CSED211: Introduction to Computer SW Architecture
2019 Fall, Mid-term Exam

Student Id:__________________ Name: ____________________

Problem	Points	Score
1	10	
2	5 + extra	
3	5	
4	10	
5	6	
6	5	
7	5	
8	5	
9	10	
10	15	
11	10	
12	12	
Total	98 + extra	

1. (10 pts) Answer to the following questions related to IEEE floating-point format. Consider the following 12-bit floating-point representation based on the IEEE floating point format. The format has 1 sign bit, 5 exponent bit (k = 5), and 6 fraction bits (n = 6). The exponent bias is 15.

   Fill in the table that follows for each of the numbers given, with the following meaning for each column. Hex: the three hexa-decimal digit, M: the significand, E: the exponent, V: the numeric value represented. Give the numeric value as whole numbers (e.g., 17) or as fractions (e.g., 17/64) if asked to provide. You may use exponential form if the value is difficult to present as whole numbers or fractions. If necessary, you should apply the round-to-even rounding rule. For your information, $V = (-1)^s \times M \times 2^E$ => s(1) + exp(5) + fraction(6)

Description	Hex	M (bit pattern) 0.XX.. or 1.XX	E(value)	V
-0				-0
Smallest value > 4				
Largest denormalized				
-Inf				-Inf
Value -63/128				
Value	0X 3BB			

2. (1 pt each) Evaluate the following expression as 'Always True' or 'Not'. If your answer is 'Not', show a counterexample (1 extra point).

```
a. (True/Not)  X<=0  ➔  -X>=0    # Initialized  int X = foo();

b. (True/Not) (X | -X)>>31 == -1

c. (True/Not)  F == (float)(double) F   #    D:double,    F:single
   precision f-p variables

d. (True/Not)  F = (float) D  ➔  F * F == (float) D * D

e. (True/Not)  X == (int)(double) X
```

3. (5 pts) The followings are the items to be considered in designing CPU. Point out the item(s) that is (are) not included in ISA (Instruction Set Architecture) definition.

   Word size, Big endian/Little endian, Supported Basic Data Types,
   Assembler Instructions, Addressing mode,
   Pipeline depth, Number of functional units in ALU
   General-Purpose Registers, Condition Flags, Brach Prediction Unit
   Memory address space, Stack size, Memory Layout

4. Answer to the following questions.

   a. (6 pts) Calculate the memory space allocated for each definition. (X86-64)

`union Unode {`
` float f;`
` unsigned long u;`
` char c;`
`}  Ud;`
	`struct Snode {`
	` char   c;`
	` char   d;`
	` short int`
	`        sary[2];`
	` char e;`
	`} Sd;`
		`struct  Snode  ArSn[3];`
		
		`union Unode  * ArUn[3];`

```
Ud : (     );   Sd:  (     );   ArSn: (       ),  ArUn: (     )
```

   b. (4 pts) For the following machine code and disassembled instructions, find the value of XXXXXX and YYYYYY in hexadecimal. For your information, the first byte of machine codes for `'je'` and `'callq'` represents the operation code and remaining part is offset.

```
4003fa: 74  02                je     XXXXXX  ____________
4003fc:  e8  eb ff ff ff      callq  YYYYYY  ____________
400401:  48 89 e7             mov   %rsp,%rdi
```

5. (1 or 2 pts each) Explain concisely **when** and **why** the following scheme enhances program execution performance.

   a. Code motion. – **when and why**

   b. Loop unrolling even it has single functional unit. – **when and why**

   c. Inline substitution for procedure call (before applying code optimization) -- **why**

   d. When finding column-wise sum of 2-D array and storing them into a 1-D vector, use of a temporal variable instead of updating 1-D vector directly. – **why**

6. (5 pts) What is **the result** of array B when the following code is executed? What is **the problem of this code**?

