Question 1
4 / 4 pts
Match the following assembly instructions(left side) to the appropriate description(right side).
Correct!
movl $0x20, %eax
Immediate to register
Correct!
movl %eax, %ebx
Register to register
Correct!
movl (%eax), %ebx
Memory to register
Correct!
movl %eax, (%ebx)
Register to memory
Other Incorrect Match Options:
movl %eax, $0x20

Question 2
4 / 4 pts
Consider the following C program and assembly code of echo. .

```
#include <stdio.h>
void echo(void){
   char buf[T];
   gets(buf);
   puts(buf);
}
```

Dump of assembler code for function echo:

```
0x0804846b <+0>: push %ebp
0x0804846c <+1>: mov %esp,%ebp
0x0804846e <+3>: sub $0x20,%esp
0x08048471 <+6>: lea -0x20(%ebp),%eax
0x08048474 <+9>: push %eax
0x08048475 <+10>: call 0x8048330 <gets@plt>
0x0804847a <+15>: add $0x4,%esp
0x0804847d <+18>: lea -0x20(%ebp),%eax
0x08048480 <+21>: push %eax
0x08048481 <+22>: call 0x8048340 <puts@plt>
0x08048486 <+27>: add $0x4,%esp
0x08048489 <+30>: nop
0x0804848a <+31>: leave
0x0804848b <+32>: ret
```

What is the value of T in the function echo

25
16
20
Correct!
32

Question 3
6 / 6 pts
Choose missing C statements based on the given assembly instructions.

```
movl   16(%ebp), %eax

movl   12(%ebp), %edx

subl    %eax, %edx

movl   %edx, %eax

imull    8(%ebp), %edx

sall     $31, %eax

sarl     $31, %eax

xorl     %edx, %eax
```

```
int decode(int x, int y, int z) {

    int t1 = y - z;

    int t2 = x * (y-z)

    int t3 = (t1 << 31) >> 31;

    return [E2];

}
```

Correct!

E2 = t3^t2
E2 = t3 | t2

Question 4
10 / 10 pts
Consider the following disassembly of the sum function which adds two given numbers. Which instructions can be safely removed without affecting the functional behavior of sum?

```
080483b4 <sum>:
80483c0: 55            push %ebp
80483c1: 89 e5         mov %esp,%ebp
80483c3: 83 ec 10      sub $0x10,%esp
80483c6: 8b 45 0c      mov 0xc(%ebp),%eax
80483c9: 8b 55 08      mov 0x8(%ebp),%edx
80483cc: 01 d0         add %edx,%eax
80483ce: 89 45 fc      mov %eax,-0x4(%ebp)
80483d1: 8b 45 fc      mov -0x4(%ebp),%eax
80483d4: c9            leave
80483d5: c3 ret
```

```
int sum(int x, int y) {
int t = x + y; /* Ignore integer arith. issue */
return t;
}
```

Correct!
0x80483ce
0x80483c6

Correct!
0x80483d1
Correct!
0x80483c3

Question 5
6 / 6 pts
What is the exit status if the user does not pass sufficient arguments to the assembler routine below?

```
0x0804855a <+0>:    push %ebp
0x0804855b <+1>:    mov %esp,%ebp
0x0804855d <+3>:    cmpl $0x2,0x8(%ebp)
0x08048561 <+7>:    je  0x8048577 <ArgCheck+29>
0x08048563 <+9>:    push $0x8048760
0x08048568 <+14>:   call 0x8048330 <puts@plt>
0x0804856d <+19>:   add $0x4,%esp
0x08048570 <+22>:   push $0x1
0x08048572 <+24>:   call 0x8048340 <exit@plt>
0x08048577 <+29>:   nop
0x08048578 <+30>:   leave
0x08048579 <+31>:   ret
```

2
0
-1
Correct!
1

Question 6
12 / 12 pts
What is the length of the smallest input required overflow the buffer in the function below?

```
0x0804852c <+0>:    push %ebp
0x0804852d <+1>:    mov %esp,%ebp
0x0804852f <+3>:    sub $0x14,%esp
0x08048532 <+6>:    movl $0x1e,-0x4(%ebp)
0x08048539 <+13>:   pushl 0x8(%ebp)
0x0804853c <+16>:   lea -0x12(%ebp),%eax
0x0804853f <+19>:   push %eax
0x08048540 <+20>:   call 0x8048320 <strcpy@plt>
0x08048545 <+25>:   add $0x8,%esp
0x08048548 <+28>:   mov -0x4(%ebp),%eax
0x0804854b <+31>:   movzbl %al,%eax
0x0804854e <+34>:   push %eax
```

0x0804854f <+35>: call 0x80484ca <secret>
0x08048554 <+40>: add $0x4,%esp
0x08048557 <+43>: nop
0x08048558 <+44>: leave
0x08048559 <+45>: ret
15
16
Correct!
14
18

