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In the following excerpts from a disassembled binary, some of the information has been replaced by Xs.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	8048296 80482b4

XXXXXXX = 0x8048291 + 0x05 = 0x8048296

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

8048357: 8048359:	72 e7 c6 05 10 a0 04 08 01	jb movb	8048340 $0x1,0x804a010

XXXXXXX = 0x8048291 + 0xe7 = -0x8048359 - 25 = 0x8048359 - 0x19 = 0x8048340

C. What is the address of the mov instruction?

804837d: 804837f:	74 12 b8 00 00 00 00	je mov	8048391 $0x0,%eax

XXXXXXX = 0x8048391 - 0x12 = 0x804837f

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	80482a4

XXXXXXX = 0x80482c4 + 0xfffffffe0 = 0x80482c4 – 32 = 0x80482c4 – 0x20 = 0x80482a4

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

ff 25表示jmp命令的绝对跳转，fc 9f 04 08为传入地址，倒过来读为0x8049ffc，即jmp *0x8049ffc

3.34

For a C function having the general structure

```
int rfun(unsigned x) {
    if (x==0)
    return 0;
    unsigned nx = x>>1;
    int rv = rfun(nx);
    return x + rv;
}
```

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:
```

%ebx=x →%eax=0 →

%ebx&%ebx==0 →return 0

%ebx&%ebx!=0 →%eax=%ebx=x →%eax<<=1 →(%esp)= %eax=x>>1

Rfun →%ebx=x>>1, %eax=x>>2

%edx=%ebx,=x>>1 →%edx=%edx&1 →%eax=%edx+%eax=x>1>+x>>2

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

rfun store the value of “x” in the callee-save register %ebx.

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

C. Describe in English what function this code computes

Returns the sum of the numbers passed in as arguments shifted right to zero bit by bit.

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
```

%esi=x→%ebx=n→%edi=-1→%edx=1

6 %eax=mask 7 mask=mask & x 8 result= mask ^ result

9 %ecx=n 10 mask << n

11 mask!=0

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 = 0xffffffff;
4     int mask;
5     for (mask = 1; mask != 0; mask = mask<<n) {
6         result ^= mask & x;
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?

%esi hold program values x,

%ebx hold program values n,
%edi hold program values result,
%edx hold program values mask.

B. What are the initial values of result and mask?

the initial values of result is -1/0xffffffff
the initial values of mask is 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

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 nil = 0;
    if(!xp)
      x = &nil;
   return *xp;
}
```