Introduction to Computer Systems
2017 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]

2.

3.

4.

Problem 4: (11 points)

1. [1] [2] [3]

2.

3. [1] [2]

4.

Problem 5: (22 points)

1 [1] [2]

[3] [4]

[5] [6]

[7] [8]

[9] [10]

[11] [12]

2

3

Problem 6: (23 points)

1 [1]

2 [1] [2]

3 [1] [2]

[3] [4]

[5] [6]

[7]

[8] [9]

[10] [11]

[12] [13]

[14] [15]

4 [16]

Problem 1: (14 points)

1. Consider the following C program

```
int a = 0x18;
unsigned short ua = a;
int b = ua >> 1;
short c = (b && 0) + 1;
unsigned int d = (~(unsigned int)a) ^ 0xe5;
int e = 0xf4 & 0xe3;
```

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
a	0001 1000
ua	[1]
b	[2]
c	[3]
d	[4]
e | (!0)	[5]
(e + 0x20) + (0x11 >> 2)	[6]
(ua >> d) + (b << d)	[7]

Problem 2: (12points)

Suppose a **64-bit little endian** machine has the following memory and register status. (NOTE: **Instructions are independent**). (1'*12=12')

Memory status

Address	Low							High
0x4000	0x00	0x00	0x00	0x00	0x35	0x00	0x00	0x00
0x4008	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x71
0x4010	0x00	0x00	0x00	0x00	0x80	0xff	0x04	0x08
0x4018	0x00	0x00	0x00	0x00	0xde	0xad	0xbe	0xef
0x4020	0xde	0xad	0xbe	0xef	0xab	0xcd	0xdc	0xba

Register status

Register	Hex Value
%rax	0x00000000 00004000
%rcx	0xffffffff ffffffff
%rdx	0xaabbccdd ababcdcd
%rbx	0x00000000 00000002
%rsp	0x00000000 00004010

Please fill in the blanks below. For **'Value'**, write in **8-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
subq %rcx, %rax	[1]	[2]
movl $17, (%rax, %rbx, 8)	[3]	[4]
sarq $32, %rdx	[5]	[6]
cmpq $0x71, 8(%eax)	[7]	[8]
leaq 8(%rax, %rcx, 8), %rdx	[9]	[10]
popq %rcx	[11]	[12]

Problem 3: (18points)

Please answer the following questions according to the definition of heterogeneous data structures. (NOTE that the size of data types in x86-64 is shown in the Figure 3.1 in ICS book.))

```
struct node_t {
    char type;
    union data_t {
        struct {
            long lsn;
            char loaded;
            struct node_t **child_cache;
            short keys[7];
            long children[8];
        } intern;
```

```
        struct {
            struct node_t * (*split) (int);
            short keys[7];
            char values[32];
        } leaf;
    } data;
    char status;
} node;

union data_t *data = &(node.data);
```

This declaration illustrates that structures can be embedded within unions.

1. Fill in the following blocks. (please represent address with **Hex**) (10’)

Representation	x86-64
sizeof(node)	[1]
sizeof(node.data)	[2]
sizeof(node.data.leaf)	[3]
sizeof(node.data.intern)	[4]
node	0x601060
&(node.data)	[5]
&(data->intern.loaded)	[6]
&(data->intern.children)	[7]
&(data->leaf.keys)	[8]
&(data->leaf.values)	[9]
&(node.status)	[10]

2. How many bytes are **WASTED** in `struct intern` under x86-64? Explain your solution. (3’)

3. If you can rearrange the declarations in the `struct intern`, how many bytes of memory can you **SAVE** in `struct intern` compared to the original declaration under x86-64? Explain your solution (3’)

4. Alice thought the struct leaf is too small, so he changed “`struct {...} leaf;`” into “`struct {...} leaf[2];`” based on the **UNOPTIMIZED** data structure, what’s the size of node now? (2’)

Problem 4: (11 points)

