Introduction to Computer Systems
2015 Fall Middle Examination

Name_____________ Student No._______________ Score___________

Problem 1: (14 points)

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

Problem 2: (10 points)

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

Problem 3: (16 points)

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

2.

3.

Problem 4: (10 points)

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

2

3.

Problem 5: (23 points)

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

2

Problem 6: (27 points)

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

2

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

4

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

Problem 1: (14 points)

1. Consider the following C program

```
unsigned int ua = 0xe5;
short a = ua;
int b = a;
short c = (~a & 7) + 1;
int d = 0x69 && 0x55;
int e = ua >> 3;
```

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
ua	1110 0101
a	[1]
b	[2]
c	[3]
e	[4]
d & (!0x41)	[5]
(e & 0x69) | (a >> 2)	[6]
(e >> c) - 1 + (d << 1)	[7]

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	Low			High
0x610	0xff	0x33	0x22	0x10
0x614	0xab	0x00	0xff	0x11
0x618	0x13	0x01	0xab	0xcd
0x61c	0x01	0x54	0x76	0x98

Register status

Register	Value
%eax	0x00000610
%ecx	0x00000001
%edx	0x00000003

Fill in the blanks using **4 byte size** and **hex**.

Operation	Destination	Value
addl %ecx, (%eax)	[1]	[2]
subl %edx, 4(%eax)	[3]	[4]
imul $16, (%eax,%edx,4)	[5]	[6]
incl 8(%eax)	[7]	[8]
leal (%eax, %ecx,c), %eax	[9]	[10]

Problem 3: (16points)

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.)

```
struct {
    short s1;
    short *ps[2];
    char ca[2];
    union {
        char c1;
        int *pi[2];
    } u;
    short s2;
    int (*p[2])();
    char c2;
} str[2];
```

1. Fill in the following blocks. (12')

Representation	**x86**	**x86-64**
`sizeof(str[0])`	[1]	[2]
`sizeof(str[1].u)`	[3]	[4]
`str`	0x8049780	0x600a00
`&(str[0].u)`	[5]	[6]
`&(str[0].ca[1])`	[7]	[8]
`&(str[0].u.pi[1])`	[9]	[10]
`&(str[1].c2)`	[11]	[12]

2. How many bytes are **wasted** in **`str[0]`** in x86 and x86-64? (1'*2=2')

3. If you can rearrange the declarations in the **`struct`** and **`union`**, how many bytes of memory can you **save** in **`str[0]`** compared to the original declaration in x86 and x86-64? (1'*2=2')

Problem 4: FP (10points)

The following figure shows the floating-point format we designed for the exam, called float12. Except for the length, it's the same as the IEEE 754 single-precision format you have learned in the class.

S	Exp(5bits)	Fract.(6bits)

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

   1) **Normalized**: $(-1)^{\mathbf{S}} \times (1.\text{Fract}) \times 2^{\mathbf{Exp.-bias}}$, where **bias**= [1] ;
   2) **-Infinity(-∞)** (in **binary** form): ___[2]___ ;
   3) **Smallest Negative Denormalized Value** (in **binary** form):___[3]___;
   4) **Smallest Positive Normalized Value** (in **binary** form):___[4]___;

2. Convert the number $(-0.375)_{10}$ into the float12 representation (in **binary**). (3')

3. Assume we use IEEE **round-to-even** mode to do the approximation. Please calculate the addition: $(0\ 10000\ 010110)_2 + (0\ 10010\ 100100)_2$ and write the answer in **binary**. (The answer is represented in float12 as well) (3')