`/* Sum rows is of n X n matrix a`
`   and store in vector b  */`
`void sum_rows1(double *a,`
`  double *b, long n) {`
`    long i, j;`
`    for (i = 0; i < n; i++) {`
`      b[i] = 0;`
`      for (j = 0; j < n; j++)`
`          b[i] += a[i*n + j];`
`    }`
`}`
	`double A[9] =`
	`       { 4,   9,   2,`
	`         3,   5,  7,`
	`         8,   1,  6};`
	
	`double B[] = A+2;`
	
	`sum_rows1(A, B, 3);`

7. (5 pts) Fill in the table for register and flags

%rax = 1

Instruction	%rax	SF	CF	OF	ZF
`orq   %rax, %rax`					
`subq   $-1, %rax`					
`cmpq   $1, %rax`					
`setb     %al`					
`movzblq  %al, %eax`					

8. (5 pts) The following is a list of compiled assembler code. For each assembler code, identify whether it accesses memory or not. Do not count memory access for instruction read (assume x86-64)

```
movss        (%r8,%rax), %xmm1     // a. ( Y/N )
leaq         (%rdi,%rbx), %r8      // b. ( Y/N )
movq         %rdx, %r11            // c. ( Y/N )
jmp          *.L6(,%rax,8)         // d. ( Y/N )
popq         %rbx                  // e. ( Y/N )
```

9. The following is the matrix-matrix multiplication c code and its compiled assembly code (Assume x86-64).

```c
void MMC (float W[L][M],
          float X[M][N],
          float Y[L][N] ) {

 float temp;
 int i, j, k;

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

Hint) %rdi = W, %rsi = X, %rdx
= Y
```

```asm
MMC: pushq %rbx
  movl  $0, %ebx
  movl  $0x00000000, %r10d
  jmp   .L2
.L3:
  movss  (%r8,%rax), %xmm1
  mulss  (%rcx), %xmm1
  adds    %xmm1, %xmm0
  addq    $4, %rax
  addq    $24, %rcx
  cmpq    $20, %rax
  jne   .L3
  movss   %xmm0, (%r11,%r9,4)
  addq    $1, %r9
  cmpq    $6, %r9
  je    .L4
.L6:
  movslq  %r9d, %rax
  leaq    (%rsi,%rax,4), %rcx
  movl    $0, %eax
  movl    %r10d, -4(%rsp)
  movss   -4(%rsp), %xmm0
  jmp   .L3
.L4:
  addq     $24, %rdx
  addq     $20, %rbx
  cmpq     $80, %rbx
  je    .L1
.L2:
  leaq    (%rdi,%rbx), %r8
  movq    %rdx, %r11
  movl    $0, %r9d
  jmp   .L6
.L1:
  popq    %rbx
  ret
```

a. (6 pts) Find the value of **L, M**, and **N** from the compiled assembler code.

b. (4 pts) Where are those local variables **temp, i, j,** and **k** stored?

temp ______________

i ______________

j ______________

k ______________

10. 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.

<pre>struct s1 {
   short x;
   int   y;
};
struct s2 {
   struct s1 a;
   struct s1 *b;
   int   x;
   char   c;
   long   y;
   char   e[3];
   int   z;
};</pre>
	<pre>short fun1(struct s2 *s) {
	   return s->a.x;
	}
	void *fun2(struct s2 *s) {
	   return &s->z;
	}
	int fun3(struct s2 *s) {
	   return s->z;
	}
	short fun4(struct s2 *s) {
	   return s->b->x;
	}</pre>

a. (2 pts) What is the size of struct s2? And how many bytes are wasted for padding?

b. (5 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.

<pre>movq   24(%rdi), %rax
ret
 	
ANSWER: _________</pre>
	<pre>movzwl (%rdi), %eax
	ret
	
	ANSWER: _________</pre>

<pre>movl   36(%rdi), %eax
ret
 	
ANSWER: _________</pre>
	<pre>movq 8(%rdi), %rax
	movzwl (%rax), %eax
	ret
	
	ANSWER: _________</pre>

<pre>movl   4(%rdi), %eax
ret
 	
ANSWER: _________</pre>
	<pre>leaq 36(%rdi), %rax
	ret
	
	ANSWER: _________</pre>

c. Assume a variable is declared as **`struct s2 myS2`**; and the storage for this variable begins at address 0xbfb2ffc0. Each hexadecimal notation should be considered as a whole value of 32-bit. When we consider byte-ordering, the address mapping for each byte could be different. In this exam, we consider Intel X86-64.

```
(gdb) x/20w &myS2
0xbfb2ffc0: 0x0000000f      0x000000d5      0xbfb2ffe8      0x00000000
0xbfb2ffd0: 0x00000000      0xb7f173ff      0x0000012c      0x00000000
0xbfb2ffe0: 0xbf030102      0x0000000c      0xbfb30012      0x000000f3
0xbfb2fff0: 0xb7e2e0b9      0xb7f15ff4      0xbfb30058      0xb7e1adce
0xbfb30000: 0x00000001      0xbfb30084      0xbfb3008c      0xbfb30010
```

(1 pt each) Fill in all the blanks below.