The following figure shows part of codes compiled on an x86-64 machine. Please answer the following question according to the code.

```
int test[5][7];

int main(void) {
   int a[3][4];
   int b[4][2];
   int sum = 0;
   int col = ____[1]____;
   int row = ____[2]____;
   for (int i = 0; i < 4; i++)
      sum += a[row][i]
           * b[i][col];

   printf("%d\n", sum);
   return 0;
}
```

```
main:
   pushq  %rbp
   movq   %rsp, %rbp
   subq   $0x60, %rsp
   movl   $0x0, -0x4(%rbp)
   movl   $0x0, -0x8(%rbp)
   movl   $0x1, -0x10(%rbp)
   movl   $0x2, -0xc(%rbp)
   jmp    .L2
.L3:
   movl   -0x8(%rbp), %eax
   cltq
   movl   -0x10(%rbp), %edx
   movslq %edx, %rdx
   salq   $0x2, %rdx
   addq   %rdx, %rax
   movl   -0x40(%rbp,%rax,4), %edx
   movl   -0xc(%rbp), %eax
   cltq
   movl   -0x8(%rbp), %ecx
   movslq %ecx, %rcx
   addq   %rcx, %rcx
   addq   %rcx, %rax
   movl   -0x60(%rbp,%rax,4), %eax
   imull  %edx, %eax
   addl   %eax, -0x4(%rbp)
   addl   $0x1, -0x8(%rbp)
.L2:
   cmpl   $0x3, -0x8(%rbp)
   jle    .L3
   movl   -0x4(%rbp), %eax
   movl   %eax, %esi
   movl   $0x400634, %edi
   movl   $0x0, %eax
   call   printf
   movl   $0x0, %eax
   leave
   ret
```

1. Suppose the address of `test` is **0x601060**, what's the value of following expressions? (NOTE: **n** is a variable and please represent address with **Hex**) (3')

1) `&(test[1][3])`: ____[1]____;
2) `&(test[3][n])`: ____[2]____;
3) `&(test[n][2])`: ____[3]____;

2. What's the address of array **a** and array **b**? (NOTE: please represent with expressions based on **%rbp**) (2')

3. What's the value of **row** and **col**? Please complete the initialization code in C program (2')

4. Please show the value of the underlined **%rax** in line 16 and 23 when **i=2** and explain how the value is calculated using **col**, **row** and **i**. (4')

Problem 5: (22points)

①

```
// ASCII(0~9):0x30~0x39
int aaa(char *str, int len){
  int i = 0, result = len;
  for(i; i < len; i++){
    switch(str[i]){
      case '1':
        result = __[1]__;
        break;
      case __[2]__:
        result = str[i] >> 3;
      case __[3]__:
        result++;
        break;
      case '5':
        result = __[4]__;
        break;
      case '7':
        result = 9;
      default:
        result = __[5]__;
  }}
  return result;
}

int main(){
  char *str = "54749110";
  int pos = aaa(str, 8);
  printf("pos:%d\n", pos);
  long l = 0x123456789abcdef;
  unsigned char *bytel=(char *)&l;
  printf("0x%.2x%.2x\n",
        bytel[pos-1],bytel[pos]);
  return 0;
}
```

②

```
    .section
    .rodata
    .align 8
.L5:
    .quad  .L4
    .quad  .L3
    .quad  .L6
    .quad  .L7
    .quad  .L8
    .quad  .L3
    .quad  .L9

    .text
<aaa>:
   pushq  %rbp
   movq   %rsp, %rbp
   movq   %rdi, -24(%rbp)
   movl   __[6]__, -28(%rbp)
   movl   $0, -4(%rbp)
   movl   -28(%rbp), %eax
   movl   %eax, -8(%rbp)
   jmp .L2
.L13:
   movl   -4(%rbp), %eax
   movslq %eax, %rdx
   movq   -24(%rbp), %rax
   addq   %rdx, %rax
   movzbl __[7]__(%rax), %eax
   movsbl %al, %eax
   subl   $49, %eax
   cmpl   $6, %eax
   ja  .L3
   movq   __[8]__, %rax
   jmp*%rax
```

③

```
.L4:
    cmpl    $8, -8(%rbp)
    jg      .L10
    movl    -8(%rbp), %eax
    jmp     .L11
.L10:
    movl    $2, %eax
```

