University of Washington — Computer Science and Engineering

Spring 2026 - Instructor: Ruth Anderson - 2026-05-06

CSE 351 MIDTERM

Last Name:	**SAMPLE SOLUTION**
First Name:	
UWNetID:	**@uw.edu**
Name of person to your Left | Right:	
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE351 who haven't taken it yet. Violation of these terms could result in a failing grade. (**please sign**)	

Do not turn the page until you are told to begin.

Instructions

- **Fill out this page of the exam NOW**. Writing after time has been called will not be allowed and will be reported to CSSC.
- This exam contains 10 pages, including this cover page. Show scratch work for potential partial credit, but put your final answers in the boxes and blanks provided.
- A reference sheet is at the end of the exam. You may detach the reference sheet and page 9/10, but we will collect them at the end of the exam.
- The exam is closed book (no laptops, tablets, wearable devices). You are allowed one page (double-sided) of *handwritten* notes. Scientific calculators are allowed.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 70 minutes to complete this exam.

Advice

- Read questions carefully. 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. You can do it!

Question	1	2	3	4	5	Total
Possible Points	16	15	12	20	18	81

Question 1: Number Representation [16 pts]

Given the following variable declarations:

```
signed char x = 0x87;
signed char y = [SOMETHING];
```

For each line of code below, write a value for `y` that would make the largest (most positive) result in `z`. ANSWER IN HEX.

a) `signed char z = x & y;` Any answer that 0's out the sign bit and preserves the last 3 bits is accepted.	`y` = **0x 07**
b) `signed char z = x | y;` Any answer that results in z = 0xFF is accepted. We are stuck with a 1 as a sign bit.	`y` = **0x 78**
c) `signed char z = x ^ y;`	`y` = **0x F8**
d) `signed char z = x - y;`	`y` = **0x 08**
e) `unsigned char z = ((unsigned char) x) >> y;`	`y` = **0x 00**
f) `signed short z = ((signed short) x) << y;`	`y` = **0x 0C**
g) `signed char z = ~(x - y);`	`y` = **0x 07**
h) `signed char z = !(x && y);`	`y` = **0x 00**

Question 2: Pointers and Memory [15 pts]

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

```
// scores starts at address 0x70
short scores[3];
char* name = 0x89;
long* res = 0x78;
```

Word Addr	+0	+1	+2	+3	+4	+5	+6	+7
0x70	05	01	A0	BA	AA	06	3D	4C
0x78	14	5D	09	DA	A2	C8	C9	D4
0x80	1E	02	C3	A9	BA	D9	B6	04
0x88	28	74	00	00	1A	C0	CA	17
0x90	02	10	DF	07	DC	AB	9F	00

(A) **How many bytes** are allocated by the declarations and initializations in the code above?

22 bytes

(B) Consider the following C expressions. What is their **C type** and **Value (in hex)**? Write UNKNOWN if the value cannot be determined from the given information.

Expression (in C)	C Type	Value (in hex)
`scores[2] + 4`	short	0x06AE
`name + 3`	char*	0x8C
`*(res + 2) - 5`	long	0x17CA C01A 0000 7423

(C) Using the memory above, what are the values (in hex) stored in each register after the following x86 instructions are executed? Write UNKNOWN if the value cannot be determined from the given information. *Remember to use the appropriate bit widths.*

```
movq (%rsi), %rbx
movswl 4(%rax), %edi
leaw 4(%rsi,%rax,2), %cx
```

Register	Value (in hex)
%rax	0x0000 0000 0000 0080
%rsi	0x0000 0000 0000 0078
%rbx	0xD4C9 C8A2 DA09 5D14
%edi	0xFFFF D9BA
%cx	0x017C

Question 3: Design Questions [12 pts]

Consider the following two 8-bit floating point formats. **Assume these operate similar to the 32-bit floating point numbers described in class and follow IEEE 754 standards**, just with different bit widths for E and M.

