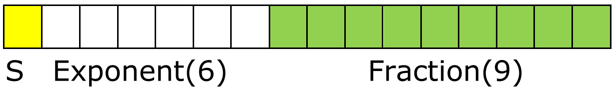
Introduction to Computer Systems 2011
Second Midterm Examination

Name_____________ Student No._______________ Score__________

Problem 1: (16 points)

Number Conversion: IEEE 754 single precision float standard with a little change is illustrated below.



1) Filling the blanks with proper values. (1' * 2)

   Normalized: $(-1)^{\textbf{sign}}$ * (1.fraction) * $2^{\textbf{exponent-bias}}$, where **bias**=_____;
   Denormalized: $(-1)^{\textbf{sign}}$ * (0.fraction) * $2^{\textbf{E}}$, where **E**=_____;
   Zero: all 0's in all 3 fields

2) Convert the number **$(-4.1171875)_{10}$** into IEEE 754 FP single precision representation (in hex). (4')

3) What is the equivalent value as a decimal number? (3'*2 = 6')

   $(0\ 000000\ 000001000)_2$ and **$(0\ 000100\ 000010000)_2$**

4) Calculate both the sum of **$(0\ 000000\ 000001000)_2$** and **$(0\ 000100\ 000010000)_2$**, and then round the results to **5** bits to the right of the binary point with **Round-to-Even** rounding modes. (NOTE: Please give your steps detailed) (4')

Problem 2: (24points)

Suppose the following code is executed on a 32-bit machine, where **long long** is 8-byte**, int** is 4-byte, **short** is 2-byte, **char** is 1-byte and **pointer** is 4-byte. Please read the code and answer the following questions.

```
#include <stdio.h>

struct data {
    char *p;
    int i;
    union{
        struct{
            int ii;
            short s[3];
        }s1;
        long long l;
    }u1;
    char c;
};

int main(void)
{
    struct data array[2];
    struct data *d = &array[1];
    printf("length: %d, start: %p \n", sizeof(struct data), d);
    printf("data: %p %p %p %p %p %p\n", &d->p, &d->i,
            &d->u1.s1.ii, &d->u1.s1.s, &d->u1.l, &d->c);
    return 0;
}
```

1. Suppose the start address of array is **0xbfbf5224**, please fill the output of above program. (1′ * 8 = 8′)

```
length: __[1]__, start: __[2]__
data: __[3]___, __[4]__, __[5]__, __[6]__, __[7]__, __[8]__
```

```
struct {
   char a[9];
   short b[3];
   float c;
   char d;
   int e;
   char *f;
   short g;
} foo;
```

2. Show how the **struct foo** above would appear on a 64-bit ("x86-64") Windows machine (NOTE primitives of size k are k-byte aligned). Label the bytes that belong to the various fields with their names and clearly mark the end of the **struct foo**. Use hatch marks or x's to indicate bytes that are allocated in the **struct foo** but are not used. (6')

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**

3. Rearrange the above fields in **struct foo** to conserve the most space in the memory below. Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks or x's to indicate bytes that are allocated in the **struct foo** but are not used. (6')

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**

4. How many bytes are wasted in **original** struct foo, inside and after the struct? (suppose the next memory value is a pointer) (1' * 2 = 2')

5. How many bytes are wasted in **rearranged** struct foo, inside and after the struct? (suppose the next memory value is a pointer) (1' * 2 = 2')

Problem 3: (16points)

Suppose the following C code and assembly code are executed on a 32-bit **little endian** machine. Read the code and fill in the blanks. (2’*8=16’)

C code:

```
int lolwut(void)
{
   int i, n = 20;
   switch(__[1]__){
      case 5: i = __[2]__; break;
      case 7: i = __[3]__; break;
      case 6: i = 7; break;
      default: i = -1;
   }
   return i;
}
```

Assembly Code:

```
lolwut:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $20, %esp
   movl    $20, -4(%ebp)
   movl    -4(%ebp), %edx
   movl    %edx, %eax
   shrl    $1, %eax
   subl    __[4]__, %eax
   movl    %eax, -20(%ebp)
   cmpl    $5, -20(%ebp)
   ja     __[5]__
   movl    -20(%ebp), %edx
   movl    __[6]__, %eax
   jmp     *%eax
.section .rodata
.align 4
.L9:
   .long   __[7]__
   .long   .L6
   .long   __[8]__
   .text
.L3:
   movl    $9, -8(%ebp)
   jmp     .L10
```

```
.L7:
   movl    $8, -8(%ebp)
   jmp     .L10
.L6:
   movl    $7, -8(%ebp)
   jmp     .L10
.L2:
   movl    $-1, -8(%ebp)
.L10:
   movl    -8(%ebp), %eax
   leave
   ret
size   lolwut, .-lolwut
```

Problem 4: (24points)

This problem covers Lab2. As in Lab2 we get the BOMB program which reads a hex-string from standard input with a function getbuf having the following C code.

```
int getbuf(void)
{
   char buf[24];
   gets(buf);
   return 0;
}
```

Suppose that Your **cookie number** is **1234(0x04D2)** and there are three functions in the BOMB program: getbuf(), test(), and ICSExam().

