Tips for Exam #2

Concepts you must know

- Assembly-code suffix
- Assembly addition, subtraction, LEA instruction
- If-else condition, while-loop, for-loop, ↔ assembly codes
- Recursion assembly
- C codes ↔ assembly codes
- Array size and address
- Data structures
- Buffer in callee (registers) and overflow
- The BombLab should have been a great practice

Question 1

Determine the appropriate instruction suffix based on the operands.

mov **b** $0xFF, %bl

b | w | l | q

Instruction		Effect	Description
MOV	*S*, *D*	*D* ← *S*	Move
movb			Move byte
movw			Move word
movl			Move double word
movq			Move quad word
movabsq	*I*, *R*	*R* ← *I*	Move absolute quad word

Question 2

If %rdx=42 and %rdi=6, what is the value of %r11 after executing

```
leaq (%rdx, %rdi), %r11
```

You can use an expression is that's useful.

48

Number A (base_reg, index_reg, number B) = base_reg + (B * index_reg) + A

Question 3

Given the following memory contents:

Memory

Addr	Contents
300	10
304	17
308	3
312	30
316	22
320	27
324	2
328	14
332	49
336	6

and given that the following instruction is being executed

```
subl 28,28(%rsi)
```

when **%rsi** = 296

a) What memory address is modified (decimal, equation ok) : 324

b) What is the updated value at that address (decimal, equation ok) : -26

RSI + 28 = 296 + 28 = 324

At the address of 324, the content is 2.

So, 2 – 28 = -26.

Question 4

signed: l, g, le, ge
unsigned: a, b, ae, be

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.

```
#include <stdio.h>

int main() {
    long int a=5;

    if (a < 10) {
        return 3;
    }
    else {
        return 0;
    }
}
```

```
main:
    push %rbp
    mov  %rsp, %rbp
    movq $0x5, -0x8(%rbp)
    cmpq $0x9, -0x8(%rbp)
    jg   L1
    mov $0x3, %eax
    jmp   L2
L1:
    mov $0x0, %eax
L2:
    pop %rbp
    retq
```

je | jl | jbe | ja | js | jg | jns | jb | jle | jne | jae | jge

c | a | b | x | z | y

!= | < | <= | > | == | >=

A – 9 > 0 => go to L1
Otherwise, go to the next line.

Let's think about the edge case, 'jge'.
In order to go to the else condition, A should be at least 10.
But, if we use 'jge', we will jump L1 when A is equal to 9.
So, the correct answer is jg.

Question 5

If we compile the following C function, we get the assembly code shown below. Fill in the missing values in the assembly to match the C code.

```
int ii;
int limit;
int foo() {
  int sum = 0;
  for (int i = ii; i >= 4 ; i -= 1) {
    sum += bar(sum,i);
  }
  return sum;
}
```

ii < ? => does not go inside the for loop.
The condition is >= 4, so the first blank is 4.
Each iteration I is subtracted by 1.
I > ? => do another iteration
Again, the condition is I >=4
So, ? is 3.

```
foo:
.LFB0:
        pushq   %rbx
        movl    \$0, sum
        movl    ii, %ebx
        cmpl    \$ 4 , %ebx
        jl      .L2
.L3:
        movl    %ebx, %esi
        movl    sum, %edi
        call    bar
        addl    %eax, sum
        subl    \$ 1 , %ebx
        cmpl    \$ 3 , %ebx
        jg      .L3
.L2:
        movl    sum, %eax
        popq    %rbx
        ret
```

Question 6

Write a C function **func** that performs the actions of the following assembly code.

The function takes three arguments passed in registers **%edi, %esi, %edx.**

1 2 3

```
func:
        movl    %edx, %eax
        movl    %esi, %edx
.L2:
        addl    %esi, %edi
        addl    $1, %eax
        cmpw    %di, %dx
        jg      .L2
        ret
```

c – a > 0

Edi is 1st Esi is 2nd Edx is 3rd argument register.

