1 Purpose

Develop a password verification program for a hypothetical bank system using LC-3 assembly language.

- Each input must be preceded by a specific text prompt;
- It is able to determine whether the password entered is correct and output the result for whether it is correct or not;
- Only 3 chances to enter password, program should judge if the user can still enter password.

2 Principles

2.1 The output of the prompt text

- Using `.STRINGZ` to load the strings into memory. e.g. `WELCOME .STRINGZ "Welcome!"`
- Using `LEA` and `PUTS` instructions to output the strings. e.g. `LEA R0, LABLE` then `PUTS`.

2.2 Get input from the keyboard

- Using `TRAP x23` to read the input from keyboard and store it into `R0`.
- Store the content in `R0` into memory.

2.3 Check the input information

- To check if the input is "W" or "Y", we need to subtract the content in `R0` by the ASCII of "W" or "W", and if check the result is 0.
- To compare the input password with the correct password, we could using two registers to record pointers of their memory addresses, and compare the characters one by one using the same methd that is used to check "W" or "W".

2.4 Record the number of entries made

- Using a register to store the total number of input chances;
- Subtract it by 1 after every time we checked the input.

3 Procedure

Step (1) **Init**: Load `#3` into `R6`; set those prompt strings by using `.STRINGZ` instruction.

Step (2) **Start**: Output the "Welcome" prompt by using `LEA` and `PUTS`; using `TRAP x23` to fetch the input from keyboard.

Step (3) Check if the input is "W". If the input is not "W", do Step (3) again.

Step (4) **Input password**: Using `TRAP x23` to fetch the input from keyboard, check if the input is "Y", if it is, jump to Step (5), otherwise store the content in `R0` into memory and do Step (4) again.

Step (5) **Before compare**: Load the addresses of correct password and input password into two defferent registers; add a spare register by 10 which is the number of characters in correct password.

Step (6) **Compare** Using `LDR` instruction to fetch letters from memory based on the addresses stored in registers mentioned last step; subtract one by another and jump to Step (8) if the result is not 0; if the result is 0, `ADD` those addresses by 1 and subtract the register that stores 10 by 1; then check the content stored in this register, jump to Step (7) if the result is 0, otherwise do Step (6) again.

Step (7) **If success**: Output the "Success" prompt by using `LEA` and `PUTS`, jump to Step (10).

Step (8) **If incorrect**: Output the "Wrong input" prompt by using `LEA` and `PUTS`; subtract the register that stores the input chances by 1; check the result and jump to Step (4) if the result is not 0.

Step (9) **Fail**: Output the "Fail" prompt by using `LEA` and `PUTS`; jump to Step (1).

Step (10) **Stop**: `TRAP x25`.

4 Results

I've recorded a video of the program running and it will be submited with code and report. The video contains the following sections:

- The beginning of program: output the prompt.
- Input the incorrect password for 3 times and output the prompt.
- Input the correct password and output the prompt.

To ensure that my video can be seen, I will also upload it to my network disk.
Link: https://rec.ustc.edu.cn/share/6d8d3e30-a196-11ee-9932-8121ec0544a2

5 Discussion Questions

- Do you use function definition/call in your program, why or why not?

  No, I only used `BR` instruction. Because the logical structure of this program is basically sequential, there is no need for complex function calls and jumps.

- Do you use a recursive function in your program, why or why not? If not, will you use this trick when the stack mechanism is provided?

  No I didn't. The reason is the same as the previous one. And I won't use this trick when the stack mechanism is provided. I don't this program needs that.

- How do you store these preset prompts? If you use a recursive function, can you conclude how many parts should a typical program assembled?

I used `.STRINGZ` instruction directly.

- Assess the security of your program with potential vulnerability scenarios. For example, what if the user types a super long password to your program?

  In my program, if the length of input is greater than a certain threshold, the input is automatically stopped, and the results are compared directly, so there should be no errors due to a large number of inputs.

- Share challenges faced during development and how they were resolved.

  I got an error when converting ASCII code to decimal numbers when outputting the remaining attempts, because the program can't calculate the number 48 directly, so I changed it to the sum of 4 12s.