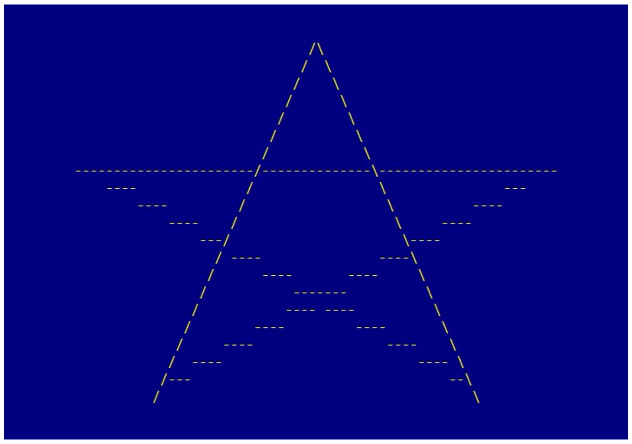
ECE 120: Introduction to Computing **Spring 2019 ZJUI**
due by 11:59:59 p.m. on Saturday 25 May 2019

Lab 6: Decrypting a Secret Image with LC-3



Your task in this lab is to write a program in LC-3 assembly language to decrypt a secret image stored in memory, then dump the decrypted image to the screen for viewing. The data in the image have been encrypted using a simple, reversible numeric function based on the X and Y coordinates of each location within the 79×25 screen (79 columns, 25 rows). The program requires about 70 instructions and some data, but you may need more.

The objective for this lab is for you to gain some experience with assembly language programming, to implement a few simple numeric operations in LC-3 assembly, and to understand how two-dimensional data are mapped into memory. Towards that end, your program must be written in LC-3 assembly language and converted into executable form using the **`lc3as`** tool. You may wish to read and perform some of the exercises in the **Lab 6 Preparation Guide** in order to prepare you for the lab, but these steps are not required and will not be graded.

The Task

Start by updating your Subversion repository to obtain the **`lab6`** subdirectory, which contains all necessary files except for your program, which you must write in the file **`lab6.asm`**, add to the subdirectory, and commit yourself.

The screen is stored in 1975 consecutive memory locations starting at x5000. The first memory location represents the upper left corner of the screen, which we call (0,0) in (X,Y) coordinates. X coordinates get larger to the right, and Y coordinates get larger in the downward direction (the standard convention for screen data, and the opposite convention from mathematics). The first row, with Y coordinate 0 and X coordinates 0 through 78, are stored in X order in the first 79 memory locations. The second row, with Y coordinate 1 and X coordinates 0 through 78, are stored in the next 79 memory locations. Thus, to obtain the memory address for a screen location (X,Y), one calculates the expression x5000 + 79 Y + X. Your program will only need to access the 1975 locations in order, so you will not need to compute the value of the expression, but you do need to understand the mapping.

When your program starts, each memory location corresponding to a point (X,Y) in the screen data will contain an encrypted value T, an 8-bit unsigned value zero-extended to 16 bits. Given X, Y, and T, you must use the following equations to decrypt the data into an ASCII character V, which you should store back into the same memory location (in the screen data):

$$\mathbf{S = T\ XOR\ X}$$

$$\mathbf{Q = (X + 2Y)\ \%\ 10 \quad + 7}$$

$$\mathbf{V = (S\ AND\ x000F) + (S >> 4)\ Q}$$

where A % B represents the remainder of A when divided by B (X + 2Y is always non-negative) and A >> B represents the logical right shift of A by B bits.

After translating the screen data, your program must output the screen data to the display using the OUT trap. Since the screen data are stored in row-major order (the ordered described above, with the contents of each row in stored in consecutive memory locations), your program can iterate over all screen locations, but you must keep track of the X coordinate so as to output an ASCII line feed (x0A) character at the end of each row.

