The next problem concerns the following C code. This program reads a string on standard input and prints an integer in hexadecimal format based on the input string it read.

```
#include <stdio.h>

/* Read a string from stdin into buf */
int evil_read_string()
{
   int buf[2];
   scanf("%s",buf);
   return buf[1];
}

int main()
{
   printf("0x%x\n", evil_read_string());
}
```

Here is the corresponding machine code on a Linux/x86 machine:

```
08048414 <evil_read_string>:
 8048414: 55
                                    push
                                           %ebp
 8048415: 89 e5
                                   mov
                                           %esp,%ebp
 8048417: 83 ec 14
                                   sub
                                           $0x14,%esp
 804841a: 53
                                   push
                                           %ebx
 804841b: 83 c4 f8
                                   add
                                           $0xfffffff8,%esp
 804841e: 8d 5d f8
                                           0xfffffff8(%ebp),%ebx
                                   lea
 8048421: 53
                                   push
                                                                  address arg for scanf
 8048422: 68 b8 84 04 08
                                           $0x80484b8
                                   push
                                                                  format string for scanf
 8048427: e8 e0 fe ff ff
                                   call
                                           804830c <_init+0x50>
                                                                  call scanf
 804842c: 8b 43 04
                                           0x4(\%ebx),\%eax
                                   mov
 804842f: 8b 5d e8
                                           0xffffffe8(%ebp),%ebx
                                   mov
 8048432: 89 ec
                                           %ebp,%esp
                                    mov
 8048434:
                                    pop
           5d
                                           %ebp
 8048435: c3
                                    ret
08048438 <main>:
 8048438: 55
                                    push
                                           %ebp
 8048439: 89 e5
                                   mov
                                           %esp,%ebp
 804843b: 83 ec 08
                                           $0x8, %esp
                                    sub
 804843e: 83 c4 f8
                                    add
                                           $0xfffffff8,%esp
 8048441: e8 ce ff ff ff
                                           8048414 <evil read string>
                                   call
 8048446: 50
                                   push
                                           %eax
                                                                  integer arg for printf
 8048447: 68 bb 84 04 08
                                           $0x80484bb
                                   push
                                                                  format string for printf
 804844c: e8 eb fe ff ff
                                   call
                                           804833c <_init+0x80> call printf
 8048451: 89 ec
                                   mov
                                           %ebp,%esp
 8048453: 5d
                                           %ebp
                                   pop
 8048454: c3
                                    ret
```

Problem 32. (12 points):

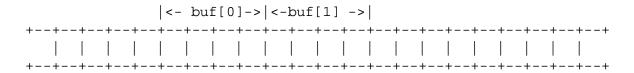
This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- scanf("%s", buf) reads an input string from the standard input stream (stdin) and stores it at address buf (including the terminating '\0' character). It does **not** check the size of the destination buffer.
- printf("0x%x", i) prints the integer i in hexadecimal format preceded by "0x".
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
'd'	0x64	' V '	0x76
'r'	0x72	'i'	0x69
' · '	0x2e	'1'	0x6c
'e'	0x65	'\0'	$0 \times 0 0$
		's'	0x73

A. Suppose we run this program on a Linux/x86 machine, and give it the string "dr.evil" as input on stdin.

Here is a template for the stack, showing the locations of buf[0] and buf[1]. Fill in the value of buf[1] (in hexadecimal) and indicate where ebp points just after scanf returns to evil read string.



What is the 4-byte integer (in hex) printed by the printf inside main?

0x

В.	Suppose now v	we give it the input "d	r.evil.lives"	(again on a Linux/x86	5 machine).

(a) List the contents of the following memory locations just **after** scanf returns to evil read string. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

(b) Immediately **before** the ret instruction at address 0x08048435 executes, what is the value of the frame pointer register %ebp?

```
%ebp = 0x_____
```

You can use the following template of the stack as *scratch space*. *Note*: this does **not** have to be filled out to receive full credit.