Question 7
7 / 14 pts
Consider the following assembly code of a C function to answer the following questions.

```
void illusion(unsigned char x)
{
  switch(x)
  {
.......

 }
}
```

Dump of assembler code for function secret:

```
0x080484ca <+0>:  push %ebp

0x080484cb <+1>:  mov %esp,%ebp

0x080484cd <+3>:  sub $0x4,%esp

0x080484d0 <+6>:  mov 0x8(%ebp),%eax

0x080484d3 <+9>:  mov %al,-0x4(%ebp)

0x080484d6 <+12>:  movzbl -0x4(%ebp),%eax

0x080484da <+16>:  sub $0x58,%eax

0x080484dd <+19>:  cmp $0x22,%eax

0x080484e0 <+22>:  ja 0x804851c <secret+82>

0x080484e2 <+24>:  mov 0x80486d4(,%eax,4),%eax
```

```
0x080484e9 <+31>:    jmp *%eax
0x080484eb <+33>:    push $0x41
0x080484ed <+35>:    push $0x804a024
0x080484f2 <+40>:    call 0x804846b <show_flag>
0x080484f7 <+45>:    add $0x8,%esp
0x080484fa <+48>:    jmp 0x8048529 <secret+95>
0x080484fc <+50>:    push $0x8048683
0x08048501 <+55>:    call 0x8048330 <puts@plt>
0x08048506 <+60>:    add $0x4,%esp
0x08048509 <+63>:    jmp 0x8048529 <secret+95>
0x0804850b <+65>:    push $0x41
0x0804850d <+67>:    push $0x804a024
0x08048512 <+72>:    call 0x804846b <show_flag>
0x08048517 <+77>:    add $0x8,%esp
0x0804851a <+80>:    jmp 0x8048529 <secret+95>
0x0804851c <+82>:    push $0x80486a0
0x08048521 <+87>:    call 0x8048330 <puts@plt>
0x08048526 <+92>:    add $0x4,%esp
0x08048529 <+95>:    nop
0x0804852a <+96>:    leave
0x0804852b <+97>:    ret
```

(gdb) x/50xw 0x80486d4
0x80486d4: 0x0804850b 0x080484fc 0x080484eb 0x0804851c
0x80486e4: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x80486f4: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048704: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048714: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048724: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048734: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048744: 0x0804851c 0x0804851c 0x0804851c 0x0804851c
0x8048754: 0x0804850b 0x080484fc 0x080484eb 0x67617355
0x8048764: 0x2e203a65 0x64696d2f 0x6d726554 0x6f793c20
A. What is the base address of the jump table?

Answer: [jump_table]

B. How many case statements N are defined in the C function (excluding default)?

Answer: [N]

C. What is the address of the default statement code block?

Answer: [default]

D. What is the address of the code block for case label B in decimal?

Answer: [label_B]

default=0x0804842e
jump_table=0x080483fc
Correct!
jump_table=0x80486d4
label_B=0x0804841c
Correct Answer
N=6
You Answered

N=5
Correct!
label_B=0x0804851c
Correct!
default=0x804851c

Question 8
10 / 10 pts
Consider the following IA32 code for a procedure foo():

```
foo:
    pushl   %ebp
    movl    %esp,%ebp
    movl    8(%ebp),%ecx
    movl    16(%ebp),%edx
    movl    12(%ebp),%eax
    decl    %eax
    js .L3
.L7:
    cmpl    %edx,(%ecx,%eax,4)
    jne .L3
    decl    %eax
    jns .L7
.L3:
     movl    %ebp,%esp
     popl %ebp
     ret
```

Based on the assembly code above, choose expressions in its corresponding C source code.

Fill in the blanks of the bar:

```
int foo(int *a, int n, int val) {
    int i;
    for (i = [E1]; [E2] ; i =[E3]) {
          ;
     }
```

```
    return i;

}
```

Correct!
E2= (i >=0 && a[i] == val)
E3 = (1-i)
Correct!
E1 = (n-1)
E2= (i >=0 && a[i] != val)
E1 = n
Correct!
E3 = (i-1)

Question 9
8 / 9 pts
The following problem concerns the following, low-quality code:

```
void foo(int x) {
    int a[ 3 ];
    char buf[ 4 ];
    a[ 0 ] = 0xF0F1F2F3;
    a[ 1 ] = x;
    gets(buf);
    printf("a[ 0 ] = 0x%x, a[ 1 ] = 0x%x, buf = %s\n", a[ 0 ], a[ 1 ], buf);
}
```

