Suppose R# is an 8-bit register. The data stored in R# is treated as an unsigned binary number. We want to calculate the following for given input R_1 (% is the MOD operation):

$$R_2 = 2 \cdot R_1 \text{ if } R_1 \leq 127$$
$$R_2 = 2 \cdot (R_1 - 128) \text{ if } R_1 \geq 128$$

R_1 is stored in register R1 (input) and R_2 is the result that will be stored in register R2. The above function can be implemented by two assembly instructions with two constants C1 and C2 as follows:

☐	R2, R1, #C1
☐	R2, R2, #C2

Find the instructions and the constants. Notice that the instructions must be the ones shown in the instruction page.

Suppose R# is an 8-bit register. The data stored in R# is treated as an unsigned binary number. We want to calculate the following for given input R_1 (% is the MOD operation):

$$R_2 = 85 - R_1 + 2 * [\{(R_1\%256) + (R_1\%64) + (R_1\%16) + (R_1\%4)\} - \{(R_1\%128) + (R_1\%32) + (R_1\%8) + (R_1\%2)\}]$$

R_1 is stored in register R1 (input) and R_2 is the result that will be stored in register R2. The above function can be implemented by one assembly instruction with a constant C as follows:

[ ] R2, R1, #C1

Find the instruction and the constant. (Don’t care about overflows.)

Now, let’s use the 32-bit ARM architecture, i.e., R# is a 32-bit register and int is a 32-bit signed integer. How will the main memory look like after the following code is executed? Draw a figure for the main memory.

```
int x[8];

for ( int i = 0 ; i < 8 ; i++ )
  x[i] = i;
```

Use the 32-bit ARM architecture. Write an assembly code for the following C code. The starting address of array x is 0x5000.

```
int x[8];

for ( int i = 0 ; i < 8 ; i++ )
  x[i] = i;
```

Use the 32-bit ARM architecture.

```
int x[4][8];
```

The address of x[0][0], i.e., &(x[0][0]), is 0x6000.

What is the address of x[1][2]?

What is the address of x[3][5]?

```
int x[8];

for ( int i = 0 ; i < 8 ; i++ )
  x[i] = i;
```