Introduction to Computer Systems
2014 Spring Midterm Examination

Name_____________ Student No._______________ Score___________

Problem 1: (16 points)

[1]	[2]	[3]
[4]	[5]	[6]
[7]	[8]	

Problem 2: (10 points)

[1]	[2]
[3]	[4]
[5]	[6]
[7]	[8]
[9]	[10]

Problem 3: (15 points)

1.	[1]	[2]
	[3]	[4]
	[5]	[6]
	[7]	[8]
	[9]	[10]

2.

3.

Problem 4: (10 points)

1	[1]	[2]
	[3]	[4]

2

3.

Problem 5: (25 points)

1	[1]	[2]
	[3]	[4]
	[5]	[6]
	[7]	[8]
	[9]	[10]
	[11]	[12]
	[13]	[14]

2

Problem 6: (24 points)

1	[1]	[2]
	[3]	[4]
	[5]	[6]
2	[1]	[2]
	[3]	[4]
	[5]	[6]

3	[1]	[2]
	[3]	[4]
	[5]	[6]
	[7]	[8]

4

Problem 1: (16 points)

1. Consider the following C program

```
short s = -7;   unsigned short us = s;
int i = -26;
unsigned int ui = i;
unsigned int k = ~0xf7;
unsigned int a = !ui ^ k;
unsigned int result = ((unsigned int )-1 >=  a);
```

Assume we are running code on an **8-bit** machine using two's complement arithmetic for signed integers. Also assume that right shifts of signed values are performed **arithmetically**. A "short" integer is encoded using **4 bits**. Please fill in the blanks of table below. (2'*8=16')

Expression	Binary Representation
i	1110 0110
us	[1]
k	[2]
a	[3]
(us>>2) + (s>>2)	[4]
-7 || us	[5]
TMIN	[6]
result	[7]
i = (unsigned int)-1 + ui	OF = [8]

Problem 2: (10points)

Suppose a **32-bit little endian** machine has the following memory and register status. (NOTE: **Instructions are independent**). (1'*10=10')

Memory status

Address	Value
0x100	0xabcdef12
0x104	0x12345678
0x108	0x10011001
0x10c	0x00000007

Register status

Register	Value
%eax	0x00000102
%ebx	0x00000002
%ecx	0x00000104
%edx	0xa1b2c3d4

Fill in the blanks using **4 byte size** and **hex**. (1' * 10 = 10')

Operation	Destination	Value
movzwl %dx,%eax	[1]	[2]
imull %ebx,%eax	[3]	[4]
leal 0xfffffffc(%ecx),%edx	[5]	[6]
mull %ebx	[7]	[8]
dec 0xa(%eax)	[9]	[10]

Problem 3: (15points)

Please answer the following questions according to the definition of the union. (NOTE that the size of different types in x86 and x86-64 is shown in the Figure 3.34 in ICS book.)

```
union u_t{
    char z;
    struct s_t {
        int a;
        char b;
        short *(*d)(char, long, double, float);
        short c;
        char *e[2];
    } str[2];
    char f[2];
} u;
```

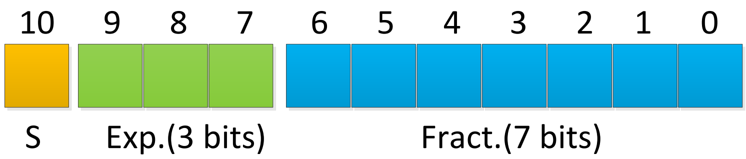
1. Fill in the following blocks. (10')

Representation	x86	x86-64
sizeof(u)	[1]	[2]
sizeof(u.str[1])	[3]	[4]
&(u)	0x8049780	0x6009e0
&(u.f[1])	[5]	[6]
&(u.str[0].d)	[7]	[8]
&(u.str[1].e[1])	[9]	[10]

2. How many bytes are WASTED in **struct s_t** in x86 and x86-64? (2')
3. If you can re-arrange the declarations of **struct s_t**, how many bytes of memory can you SAVE compared to the original one in x86and x86-64? (3')

Problem 4: FP (10points)

The following figure shows the floating-point format we designed for the exam, called float11. Except for the length, it's the same as the IEEE 754 single-precision format you have learned in the class.



