1. Use the Load Effective Address instruction to implement the following arithmetic operations:

a. Add 20 to a variable

```
LEA eax,[eax+20]
```

b. Multiply a variable by 52

```
LEA eax,[ebx+ebx*8]//9
LEA eax,[eax+eax*4]//9+36=45
LEA eax,[eax+ebx*4]//45+4=49
LEA eax,[eax+ebx*2]//49+2=51
LEA eax,[eax+ebx]//51+1
```

c. Multiply a variable by 9

```
LEA eax,[eax+eax*8]
```

2. Use SHL/SAL, LEA and ADD to implement the following arithmetic operations:

a. Multiply a variable by 24

```
MOV eax,[variable]
MOV ebx,[variable]
//3x ebx
ADD ebx,ebx
ADD ebx,eax

//mul by 3
LEA     eax,[ebx]
// mul by 8
SHL eax,3
```

b. Multiply a variable by 1000

```
MOV eax,[variable]
MOV ebx,[variable]
MOV edx,[variable]


//shift by 10==1024
SHL eax,10

// 24 times

ADD ebx,ebx
ADD ebx,edx
```

```
                    LEA ebx, [ebx*8]

                    ADD eax ,-ebx
```

3. Implement the following while loop in IA-32 assembly language using a post-tested loop.
Why is a post-tested loop implementation possible in this situation?

```
int x = 1;
while (x <= 10)
{

      if (x != 5)
   {
       /* CodeBlock */
       x = x * 2;
   }
   x = x + 1;
}
                    //int x = 1
                    MOV eax, 1


                    LoopStart:
                              //  if (x != 5)
                              Cmp eax,5
```

```
                JNE IFCondition
                //x=x+1
                INC eax
                //While condition

                Cmp eax,10
                Jle LoopStart

        IFConsition:
                //x=x*2
                LEA eax, DWORD PTR [eax+eax]
```

Why is a post-tested loop implementation possible in this situation?

Because the condition for the while loop is satisfied, so the post-tested loop is possible.

4. Implement the following C code segment in IA-32 assembly language as efficiently as possible.

```
int j = 5;
for (int i = 0; i < 25; i += 2)
{

        if (i < 3 && j > 23)
        {

                if (i <= 30 && j <= 35 && i%2==0)
                {

                break;

                }

        i = i + 1;

        }
j += i * 2;
}

// int j=5
Mov eax,5


ForLoop:
        //int i=0
        Mov ebx,0

        // j<=25
        Cmp eax, 25
        JG AfterLoopCondition

        //if (i < 3 && j > 23)
        Cmp ebx,3
        Jge AfterIfCondition
        Cmp eax,23
        Jle AfterIfCondition

        //i = i + 1;
        INC ebx
        //j += i * 2;
        LEA eax, [eax+ebx*2]

        //i+=2
        LEA ebx,[exa+2]
AfterIfCondition:
        //j += i * 2;
        LEA eax,[eax+ebx*2]
        //i += 2
        LEA ebx,[ebx+2]
        JMP ForLoop
AfterLoopCondition:
```

5. Implement the following C code segment in IA-32 assembly language. Use the cdecl calling convention to implement (Note: treat func1 as both a caller and a callee)

```
int func1(int y)
{

int a = 5;
int b = functionOne(a);
int c = functionTwo(a);
int d = b + c + y;
return d;

}
```

```
Func1:
        push ebp
        mov ebp, esp

        MOV [EBP+8],[y]
```

```
//local variant a=5
sub ebp, 0x4
Mov [EBP-4],5


//int b = functionOne(a)
sub EBP, 0x4
Mov [ESP-8],[b]
//push a
Push [ESP-4]
Call functionOne


//int c = functionTwo(a);
sub EBP, 0x4
Mov [EBP-12],[c]
//push a
Push [ESP-4]
Call functionTwo

//int d = b + c + y;
sub EBP, 0x4
Mov [ESP-16],[d]
//d=b+c+y
LEA [ESP-16] ,[       [ESP-8]+[ESP-12]     ]
LEA [ESP-16] ,[       [ESP-16]+[ESP+8]]
MOV    [EBP+4],[ESP-16]


MOV esp, ebp
POP ebp
RET
```

6. Generate the equivalent C code for the following IA-32 assembly code.

```
Label1:

cmp [var1], 0x12
jle Label2
jmp Label6

Label2:

cmp [var2], 0x27
jg Label3
cmp [var3], 0x19
jg Label4

Label3:

cmp [var2], 0x10
jle Label5
cmp [var3], 0x16
jg Label4
jmp Label5

Label4:
```

```
; CodeBlock

Label5:

mul [var3], 0x06
add [var2], 0x02
inc [var1]
jmp Label1

Label6:


Void label1(){
        If( var1<= 18) label2();
        else label6();
}

Void label2(){
        If (var2 >39) label3();
        elseIf (var3 >25) Label4

}

Void label3(){
        If (var2<= 16) label5();
        elseif(var3 >22) label4();
        Else: label5;



}
Void label4(){
        codeblock;
}

Void Label5(){
        var3= var3*6;
        var2+=2;
        var1+=1;
        label1();
}



Void Label6(){
}
```

7. Generate the assembly and also evaluate the output of the following program.

NOTE: Please make sure that you don't have leading whitespaces or comma(",") as a seperator for the output. Write down the exact output as your answer

Assume CDECL calling convention and the order of evaluation of arguments from right to left.

```
int main()
{

int c = 0;
printf("%d\n",c);
c++;
printf("%d\n",c);
++c;
printf("%d\n",c);
printf("%d %d %d %d\n", ++c,++c,c++,c++);
printf("%d ", c++);

return 0;

}
```

```
Push ebp
mov ebp, esp
//int c = 0;
Sub esp,0x4
MOV eax,0

//printf("%d\n",c);
Push eax
Call printf
```

```
//c++;
Inc eax

//printf("%d\n",c);
Push eax
Call printf
//++c;
Inc eax
Push eax



//printf("%d\n",c);
Call printf
Push eax

//printf("%d %d %d %d\n", ++c,++c,c++,c++);
Push eax
INC eax
Push eax
Inc eax
Inc eax
Inc eax
Push eax
Push eax
Call prinft

//printf("%d ", c++);

Push eax
Inc eax
Call prinft

//set the return value to the return address
Mov [ebp+4],0

mov esp, ebp
pop ebp
Ret
```

Output:

0
1
2
6 6 3 2
6