E5M2:

S (1 bit)	E (5 bits)	M (2 bits)

E4M3:

S (1 bit)	E (4 bits)	M (3 bits)

(A) Which format can represent a larger **range** of numbers? **Circle your choice** and justify your response.

E5M2 E4M3 No Difference

Explanation (1 sentence):

E5M2 has more exponent bits and therefore can represent a larger range of numbers.

(B) Which format can represent a greater **precision** of numbers? **Circle your choice** and justify your response.

E5M2 **E4M3** No Difference

Explanation (1 sentence):

E4M3 has more mantissa bits and therefore can represent numbers with a higher precision with the same exponent.

(C) Of the two 8-bit FP representations, which one (E5M2 or E4M3) has the larger bias? (**Circle your choice)** And **what is the value** of the larger bias?

E5M2 E4M3 No Difference

Bias Value (give an exact number not an expression): _____**15**_____

(D) Of the two 8-bit FP representations, which one (E5M2 or E4M3) has the larger max (most positive) normalized value? (**Circle your choice)** And **what is that value**? Give your answer as Decimal Value (in base 10) * 2 ^ exponent.

E5M2 E4M3 No Difference

Max Value: _____**1.75**_____ * 2 ^(_____**15**_____)

Decimal Value (base 10) | Exponent (base 10)

(E) Suppose you're trying to store the decimal value 256.5. Which format can represent it more accurately (represent the value closest to 256.5)? **Circle your choice** and justify your response.

E5M2 E4M3 No Difference

Justification (1-2 sentences):

The maximum value representable by E4M3 is 240. E5M2 can represent 256 (1.00 x 2^8). 256 is closer to 256.5 than 240.

Question 4: C & Assembly [20 pts]

Consider the following x86-64 assembly code:

Note: the xor instruction on line 4 performs a bitwise xor (^) operation

```
1   mystery:
2           testl   %ecx, %ecx
3           jle     .L2
4           xorl    %eax, %eax
5   .L1:
6           movl    (%rdx,%rax,4), %r10d
7           addl    (%rsi,%rax,4), %r10d
8           sarl    $1, %r10d
9           movl    %r10d, (%rdi,%rax,4)
10          addl    $1, %eax
11          cmpq    %rax, %rcx
12          jne     .L1
13          movq    %rdi, %rax
14          retq
15  .L2:
16          movq    %rdi, %rax
17          retq
18
```

(A) Specify the C types of `mystery`'s parameters in the blanks, using however many blanks as are necessary. If an argument is not used, write NA.

Arg1: ___int*_____ Arg2: ___int*_____ Arg3: ___int*_____ Arg4: ___int_____

(B) Here is part of the function `mystery` in the C code. Fill in the blanks for the equivalent for-loop, using **registers as variable names**:

```
for (int i = ____0___; i < ___ecx____; i += __1____)  {

      int sum = ___rdx[i]_____ + _____rsi[i]_____ ;

      _____rdi[i]____ = sum / ___2_____ ;

 }
```

(C) Rewrite line 4 of the assembly using a different instruction (*NOT* `xor`) that has the same effect as the original. Be sure to use the correct instruction suffix and register names.

Line 4: `xorl %eax, %eax` ➜ **movl $0, %eax (or equiv)**

(D) Name a specific value of `sum` (computed on line 7 of the assembly) that would result in incorrect behavior with the C code if we were to replace instruction `sarl` with instruction `shrl` on line 8.

`sarl $1, %r10d` ➜ `shrl $1, %r10d`

Briefly explain how this instruction change would change the behavior of `mystery` for your chosen value of `sum`.

Value of sum (in decimal): any negative number

Brief Explanation (1-2 sentences): This would convert negative values into positive ones so the average stored in rdi[i] would go up.

Question 5: Procedures & The Stack [18 pts]

This question deals with a C program that bakes a cake for a very hungry caterpillar. The C code and disassembly are found on the next page. **Please read through the code before answering the following questions!**

(A) Assume `printf` doesn't call any other functions. If main is executed as it is written, how many total stack frames are created (including `main`)?