1. Fill the blanks with proper values. (4')

   1) **Normalized**: $(-1)^{\mathbf{S}} \times (1.\mathrm{Fract}) \times 2^{\mathbf{Exp.-bias}}$, where **bias**= [1] ;
   2) **NaN** (any correct **binary** form): __[2]___ ;
   3) **Smallest Normalized Value** (s = 0 and in **binary** form):___[3]__;
   4) **Largest Denormalized Value** (s = 0 and in **binary** form):__[4]___;

2. Convert the number $(-4.9375)_{10}$ into the float11 representation (in **binary**). (3')

3. What is the equivalent value to $(1\ 000\ 1010101)_2$ as a **decimal** number? (3')

Problem 5: (25points)

```
int cal(int *val, char type, int x)
 {
   int result = 0;
   for(int i=0; i < R; i++) {
      switch(type) {
         case ___[1]___:
            val[i][x]++;
            break;
         case 'b':
            ___[2]___;
         case ___[3]___: case 'f':
            result += x;
            break;
         default: ___[4]___;
       }
   }
   return result;
}
```

```
#define R  ___[5]___
#define S  ___[6]___

/* sizeof(val) == 200 */
int val[R][S];
int x;
```

```
.section .rodata
.align 4
.L7:
  .long  .L4
  .long  .L5
  .long  .L6
  .long  ___[7]___
  .long  .L3
  .long  ___[8]___
```

```
<cal>:
      pushl   %ebp
      movl    %esp, %ebp
      pushl   ___[9]___
      subl    $20, %esp
      movl    12(%ebp), %ebx
      movb    %bl, -12(%ebp)
      movl    $0, %eax
      movl    $0, %ecx
      jmp     .L2
.L9:  movsbl  -12(%ebp), %ebx
      subl    $97, %ebx
      cmpl    ___[10]___, %ebx
      ja      .L3
      jmp     ___[11]___
.L4:  movl    16(%ebp), %edx
      movl    %ecx, %ebx
      sall    $2, %ebx
      addl    %ecx, %ebx
      addl    ___[12]___, %ebx
```

```
      movl   8(%ebp,%ebx,4),%esi
      addl   $1, %esi
      movl   %esi, 8(%ebp,%ebx,4)
      jmp    .L8
.L5:  movl   16(%ebp), %ebx
      cmpl   %ecx, %ebx
      cmova  %ecx, %ebx
      addl   %ebx, %eax
.L6:  movl   16(%ebp), %ebx
      addl   %ebx, %eax
      jmp    .L8
.L3:  movl   $-1, %eax
      jmp    .L1
.L8:  addl   $1,%ecx
.L2:  cmpl   ___[13]___,___[14]___
      jle    .L9
.L1:  addl   $20, %esp
      popl   %ebx
      popl   %ebp
      ret
```

Suppose the C and assembly code are executed on a 32-bit little endian machine. Read the code and answer the following questions.

Reference:

Char	'0'	'9'	'A'	'Z'	'a'	'z'
ASCII	48(0x30)	57(0x39)	65(0x41)	90(0x5A)	97(0x61)	122(0x7A)

1. Please fill in the blanks within C and assembly code. (NOTE: no more than one instruction per blank) (1.5' * 14)
2. Here we have two instructions underlined:

   `movb   %bl, -12(%ebp)`
   `movsbl -12(%ebp), %ebx`

   What is the purpose of executing those two instructions? (2').