Specifics

- Your code must be written in LC-3 assembly language and must be contained in a single file called `lab6.asm`. We **will not grade** files with any other name.
- Your program must translate the screen data in memory locations x5000 through x57B6 using the equations given earlier.
- Your program must draw each line of the translated screen data to the monitor using the LC-3 OUT trap.
- Your program must output a single ASCII line feed (x0A) character after each row of screen data.
- The screen-printing portion of your code must begin with the label **`PSCR`**. We will use this label to test that part of your program—if it is missing, you will lose all points for that section.
- You may assume that each encrypted screen value (T values) is between 0 and 255 (8 bits).
- You may use any registers for this program, but we suggest that you avoid using R7.
- Your code must be well-commented:
    - at the top of your program, you must include a paragraph explaining the purpose of your program and how you approach the problem (**in your own words**);
    - below the description, you must include a table describing how each register is used (only the registers that you use)—note that you may need separate register tables for the decryption and screen-drawing parts of your code; and
    - your code should be thoroughly commented (not necessarily every line, but close—for example, you need only write one comment for the two instructions needed to negate a register value).
    - Follow the style of examples provided to you in class and in the textbook.
- Do not forget to include a HALT (TRAP x25) at the end of your program.

Step 1: Develop and Test the Screen Drawing Code

We suggest that you first write the code to draw the screen data to the display. Put the label `PSCR` at the start of your code—our tests use this label in your final program.

Once you have written and assembled the code to draw the screen data, use the screen.hex image included in the `lab6` subdirectory as a test. First, convert the file to an LC-3 object file using the command

```
lc3convert -b16 screen.hex
```

The "`-b16`" argument tells the conversion tool that the input file is in hexadecimal rather than binary.

When you start `lc3sim`, first load the `screen.obj` file, then load your `lab6.obj` file. If you load the two files in the wrong order, the LC-3's PC register will point to the screen data. Note that if you reset the LC-3, the simulator reloads only the last file.

Debug your screen drawing code before moving on to write the decryption code! Step through the instructions using the simulator and make sure that it is doing what you expect. If you let the code execute, you should see the image on the first page of this specification.

When you are fairly sure that your code works, you can use the input script `testdraw` to check. Run the simulator with the script as input:

```
lc3sim -s testdraw > mydraw
```

Be sure that you have both object files in your directory before running the script. Note that your code must terminate properly, or the simulator will simply keep running. You can then compare your program's output with the correct output using the command

```
diff mydraw golddraw
```

The final register values should be different, and the value of `PSCR` will be different ("`Setting PC to` …"), but the two files should otherwise be identical (no further output).

Step 2: Develop and Test the Decryption Code

Now you can write the code to calculate the decrypted value V for each encrypted T value. You may run out of registers, in which case you must store some information in memory. Note that you have already implemented several of the necessary operations in homework and in Lab 5.

Once you have written and assembled your program, use the encrypted image `cryptic.hex` included in the `lab6` subdirectory as a test. As before, load the `cryptic.obj` file into the simulator before loading your `lab6.obj` file. And remember that if you reset the LC-3, the simulator reloads only the last file.

You may want to calculate the first few screen values by hand in order to make sure that your program is producing the correct values. You may also want to check the first few values in the second and/or third rows (with non-zero Y values). You can step through your code or set breakpoints at appropriate places to validate your calculations.

Once you are fairly certain that your code works, let it continue to execute and you should see the decrypted image. To verify it more thoroughly, use the input script **`testdecode`** to check. Your output should match the golddecode output other than for the final register values.

Submitting your Lab

Add your **`lab6.asm`** file to your Subversion **`lab6`** subdirectory, then commit it to hand in your final program. Do not add **`.obj`** nor **`.sym`** files to your repository. Note that we will use computer-based comparison tools, so please write your own program and **do not share code with any other student**.

Grading Rubric

Functionality (55%)

- 35% program correctly decrypts screen data
- 15% program draws all screen data to the display after decryption (for credit, program MUST include the label PSCR before this part of the code)
- 5% program outputs a single line feed after each row

Style (20%)

- 5% - program calculates remainder of division by 10 using an iteration rather than repeated conditionals
- 10% - program performs right shift using an iteration rather than one conditional per bit shifted
- 5% - program keeps track of X and Y using nested loops while walking in order through screen data to avoid calculating screen data offset from X and Y

Comments, Clarity, and Write-up (25%)

- 5% - program includes a paragraph explaining what it does and how it works (these are given to you; you just need to document your work **in your own words**)
- 10% - program includes a table of registers explaining how each is used in the program (two tables if uses are different between decryption and screen-drawing)
- 10% - code is clear and well-commented

Note that some categories in the rubric may depend on other categories and/or criteria, and that you must comment clearly in order to receive style points.