复旦大学
2022~2023 学年第 2 学期考试试卷

课程名称：计算机系统基础　　　　　　　　　　考试形式：开卷

姓名：________________ 学号：____________________专业：____________________

提示：请同学们秉持诚实守信宗旨，谨守考试纪律，摒弃考试作弊。学生如有违反学校考试纪律的行为，学校将按《复旦大学学生纪律处分条例》规定予以严肃处理。

题号	1	2	3	4	5	6	7	8	总分
得分									

Part 1: Information Representation (30’)

1. Assume we are running code on a 7-bit machine using two’s complement arithmetic for signed integers.

```
int x = -16;
unsigned uy = x;
```

Fillin the empty boxes in the table below. The following definitions are used in the table.
You need not fill in entries marked with “–”. (8’)

- TMax denotes the largest positive two’s complement number.
- TMin denotes the smallest negative two’s complement number.

Hint: Be careful with the promotion rules that C uses for signed and unsigned ints.

Expression	Decimal Representation	Binary Representation
-	-2	
-		001 0011
x		
uy		
x - uy		
TMax + 1		
TMin - 1		
-TMin		
TMin + TMin		

2. Consider the following two 7-bit floating point representations based on the IEEE floating point format. Neither of them has sign bits—they can only represent nonnegative numbers.
Numeric values are encoded in both of these formats as a value of the form $V = M \times 2^E$, where E is exponent after biasing, and M is the significand value. The fraction bits encode the significand value M using either a denormalized (exponent field 0) or a normalized representation (exponent field nonzero).

1. Format A
   - There are $k = 3$ exponent bits.
   - There are $n = 4$ fraction bits.
2. Format B
   - There are $k = 4$ exponent bits.
   - There are $n = 3$ fraction bits.

a) Write the bit pattern of the given value in the following table. (4')

Value	Format A
0	
Smallest positive	
Largest normalized	
Infinity	
NaN	111 $x_1x_2x_3x_4$(>0)

b) You are given some bit patterns in Format A, and your task is to convert them to the closest value in Format B. If rounding is necessary, you should *round upward*. In addition, give the values of bit patterns in each Format, as a whole number (e.g., 17) or as a fraction (e.g., 17/64). (6')

Format A		Format B	
Bits	Value	Bits	Value
011 0000	1	0111 000	1
101 1110			
010 1001			
110 1111			
000 0001			

3. Given the following code fragment,

```
unsigned char __(1)___8(unsigned char n) {
    n = (n & 0x55) << 1 | (n & 0xAA) >> 1;
    n = (n & 0x33) << 2 | (n & 0xCC) >> 2;
    n = (n & 0x0F) << 4 | (n & 0xF0) >> 4;
    return n;
}

unsigned short __(1)___16(unsigned short n) {
    n = (n & 0x5555) << 1 | (n & 0xAAAA) >> 1;
    n = (n & 0x3333) << 2 | (n & 0xCCCC) >> 2;
    n = (n & 0x0F0F) << 4 | (n & 0xF0F0) >> 4;
    n = ____(2)_____;
    return n;
}
```

These two functions do the same work with an input in different type. Answer the following questions:

a) Describe briefly what these two functions do. Find a proper name for blank (1). (6’)

   Hint: Notice the binary representation of `n`. It’s better to provide illustration with some typical patterns.

b) Fill in the blank (2) to make the second function work. (6’)

Part 2: Machine Level Representation (70’)

4. We consider an illustrative program for multiplication of two unsigned-int’s, returning an unsigned long int holding the product.

```
unsigned long mult(unsigned i,unsigned k)
{
    unsigned long p = 0;
    unsigned long q = k;
    while (i != 0) {
        if (i & 1)
            p = p + q;
        q = q << 1;
        i = i >> 1;
    }
    return p;
}
```

```
mult:
    xorl %ecx, %ecx
    mov %esi, %edx
    testl %edi, %edi
    jmp .L8
.L10:
    leaq (%rcx,%rdx), %rax
    testb $1, %dil
    _____(1)_______
    addq %rdx, %rdx
    shrl %edi
.L8:
    jne .L10
    _____(2)_______
    ret
```

Hints: (i) The register `%dil` is the name of the lowest 8 bits of register `%rdi`. (ii) For blank (1), think about what instruction can be used to eliminate conditional jump in case of simple branch.

