Lab05: Password Verification

1 Objective

Develop a password verification program for a hypothetical bank system using LC-3 assembly language. This program should validate user passwords during sensitive operations, like withdrawing funds, with a limit of three attempts.

2 Instructions

1. **Initial Prompt**: On starting, display `Welcome to the bank system! Type 'W' to withdraw some fund.` Wait for the user to input 'W'.
2. **Password Input**: Once 'W' is entered, prompt `Please input your password:`.
3. **Password Verification**:
   - The correct password is your student ID (format: `PB22XXXXXX`). After entering the password, type 'Y' to submit.
   - Users get three attempts to enter the correct password.
   - Display `Success!` for a correct password or `Incorrect password! X attempt(s) remain.` for an incorrect attempt, where `X` is the number of remaining attempts.
4. **Attempt Limit**: After three incorrect attempts, display `Fails.` and restart from step 1, which means the prompt `Welcome ...` will be output again and the user should call for a new job.
5. **Successful Entry**: On correct entry, the program should HALT immediately.

2.1 Programming Guidelines

- Begin with `.ORIG x3000` and end with `.END`.
- Always include a HALT instruction.
- Use uppercase for keywords and labels, e.g., `ADD`.
- Maintain clarity with spaces after commas.
- Prefix decimal constants with `#` and hexadecimal with `x`.
- Comment your code for clarity.

3 Report Requirements

Your report should include:

1. **Program Design**: Describe the principles of your program. Diagrams or automata preferred over code comments.
2. **Testing Evidence**: Provide screenshots or a video link demonstrating the program's functionality.

3.1 Discussion Questions

- Do you use function definition/call in your program, why or why not?
- 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?
- How do you store these preset prompts? If you use a recursive function, can you conclude how many parts should a typical program assembled?
- Assess the security of your program with potential vulnerability scenarios. For example, what if the user types a super long password to your program?
- Share challenges faced during development and how they were resolved.

```
      .ORIG  x3000
      ;      R0            ; input & output info
      LDI  R1, TABLE       ; string table
      AND  R2, R2, #0      ; as a counter for attempt times
      AND  R3, R3, #0      ; as a counter for matching chars
      AND  R4, R4, #0      ; Trash can

START
   LEA R0, WELCOME
   TRAP x22









LOOP  TRAP x23
       LD  R4, NEG_W
       ADD R4, R4, R0
       BRz  WITHDRAW    ; 入力 "W"
       LEA R0, TYPO     ; typo! retry!
       TRAP x22
       BRnzp LOOP
```

```
WITHDRAW
        LEA R0, IN_PROMPT
        TRAP x22
        AND R3 R3 #0
        ADD R3, R3, #10
        LDI R1, R1, TABLE
INPUTING
        TRAP x23
        LDR R0, R1, #0
        ADD R1, R1, #1
```

Load the string into a stack

input respectively untill "Y" appears

```
        LD R4, NEG_Y
        ADD R4, R4, R0
        BRz SUBMITTED
        ADD R3, R3, #-1
        BRnzp INPUTING
SUBMITTED
        ADD R3, R3, #0
        BRz RIGHTLENGTH
```

```
    BRnzp INCORRECT    ;错啦!
RIGHTLENGTH
    ADD R3, R3, #10

    ;逐一对比


INCORRECT
    LEA R0, WRONG
    TRAP x22
    ADD R2, R2, #-1
    BRz ENDALL
    BRp LOOP
    ADD R0, R2, #0
    TRAP x22
    LEA R0, LEFT
    TRAP x22
    BRnzp INPUTTING

ENDALL
    LEA R0, FAIL
    TRAP x22
    BRnzp START
```

```
WELCOME     .STRINGZ  "Welcome...fund."
NEG_W       .FILL   x ?           ; #-87
TYPO        .STRINGZ  "You didn't enter "W",
                              try again!"

FAIL        .STRINGZ  "Fails."
IN-PROMPT   .STRINGZ  "Please...password:"
~~PSWDLEN   .FILL   x000A         ; #10~~
NEG_Y       .FILL   x ?           ; #-89
WRONG       .STRINGZ  "Incorrect password!"
LEFT        .STRINGZ  " attempt(s) remain."
TABLE       .FILL   x6001
```