4-entry TLB that is fully associative with LRU replacement
- A page table entry contains a valid bit and protection bits for read (R), write (W), execute (X)

(A) Compute the following values:  [2 pt]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page offset width</td>
<td>_____</td>
<td></td>
</tr>
<tr>
<td>PPN width</td>
<td>_____</td>
<td></td>
</tr>
<tr>
<td>Entries in a page table</td>
<td>_____</td>
<td></td>
</tr>
<tr>
<td>TLBT width</td>
<td>_____</td>
<td></td>
</tr>
</tbody>
</table>

(B) Briefly explain why we make the page size so much larger than a cache block size.  [2 pt]

(C) Fill in the following blanks with “A” for always, “S” for sometimes, and “N” for never if the following get updated during a page fault.  [2 pt]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page table</td>
<td>_____</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swap space</td>
<td>_____</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLB</td>
<td>_____</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>_____</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(D) The TLB is in the state shown when the following code is executed. Which iteration (value of i) will cause the protection fault (segfault)? Assume sum is stored in a register.  [4 pt]

```c
long *p = 0x7F0000, sum = 0;
for (int i = 0; i < 1; i++) {
    if (i%2)
        *p = 0;
    else
        sum += *p;
    p++;
}
```

<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>0x31</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x7F2</td>
<td>0x15</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x004</td>
<td>0x1D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0x7F1</td>
<td>0x2D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

```
```
Question F8: Memory Allocation [9 pts]

(A) Briefly describe one drawback and one benefit to using an implicit free list over an explicit free list. [4 pt]

<table>
<thead>
<tr>
<th>Implicit drawback:</th>
<th>Implicit benefit:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(B) The table shown to the right shows the value of the header for the block returned by the request: \((\text{int}*)\text{malloc}(\text{N} * \text{sizeof(int)})\)
What is the alignment size for this dynamic memory allocator? [2 pt]

<table>
<thead>
<tr>
<th>N</th>
<th>header value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>49</td>
</tr>
<tr>
<td>12</td>
<td>65</td>
</tr>
</tbody>
</table>

(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;
}
```

\[ \text{ZERO} \quad \_\_\_\_\_ \quad \&\text{ZERO} \]
\[ \text{foo} \quad \_\_\_\_\_ \quad \&\text{foo} \]
\[ \text{foo} \quad \_\_\_\_\_ \quad \&\str \]
Wi16 Final Q10

10. C vs. Java (11 points) Consider this Java code (left) and somewhat similar C code (right) running on x86-64:

```java
public class Foo {
    private int[] x;
    private int y;
    private int z;
    private Bar b;
    public Foo() {
        x = null;
        b = null;
    }
}
```

```c
struct Foo {
    int x[6];
    int y;
    int z;
    struct Bar * b;
};
struct Foo * make_foo() {
    struct Foo * f = (struct Foo *)malloc(sizeof(struct Foo));
    f->x = NULL;
    f->b = NULL;
    return f;
}
```

(a) In Java, new Foo() allocates a new object on the heap. How many bytes would you expect this object to contain for holding Foo’s fields? (Do not include space for any header information, vtable pointers, or allocator data.)

(b) In C, malloc(sizeof(struct Foo)) allocates a new object on the heap. How many bytes would you expect this object to contain for holding struct Foo’s fields? (Do not include space for any header information or allocator data.)

(c) The function make_foo attempts to be a C variant of the Foo constructor in Java. One line fails to compile. Which one and why?

(d) What, if anything, do we know about the values of the y and z fields after Java creates an instance of Foo?

(e) What, if anything, do we know about the values of the y and z fields in the object returned by make_foo?