In a program containing this code, procedure foo has the following disassembled form on an IA32 machine:

```
080485d0 <foo>:
80485d0: 55                              pushl   %ebp
80485d1: 89 e5                           movl    %esp,%ebp
80485d3: 83 ec 10                        subl    $0x10,%esp
80485d6: 53                              pushl   %ebx
80485d7: 8b 45 08                        movl    0x8(%ebp),%eax
80485da: c7 45 f4 f3 f2                  movl    $0xf0f1f2f3,0xfffffff4(%ebp)
80485df: f1 f0 80485e1: 89 45 f8                 movl    %eax,0xfffffff8(%ebp)
80485e4: 8d 5d f0                        leal    0xfffffff0(%ebp),%ebx
80485e7: 53                              pushl   %ebx
80485e8: e8 b7 fe ff ff                  call    80484a4 <_init+0x54> # gets
80485ed: 53                              pushl   %ebx
80485ee: 8b 45 f8                        movl    0xfffffff8(%ebp),%eax
80485f1: 50                              pushl   %eax
80485f2: 8b 45 f4                        movl    0xfffffff4(%ebp),%eax
```

```
80485f5: 50                    pushl    %eax
80485f6: 68 ec 90 04 08              pushl    $0x80490ec
80485fb: e8 94 fe ff ff          call   8048494 <_init+0x44> # printf
8048600: 8b 5d ec                  movl     0xfffffffec(%ebp),%ebx
8048603: 89 ec                   movl    %ebp,%esp
8048605: 5d                     popl     %ebp
8048606: c3                     ret
8048607: 90                     nop
```

For the following questions, recall that:

gets is a standard C library routine.

IA32 machines are little-endian.

C strings are null-terminated (i.e., terminated by a character with value 0x00).

Characters ‘0’ through ‘9’ have ASCII codes 0x30 through 0x39.

Consider the case where procedure foo is called with argument x equal to 0xE3E2E1E0, and we type “123456789” in response to gets.

A. Fill in the following table indicating which program values are/are not corrupted by the response from gets, i.e., their values were altered by some action within the call to gets. (1 pt each)

Please type Y or N for the fill in the blanks questions that ask for yes or no. Please do not add anything else, otherwise you may loose points due to automation.

Program Value	Corrupted? (Y/N)
a[0]	
[Select]	
a[1]	
[Select]	
a[2]	

[Select]

x
[Select]

Saved value of register %ebp
[Select]

Saved value of register %ebx
[Select]

B. What will the printf function print for the following: (2 pts each). Please fillup addresses in the right format: 0x<your address choice>.

a[0] (hexadecimal):
[Select]

a[1] (hexadecimal):
[Select]

buf (ASCII):
[Select]

Answer 1:
Correct!
Y
Answer 2:
Correct!
Y
Answer 3:
Correct!
N
Answer 4:
Correct!
N
Answer 5:
Correct!
N
Answer 6:
Correct!
N
Answer 7:

Correct Answer
0x38373635
You Answered
0x31323334
Answer 8:
Correct!
E3E20039
Answer 9:
Correct!
123456789

Question 10
8 / 8 pts
Consider the following disassemly of a C function named getData.

```
(gdb) disass getData
Dump of assembler code for function getData:
   0x0804840c <+0>:  push   %ebp
   0x0804840d <+1>:  mov    %esp,%ebp
   0x0804840f <+3>:   sub    $0x30,%esp
   0x08048412 <+6>:  lea    -0x13(%ebp),%eax
   0x08048415 <+9>:  mov    %eax,(%esp)
   0x08048418 <+12>: call    0x80482f0 <gets@plt>
   0x0804841d <+17>: leave
   0x0804841e <+18>: ret
End of assembler dump.
```

A. What is the least number of bytes B needed to overflow the buffer in order to corrupt the least significant byte of EBP? (Hint: NULL byte terminates the input to the gets function)

Answer: 19

B. Assume that some shellcode is placed at 0x08048521 and you are planning to exploit the buffer overflow vulnerability as follows:

```
payload = "A" x 23 . "\x21\x85\x04\x08"
```

What will be the value of N? That is, how many A's have to be filled in as part of the payload.

Answer: [N]

Answer 1:
Correct!
19
Answer 2:
Correct!
23

Question 11
8 / 8 pts
Consider the disassembly of a C function to answer the following questions.

00000000 <foo>:

```
 0:55                    push   %ebp
 1:89 e5                 mov    %esp,%ebp
 3:83 ec 04              sub    $0x4,%esp
 6:8b 45 14              movsbl 0x10(%ebp),%eax
 9:88 45 fc              mov    %eax,-0x4(%ebp)
 c:0f be 55 fc           movsbl -0x4(%ebp),%edx
10:8b 45 08              mov    0x8(%ebp),%eax
13:01 d0                 add    %edx,%eax
15:c9                    leave
16:c3                    ret
```

A. What is the minimum number of function parameters P of foo or the number of arguments foo takes?

Answer:
[Select]

B. What is the data type T of the argument pointed by 0x10+%ebp?

Answer:
[Select]

Answer 1:
Correct!
3
Answer 2:
Correct!
char

Question 12
9 / 9 pts
What does it indicate when the condition flag ZF is set to 1?

Last executed operation had resulted in integer overflow
Correct!
Last executed operation had a result of zero
Last executed operation had resulted in error
Last executed operation had a negative result