Problem 5: (23points)

```
char TX(char *str, int len) {
  int i = 0;  char ret = 'A';
  while (i < len) {
    switch (str[i]) {
      case 'a':
        ret += 1;  break;
      case 'b':
        ret -= 32;  break;
      case 'c':
        ret += 32;  break;
```

```
/* ASCII(0~9):0x30~0x39
 * ASCII(A~Z):0x41~0x5a
 * ASCII(a~z):0x61~0x7a
 */
int main(void) {
  char str[4] =
        {'a','c','f','b'};
  return TX((char *)str,4);
}
```

```
      case 'd':
        ret -= ___[1]___;  break;
      case __[2]__:
        ret |= 3;  __[3]__;
      default:
        ret = __[4]__;
    }
    i = i + 1;
  }
  return ret;
}
```

```
.section
.rodata
.align 4
.L7:
  .long  .L1
  .long  .L2
  .long  .L3
  .long  .L4
  .long  __[5]__
  .long  .L6
```

```
<TX>:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx
    subl    $0x10,%esp
    _[6]_   $0x41,%dl
    movl    $0x0, -0x8(%ebp)
    jmp     .L10
.L0:
    addl    _[7]_, -0x8(%ebp)
.L10:
    movl    -0x8(%ebp), %eax
    cmp     0xc(%ebp), %eax
    jl      _[8]_
    movzbl  %dl, %eax
    addl    $0x10, %esp
    popl    _[9]_
    leave
    ret
.L9:
    movl    -0x8(%ebp), %ecx
    addl    0x8(%ebp), %ecx
    movzbl  (%ecx), %ecx
```

```
    subl    _[10]_, %ecx
    cmp     $0x5, %ecx
    ja      .L8
    jmp     _[11]_
.L1:
    addl    $0x1, %edx
    jmp     .L0
.L2:
    _[12]_  $0x20, %edx
    jmp     .L0
.L3:
    _[13]_  $0x20, %edx
    jmp     .L0
.L4:
    subl    $0x4, %edx
    jmp     .L0
.L6:
    _[14]_  $0x3, %edx
.L8:
    movl    $0x0, %ebx
    movb    $0x68, %bl
    movzbl  %bl, %edx
    jmp     .L0
```

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

1. Please fill in the blanks within C and assembly code. (1.5' * 14)
   NOTE: no more than one instruction/statement per blank. If you think nothing is required to write, please write NONE.

2. What is the return value of the **`main`** function? If it is a digit (0-9) or letter (A-Z, a-z), please use the digit or letter instead of its ASCII code. (2')

Problem 6: (27points)

