Introduction to Computer Systems
2012 Spring Midterm Examination

Name____________ Student No.______________ Score__________

Problem 1: (12 points)

1. Consider the following C program

```
short s = -8;
unsigned short us = s;
int i = -64;
unsigned int ui = i;
unsigned int j = 0xfe;
unsigned int k = ui ^ j;
```

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**. Fill in the empty boxes in the table below. (1'*12)

Expression	Binary Representation
-12	[1]
us	[2]
ui	[3]
(i << 1) - 1	[4]
(s >> 2) << 1	[5]
us >> 2	[6]
k	[7]
ui & k	[8]
(i >> 1)-(ui >> 1)	[9]
UMAX	[10]
TMAX	[11]
TMIN	[12]

Problem 2: (10points)

Suppose a **32-bit little endian** machine has the following memory and register status. Fill in the blanks using **4 byte size** and **hex**.
(NOTE: **Instructions are independent)**. (1' * 10 = 10')

Memory status:

Address	Value
0x100	0xf0f0f0f0
0x104	0x78563412
0x108	0x00001000

Register status:

Register	Value
%eax	0x00000104
%ebx	0x00000001
%ecx	0xfffffffc
%edx	0x87654321

Fill in the blanks

Operation	Destination	Value
subl (%eax), %edx	[1]	[2]
imull $2, (%eax, %ebx, 4)	[3]	[4]
notl (%eax, %ecx)	[5]	[6]
leal 8(%eax, %ebx, 4), %edx	[7]	[8]
movb $0x1, %al	[9]	[10]

Problem 3: (20points)

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

```
#include <stdio.h>
typedef unsigned char *byte_pointer;
void show_bytes(byte_pointer start, int len)
{
    int i;
    for (i = 0; i < len; i++)
        printf(“0x%.2x ”, start[i]);
    printf(“\n”);
}
int main(void)
{
    int x = 0x1234567;
    unsigned char one[8] = {0x11, 0x11, 0x11, 0x11,
                            0x11, 0x11, 0x11, 0x11};
    unsigned char two[8] = {0x22, 0x22, 0x22, 0x22,
                            0x22, 0x22, 0x22, 0x22};
    void *vp;
    int *ip;
    short *sp;

    printf("x:");
    show_bytes((byte_pointer)&x, 2);

    ip = (int *)(one + 5);
    ip[-1] = 0x11;
    printf("new one:");
    show_bytes((byte_pointer)one, 4);

    vp = (void *)(two + 3);
    vp ++;
    sp = (short *)vp;
    *sp = 0xff;
    printf("new two:");
    show_bytes((byte_pointer)(&two[4]), 4)

    return 0;
}
```

1. Suppose the machine uses LITTLE-ENDIAN, please write the output of above program (1'*10)

   x: 0x__[1]__ 0x__[2]__
   new one: 0x__[3]__ 0x__[4]__ 0x__[5]__ 0x__[6]__
   new two: 0x__[7]__ 0x__[8]__ 0x__[9]__ 0x__[10]__

2. Suppose the machine uses BIG-ENDIAN, please write the output of above program (1'*10)

   x: 0x__[1]__ 0x__[2]__
   new one: 0x__[3]__ 0x__[4]__ 0x__[5]__ 0x__[6]__
   new two: 0x__[7]__ 0x__[8]__ 0x__[9]__ 0x__[10]__

Problem 4: (14points)

Suppose the following code is executed on a 32-bit machine, where "**int**" is 4 bytes, "**short**" is 2 bytes, "**char**" is 1 byte and "**pointer**" is 4 bytes.

```
struct data {
    unsigned char *p;
    int i;
    short s[3];
    union {
        char j;
        int k;
    } u;
    char c;
};
struct data d[2];
```

Suppose the address of global variable **d** is **0x8049600**, please answer the following questions. (2'*7)

variable	start address
d[0]	0x8049600
d[1]	[1]
d[0].p	[2]
d[0].i	[3]
d[0].s[1]	[4]
d[0].u.j	[5]
d[0].u.k	[6]
d[0].c	[7]

Problem 5: (21points)

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

C code:

```
int switch_example(int op, int a, int b)
{
    int result;
    switch (op) {
        case 80:
            result = a * 5;
            break;
        case __[1]__:
            result = b + 10;
            break;
        case 83:
            result = b >> 2;
            break;
        case __[2]__: case __[3]__:
            if (__[4]__)
                result = __[5]__;
            else
                result = __[6]__;
            break;
        default:
            result = 0;
            break;
    }
    return result;
}
```

Assembly code:

```
_switch_example:
      pushl %ebp
      movl %esp, %ebp
      subl $16, %esp
      movl 8(%ebp), %eax
      subl __[10]__, %eax
      cmpl __[11]__, %eax
      ja   __[12]__
      jmp  __[13]__
L3:   movl 12(%ebp), %eax
      imull $5, %eax
      movl %eax, -4(%ebp)
```

```
.section .rodata
    .align 4
L7:
    .long  L3
    .long  L2
    .long  L4
    .long  __[7]__
    .long  __[8]__
    .long  L2
    .long  L2
    .long  __[9]__
```

<pre>   jmp L11
L4: movl 16(%ebp), %eax
     addl $10, %eax
     movl %eax, -4(%ebp)
     jmp L11
L5: movl 16(%ebp), %eax
     sarl $2, %eax
     movl %eax, -4(%ebp)
     jmp L11
L6: movl 12(%ebp), %eax
     cmpl 16(%ebp), %eax
     jge __[14]__
     movl 12(%ebp), %eax
     subl $3, %eax
     movl %eax, -4(%ebp)
     jmp L11
L9: movl 16(%ebp), %eax
     imull $4, %eax
     movl %eax, -4(%ebp)
     jmp L11
L2: movl $0, -4(%ebp)
L11: movl -4(%ebp), %eax
     leave
     ret</pre>	

1. Please fill in the blanks within C and assembly code. (1'*14)
2. Please explain the **advantage** and **limitation** of "Jump Table" (2'+2'), and provide a simple code which is not suitable to be translated into a "Jump Table" (3').

