Introduction to Computer Systems
2016 Fall Middle Examination

Name______________ Student No.________________ Score___________

Problem 1: (14 points)

[1] [2] [3]

[4] [5] [6]

[7]

Problem 2: (12 points)

[1] [2]

[3] [4]

[5] [6]

[7] [8]

[9] [10]

[11] [12]

Problem 3: (18 points)

1. [1] [2]

[3] [4]

[5] [6]

[7] [8]

[9] [10]

[11] [12]

[13] [14]

2.

3.

Problem 4: (9 points)

1. [1] [2] [3]

2.

3.

Problem 5: (25 points)

1 [1] [2]

[3] [4]

[5] [6]

[7] [8]

[9] [10]

[11] [12]

[13] [14]

2

3

Problem 6: (22 points)

1 [1] [2]

[3] [4]

[5] [6]

2 [1] [2]

[3] [4]

[5] [6]

[7] [8]

3 [1] [2]

[3] [4]

[5] [6]

4

Problem 1: (14 points)

1. Consider the following C program

```
unsigned short u = 0x7;
short a = -6;
unsigned short b = a;
unsigned int c = a;
unsigned int d = (~(unsigned int)u | 0x11) + 2;
int e = 0x35;
```

Assume the program will run on an **8-bit** machine and use two's complement arithmetic for signed integers. A 'short' integer is encoded in **4 bits**, while a normal 'int' is encoded in **8 bits**. Please fill in the blanks below. (2'*7=14')

Expression	Binary Representation
u	0111
13	[1]
c	[2]
d	[3]
e >> 2	[4]
(e & 0xff) ^ 0x3	[5]
(e - 5) + (0x11 << 1)	[6]
!(e && 0x10)	[7]

Problem 2: (12points)

Suppose a **32-bit little endian** machine has the following memory and register status. (NOTE: **Instructions are independent**). (1'*12=12')

Memory status

Address	Low			High
0x4000	0x35	0x00	0x00	0x00
0x4004	0x91	0x06	0x21	0x91
0x4008	0x80	0xff	0x04	0x08
0x400c	0xde	0xad	0xbe	0xef
0x4010	0xac	0xde	0x21	0x08

Register status

Register	Hex Value
%eax	0x00004004
%ecx	0x00000001
%edx	0x0000042d
%ebx	0x00000002
%esp	0x00004010

Please fill in the blanks below. For **'Value'**, write in **4-byte hex value**. If the instruction does not change any register or memory, fill the corresponding two blanks with '--'. If the instruction changes **multiple** destinations, write all of them in blanks and make sure the destinations and updated values are listed in the **same** order.

Operation	Destination	Hex Value
andl %ecx, %edx	[1]	[2]
sarl $4, %edx	[3]	[4]
movl $10, (%eax, %ebx, 4)	[5]	[6]
cmpl $0x2, (%eax)	[7]	[8]
leal (%eax, %ecx, 4), %eax	[9]	[10]
push %ecx	[11]	[12]

Problem 3: (18points)

Please answer the following questions according to the definition of the union. (NOTE that the size of different types in x86 (32-bit) and x86-64 is shown in the Figure 3.34 in ICS book.)

```
union ele {
    struct s1 {
        char cc;
        union ele *next;
        short ss;
        long long int li;
    } e1;

    int i;

    struct s2 {
        char c;
        struct s1 (*f) (int i, short ss, long long int li);
        char str[3];
        short s;
        int *p[2];
        char c2;
        int ii;
    } e2;
} u;
```

This declaration illustrates that structures can be embedded within unions.

1. Fill in the following blocks. (please represent address with **Hex**) (14')

Representation	**x86**	**x86-64**
`sizeof(u.e1)`	[1]	[2]
`sizeof(u.e2)`	[3]	[4]
`sizeof(union ele)`	[5]	[6]
`u`	0x804a040	0x601060
`u.e1.next`	[7]	[8]
`u.e1.li`	[9]	[10]
`u.e2.f`	[11]	[12]
`u.e2.p[1]`	[13]	[14]

2. How many bytes are **WASTED** in **`struct s2`** under x86 and x86-64? (2')

3. If you can rearrange the declarations in the **`struct s2`**, how many bytes of memory can you **SAVE** in **`struct s2`** compared to the original declaration under x86 and x86-64? (2')

