# Ense 352 – Fall 2016 – Assign 4

Handed Out: 2016-10-11
Due: 2016-10-18 by 23h55

1. (Similar to CS:APP/2e 2.69) **(5 marks)**

   Write code for an assembly language function named `rot_left` which does a rotating shift of a given integer (provided in `R6`) by an amount given in `R12`. The result should be returned in `R4`.

   Your function should assume that the amount to rotate is between 0 and 31 inclusive. You are *not* allowed to use the rotation assembly language instructions: neither `ROR` nor `RRX`.

   For example, if you call your function with `R6=0x12345678` and the rotate amount `R12=4`, the result should be rotated left by 4 bits: `R4=0x23456781`.

   Call your function from your mainline code, providing various parameters and checking the result for correctness.

2. (Not taken from the text) **(10 marks)** Write a function in assembly language named `encrypt` which takes a pointer to a buffer containing some cleartext, and modifies the buffer to contain the encrypted version of that text.

   The encryption scheme is to add a (wrap-around) displacement $n$ to each alphabetic char where the letters of the alphabet are numbered `A`=1, `B=2`, etc. The same encoding is used for lower-case letters: `a=1`, `b=2`, etc. For example given the clear text

   ```
   AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
   0123456789
   Hello, my boots are layered like an onion.
   !@#$%^&*()_-+={}[]\|
   <>,.?/~‘;:"’
   ```

   and assuming for example $n = 13$, the result should be

   ```
   NnOoPpQqRrSsTtUuVvWwXxYyZzAaBbCcDdEeFfGgHhIiJjKkLlMm
   0123456789
   ```

```
Uryyb, zl obbgf ner ynlrerq yvxr na bavba.
!@#$%^&*()_-+={}[]\|
<>,.?/~‘;:"’
```

Note that only the alphabetic characters have changed, case has been preserved, and each alphabetic character has been displaced by 13, for instance a `Z` being character 26 becomes `M` (character 13), and a `b` (character 2) becomes `o` (character 15).

Your code should accept any $n$ from 0 to 25. Of course 0 means no change.

The buffer pointer must be stored in `R3` and the size of the buffer must be stored in the most significant 27 bits of `R4`, while the value of $n$ must be stored in the lower 5 bits of `R4`.

3. (**5 marks)** Write the complementary function `decrypt` which takes exactly the same parameters as `encrypt` in the same format, and *undoes* what `encrypt` did. Your `decrypt` should expect the *same* value of $n$ to be passed as was used originally in `encrypt`. So if you called `encrypt` with $n = 6$ you must call `decrypt` also with $n = 6$ to decrypt that message. You should call `encrypt` as part of this process.

Submit your solutions via urcourses in a single zipfile. Place the solution to the questions in subfolders named `q1`, `q2`, etc. Your code should be written in standard cortex-m3 assembly language.