Problem 6: (23points)

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

```c
#include <stdio.h>

int main(void)
{
    int a = 3, b = 4;
    foo(&a,&b);
    return 0;
}
```

```c
void foo(int *p, int *q)
{
    char str[4];
    int s = *p, t = *q, temp;
    if (s < t) {
        temp = s;
        s = t;
        t = temp;
    }
    gets(str);
}
```

Assembly code

```
080483c4 <foo>:
 80483c4: 55                      push   %ebp
 80483c5: 89 e5                   mov    %esp,%ebp
 80483c7: 83 ec 28                sub    $0x28,%esp
 80483ca: 8b 45 08                mov    0x8(%ebp),%eax
 80483cd: 8b 00                   mov    (%eax),%eax
 80483cf: 89 45 ec                mov    %eax,-0x14(%ebp)
 80483d2: 8b 45 0c                mov    0xc(%ebp),%eax
 80483d5: 8b 00                   mov    (%eax),%eax
 80483d7: 89 45 f0                mov    %eax,-0x10(%ebp)
 80483da: 8b 45 ec                mov    -0x14(%ebp),%eax
 80483dd: 3b 45 f0                cmp    -0x10(%ebp),%eax
 80483e0: 7d 12                   jge    80483f4 <foo+0x30>
 80483e2: 8b 45 ec                mov    -0x14(%ebp),%eax
 80483e5: 89 45 f4                mov    %eax,-0xc(%ebp)
 80483e8: 8b 45 f0                mov    -0x10(%ebp),%eax
 80483eb: 89 45 ec                mov    %eax,-0x14(%ebp)
 80483ee: 8b 45 f4                mov    -0xc(%ebp),%eax
 80483f1: 89 45 f0                mov    %eax,-0x10(%ebp)
 80483f4: 8d 45 e8                lea    -0x18(%ebp),%eax
 80483f7: 89 04 24                mov    %eax,(%esp)
 80483fa: e8 e9 fe ff ff          call   80482e8 <gets@plt>
 80483ff: c9                      leave
 8048400: c3                      ret

08048401 <main>:
 8048401: 55                          push   %ebp
 8048402: 89 e5                       mov    %esp,%ebp
 8048404: 83 e4 f0                    and    $0xfffffff0,%esp
 8048407: 83 ec 20                    sub    $0x20,%esp
 804840a: c7 44 24 1c 03 00 00 00     movl   $0x3,0x1c(%esp)
```

```
8048412: c7 44 24 18 04 00 00 00   movl   $0x4,0x18(%esp)
804841a: 8d 44 24 18               lea    0x18(%esp),%eax
804841e: 89 44 24 04               mov    %eax,0x4(%esp)
8048422: 8d 44 24 1c               lea    0x1c(%esp),%eax
8048426: 89 04 24                  mov    %eax,(%esp)
8048429: e8 96 ff ff ff            call   80483c4 <foo>
804842e: b8 00 00 00 00            mov    $0x0,%eax
8048433: c9                        leave
8048434: c3                        ret
```

Suppose **_BEFORE_** the instruction at address **0x8048401**(“**push %ebp**”) executed, the register values are:

%esp = 0xbffff42c %ebp = 0xbffff4a8

1. What are values of the **%esp** and **%ebp**, **_BEFORE_** the instructions at address **0x8048426** and **0x80483d2** executed. (2’*4)

register	before 0x8048426	before 0x80483d2
%esp	[1]	[2]
%ebp	[3]	[4]

2. Please fill the following stack with the address, suppose the current status is just **_BEFORE_** executing the instruction at **0x80483fa** (“**call 80482e8 <gets@plt>**”). If the value can’t be determined, then fill ‘**--**’ instead. (1’*12)

address	value
0xbffff42c	[1]
0xbffff428	[2]
.....	.....
0xbffff420	[3]
0xbffff41c	[4]
0xbffff418	[5]
.....	.....
0xbffff404	[6]
0xbffff400	[7]
0xbffff3fc	[8]
0xbffff3f8	[9]
...	.....
[10]	0x4
[11]	0x3
...	.....
0xbffff3d0	[12]

3. Express how to change the return address of function “**foo()**” by using stack overflow attack. (3’)

Solution (Name: Student No.)

Problem 1: (12 points)

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

Problem 2: (10 points)

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

Problem 3: (20 points)

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

Problem 4: (14 points)

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

Problem 5: (21 points)

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

2 <u>Advantage</u>:

<u>Limitation</u>:

<u>Example</u>:

Problem 6: (23 points)

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

3