The start address of the function getbuf() is **0x8048ab0**
The start address of the function test() is **0x8048a40**
The start address of the function ICSExam() is **0x80489ec**

The function getbuf() is called within **BOMB** by a function test having the following C code:

```
void test(void)
{
   int val, i=0;
   printf("Type Hex string: ");
   val = getbuf();
   if(val == cookie)
      printf("\n Bomb! You have passed level 2!\n");
}
```

While the function ICSExam having the following C code:

```
void ICSExam(int val)
{
   if (val == cookie)
      printf("\n ICSExam! You passed ICS Exam!\n ");
   else
      printf("\nMisfire!\n");
   exit(0);
}
```

When the program BOMB stops at the breakpoint which was set within the function getbuf() using the command "breakpoint getbuf". We get the following information: **%esp= 0xbfbe64f0** and **%ebp= 0xbfbe6518**. (NOTE: these information can be used both in question 1 and 2)

1. In order to "pass ICS Exam", you must complete the following steps:

   i. First you must make the program run function **ICSExam()**, so you should replace the return address with ____[1]____ (2')

   ii. How can you make it appear to ICSExam as if you have passed your cookie number as its argument. ____[2]____ (4')

   iii. Last write down your input hex-string here (use code 0x00 to fill in the hex-digital you doesn't mind its value) ____[3]____ (4')

2. Now, change the ICSExam function as follows:

```
int global_value = 0;
void ICSExam(){
    if (global_value == cookie)
        printf("\n ICSExam! You passed ICS Exam!\n ");
    else
        printf("\nMisfire!\n");
    exit(0);
}
```

Again, you need to "pass ICS Exam". Write the right answers in the following blanks to achieve this goal.

Given:

Some memory values are provided in **Table1**

The assemble code of ICSExam is shown in **Table2**.

Some assemble code to binary code mapping in **Table3**.

Table 1: Piece of memory

Address(Hex)	Value(Hex)
0804ba1c	36b5
0804ba20	a02c
0804ba24	04d2
0804ba28	1234
0804ba2c	090b
0804ba30	0000

Table 2: Piece of assemble code

```
<ICSExam>:
 push  %ebp
 mov   %esp, %ebp
 sub   $0x14, %esp
 push  $0x2
 mov   0x804ba24, %eax
 add   $0x10, %esp
 cmp   0x804ba30, %eax
 je    <pass_exam>
 jmp   <mis_fire>
 .....
```

Table 3

```
c7 05 xx xx xx xx D2 04 00 00   ⇔ movl $04D2, xx xx xx xx
68 xx xx xx xx                  ⇔ push xx xx xx xx
c3                              ⇔ ret
```

(1) First, you must find the address of **global_value**, the address is __**[1]**__. (2’)

(2) In order to use “ret” to goto ICSExam(), You must push an address to the stack, the address is __**[2]**__ (2’)

(3) Assume we will put the binary code at the head of “char buf[24]” in the function getbuf(), that means the first instruction executed after getbuf() return will at buf[0]. So, we need to make getbuf() return to the start address of our instructions, We need to rewrite the return address of getbuf to __**[3]**__ (4’)

(4) Last write down your input hex-string here according to Table3 (use code 0x00 to fill in the hex-digital you doesn’t mind its value) __**[4]**__ (6’)

Problem 5: (10points)

Suppose function "**int max(int array[], int len)**" accepts an integer array of length **len** as arguments and return the **biggest** value in the array. The following is the Y86 implementation of this function.

