Lab 1: Unfold the Secret

Brief

> The magic of LC-3 grows.

Welcome to the **LABS** of ICS! As this is the first lab of the entire series, you might be wondering, "How do I complete theses labs?", or even, "What do these labs do?". Don't worry, there are no dumb questions —— the documents will tell you everything you need.

To not scare you out, I'll put it in the beginning: **This lab, lab 1, is really an easy cake.** And you know, Lab 1 is probably the easiest one of the entire series. At everywhere else, there will be assembly, bugs, and failures. You really won't want to get trapped by any, so get geared up, and take a look at this very first lab.

Intro

Even the most complex project in the world was born from a simple idea. Suppose you're exchanging some message with your pal, while fearing that others might be able to access your super secret code, you've decided to encrypt it. The encryption works very well, but you also need to design a **decryption program** for your friend. There is no time to get your laptop or PC. The only tool that you have access to is the LC-3 machine. Therefore, you've arrived at the LC-3 laboratory, where we are now.

Tasks

Create a program to **decrypt** the secret number:

- The secret number will be placed in the register `R0`.
- To decode the number, you need a **secret key**. Create your secret key as follows:

  1. Use your student ID, remove the letters. (`PB12345678` becomes `12345678`)
  2. Convert even digits to `0` and odd ones to `1`. (`12345678` becomes `10101010`)
  3. This is the **binary** form of your secret key. Convert it to decimal or hexadecimal. (`10101010` becomes `#170` or `xaa`)
  4. Remember it or write it down as we'll use it later.

- Performing a **bitwise XOR** operation on the number and your secret:

  ```
  Output = Secret ^ R0
  ```

- Put your output into register `R3`.

Examples

Suppose your student ID is `PB12345678`, and the input is (in `R0`):

```
x00c2
```

The decrypted number will be (in `R3`):

```
x0068
```

Because:

```
x0068 = x00c2 ^ xaa
```

Requirements

- The program should be created using **machine code** and coded in **text form**. (Not assembly!)
  - Instead of typing `AND R0, R0, x0`, use `0101000000100000`.
  - Make sure to add line breaks for each instruction.
  - It's sufficient to complete the program using Notepad, TextEdit or Vim, but you can pick any editor you like.
  - The final code of your program should look like below (content may differ):

    ```
    1010110000010000
    0011111101011100
    0111101011111101
    (And more...)
    ```

- The code is loaded at `x3000`.
  - That means the first instruction of your program will be placed at `x3000`, the second at `x3001`, and so on.
  - This is done automatically when loading the program. No manual operations needed.
  - This should not affect your code now, but it's important to know this for future labs.
- After you've completed your program, make sure to **add two extra lines**:
  - `0011000000000000` at the **beginning**.
    - This is a convention. It tells everyone that this program begins at `x3000`.
  - `1111000000100101` at the **end**.
    - This means `HALT` which stops the machine. Similar to `return 0;` in C.

  Or your code won't run correctly.
- Please follow academic ethics and morals. **Do not pirate code that does not belong to you.**

How To

Set the Secret

Suppose your secret is `10101010` (`xaa` in hexadecimal), you may find it really tempting to write:

```
AND R2, R2, x0
ADD R2, R2, xaa
```

Well, this will not work, as `xaa` is too large to fit into `imm5` (used by `ADD`).

There are several ways to deal with it, but most of them require memory accessing. I know you'll yell out "Oh, no memory please!". And, yes, there **do exists** a solution using only the `ADD` instruction.

First put 4 bits into the register, shift them left by 4 bits, then add the register by the last 4 bits.

```
AND R2, R2, x0
ADD R2, R2, xa
ADD R2, R2, R2
ADD R2, R2, R2
ADD R2, R2, R2
ADD R2, R2, R2
ADD R2, R2, xa
```

`ADD` something by it self doubles it. After adding `R2` by itself for 4 times, we have `10100000` in `R2`. Then we add `xa` into `R2` to make it become `10101010`. Done!

In future labs you'll learn how to use `LD` to load values into registers.

Test and Run

There are numbers of tools which runs LC-3 assembly, but few of them **runs LC-3 machine code**. However, I believe 99% of you have created your program in assembly before translating them to binaries. By testing the assembly code, we can verify that our algorithm works. The only thing left after which is to translate the instructions correctly.

For LC-3 assembly, it's highly recommended to use **LC3Tools** for testing. LC3Tools embeds features like code editing, syntax highlighting, debugging and memory inspecting. It's cross-platform, portable and also super powerful! If you're still new to LC3Tools, just make sure to consult your TAs to know how to use it! Here's also a link to get the latest release: https://github.com/chiragsakhuja/lc3tools/releases/latest

Tips and Tricks

- As mentioned above, try to **use assembly first** to make life easier. Just **make sure to translate it to machine code** correctly.
- `AND` something with `0` clears it.
- Although not enforced, it's possible to complete the program with no more than 20 lines. If you're finding your program being super long, **consider using a better approach**.
- It's also possible to complete the program using only `AND`, `ADD` and `NOT` instructions, but if you're finding other instructions being handy, feel free to use them.

Lab Report

Completed your program? Congratulations I must say, but that's only half way to success. To get a full mark (hopefully!), you'll need to compose a report summarizing your work.

- **Make sure to explain your code** in your report. This is essential to show your thoughts and to not get misjudged when your code happens to be similar to others.
- Contents that are recommended to include:
  - Name and date
  - The core algorithm
  - Critical code to implement the algorithm
  - Debug process (if any)
  - Run results
  - Traps and pitfalls you've found (if any)
  - Suggestions (if any)
- It's recommended to use Markdown to compose your report. **Markdown fits very well for our labs, while also being super easy to use!** If you haven't picked it up already, make sure to check out these helpful links:
  - Learn markdown: https://learnxinyminutes.com/docs/markdown or https://www.runoob.com/markdown/md-tutorial.html
  - MarkText, a markdown editor app: https://www.marktext.cc
  - StackEdit, an online markdown editor: https://stackedit.io/app
- Please export your report as a PDF file. Here's how to:
  - **(Recommended) Markdown: Use MarkText to print Markdown files as PDF files.**
  - Word Documents: Both Microsoft Word and LibreOffice Writer can export these documents as PDF files.
  - LaTeX: Use a LaTeX compiler (e.g. XeLaTeX) to create a PDF document.
  - Notability: Share your document as PDF, then save it to local files for uploading.
  - (Not recommended) Handwriting: Office Lens can help you to scan your work.

The above content should also be helpful for future labs.

Submission

> **Note:** This section varies according to your TAs. **Make sure to consult them before submitting!**

Grab your report (say `report.pdf`) and your code (`lab1.txt`):

1. Rename the files. Your TAs may have different requirements but generally:
    - `StudentID_Name.pdf` for the report. (e.g. `PB12345678_JohnDoe.pdf`)
    - `lab1.txt` for the code.
2. Create an archive (usually ZIP) and put both files in.
3. Name the archive according to your TAs. (Usually `StudentID_Name.zip`)
4. Upload the archive to the desired location. (Usually BlackBoard)