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 $R_1 = x_7 x_6 \ldots x_0$. If $R_1 \leq 127$, x_7 is 0, so R_2 is just $2 \cdot R_1$.
However, if $R_1 \geq 128$ (i.e., $x_7 = 1$), we should subtract 128 from R_1, which is setting x_7 to 0. Thus, x_7 should always be set to zero => masking. Thus, the first instruction is

AND R2, R1, #0x7F.

Now, we multiply it by 2. It can be "MUL R2, R2, #2" or "LSL R2, R2, #1".

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.)

Suppose $R_1 = x_7x_6 \ldots x_0$. 85 is 01010101 in binary form.
$(R_1\%256)$ is just $x_7x_6 \ldots x_0$. $(R_1\%64)$ is $00x_5x_4 \ldots x_0$. $(R_1\%16)$ is $0000x_3x_2x_1x_0$.
Thus, $[\{(R_1\%256)+\ldots+(R_1\%4)\} - \{(R_1\%128) + \cdots + (R_1\%2)\}]$ is $x_70x_50x_30x_10$.

$$R_2 + R_1 = 2 * (x_70x_50x_30x_10) + 01010101$$

Suppose $R_2 = y_7y_6 \ldots y_0$. Then,

$$\begin{array}{r} y_7y_6y_5y_4y_3y_2y_1y_0 \\ +\ x_7x_6x_5x_4x_3x_2x_1x_0 \\ \hline x_7\ 0\ x_5\ 0\ x_3\ 0\ x_1\ 0 \\ +\ x_7\ 1\ x_5\ 1\ x_3\ 1\ x_1\ 1 \end{array} \quad \Rightarrow \quad \begin{array}{r} y_7y_6y_5y_4y_3y_2y_1y_0 \\ +\ 0\ x_6\ 0\ x_4\ 0\ x_2\ 0\ x_0 \\ \hline x_7\ 1\ x_5\ 1\ x_3\ 1\ x_1\ 1 \end{array}$$

Thus, $R_2 = x_7\overline{x_6}x_5\overline{x_4}x_3\overline{x_2}x_1\overline{x_0}$.

Answer: EOR (XOR) R2, R1, #85 // 85 is 01010101

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;
```

I just randomly chose 0x6000 for the starting address. It doesn't matter what number you choose for that, but it should be an integer multiple of 4.

Address	Data (31 … 0)	
0x6024	...	
0x6020	...	
0x601C	7	x[7]
0x6018	6	
0x6014	5	
0x6010	4	x[4]
0x600C	3	
0x6008	2	
0x6004	1	
0x6000	0	x[0]
...	...	

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;
```

```
LDR R1, =#0x5000
MOV R2, #0  // i

loop:
  STR R2, [R1]  // x[i] = i
  ADD R2, R2, #1  // i++
  ADD R1, R1, #4  // the address for the next x[i]. It is increased by 4 (bytes).

  CMP R2, #8
  BNE loop  // if (i < 8), go back to the loop

  // done
```

* Notice that we cannot use something like STR R2, [R1, R3] for the offset. The instruction format is STR R#, [R#, #imm] where #imm is a constant. Thus, I am increasing R1 itself to access the next element.

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]?
The address of x[1][0] is 0x6000 + 8*(4 bytes) = 0x6000 + 32 = 0x6020.
Thus, the address of x[1][2] is 0x6020 + 2*(4 bytes) = 0x6028.

What is the address of x[3][5]?
The address of x[3][0] is 0x6000 + 8*(4 bytes)*3 = 0x6000 + 96 = 0x6060.
Thus, the address of x[3][5] is 0x6060 + 5*(4 bytes) = 0x6074.