What would be returned by:

```
fun1(&myS2) = 0x____________
fun2(&myS2) = 0x____________
fun3(&myS2) = 0x____________
fun4(&myS2) = 0x____________
```

What is the value of:

```
myS2.b->y =       0x____________
myS2.a.y =  0x____________
myS2.z =    0x____________
myS2.e[1] =       0x____________
```

11. (10 pts) Explain the following terminologies which refer the attacks using runtime stack and schemes to prevent them.

Buffer overflow	
Return-oriented programming	
Stack canary	
Randomized stack offset	
Non-executable data segment	
Secure coding (use of safe functions)	Avoid the use of vulnerable functions. At the coding time, a tool is used to find whether unsafe functions are used and tries to replace it with a safe function.

12. (12 pts) Consider the following assembly code for a strange but simple function that is implemented with switch statement. Fill in (a) ~ (j) with address label and (k) – (l) with meaningful or empty statement.

```
swing:   movq     %rbx, -16(%rsp)
         movq     %rbp, -8(%rsp)
         subq     $24, %rsp
         movq     %rsi, %rbx
         movl     $0, %eax
         testq    %rsi, %rsi
         je       .L2
         cmpl     $9, %edi
         ja       .L12
         movl     %edi, %edi
         jmp      *.L10(,%rdi,8)
.L3:     movswl   32(%rsi), %eax
         jmp      .L2
.L4:     movq     16(%rsi), %rsi
         movl     $1, %edi
         call     swing
         jmp      .L2
.L5:     movq     (%rsi), %rsi
         movl     $1, %edi
         call     swing
         jmp      .L2
.L6:     movq     (%rsi), %rsi
         movl     $1, %edi
         call     swing
         movl     %eax, %ebp
         jmp      .L7
.L13:    movl     $0, %ebp
.L7:     movq     8(%rbx), %rsi
         movl     $1, %edi
         call     swing
         addl     %ebp, %eax
         jmp      .L2
.L8:     movq     24(%rsi), %rsi
         movl     $7, %edi
         call     swing
         jmp      .L2
.L9:     movq     16(%rsi), %rsi
         movl     $1, %edi
         call     swing
         movl     %eax, %ebp
         movq     24(%rbx), %rsi
         movl     $1, %edi
         call     swing
         addl     %ebp, %eax
         jmp      .L2
.L12:    movl     $-1, %eax
.L2:     movq     8(%rsp), %rbx
         movq     16(%rsp), %rbp
         addq     $24, %rsp
         ret

Hint) %rdi = i, %rsi = snode
```

```
struct Node {
   struct Node * left;
   struct Node * right;
   struct Node * up;
   struct Node * down;
   short  val;
} N_data;

int swing (int i, struct Node * snode)
{

  int temp = 0;
  if (!snode) return 0;

  switch(i)
  { case 1:
      temp = snode->val;
      break;
    case 2:
      temp = swing (1, snode->up);
      break;
     case 4:
      temp = swing (1, snode->left);
      break;
     case 5:
      temp = swing (1, snode->left);
      _(k)__________
     case 6:
      temp += swing (1, snode->right);
      _(l)__________
     case 8:
      temp = swing (i - 1, snode->down);
      break;
     case 9:
      temp = swing (1, snode->up) +
             swing (1, snode->down);
      break;
     default:
      temp = -1;
     };
 return  temp;
}
```

(a)__________: ____.L12______ (a)+0x08: (b)______________

(a)+0x10: (c)______________ (a)+0x18: (d)______________

(a)+0x20: (e)______________ (a)+0x28: (f)______________

(a)+0x30: (g)______________ (a)+0x38: (h)______________

(a)+0x40: (i)______________ (a)+0x48: (j)______________