The questions shown below are provided by the textbook authors. Not all of them are relevant to CSCI 237. You may safely ignore the ones that have been greyed out or removed.

CSCI 237 Sample Midterm

Problem 1. (10 points):

General systems concepts. Write the correct answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

1. Consider the following code, what is the output of the printf?

```
int x = 0x15213F10 >> 4;
char y = (char) x;
unsigned char z = (unsigned char) x;
printf("%d, %u", y, z);
```

   (a) -241, 15
   (b) -15, 241
   (c) -241, 241
   (d) -15, 15

2. In two's compliment, what is $-TMin$?

   (a) $Tmin$
   (b) $Tmax$
   (c) 0
   (d) -1

3. Let $int\ x = -31/8$ and $int\ y = -31 >> 3$. What are the values of x and y?

   (a) $x = -3, y = -3$
   (b) $x = -4, y = -4$
   (c) $x = -3, y = -4$
   (d) $x = -4, y = -3$

4. In C, the expression "$15213U > -1$" evaluates to:

   (a) True (1)
   (b) False (0)

5. In two's compliment, what is the minimum number of bits needed to represent the numbers -1 and the number 1 respectively?

   (a) 1 and 2
   (b) 2 and 2
   (c) 2 and 1
   (d) 1 and 1

6. Consider the following program. Assuming the user correctly types an integer into stdin, what will the program output in the end?

```
#include <stdio.h>
int main(){
    int x = 0;
    printf("Please input an integer:");
    scanf("%d",x);
    printf("%d", (!!x)<<31);
}
```

   (a) 0
   (b) $TMin$
   (c) Depends on the integer read from stdin
   (d) Segmentation fault

7. By default, on Intel x86, the stack

   (a) Is located at the bottom of memory.
   (b) Grows down towards smaller addresses
   (c) Grows up towards larger addresses
   (d) Is located in the heap

8. Which of the following registers stores the return value of functions in Intel x86_64?

   (a) %rax
   (b) %rcx
   (c) %rdx
   (d) %rip
   (e) %cr3

9. The `leave` instruction is effectively the same as which of the following:

   (a) `mov %ebp, %esp`
       `pop %ebp`
   (b) `pop %eip`
   (c) `mov %esp, %ebp`
       `pop %esp`
   (d) `ret`

10. Arguments to a function, in Intel IA32 assembly, are passed via

   (a) The stack
   (b) Registers
   (c) Physical memory
   (d) The `.text` section
   (e) A combination of the stack and registers.

11. A buffer overflow attack can only be executed against programs that use the `gets` function.

   (a) True
   (b) False

12. Intel x86_64 systems are

   (a) Little endian
   (b) Big endian
   (c) Have no endianess
   (d) Depend on the operating system

13. Please fill in the return value for the following function calls on both an Intel IA32 and Intel x86_64 system:

Function	Intel IA32	Intel x86_64
`sizeof(char)`		
`sizeof(int)`		
`sizeof(void *)`		
`sizeof(int *)`		

14. Select the two's complement negation of the following binary value: `0000101101`:

   (a) `1111010011`
   (b) `1111010010`
   (c) `1000101101`
   (d) `1111011011`

15. Which line of C-code will perform the same operation as `leal 0x10(%rax,%rcx,4),%rax`?

   (a) `rax = 16 + rax + 4*rcx`
   (b) `rax = *(16 + rax + 4*rcx)`
   (c) `rax = 16 + *(rax + 4*rcx)`
   (d) `*(16 + rcx + 4*rax) = rax`
   (e) `rax = 16 + 4*rax + rcx`

16. Which line of Intel x86-64 assembly will perform the same operation as `rcx = ((int *)rax)[rcx]`?

   (a) `mov (%rax,%rcx,4),%rcx`
   (b) `lea (%rax,%rcx,4),%rcx`
   (c) `lea (%rax,4,%rcx),%rcx`
   (d) `mov (%rax,4,%rcx),%rcx`

17. If `a` is of type `(int)` and *b* is of type `(unsigned int)`, then `(a < b)` will perform

   (a) An unsigned comparison.
   (b) A signed comparison.
   (c) A segmentation fault.
   (d) A compiler error.