Problem 4: FP (9points)

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(6bits)	Fract.(5bits)

1. Fill the blanks with proper values. (3')

1) **Denormalized**: $(-1)^{\mathbf{S}} \times (0.\text{Fract}) \times 2^{\mathbf{E}}$, where **E** = _[1]_;

2) **-Infinity(-∞)** (in **binary** form): __[2]___ ;

3) **Biggest Negative Normalized Value** (in **binary** form):___[3]__;

2. Convert the number $(-6.5)_{10}$ into the **Float12** representation (in **binary**). (3')

3. Assume we use IEEE **round-to-even** mode to do the approximation. Please calculate the addition: $(0\ 000000\ 00110)_2 + (0\ 000011\ 10100)_2$ and write the answer in **binary**. (The answer is represented in **Float12** as well) (3')

Problem 5: (25points)

①

```
char sw(char **str,
          int *sz, int len) {
  char result = 'a';
  int i,j;
  for (i = 0; i < len; i++) {
    for (j = 0; j < sz[i]; j++) {
      switch (str[i][j]) {
        case __[1]__:
          result += __[2]__;
        case 'b':
          result += i;
          break;
        case __[3]__:
          result = 2;
          break;
        case 'e':
          result = __[4]__;
          __[5]__;
        default:
          result = __[6]__;
  }}}
  return result;
}
```

②

```
/* ASCII(0~9):0x30~0x39
 * ASCII(A~Z):0x41~0x5a
 * ASCII(a~z):0x61~0x7a
 */
int main(void) {
  char *str[4] =
    {"a", "e", "mdzz", "aeb"};

  int sz[4] = {1, 1, 4, 3};
  char cc = sw(str, sz, 4);
  printf("cc is : %c\n", cc);
  return 0;
}
```

③

```
.section
.rodata
.align 4
.L6:
    .long  .L5
    .long  __[7]__
    .long  .L8
    .long  .L4
    .long  .L9
```

④

```
<sw>:
    pushl   %ebp
    movl    %esp, %ebp
    pushl   __[8]__
    subl    $16, %esp
    movb    $97, %ebx
    movsbl  %bl, %ebx
    movl    $0, -8(%ebp)
    jmp     .L2
.L12:
    movl    $0, -4(%ebp)
    jmp     .L3
.L11:
    movl    -8(%ebp), %eax
    movl    8(%ebp), %edx
    movl    __[9]__, %eax
    addl    __[10]__, %eax
    movsbl  (%eax), %eax
    subl    $97, %eax
```

⑥

```
.L7:
    addl    __[13]__, %ebx
    jmp     .L10
.L5:
    movl    $2, %ebx
    jmp     .L10
.L9:
    sall    $2, %ebx
.L4:
    cmpl    -4(%ebp), %ebx
    cmovg   $65, %ebx
.L10:
    addl    $1, -4(%ebp)
.L3:
    movl    -8(%ebp), %eax
    movl    12(%ebp), %edx
    movl    (%edx,%eax,4), %eax
    cmpl    -4(%ebp), %eax
    jg      .L11
```

```
    cmpl    __[11]__, %eax
    ja      .L4
    jmp     __[12]__
.L8:
    movl    -8(%ebp), %eax
    movl    8(%ebp), %edx
    movl    (%edx,%eax,4), %eax
    addl    -4(%ebp), %eax
    movsbl  (%eax), %eax
    addl    %eax, %ebx
```

⑤

```
    addl    $1, __[14]__
.L2:
    movl    -8(%ebp), %eax
    cmpl    16(%ebp), %eax
    jl      .L12
    movsbl %bl, %eax
    addl    $16, %esp
    popl    %ebx
    leave
    ret
```

⑦

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 purpose of instruction **popl %ebx**? (2')

3. What is the output of the main function? (2')

Problem 6: (22points)