   If we replace "**`movsbl`**" with "**`movzbl`**", will the function "**`cal`**" perform differently? Please explain your answer (2')

Problem 6: (24points)

Suppose the following C code and assembly code are executed on a 32-bit little endian machine. In the C code, the bar functions are **NOT explicitly** called. In order to do the invocation **implicitly** we modify the assembly code directly, as shown in the lines **labeled** by ① to ⑤ which are added to the original assembly code. In this case, only **`bar1()`** is called. Read the code and answer the following questions:

```
#include <stdio.h>

int array[4];
void foo();
void bar1();
void bar2();

int main(){
  foo();
}
```

```
void foo(void){
   array[0] = (int)bar1;
   array[2] = (int)bar2;
}
void bar1(void){
   printf("bar1\n");
}
void bar2(void){
   printf("bar2\n");
}
```

```
080483c4 <main>:
 80483c4:       55                      push   %ebp
 80483c5:       89 e5                   mov    %esp,%ebp
 80483c7:       83 e4 f0                and    $0xfffffff0,%esp
 80483ca:       e8 04 00 00 00          call   80483d3 <foo>
 80483cf:       89 ec                   mov    %ebp,%esp
 80483d1:       5d                      pop    %ebp
 80483d2:       c3                      ret

080483d3 <foo>:
 80483d3:       55                      push   %ebp
 80483d4:       89 e5                   mov    %esp,%ebp
 80483d6:       b8 ___[1]___            mov    $0x80483fe,%eax
 80483db:       a3 10 96 04 08          mov    %eax,0x8049610
 __[2]__:       b8 15 84 04 08          mov    $0x8048415,%eax
 80483e5:       a3 18 96 04 08          mov    %eax, ___[3]___
 80483ea:       5d                      pop    ___[4]___
 80483eb:       89 25 14 96 04 08       mov    %esp,0x8049614    ①
 80483f1:       89 25 1c 96 04 08       mov    %esp,0x804961c    ②
 80483f7:       8d 25 10 96 04 08       lea    0x8049610,%esp    ③
 80483fd:       c3                      ret

080483fe <bar1>:
 80483fe:       8b 24 24                mov    (%esp),%esp       ④
 8048401:       55                      push   %ebp
 8048402:       89 e5                   mov    %esp,%ebp
 8048404:       83 ec 18                sub    $0x18,%esp
 8048407:       c7 04 24 f0 84 04 08    movl   $___[5]___,(%esp)
 804840e:       e8 e5 fe ff ff          call   80482f8 <puts@plt>
 8048413:       c9                      ___[6]___
 8048414:       c3                      ret

08048415 <bar2>:
 8048415:       8b 24 24                mov    (%esp),%esp       ⑤
 8048418:       55                      push   %ebp
 8048419:       89 e5                   mov    %esp,%ebp
 804841b:       83 ec 18                sub    $0x18,%esp
 804841e:       c7 04 24 f5 84 04 08    movl   $0x80484f5,(%esp)
 8048425:       e8 ce fe ff ff          call   80482f8 <puts@plt>
 804842a:       c9                      leave
 804842b:       c3                      ret

08049610 <array>:
 ...
```

1. Fill in the blanks in the Assembly Code. (1'*6=6').

2. BEFORE executing the instruction "**push %ebp**" **(0x80483d3)**, the values of registers are **%esp = 0xffffdb9c**, **%ebp = 0xffffdba8**. Please fill the following blanks. (1'*6=6')

   NOTE: "Before 0x80483eb" means "before executing the instruction in the address 0x80483eb".

Phase	%esp	%ebp
Before 0x80483eb	[1]	[2]
Before 0x80483fe	[3]	[4]
After 0x8048404	[5]	[6]

3. Indicate the values of elements stored in the array, as well as the meaning of the value respectively. If the value can't be determined, fill with '--' instead. (1'*8=8')

Element	Value(Hex)	Meaning
array[0]	[1]	[2]
array[1]	[3]	[4]
array[2]	[5]	[6]
array[3]	[7]	[8]

4. Modify the assembly code to call function **bar2()** instead of **bar1() and explain it**. (4') HINT: Actually you need to modify only one instruction.