Problem 1 (15 points) Consider the following high level code. Assume the array temp is initialized before it is used, and that register X20 holds the base address of temp. Convert the code to LEGv8. There is no need to check if the index of the array is out of bounds.

```
for (i =0; i < 20; i++)
    temp[i] = temp[i] + i;
```

Problem 1

```
ADD X19, XZR, XZR          // X19 will represent i =0
Loop:  SUBI X9, X19, #20   // i - 20
       CBZ X9, Break       // checking if i = 20 (loop condition)
       LSL X10, X19, #3    // X10 is i*8
       ADD X10, X10, X20   // X10 hold  address of temp[i]
       LDUR X11, [X10, #0] //  X11 holds value of temp[i]
       ADD X12, X11, X19   // X12 holds value of temp[i] + i
       STUR X12, [X10, #0] // sets temp[i] value to X12 value
       ADDI X19, X19, #1   // increments i++
       B Loop              // loop until break
Break:
```

Problem 2 (15 points) For the following C statement, write the corresponding LEGv8 assembly code. Assume that the variables i, and j are assigned to registers X19 and X20, respectively. Assume that the base addresses of the arrays A and B are in registers X21 and X22, respectively.

```
B[6] = A[i + j];
```

Problem 2

```
ADD X12, X19, X20          // X12 holds value i+j
LSL X12, X12, #3           // X12 is (i+j)*8
ADD X12, X12, X21          //  X12 Holds address of A[i+j]
LDUR X13, [X12, #0]        // X13 holds value of A[i+j]
STUR X13, [X22, #48]       // sets B[6] = value of A[i+j]
```

Problem 3 (20 points) Convert the following code into LEGv8 assembly. Assume a and b are signed numbers and have been placed in registers X19, and X20 respectively:

```
if (a < b)
    a = b - a;
else
    a = 2*b;
```

Problem 3

```
        SUBS X19, X20          // uses subtract to make comparison
        B.LT Less              // check a < b
        ADD X19, X20, X20      // else : a = b + b
Less:
        SUB X19, X20, X19      // a = b - a
```

"Pb3. (-3) Missing SUBS destination register.

Problem 4 (15 points) Assume X0 holds the value 00010 1000. What is the value of X1 after the following instructions? **Explain** your answer.

```
        CMP X0, #2
        B.GE ELSE
        B DONE
ELSE:   LSR X1, X0, #2
DONE:
```

Problem 4

The instructions first compares the value of X0 (int value 40) to the integer 2. If X0 is greater-than or equal-to (signed) than 2. It is so we jump to ELSE where X0 is then Right shifted by 2 and stored in X1. Shifting 000101000 by 2 right gives us…. 000001010 which is the decimal value 10.

X1 = 000101000 >>2 =000001010 = decimal 10

Problem 5 (15 points) Consider the following LEGv8 loop:

```
        ADDI X10, XZR, #1
        ADDI X11, X10, #0
LOOP:   SUBIS X12, X11, #5
        B.GE DONE
        ADDI X11, X11, #1
        LSL X10, X10, #1
        B LOOP
DONE:
```

What is the final value in register X10? **Explain** your answer.

Problem 5

The final value in register X10 is 16 (binary 10000).. This is because both X10 and X11 start out at value 1, the loop subtracts and conditionally branches (comparing the two values).. if X11 >= 5 we exit. Before we can exit, the loop executes 4 times fully… shifting X10 left 1 on each iteration. X10's shift was as follows….
initial 00000001
end of 1st loop 00000010
end of 2nd loop 00000100
end of 3rd loop 00001000
end of 4th loop 00010000 = 16
The loop stopped because after SUBIS of the 5th iteration, X11 was equal to 5 so we were Done

Problem 6 (20 points) Consider the following high level code and convert it to LEGv8 assembly. All variables declared in a function are local. Assume that execution will start from the first line of your code and will end when caller() returns.

```
int caller(void) {
    int x = 2;
    int y = 3;
    int z;
    z = addition(x, y);
    return z;
}

int addition(int a, int b) {
    return (a +b);
```

Problem 6

```
        ADDI X0, XZR, #2    // X0 holds value x = 2
        ADDI X1, XZR, #3    // X1 holds value y = 3
        ADDI X2, X1, X0
        B return
return:
```

Pb6. You were expected to define a proper, re-usable procedure, pass arguments to it and use the link register. To finish this task, you will also need to create a stack by changing SP for saving/restoring LR. Partial credit for initializing two arguments and adding them together (+3). "

(I couldn't tell if I was either way over complicating this or did not fully understand the question)