So, let's suppose to use a, b, and c.

Then, edi is a, esi is b and edx is c.

For example:

Test	Result
test(4,5,6);	OK1

Answer: (penalty regime: 10, 20, ... %)

```
1  int func(int a, int b, int c){
2      int result = c;
3      c = b;
4      do{
5          a += b;
6          result += 1;
7      }while(c > a);
8      return result;
9  }
10
```

Question 7

Write a C function **func** that performs the actions of the following assembly code:

```
func:                1     2     edi is 1st / esi is 2nd argument register.
        cmpl    %edi, %esi
        jge     .L5
        leal    1(%rdi), %eax
        ret
.L5:
        movl    %esi, %eax
        subl    %edi, %eax
        ret
```

For example:

Test	Result
`test(5,4);`	OK1

Answer: (penalty regime: 10, 20, ... %)

```
1  int func(int a, int b)
2  {
3    if (b >= a ) {
4      return b-a;
5    } else {
6      return a+1;
7    }
8  }
9
10
```

Question 8

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.

```
int a;
int foo() {
  int rval;
  if (a <= 1)
    return 1;
  rval = foo(a - 1);
  return rval+a;
   }
}
```

```
foo:
    movl    a, %ebx
    cmpl    $1, %ebx
    jle     .L3
    leal    -1(%ebx),%eax
    pushl   %eax
    call    foo
    leal    (%ebx, %eax)
    jmp     .L2
.L3:
    movl    $1, %eax
.L2:
    ret
```

a – 1 <= 0

/ | * | << | + | >> | -

Question 9

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.

```
int a;
int foo() {
  int rval;
  if (a == 0)
    return 1;
  rval = foo(a>>1);
  return rval * a;
   }
}
```

```
foo:
    movl    a, %ebx
    cmpl    $0, %ebx
    je      L3
    movl    %ebx,%eax
    shrl    %eax
    pushl   %eax
    call    foo
    imull   %ebx, %eax
    jmp     L2
.L3:
    movl    $1, %eax
.L2:
    ret
```

signed multiply

+ | - | * | / | >> | <<

Question 10

2 byte

An array A is declared:

short A[3][3];

What is **sizeof(A)**?

18

A[3][3] =

00	**01**	**02**
10	11	12
20	21	22

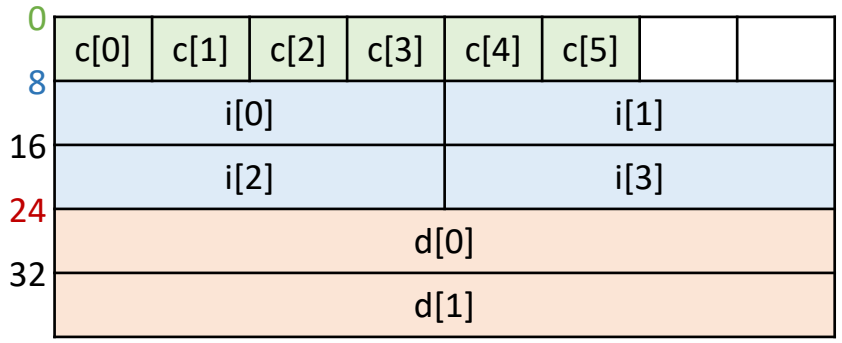
2 bytes x 3 x 3 = 18

Question 11

Memory Alignment!

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.

```
struct {
  char c[ 6 ];
  int i[ 4 ];
  double d[ 2 ];
} datum[ 4 ];
```



- char array ‘c’ starts at offset 0
- integer array ‘I’ starts at offset 8
- double array ‘d’ starts at offset 24
- size of each struct is 40

What is the offset of datum[3].c[3] relative to &datum? 123 — 40 x 3 + 3

What is the offset of datum[3].i[1] relative to &datum? 132 — 40 x 3 + 12

What is the offset of datum[3].d[0] relative to &datum? 144 — 40 x 3 + 24