8 frames

(B) What is the **maximum depth** of the stack frames generated for this program?

5 frames

(C) What is the return address to `main()` stored at the top of the stack frame for `bake_cake()`?

0x401184

(D) How large is a stack frame for a call to `bake_cake()` at its largest? ANSWER IN DECIMAL!

32 bytes

(G)What is the address of the string "`strawberry`"? Where is this address located in `main`'s disassembly?

Addr of "`strawberry`": **0x402027** Addr in disassembly: **0x401176**

(H) In the disassembly for `bake_cake`, what is the purpose of the instruction at `0x40115b`? What would happen if we did not have this instruction?

Purpose of instruction (1-2 sentences): Put n-1 into the first argument to bake_cake() following calling conventions. (partial credit given for answers that answered in assembly)
What would happen without it (1-2 sentences): Without this instruction, we would never decrement n(*) and thus would never hit the base case. We would have infinite recursion/stack overflow(**). (*) Also correct: %rdi would contain the pointer to "Frosted with %s"/“strawberry”, or (**) If `printf` clobbers %rdi correctly, we can technically get to the base case!

Note: Although you may detach this page, we will collect it at the end of the exam.

Your Name:	
Your UWNetID:	

Consider the following C functions, which is used to bake a cake for a very hungry caterpillar. Feel free to detach this page!

```
int bake_cake(int n, char* flavor) {
   if (n <= 1) {
      return n;
   }
   frost(flavor);
   return 1 + bake_cake(n - 1, flavor);
}
```

```
void frost(char* flavor){
   printf("Frosted with %s", flavor);
}
```

```
int main() {
   char* fav_flavor = "strawberry";
   int n = 3;
   int layers_baked = bake_cake(n, fav_flavor);
}
```

The disassembly for these functions are on the other side → → →

Disassembly for `bake_cake`:

```
0000000000401141 <bake_cake>:
  401141:       89 f8                   mov    %edi,%eax
  401143:       83 ff 01                cmp    $0x1,%edi
  401146:       7e 28                   jle    401170 <bake_cake+0x2f>
  401148:       55                      push   %rbp
  401149:       53                      push   %rbx
  40114a:       48 83 ec 08             sub    $0x8,%rsp
  40114e:       89 fb                   mov    %edi,%ebx
  401150:       48 89 f5                mov    %rsi,%rbp
  401153:       48 89 f7                mov    %rsi,%rdi
  401156:       e8 cb ff ff ff          call   401126 <frost>
  40115b:       8d 7b ff                lea    -0x1(%rbx),%edi
  40115e:       48 89 ee                mov    %rbp,%rsi
  401161:       e8 db ff ff ff          call   401141 <bake_cake>
  401166:       83 c0 01                add    $0x1,%eax
  401169:       48 83 c4 08             add    $0x8,%rsp
  40116d:       5b                      pop    %rbx
  40116e:       5d                      pop    %rbp
  40116f:       c3                      ret
  401170:       c3                      ret
```

And disassemblies for `frost` and `main`:

```
 0000000000401126 <frost>:
  401126:       48 83 ec 08             sub    $0x8,%rsp
  40112a:       48 89 fe                mov    %rdi,%rsi
  40112d:       bf 10 20 40 00          mov    $0x402010,%edi
  401132:       b8 00 00 00 00          mov    $0x0,%eax
  401137:       e8 f4 fe ff ff          call   401030 <printf@plt>
  40113c:       48 83 c4 08             add    $0x8,%rsp
  401140:       c3                      ret
```

```
0000000000401171 <main>:
  401171:       48 83 ec 08             sub    $0x8,%rsp
  401175:       be 27 20 40 00          mov    $0x402027,%esi
  40117a:       bf 04 00 00 00          mov    $0x3,%edi
  40117f:       e8 bd ff ff ff          call   401141 <bake_cake>
  401184:       b8 00 00 00 00          mov    $0x0,%eax
  401189:       48 83 c4 08             add    $0x8,%rsp
  40118d:       c3                      ret
```