Answer the following questions:

a) Describe the value of the given register in terms of the C programming language during iteration. (5’)

Register	C expression
`%rcx`	
`%rdx`	
`%rax`	
`%edi`	
`%dil`	

b) Fill in the missing two instructions in the code. (5’)

c) Rewrite the assembly code to use a conditional jump instead of a conditional move. (4’)

d) Explain briefly why the compiler preferred to use a conditional move instruction. (3’)

e) Assume we declared and initialized

```
int i, k;
long m;
```

and called

```
 m = (long)mult((unsigned)i, (unsigned)k);
```

using the above definition of `mult`. Will `m` hold the correct value of the signed product of `i` and `k`? Briefly explain your answer. (3’)

5. Consider the following C code,

```
int looped(int a[], int n)
{
    int i;
    int x = ____(1)____;
    for (i = ____(2)____; ____(3)____; i++) {
        if (____(4)___)
            x = ____(5)___;
    }
    return x;

}
```

Its corresponding x86-64 assembly function, called looped:

```
looped:
    movl $0, %edx
    testl %esi, %esi
    jle .L4
    movl $0, %ecx
.L5:
    movslq %ecx,%rax
    movl (%rdi,%rax,4), %eax
    cmpl %eax, %edx
    cmovl %eax, %edx
    incl %ecx
    cmpl %edx, %esi
    jg .L5
.L4:
    movl %edx, %eax
    ret
```

a) Where are parameters a and n stored in assembly code? (2')

b) Fill in the blanks in C code. (10')

- You may only use the C variable names `n`, `a`, `i` and `x`, not register names.
- Use array notation in showing accesses or updates to elements of `a`.

6. Given the following program code:

```
int main(){
    unsigned int x=0;
    unsigned int * xp =&x;
    switcher(-10, 25, xp);
    printf("1:%u,%d\n",*xp,*xp);
    switcher(-10, 27, xp);
    if (-10 > x) {
        printf("-10 bigger\n");
    } else {
        printf("x bigger\n");
    }
}
```

```
void switcher(int x, unsigned int n,
                     unsigned int *dest){
    unsigned int val = x;
    switch (n) {
        case   (1)  :
            val = val + n * 2;
            break;
        case   (2)  :
            val = val + n *  (3) ;
            break;
        case   (4)  :
            val = val + n - 30;
            break;
        case    (5)  :
            val = val + n - 110;
            break;
        case   (6)  :
            val = val + n -  (7) ;
            break;
        default:
            val = 0;
    }
    *dest = val;
}
```

We use GDB to debug it. The following table just shows the output with the given command:

```
disas switcher:
0x04005f6 <+00>: lea    -0x19(%rsi),%eax
0x04005f9 <+03>: cmp    $0x4,%eax
0x04005fc <+06>: ja     0x400625 <switcher+47>
0x04005fe <+08>: mov    %eax,%eax
0x0400600 <+10>: jmpq   *0x400758(,%rax,8)
0x0400607 <+17>: lea    (%rdi,__(8)__, __(9)__),%edi
0x040060a <+20>: jmp    0x40062a <switcher+52>
0x040060c <+22>: lea    (%rsi,%rsi,4),%eax
0x040060f <+25>: add    %eax,%edi
0x0400611 <+27>: jmp    0x40062a <switcher+52>
0x0400613 <+29>: lea    -0x1e(%rdi,%rsi,1),%edi
0x0400617 <+33>: jmp    0x40062a <switcher+52>
0x0400619 <+35>: lea    -0x6e(%rdi,%rsi,1),%edi
0x040061d <+39>: jmp    0x40062a <switcher+52>
0x040061f <+41>: lea    -0x46(%rdi,%rsi,1),%edi
0x0400623 <+45>: jmp    0x40062a <switcher+52>
0x0400625 <+47>: mov    $0x0,%edi
0x040062a <+52>: mov    %edi,(%rdx)
0x040062c <+54>: retq
```

```
x/6xg 0x400758:
0x0400758:   0x0000000000400613   0x0000000000400619
0x0400768:   0x0000000000400607   0x000000000040060c
0x0400778:   0x000000000040061f   0x0a64252c75253a31
```

Please answer the following question:

a) Please fill in the blanks (2' * 9)

b) What's the result of the main function? (5')

7. Consider the following code written in C:

```
typedef union {
    struct {
        short a[2];
        char  b[2];
        long  c;
    } t1;
    struct {
        float  x;
        double y;
    } t2;
} utype;
```

We declare a variable `utype u`. and let `u.t2.y = -2.0 and u.t2.x = 2.0.` Please answer the following question:

a) If the code is running on a **64-bit little endian** machine (x86-64/Linux):

  i. What's the size of `u`? (2')

  ii. What can we know about `u.t1`? (6')

b) If the code is running on a **32-bit big endian** machine (sparc32/Linux), what can we know about `u.t1`? (7')