One of the TA of ICS wrote a wrong Bubble Sort program and try to use gdb to find out the reason. The following C code and assembly code are executed on a **32-bit little endian** machine. In the C code, the definitions of array is omitted.

```
#include <stdio.h>

int main(void) {
  int array[SIZE];

  ... // initialize array

  sort(array, SIZE);
  return 0;
}
```

```
void sort(int array[], int size) {
  int i;
  for (i=0; i<size-1; ++i)
    if (array[i] < array[i+1])
      swap(&array[i], &array[i+1]);
}
void swap (int *left, int *right) {
  *left ^= *right;
  *right ^= *left;
  *left ^= *right;
}
```

```
080483ed <swap>:
  80483ed: 55                    push %ebp
  80483ee: 89 e5                 mov %esp,%ebp
  80483f0: 8b 45 08              mov 0x8(%ebp),%eax
  80483f3: 8b 55 0c              mov 0xc(%ebp),___[1]___
  80483f6: 8b 08                 mov (%eax),%ecx
  80483f8: 33 0a                 xor (%edx),%ecx
  80483fa: 89 08                 mov %ecx,(%eax)
  80483fc: 33 0a                 xor (%edx),%ecx
  80483fe: 89 0a                 mov %ecx,(%edx)
  8048400: 31 08                 xor %ecx,(%eax)
  8048402: 5d                    pop %ebp
  8048403: c3                    ret

08048404 <sort>:
  8048404: 55                    push %ebp
  8048405: 89 e5                 mov %esp,%ebp
  8048407: 57                    push %edi
  8048408: 56                    push %esi
  8048409: 53                    push %ebx
  804840a: 83 ec 08              sub $0x8,%esp
  804840d: 8b 7d 0c              mov ___[2]___,%edi
  8048410: 8d 47 ff              lea -0x1(%edi),%eax
  8048413: 85 c0                 test %eax,%eax
  8048415: 7e 2c                 jle 8048443 <sort+0x3f>
  8048417: 8b 45 08              mov 0x8(%ebp),%eax
```

```
  804841a: 83 ef 01                sub $0x1,%edi
  804841d: bb 00 00 00 00          mov $0x0,%ebx
  8048422: 8d 70 04                lea 0x4(%eax),%esi
  8048425: 8b 50 04                mov 0x4(%eax),%edx
  8048428: 39 10                   cmp %edx,(%eax)
  804842a: 7d 0c                   jge 8048438 <sort+0x34>
  804842c: 89 74 24 04             mov %esi,0x4(%esp)
  8048430: 89 04 24                mov ___[3]___,(%esp)
  8048433: e8 b5 ff ff ff          call 80483ed <swap>
  8048438: 83 c3 01                add $0x1,%ebx
  804843b: 39 fb                   cmp %edi,%ebx
  804843d: 74 04                   je 8048443 <sort+0x3f>
  804843f: 89 f0                   mov %esi,%eax
  8048441: eb df                   jmp 8048422 <sort+0x1e>
  8048443: 83 c4 08                add $0x8,%esp
  8048446: 5b                      pop %ebx
  8048447: 5e                      pop %esi
  8048448: 5f                      pop %edi
  8048449: 5d                      pop %ebp
  804844a: c3                      ret

0804844b <main>:
  804844b: 55                      push %ebp
  804844c: 89 e5                   mov %esp,%ebp
  804844e: 83 ec 28                sub $0x28,%esp
  8048451: c7 45 e0 09 00 00 00    movl $0x9,-0x20(%ebp)
  8048458: c7 45 e4 0a 00 00 00    movl $0xa,-0x1c(%ebp)
  804845f: c7 45 e8 0b 00 00 00    movl $0xb,-0x18(%ebp)
  8048466: c7 45 ec 0c 00 00 00    movl $0xc,-0x14(%ebp)
  804846d: c7 45 f0 0d 00 00 00    movl $0xd,-0x10(%ebp)
  8048474: c7 45 f4 0e 00 00 00    movl $0xe,-0xc(%ebp)
  804847b: c7 45 f8 0f 00 00 00    movl $0xf,-0x8(%ebp)
  8048482: c7 45 fc ___[4]____     movl $0x10,-0x4(%ebp)
  8048489: c7 44 24 04 08 00 00 00 movl $0x8,0x4(%esp)
  8048491: 8d 45 e0                lea -0x20(%ebp),%eax
  8048494: 89 04 24                mov %eax,(%esp)
  8048497: e8 68 ff ff ff          call ___[5]___ <sort>
  804849c: b8 00 00 00 00          mov $0x0,%eax
  __[6]__: c9                      leave
  80484a2: c3                      ret
```

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

2. What is the value of the size of array? (1’)

3. **BEFORE** executing the instruction "**`push %ebp`**" **`(0x804844b)`**, the value of register **`%esp`** is **`0xffffd4fc`**. Please fill the following blanks.(1'*6=6')

   AFTER executing the instruction "**`call __[5]__ <sort>`**" **`(0x8048497)`**

register	value
%esp	[1]
%ebp	[2]

   AFTER executing the instruction "**`push %edi`**" **`(0x8048407)`**

register	value
%esp	[3]
%ebp	[4]

   AFTER executing the instruction "**`ret`**" **`(0x804844a)`**

register	value
%esp	[5]
%ebp	[6]

4. We use **`gdb`** to run this program and step into the first loop of function **`sort`**. The **FIRST** time we stop before the execution of instruction at

   ```
   8048428: 39 10      cmp %edx,(%eax)
   804842a: 7d 0c      jge 8048438 <sort+0x34>
   ```

   What are the values (in hexadecimal) of **`%edx`** and **`(%eax)`**? Which instruction will be executed after "**`jge 8048438`**"? (6')

5. After that, we continue the execution of **`gbd`**. After several loops in function **`sort`**, we stop again before the execution of instruction at

   ```
   804842a: 7d 0c      jge 8048438 <sort+0x34>
   ```

   We use **`gdb`** to print some registers. Suppose the start address of **`array`** is **`0xffffd4d8`**. Please fill the table below. (2'*4=8')

register	value
%eax	0xffffd4e4
%ebx	__[1]__
%ecx	0x9
%edx	__[2]__
%esi	0xffffd4e8
%edi	0x7
%esp	0xffffd4b4
%ebp	__[3]__
%eip	__[4]__