MIPS assembly code challenge:

You are working as the team manager for a computer security company and your team has identified a possible threat: a piece of code generating a strange sequence of numbers. They all appear to be negative and have little obvious meaning. Your team has reported that the numbers are actually a sophisticated combination of binary operations and a Vigenère cipher. After running a code decompiler, they found this in the source code:

```
char[] s = "".ToCharArray(); //Cipher string did not decompile

int[] ints = new int[s.Length];

string key = "hack";
int encrypt = 0;

for (int i = 0; i < s.Length; i++)
{
    s[i] = (char)(key[encrypt] + s[i]);
    encrypt = (encrypt + 1) % 4;
}

for(int i = 0; I < s.Length; i++)
{
    ints[i] = (~(s[i]) + 1) << 1;
}
```

As you may notice, the decompiler did not identify the encoded string. Fortunately, we have the algorithm used to encrypt the string and the bitwise obfuscation attempts. The string was obfuscated in the following way:

1. The string was encrypted with a simple Vigenère that added the current encryption key value as a character to each element of the string. It cycles through the key word "hack" character by character, and resets back to 'h' after reaching 'k'.
2. Each element of the string was converted to an integer using the two's complement of its character value, and then was bit-shifted left by one. This is identifiable by the NOT (~) then subsequent addition of one performed, and followed by a left bit-shift by one.

Your task is to decode and print the string using MIPS assembly code. You should remember the order of operations involved here. Because the characters were first encrypted, then two's complemented, and lastly bit-shifted you do **not** want to start with undoing the Vigenère. Instead, work backwards to undo each

operation. Think about the logical ways to undo a left bit-shift, and ***especially*** think about how to undo two's complement (no multiplication allowed!).

The template for this challenge is attached here.

In the template, the suspicious code output is in a 24-byte long byte array. Some useful boilerplate code is already provided for you. You must use the decode table in order to decode the Vigenère. Here is an example of a way in which you can access the data stored there:

Method 1:

```
li   $t0, 0                  #first element
lbu  $t1, decode_table($t0)  #get element in table
```

Method 2:

```
li   $t0, 0                  #first element
la   $t1, decode_table       #get address of table
addu $t2, $t1, $t0           #sum table address w. element desired
lbu  $t3, 0($t2)             #get element in table
```