Y86 code:

```
max:
Line 01        00: a0 58                  pushl  %ebp
Line 02        02: 20 43                  rrmovl %esp, %ebp
Line 03        04: a0 68                  pushl  %esi
Line 04        06: 50 65 08 00 00 00      ____[1]____
Line 05        0c: a0 38                  pushl  %ebx
Line 06        0e: 50 35 ____[2]____      mrmovl 12(%ebp), %ebx
Line 07        14: 50 16 00 00 00 00      mrmovl (%esi), %ecx
Line 08        1a: 30 82 01 00 00 00      irmovl $1, %edx
Line 09        20: 61 23                  subl   %edx, %ebx
Line 10        22: 71 ____[3]____         jle    .L2
Line 11        27: 50 35 0c 00 00 00      mrmovl 12(%ebp), %ebx
            .L4:
Line 12        2d: 30 80 04 00 00 00      irmovl $4, %eax
Line 13        33: 60 06                  addl   %eax, %esi
Line 14        35: 50 06 00 00 00 00      mrmovl(%esi), %eax
Line 15        3b: 61 10                  subl   %ecx, %eax
Line 16        3d: 75 48 00 00 00         ____[4]____
Line 17        42: 50 16 00 00 00 00      mrmovl (%esi), %ecx
            .L5:
Line 18        48: 30 80 01 00 00 00      irmovl $1, $eax
Line 19        4e: 60 02                  addl   $eax, %edx
Line 20        50: 20 30                  rrmovl %ebx, %eax
Line 21        52: ____[5]____            subl   %edx, %ebx
Line 22        54: 75 2d 00 00 00         jge    .L4
            .L2:
Line 23        5a: b0 38                  popl   %ebx
Line 24        5c: 20 10                  rrmovl %ecx, %eax
Line 25        5e: b0 68                  ____[6]____
Line 26        60: b0 58                  popl   %ebp
Line 27        62: 90                     ret
```

1. Please fill the blanks in the Y86 code. (1'*6 = 6')
2. There is a logic mistake in the Y86 code of the **max** function. Please find which line of the code is buggy? And correct it as simple as possible. (2'+2'=4')

Problem 6: (10points)

Suppose the following C code are executed on a 32-bit **little endian** machine. Read the code and answer the following question:

C code (counter.c)

```
extern void sort(int* a, int len);
extern int* array;

int global_counter = 100;

int sort_counter(int* data, int len)
{
   static int counter = 0;
   global_counter --;
   if(array == data)
      counter ++;
   sort(data, len);
   return counter;
}
```

Object File (counter.o)

```
Section .text:
00000000 <sort_counter>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 08                sub    $0x8,%esp
   6:   8b 55 08                mov    0x8(%ebp),%edx
   9:   83 2d ___[1]____01      subl   $0x1,0x0
  10:   39 15 00 00 00 00       cmp    %edx,0x0
  16:   74 18                   je     30 <sort_counter+0x30>
  18:   8b 45 0c                mov    0xc(%ebp),%eax
  1b:   89 14 24                mov    %edx,(%esp)
  1e:   89 44 24 04             mov    %eax,0x4(%esp)
  22:   e8 ___[2]____           call   23 <sort_counter+0x23>
  27:   a1 00 00 00 00          mov    0x0,%eax
  2c:   c9                      leave
  2d:   c3                      ret
  2e:   66 90                   xchg   %ax,%ax
  30:   83 05 00 00 00 00 01     addl   $0x1,0x0
  37:   eb df                   jmp    18 <sort_counter+0x18>

Section .data:
00000000 <global_counter>:
   0:   ___[3]___
```

```
Section .bss:
00000000 <counter.1288>:
   0:   00 00
```

1. Please fill the blanks in the above relocatable object file. (1'*3 = '3)

2. Please fill the blanks in the table of the relocatable table of **counter.o***.* (1'*3 = 3')

OFFSET	**TYPE**	**VALUE**
0000000b	R_386_32	global_counter
00000012	R_386_32	___**[4]** ___
00000023	___ **[5]** ___	sort
00000028	___ **[6]** ___	.bss
00000032	R_386_32	.bss

3. After linking, **sort_counter** and **sort** functions are located at **0x08048390** and **0x08048354** respectively. Please answer the following questions. (2'*2 = 4')

   i. What's the relocated address of the relocated reference to **sort** in **sort_counter** ?   relocated **address**: ___**[7]**___

   ii. What's the relocated value of the relocated reference to **sort** in **sort_counter?**   relocated **value**: __**[8]**__

Solution

Problem 1: (16points)

1

2

3 [1]

[2]

4

Problem 2: (24 points)

1 [1] [2]

[3] [4]

[5] [6]

[7] [8]

2

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

3

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

4

5

Problem 3: (16 points)

1 [1] [2] [3]

[4] [5] [6]

[7] [8]

Problem 4: (24 points)

1 [1] [2]

[3]

2 [1] [2]

[3]

[4]

Problem 5: (10 points)

1 [1] [2]

[3] [4]

[5] [6]

2

Problem 6: (10 points)

1 [1] [2] [3]

2 [4] [5] [6]

3 [7] [8]