CSED211: Microprocessor & Programming, 2017 Fall, Mid-term Exam-I

Student Id:__________________ Name: ____________________

1. (10 pts) Answer to the following questions related to IEEE floating-point format.

Consider the following 8-bit floating point representation based on the IEEE floating point format.

- one sign bit
- There are k = 3 exponent bits. The exponent bias is 3.
- There are n = 4 fraction bits.

Recall that numeric values are encoded as a value of the form V = M × 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 de-normalized (exponent field 0) or a normalized representation (exponent field nonzero). The exponent E is given by E = 1 − *Bias* for de-normalized values and E = e − *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 un-representable value that lies halfway between two representable values to the nearest even representable value.

Value	Floating Point Bits	Rounded value
9/16	`0 010 0010`	9/16
-71/512		
-35/128		
34/16		
69/16		
-37/16		

2. Answer to the following questions.

a. (3 pts) Explain the difference between the following three instructions

```
- cmpq          %rax, %rbx
- testq         %rax, %rbx
- subq          %rax, %rbx
```

b. (2 pts) Look at the following code in the given address. What is the address to be jumped by executing call function? (Assume leading 0's are omitted and you can also omit leading 0's)

```
0x08028800          call    %rip+0x0213
0x08028804          movl %rax, %rdx
```

c. (15 pts) The following is a list of compiled assembler code. For each assembler code, identify whether it accesses memory or not. And what is the address of accessed memory location if it is. (assume x86-64)

```
pushq     %rbx                        ( Y / N ) (                    )
addq      $-16, %rsp                  ( Y / N ) (                    )
movq      Array1(, %rdx, 4), %rax     ( Y / N ) (                    )
leaq      (%rax,%rax,2), %rax         ( Y / N ) (                    )
addq      $16, %rsp                   ( Y / N ) (                    )
popq      %rbx                        ( Y / N ) (                    )
ret                                   ( Y / N ) (                    )
```

%rax = 2, %rbx = 32, %rcx = 4, %rdx = 1, %rbp = 0xFF0210

%rsp= 0x0000FFFFFFFE0010, Array1=0x00001000

d. (3 pts) The following is the compiled code of the program that calls a function **'foo'.** Write the following information about the function **'foo'.**

- Number of arguments passed
- Whether the function **'foo'** reserves a frame in a stack or not.
- The return value type

```
foo: addl       %edx, %esi
     imull      %esi, %ecx
     addl       %ecx, %edi
     addl       %r8d, %edi
     cvtsi2ss   %edi, %xmm0        #data convert instruction
     ret
```

3. (2 pts each) Answer to the following questions with a few sentences
   a. Code motion
   b. Loop unrolling
   c. 2-bit loop branch prediction
   d. Instruction-level parallelism
   e. Memory aliasing

4. (2 pt each) Evaluate the following expression as ‘Always True’ or ‘Couldbe False’
   a. X >> 3 == X / 8 # Initialized int X = foo();
   b. X * X >= 0
   c. UX > - 1 # Initialized unsigned int UX = X;
   d. (X | -X)>>31 == -1

5. (6 pts) The following is the commonly used code for machine learning, known as Multiplication-Add of matrix and vector. Find the value of M and N. (Assume x86-64)

```
Void MAC (float W[N][M],
float X[M], float B[N],
float Y[N] ) {

  float temp;
  int i, j;

  for (i = 0; i < N; i++) {
    temp = 0.0;
    for (j = 0; j < M; j++)
       temp +=
          X[j] *  W[i][j];

    Y[i] = temp + B[i];
  }
}
```

Hint) %rdi = W, %rsi = X, %rdx = B, %rcx = Y

```
MAC: movl    $0, %r9d
     movl    $0x00000000, %r10d
     jmp     .L2
.L3: movss   (%rsi,%rax), %xmm1
     mulss   (%r8,%rax), %xmm1
     addss   %xmm1, %xmm0
     addq    $4, %rax
     cmpq    $24, %rax
     jne     .L3
     addss   (%rdx,%r9), %xmm0
     movss   %xmm0, (%rcx,%r9)
     addq    $4, %r9
     addq    $24, %rdi
     cmpq    $32, %r9
     je      .L1
.L2: movq    %rdi, %r8
     movl    $0, %eax
     movl    %r10d, -4(%rsp)
     movss   -4(%rsp), %xmm0
     jmp     .L3
.L1: rep
     Ret
```

6. Answer to the following questions related to ‘struct’ and ‘union’ constructs in C. Consider the C code written below and compiled on Linux x86-64 system using GCC.

```
struct Node {
   char       c;
   double    value;
   struct Node* next;
   int flag;
   struct Node* left;
   struct Node* right;
};

union Unode
         {   char       sc;
             double    du;
             short int    si;
         };
```

```
static struct Node * NodeTree[N];

struct Node * fun1(int i) {
   return NodeTree[i]->right->left->left;
}
int fun2(int i) {
   return NodeTree[i]->flag;
}
int fun3(int i) {
   return NodeTree[i]->next->next->flag;
}
char fun4(int i) {
   return NodeTree[i]->left->left->c;
}
```

a. (4 pts) What is the size of struct Node? And how many bytes are wasted for padding?

b. (2 pts) What is the size of union UNode??

c. (6 pts) Which of the following corresponds to functions fun1, fun2, fun3, and fun4? There are two extra codes that do not match with the given functions.