One of TAs of ICS wrote a binary searching program. The following C code and assembly code are executed on a **32-bit little endian** machine.

```
#include <stdio.h>

int main(void){
  char array[6] = {0,0,1,4,6,7};
  int i;
  for(i = 0; i < 3; i++)
    foo(array + i);
}
```

```
void foo(char* n){
  *(int*)n += 0x10100;
  char c = n[*n];
  printf("foo: %d\n", c);
}
```

```
0804841d <foo>:
 804841d:   55                      push   %ebp
 804841e:   89 e5                   mov    %esp,%ebp
 8048420:   83 ec 28                sub    $0x28,%esp
 8048423:   8b 45 08                mov    0x8(%ebp),%eax
 8048426:   8b 00                   mov    __[1]__,%eax
 8048428:   8d 90 00 01 01 00       lea    0x10100(%eax),%edx
 804842e:   8b 45 08                mov    0x8(%ebp),%eax
 8048431:   89 10                   mov    %edx,(%eax)
 8048433:   8b 45 08                mov    0x8(%ebp),%eax
 8048436:   0f b6 00                movzbl (%eax),%eax
 8048439:   0f be d0                movsbl %al,%edx
 804843c:   8b 45 08                mov    0x8(%ebp),%eax
 804843f:   0f b6 04 10             movzbl __[2]__,%eax
 8048443:   88 45 f7                mov    %al,-0x9(%ebp)
 8048446:   0f be 45 f7             movsbl -0x9(%ebp),%eax
 804844a:   89 44 24 04             mov    %eax, __[3]__
 804844e:   c7 04 24 __[4]__        movl   $0x8048540,(%esp)
 8048455:   e8 96 fe ff ff          call   80482f0 <printf@plt>
 804845a:   c9                      leave
 804845b:   c3                      ret
0804845c <main>:
 804845c:   55                      push   %ebp
 804845d:   89 e5                   mov    %esp,%ebp
 804845f:   83 e4 f0                and    $0xfffffff0,%esp
 8048462:   83 ec 10                sub    $0x10,%esp
 8048465:   c6 44 24 0a 00          movb   $0x0,0xa(%esp)
 804846a:   c6 44 24 0b 00          movb   $0x0,0xb(%esp)
 804846f:   c6 44 24 0c 01          movb   $0x1,0xc(%esp)
 8048474:   c6 44 24 0d 04          movb   $0x4,0xd(%esp)
 8048479:   c6 44 24 0e 06          movb   $0x6,0xe(%esp)
 804847e:   c6 44 24 0f 07          movb   $0x7,0xf(%esp)
 8048483:c7 44 24 04 00 00 00 00    movl   $0x0,0x4(%esp)
 804848b:   eb 17                   jmp    80484a4 <main+0x48>
 804848d:   8b 44 24 04             mov    0x4(%esp),%eax
 8048491:   8d 54 24 0a             lea    __[5]__,%edx
 8048495:   01 d0                   add    %edx,%eax
 8048497:   89 04 24                mov    %eax,(%esp)
 804849a:   e8 7e ff ff ff          call   804841d <foo>
 804849f:   83 44 24 04 01          addl   $0x1,0x4(%esp)
 80484a4:   83 7c 24 04 02          cmpl   $0x2,0x4(%esp)
 80484a9:   7e e2                   jle    804848d <main+0x31>
 80484ab:   c9                      __[6]__
 80484ac:   c3                      ret
```

Suppose **BEFORE** the execution of instruction at `804845c(push %ebp)`, the register values are: `%esp = 0xffffcb1c %ebp = 0xffffcba8`

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

2. According to the %esp, %ebp **BEFORE** the execution of instruction at `804845c` (`push %ebp`). Please fill the following blanks.(1'*8=8')

   AFTER executing the instruction "`push %ebp`" `(804845c)`

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

   AFTER executing the instruction "`call <foo>`" `(804849a)`

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

   BEFORE executing the instruction "`leave`" `(804845a)`

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

   AFTER executing the instruction "`ret`" `(804845b)`

register	Value
%esp	[7]
%ebp	[8]

3. After that, we continue the execution. Now, we stop **After** the execution of instruction at

   `8048426:    mov    __[1]__,%eax`

   We find that the value of %eax is 0x4010000.Then we continue the execution. Please fill the table below. (1'*6=6')

   NOTE: "After `8048428`" means "after executing the instruction in the address `8048428`". "Meaning" wants you to explain what variable or C expression the register or address represent.

Phase	Register or Address	Value	Meaning
After 8048428	%edx	__[1]__	__[2]__
After 8048436	%eax	__[3]__	__[4]__
After 8048443	-0x9(%ebp)	__[5]__	__[6]__

4. Please write the output of the function. (2')