Assignment 3 — Assembly Languages

Due: Wednesday, April 27th

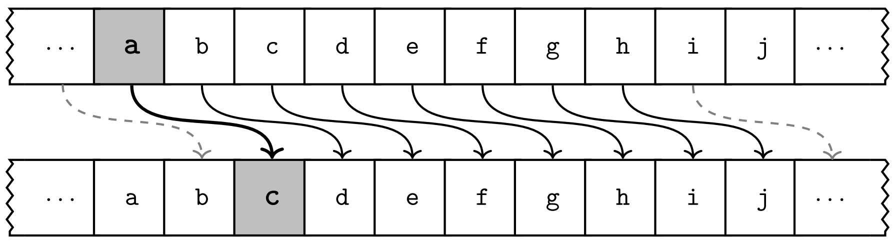
Assembly languages allow programs to be written in a format that is more human-readable than bare ‘`0`’s and ‘`1`’s, yet is still essentially a direct translation of machine code. Not only are the instructions themselves represented symbolically, but macros can be provided for common operations such as input or output.

Deliverables:

GitHub Classroom:	`https://classroom.github.com/a/K0uzu4o2`
Required Files:	`encrypt.asm`, `decrypt.asm`
Optional Files:	*none*

Part 1: Caesar Ciphers

A *Caesar cipher*[1] encrypts a string by employing a simple substitution: each character is shifted by some fixed integer, the *key*. For example, given a key of 2:



...encrypting a character ‘`a`’ results in the character ‘`c`’.

Part 2: Encrypting Strings

Complete the assembly program in `encrypt.asm`, most of which has already been written for you:

- Your program’s assembly code must begin at memory location `0x3000`.
- Your program must prompt the user to type both a key and an unencrypted string, then print the corresponding encrypted string.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

You may assume that the key will be an integer between 0 and 9, inclusive, that the string will be at most 32 characters long, and that the user will hit the ‘Enter’ key after both the key and the string. You may further assume that all input will consist of valid, printable ASCII characters.

For example:

```
Encryption key (0-9): 2
Unencrypted string: Veni, vidi, vici
Encrypted string: Xgpk."xkfk."xkek
```

Your program will be tested using `diff`, so its printed output must match *exactly*.

[1]Although this cipher is rudimentary by modern standards and easily broken by computers, it was likely effective against Julius Caesar’s enemies — most of them were illiterate.

To help you get started, note the following caveats:

- When reading and writing characters and strings, remember that `GETC`, `OUT`, and `PUTS` all assume that the relevant information is in `R0`. Organize your program's data accordingly.
- When a character is typed, it will not automatically be echoed back to the terminal[2]. Your program must do this, so that the user can see what they typed.
- When the user strikes a key, they are typing a *character*. For example, if they type '3', it will appear as the ASCII code `0x33`; your program must convert this character into an integer[3]. Similarly, when they strike the 'Enter' key (which should not be considered part of the input string), its ASCII code is `0x0A`.
- When encrypting a character, it is possible that the resulting sum is < `0x20` or > `0x7E`. The character is then *unprintable*. In such cases, your program must output '`?`' in place of the unprintable character.

Part 3: Decrypting Strings

Complete the assembly program in `decrypt.asm`:

- Your program's assembly code must begin at memory location `0x3000`.
- Your program must prompt the user to type an encrypted string, then print all possible corresponding unencrypted strings. Note that decryption involves *subtracting* the encryption key.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

You may assume that the string will be at most 32 characters long and that the user will hit the 'Enter' key after the string. You may further assume that all input will consist of valid, printable ASCII characters.

For example:

```
Encrypted string: Xgpk."xkfk."xkek
Decryption key 0: Xgpk."xkfk."xkek
Decryption key 1: Wfoj-!wjej-!wjdj
Decryption key 2: Veni, vidi, vici
Decryption key 3: Udmh+?uhch+?uhbh
Decryption key 4: Tclg*?tgbg*?tgag
Decryption key 5: Sbkf)?sfaf)?sf`f
Decryption key 6: Raje(?re`e(?re_e
Decryption key 7: Q`id'?qd_d'?qd^d
Decryption key 8: P_hc&?pc^c&?pc]c
Decryption key 9: O^gb%?ob]b%?ob\b
```

Your program will be tested using `diff`, so its printed output must match *exactly*.

Part 4: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `encrypt.asm` — A working assembly program for encrypting strings, as specified.
- `decrypt.asm` — A working assembly program for decrypting strings, as specified.

The following files are optional:

- *none*

Any files other than these will be ignored.

[2]On a real computer, some combination of the shell, the operating system, or the terminal itself would handle this.
[3]Note that all of the Hindu-Arabic numerals have consecutive ASCII codes.