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movq     40(%rax), %rax
movq     32(%rax), %rax
movq     32(%rax), %rax
ret
```

ANSWER: ________□

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movl     (%rax), %rax
movl     24(%rax), %eax
ret
```

ANSWER: ________□

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movq     32(%rax), %rax
movq     32(%rax), %rax
movzbl   (%rax), %eax
movb     %al, -1(%rbp)
ret
```

ANSWER: ________□

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movq     24(%rax), %rax
movq     24(%rax), %rax
movl     32(%rax), %eax
ret
```

ANSWER: ________□

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movl     24(%rax), %eax
ret
```

ANSWER: ________□

```
movslq   %edi, %rdi
movq     NodeTree(,%rdi,8), %rax
movq     16(%rax), %rax
movq     16(%rax), %rax
movl     24(%rax), %eax
ret
```

ANSWER: ________□

7. (10 pts) Answer to the questions related to following function (Assume x86-64) The assembly source code on the right column is an optimized code. Fill in the left blank part labeled (a) to (e). (Assume x86-64)

```
void loop(char *h, int len)
{
  char *t;
  for(_(a)_; _(b)_; h++,t--)
  {
     __(c)__________;
     __(d)__________;
     __(e)__________;
  }
  return;
}

Hint) xor operator ^
  %rdi = h
  %rsi = len
```

```
loop: movslq  %esi, %rsi
      addq    %rdi, %rsi
      cmpq    %rsi, %rdi
      jae     .L1
.L4:  movzbl  (%rsi), %eax
      xorb    (%rdi), %al
      movb    %al, (%rdi)
      xorb    (%rsi), %al
      movb    %al, (%rsi)
      xorb    %al, (%rdi)
      addq    $1, %rdi
      subq    $1, %rsi
      cmpq    %rsi, %rdi
      jb     .L4
.L1:  rep
      Ret
```

8. (6 pts) The followings are the schemes to avoid the buffer overflow attack. Match the correct ones. Also, indicate whether the protection scheme is working for ROP attack or not.

Stack canary (/ Y or N)	a) Stack start address is randomized so that it is hard to change the control to the injected code.
Use a safe function (/ Y or N)	b) System puts a control bit to each memory section so that code in a data section cannot be executed.
Non-executable code segment (/ Y or N)	c) Vulnerable functions are replaced with non-vulnerable functions that check input length or input parameters
Randomized stack offset (/ Y or N)	d) Put a special data into a stack as soon as the code enters a called function and check the special data unchanged before returning to the calling function.

9. (15 pts) Consider the following assembly code for a strange but simple function that is implemented with switch statement. Fill in each underbar with either meaningful or empty statement. And also fill in the jump table.

```
<lol>:
.L1:   leal   -10(%rdi), %edx
       cmpl   $9, %edx
       ja     .L15
.L2:   movl   %edx, %edx
       jmp    *.L30(,%rdx,8)
.L3:   movl   %esi, %eax
.L4:   sall   $4, %eax
.L5:   subl   %esi, %eax
.L6:   ret
.L7:   movl   %esi, %eax
.L8    imull  %esi, %eax
.L9    ret
.L10:  movl   %esi, %eax
.L11:  subl   %edi, %eax
.L12:  ret
.L13:  leal   15(%rsi), %eax
.L14:  ret
.L15:  leal   -10(%rsi), %eax
.L16:  ret
.L17:  movl   $12345, %eax
       ret


Hint) %rdi = a, %rsi = b
```

```
int lol(int a, int b)
{    switch(a)
     {  case 10:
            b *= 15;
            __(1)___;
        case 13:
            b = 12345;
            __(2)___;
        case 14:
            b *= b;
            __(3)___;
        case 16:
            __(4)___;
            __(5)___;
        case 18:
            b -= a;
            __(6)___;
        case 19:
            b += 15;
            __(7)___;
        default:
            b -= 10;
            __(8)___;
     };
     return b;
}
```

.L30: __________________ .L30+0x08: __________________

.L30+0x10: __________________ .L30+0x18: __________________

.L30+0x20: __________________ .L30+0x28: __________________

.L30+0x30: _________.L10_____ .L30+0x38: ___.L15__________

.L30+0x40: _________.L10_____ .L30+0x48: __________________