1. Finish your homework independently
2. Convert this docx to pdf: “stuID_name_csapp3.pdf”
   Example: “201X010000_zhangsan_csapp3.pdf”
3. Submit this pdf: learn.tsinghua.edu.cn

3.1
Assume the following values are stored at the indicated memory addresses and registers:

Address	Value	Register	Value
0x100	0xFF	%eax	0x100
0x104	0xAB	%ecx	0x1
0x108	0x13	%edx	0x3
0x10C	0x11		

Fill in the following table showing the values for the indicated operands:

Operand	Value
%eax	**0x100**
0x104	**0xAB**
$0x108	**0x108**
(%eax)	**0xFF**
4(%eax)	**0xAB**
9(%eax,%edx)	**0x11**
260(%ecx,%edx)	**0x13**
0xFC(,%ecx,4)	**0xFF**
(%eax,%edx,4)	**0x11**

3.15
In the following excerpts from a disassembled binary, some of the information has been replaced by Xs. Answer the following questions about these instructions.

A. What is the target of the je instruction below? (You don’t need to know anything about the call instruction here.)

804828f: 8048291:	74 05 e8 1e 00 00 00	je call	XXXXXXX 80482b4

8048296

B. What is the target of the jb instruction below**?**

8048357: 8048359:	72 e7 c6 05 10 a0 04 08 01	jb movb	XXXXXXX $0x1,0x804a010

8048340

C. What is the address of the mov instruction?

XXXXXXX: XXXXXXX:	74 12 b8 00 00 00 00	je mov	8048391 $0x0,%eax

804837f

D. In the code that follows, the jump target is encoded in PC-relative form as a 4-byte, two's-complement number. The bytes are listed from least significant to most, reflecting the little-endian byte ordering of IA32. What is the address of the jump target?

80482bf: 80482c4:	e9 e0 ff ff ff 90	jmp nop	XXXXXXX

80482a4

E. Explain the relation between the annotation on the right and the byte coding on the left.

80482aa:	ff 25 fc 9f 04 08	jmp	*0x8049ffc

ff25是间接跳转的指令代码， fc9f0408是要跳转的行号8049ffc，由于小端，所以呈现出倒置的样子

3.34

For a C function having the general structure

```
int rfun(unsigned x) {
    if (________)
    return ________;
    unsigned nx = ________;
    int rv = rfun(nx);
    return ________;
}
```

gcc generates the following assembly code (with the setup and completion code omitted):

```
1 movl 8(%ebp), %ebx
2 movl $0, %eax
3 testl %ebx, %ebx
4 je .L3
5 movl %ebx, %eax
6 shrl %eax                    Shift right by 1
7 movl %eax, (%esp)
8 call rfun
9 movl %ebx, %edx
10 andl $1, %edx
11 leal (%edx,%eax), %eax
12 .L3:
```

A. What value does rfun store in the callee-save register %ebx?

x

B. Fill in the missing expressions in the C code shown above.

```
int rfun(unsigned x) {
    if (___x == 0_____)
    return ____0____;
    unsigned nx = ____x >> 1____;
    int rv = rfun(nx);
    return ___rv + (x & 1)_____;
}
```

C. Describe in English what function this code computes

Compute the sum of all bits in binary representation of x

3.56

Consider the following assembly code:

```
x at %ebp+8, n at %ebp+12
1 movl 8(%ebp), %esi
2 movl 12(%ebp), %ebx
3 movl $-1, %edi
4 movl $1, %edx
5 .L2:
6 movl %edx, %eax
7 andl %esi, %eax
8 xorl %eax, %edi
9 movl %ebx, %ecx
10 sall %cl, %edx
11 testl %edx, %edx
12 jne .L2
13 movl %edi, %eax
```

The preceding code was generated by compiling C code that had the following overall form:

```
1 int loop(int x, int n)
2 {
3     int result = _________;
4     int mask;
5     for (mask = _________; mask _________; mask = _________) {
6         result ^= _________;
7     }
8     return result;
9 }
```

Your task is to fill in the missing parts of the C code to get a program equivalent to the generated assembly code. Recall that the result of the function is returned in register %eax. You will find it helpful to examine the assembly code before, during, and after the loop to form a consistent mapping between the registers and the program variables.

A. Which registers hold program values x, n, result, and mask?
x:esi; n:ebx; result:edi; mask: edx
B. What are the initial values of result and mask?
-1 1
C. What is the test condition for mask?
mask != 0
D. How does mask get updated?
mask = mask << n
E. How does result get updated?
result ^= (mask & x)
F. Fill in all the missing parts of the C code

```
1 int loop(int x, int n)
```

```
{
    int result = ____-1_____;
    int mask;
    for (mask = ____1___; mask ___!= 0 ____; mask = __mask_<< n ______) {
        result ^= ___(mask & x)______;
    }
    return result;
}
```

3.57

In Section 3.6.6, we examined the following code as a candidate for the use of conditional data transfer:

```
int cread(int *xp) {
    return (xp ? *xp : 0);
}
```

We showed a trial implementation using a conditional move instruction but argued that it was not valid, since it could attempt to read from a null address. Write a C function cread_alt that has the same behavior as cread, except that it can be compiled to use conditional data transfer. When compiled with the command-line option '-march=i686', the generated code should use a conditional move instruction rather than one of the jump instructions.

```
int cread_alt(int *xp)
{
    int temp = 0;
    int *ret = xp ? xp : &temp;
    return *ret;
}
```