④

```
.L11:
    movl    %eax, -8(%rbp)
    jmp     .L12
.L7:
    movl    -4(%rbp), %eax
    movslq  %eax, %rdx
    movq    ___[9]___, %rax
    addq    %rdx, %rax
    movzbl  (%rax), %eax
    sarb    $3, %al
    movsbl  %al, %eax
    movl    %eax, -8(%rbp)
.L6:
    addl    $1, -8(%rbp)
    jmp     .L12
```

⑤

```
.L8:
    movl    -4(%rbp), %eax
    addl    %eax, -8(%rbp)
    jmp     .L12
.L9:
    movl    $9, -8(%rbp)
    ___[10]___
```

⑥

```
.L3:
    movl    -8(%rbp), %eax
    addl    %eax, %eax
    subl    $1, %eax
    movl    %eax, -8(%rbp)
.L12:
    addl    $1, -4(%rbp)
.L2:
    movl    -4(%rbp), %eax
    cmpl    -28(%rbp), %eax
    ___[11]___
    movl    ___[12]___, %eax
    popq    %rbp
    ret
```

Suppose the C and assembly code are executed on a 64-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’ * 12)
   NOTE: no more than one instruction/statement per blank. If you think nothing is required to write, please write NONE.

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

3. Suppose the machine is **big-endian**, what will be the output of the main function? (2’)

Problem 6: (23points)

One of TAs of ICS wrote a stupid program. The following C code and assembly code are executed on a **64-bit little endian** machine.

```
void f(long a, long b, long c,
       long d, long e, long f,
       int g, int h) {
    h = g;
    printf("%p\n", ___[16]___);
}
```

```
int main(void) {
    f([8], [9], [10], [11],
      [12], [13], [14], [15]);
    return 0;
}
```

```
 400526 <f>:
  400526:   55                      pushq  %rbp
  400527:   48 89 e5                movq   %rsp, %rbp
  40052a:   8b 45 10                movl   0x10(%rbp), %eax
  40052d:   89 45 14                movl   %eax, 0x14(%rbp)
  ......    // Assembly code for calling printf
  40054d:   c9                      leaveq
  40054e:   c3                      retq

 40054f <main>:
  40054f:   55                      pushq  %rbp
  400550:   48 89 e5                movq   %rsp, %rbp
  ......    // Assembly code for preparing arguments
  400577:   e8 aa ff ff ff          callq  400526 <f>
  40057c:   48 83 c4 10             ___[1]___
  400580:   b8 00 00 00 00          movl   $0, %eax
  400585:   c9                      leaveq
  400586:   c3                      retq
```

1. Fill in the blank in the **assembly code**. (2′).

```
40057c:   48 83 c4 10        ___[1]___
```

2. There is a bug in assembly code of **f**. Please give the address of the instruction and fix it. You don't need to give the binary code of fixed instruction. (2′*2=4′)

```
___[1]___:   ......           ___[2]___
```

3. Assume **BEFORE** the execution of instruction at **400577(callq <f>)**, the memory and register states are as follows:

register	value
%rax	0x00000000 00000000
%rbx	0x00000000 0000000a
%rcx	0x00000000 00000007
%rdx	0x00000000 00000002
%rsi	0x00000000 00000008
%rdi	0x00000000 00000005
%rbp	0x00000000 1000df40
%rsp	0x00000000 1000df30
%r8	0x00000000 00000001
%r9	0x00000000 00000003
%r10	0x00000000 00000004

memory	value
0x1000df40	0x00000000 00400590
0x1000df38	0x00000000 00000004
0x1000df30	0x00000000 00000006

Please fill in the following table that show the state **AFTER** the execution of instruction at **`40052a (movl 0x10(%rbp), %eax)`**: (1'*7=7')

register	value
%rax	[1]
%rbp	[2]
%rsp	[3]

memory	value
0x1000df38	[4]
0x1000df30	[5]
0x1000df28	[6]
0x1000df20	[7]

Please fill in the C code (**[8]~[15]**) of **main** function: (1'*8=8')

4. Please fill in the blank in C code (**[16]**) of **f** function, so that the **printf** will always output the return address of **f**. (2')