18. Denormalized floating point numbers are

   (a) Very close to zero (small magnitude)
   (b) Very far from zero (large magnitude)
   (c) Un-representable on a number line
   (d) Zero.

19. What is the difference between an arithmetic and logical right shift?

   (a) C uses arithmetic right shift; Java uses logical right shift.
   (b) Logical shift works on 32 bit data; arithmetic shift works on 64 bit data.
   (c) They fill in different bits on the left
   (d) They are the same.

20. Which of the following assembly instructions is invalid in Intel IA32 Assembly?

   (a) `pop %eip`
   (b) `pop %ebp`
   (c) `mov (%esp),%ebp`
   (d) `lea 0x10(%esp),%ebp`

Problem 2. (10 points):

Floating point encoding. Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.

- There are $k = 3$ exponent bits. The exponent bias is 3.
- There are $n = 2$ fraction bits.

Recall that numeric values are encoded as a value of the form $V = M \times 2^E$, where E is the exponent after biasing, and M is the significand value. The fraction bits encode the significand value M using either a denormalized (exponent field 0) or a normalized representation (exponent field nonzero). The exponent E is given by $E = 1 - \textit{Bias}$ for denormalized values and $E = e - \textit{Bias}$ for normalized values, where e is the value of the exponent field `exp` interpreted as an unsigned number.

Below, you are given some decimal values, and your task it to encode them in floating point format. In addition, you should give the rounded value of the encoded floating point number. To get credit, you must give these as whole numbers (e.g., 17) or as fractions in reduced form (e.g., $3/4$). Any rounding of the significand is based on ***round-to-even***, which rounds an unrepresentable value that lies halfway between two representable values to the nearest even representable value.

Value	Floating Point Bits	Rounded value
$9/32$	`001 00`	1/4
1		
12		
11		
$1/8$		
$7/32$		

This one is more complicated than anything you'll be asked.

Problem 5. (10 points):

Switch statement encoding. Consider the following C code and assembly code for a strange but simple function:

```
int lol(int a, int b)
{
    switch(a)
    {
        case 210:
            b *= 13;
            ________
        case 213:
            b = 18243;
            ________
        case 214:
            b *= b;
            ________
        case 216:
        case 218:
            b -= a;
            ________
        case 219:
            b += 13;
            ________
        default:
            b -= 9;
    }

    return b;
}
```

```
40045c <lol>:
40045c: lea    -0xd2(%rdi),%eax
400462: cmp    $0x9,%eax
400465: ja     40048a <lol+0x2e>
400467: mov    %eax,%eax
400469: jmpq   *0x400590(,%rax,8)
400470: lea    (%rsi,%rsi,2),%eax
400473: lea    (%rsi,%rax,4),%eax
400476: retq
400477: mov    $0x4743,%esi
40047c: mov    %esi,%eax
40047e: imul   %esi,%eax
400481: retq
400482: mov    %esi,%eax
400484: sub    %edi,%eax
400486: retq
400487: add    $0xd,%esi
40048a: lea    -0x9(%rsi),%eax
40048d: retq
```

Using the available information, fill in the jump table below. (Feel free to omit leading zeros.) Also, for each case in the `switch` block which should have a `break`, write `break` on the corresponding blank line.

Hint: `0xd2 = 210` and `0x4743 = 18243`.

`0x400590:` ________________ `0x400598:` ________________

`0x4005a0:` ________________ `0x4005a8:` ________________

`0x4005b0:` ________________ `0x4005b8:` ________________

`0x4005c0:` ________________ `0x4005c8:` ________________

`0x4005d0:` ________________ `0x4005d8:` ________________