Homework 7

1. Buffer Overflow

One of TAs of ICS wrote a buggy program. The following C code and assembly code are executed on a **64-bit little endian** machine. He used `gets()` functions in section 3.10.3 on CSAPP.

```
void buggy(){
    char buf[0x10];
    gets(buf);
}
int main(){
    buggy();
    return 0;
}
```

```
00000000004004e6 <buggy>:
  4004e6:      55                    push   %rbp
  4004e7:      48 89 e5              mov    %rsp,%rbp
  4004ea:      48 83 ec 10           sub    $0x10,%rsp
  4004ee:      48 8d 45 f0           lea    -0x10(%rbp),%rax
  4004f2:      48 89 c7              mov    %rax,%rdi
  4004f5:      e8 17 00 00 00        callq  400511 <gets>
  4004fa:      c9                    leaveq
  4004fb:      c3                    retq

00000000004004fc <main>:
  4004fc:      55                    push   %rbp
  4004fd:      48 89 e5              mov    %rsp,%rbp
  400500:      b8 00 00 00 00        mov    $0x0,%eax
  400505:      e8 dc ff ff ff        callq  4004e6 <buggy>
  40050a:      b8 00 00 00 00        mov    $0x0,%eax
  40050f:      5d                    pop    %rbp
  400510:      c3                    retq
```

Now the TA uses different strings to feed the `gets()` in `buggy()`. Give the corresponding return address of function `buggy()` to each return address. (NOTE: the ASCII number of '0' is 48.

a. "" 0x40050a

b. "0123456789" 0x40050a

c. "01234567890123456789" 0x40050a

d. "012345678901234567890123" 0x400500

e. "0123456789012345678901234567890123456789" 0x393837363534

2. Floating point

Consider a 16-bit floating-point representation based on the IEEE floating-point format, with 1 sign bit, 5 exp bits, 10 frac bits, called Float16.

(1) Fill in the following table. Represent M in the form x or x/y where x is an integer and y is an integral power of 2, and represent Value in the form a or a * 2^b where a and b are integers.

Description	Hex	M	E	Value
-0	0X8000	0	-14	--
Largest negative Normalized value	0X8400	1	-14	-1*2^(-14)
+∞	0X7C00	--	--	--
Largest Denomalized value	0X03FF	1023/1024	-14	1023*2^(-24)
$(11.375)_{10}$	0x49B0	91/64	3	91*2^(-3)
Number with hex representation 0x4BF7	0x4BF7	2039/1024	3	2039*2^(-7)

(2) Assume we use IEEE round-to-even mode to do the approximation. Now a, b are both Float16, with a = 0x4663 and b = 0x394c represented in hex. Compute a+b and represent the answer in hex.

a: 0|100 01|10 0110 0011 S1=0, E1=17-15=2, M1=1.1001100011

b: 0|011 10|01 0100 1100 S2=0, E2=14-15=-1, M2=1.0101001100

a+b:

```
  1.1001100011
 +0.0010101001100
  1.1100001100100
```

$M=(1.1100001100100)_2$; $M'=(1.1100001100|100)_2$

= (Round to even)$(1.1100001100)_2$

E=E1=2

Exp = E + bias = 17

a+b = $(0\ 10001\ 1100001100)_2$ = 0x470c

(3) Using Float16, what's the difference between 2^15 + 0.5 – 2^15 and 2^15 – 2^15 + 0.5? Calculate them to explain why.

2^15 : 0|111 10|00 0000 0000
0.5: 0|011 10|00 0000 0000
2^15+0.5:
  1.0000 0000 00
+ 0.0000 0000 0000 0001 0000 0000 00
= 1.0000 0000 0000 0001 0000 0000 00
M=$(1.0000\ 0000\ 00|00\ 0001\ 0000\ 0000\ 00)_2$ =1.0000 0000 00
E=30
2^15+0.5 = 0|111 10|00 0000 0000 = 2^15
So 2^15+0.5-2^15 = 0

But 2^15-2^15=0
2^15-2^15+0.5=0.5

0.5 is rounded during the calculation of 2^15 + 0.5.