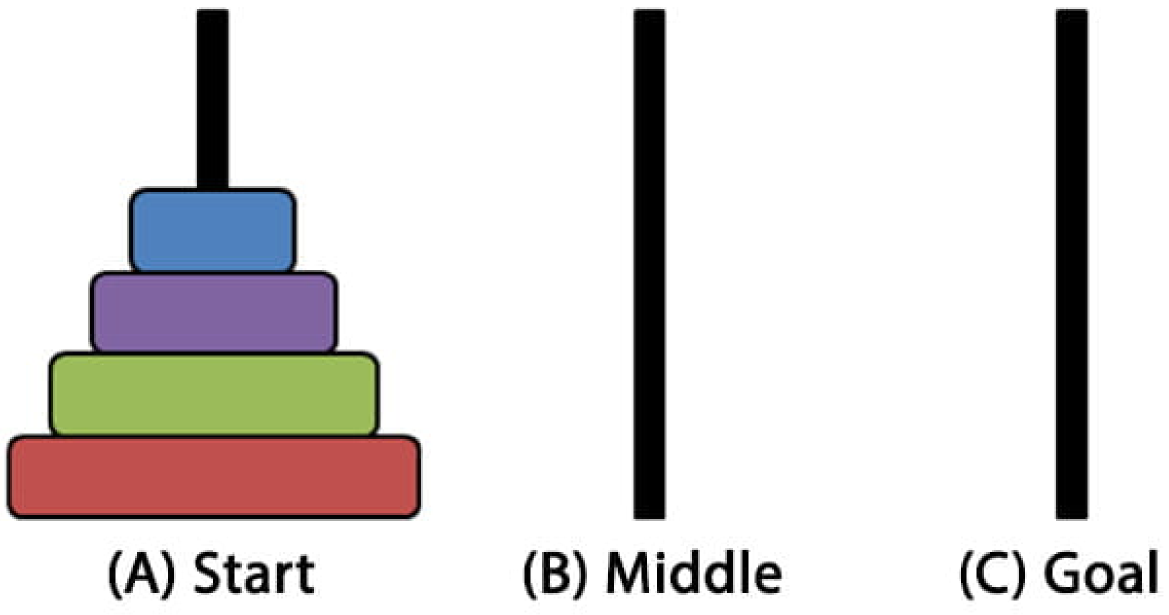
Lab 5：Tower of Honoi

Introduction:

The purpose of this assignment is to show how interrupt-driven Input/Output can interrupt a program that is running, execute the interrupt service routine, and return to the interrupted program, picking up exactly where it left off (just as if nothing had happened). In this assignment, we will use the Keyboard as the input device for interrupting the running program.

As the last LC3 assembly language programming assignment, it's also important to practice using recursion and subroutine in program. After receiving keyboard interrupt, you need to echo the input character and determine whether the keyboard input is a legal decimal number. If it is a legal decimal number, you need to store its value in memory and input it as the variable N of the Hanoi problem into a **recursive** function to solve the Hanoi problem.



The Tower of Hanoi is a mathematical game or puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the last rod, obeying the following rules:

> 1. Only one disk may be moved at a time.
> 2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
> 3. No disk may be placed on top of a disk that is smaller than it.

The key to solving a problem recursively is to recognize that it can be broken down into a collection of smaller sub-problems, to each of which that same general solving procedure that we are seeking applies, and the total solution is then found in some simple way from those sub-problems' solutions. Each of these created sub-problems being "smaller" guarantees that the base case(s) will eventually be reached. For the Towers of Hanoi:

> 1. label the pegs A, B, C,
> 2. let n be the total number of disks,
> 3. number the disks from 1 (smallest, topmost) to n (largest, bottom-most).

Assuming all n disks are distributed in valid arrangements among the pegs; assuming there are N top disks on a source peg, and all the rest of the disks are larger than m, so they can be safely ignored; to move m disks from a source peg to a target peg using a spare peg, without violating the rules:

> 1. Move n − 1 disks from the source to the spare peg, by the same general solving procedure. Rules are not violated, by assumption. This leaves the disk n as a top disk on the source peg.
> 2. Move the disk n from the source to the target peg, which is guaranteed to be a valid move, by the assumptions — a simple step.
> 3. Move the n − 1 disks that we have just placed on the spare, from the spare to the target peg by the same general solving procedure, so they are placed on top of the disk n without violating the rules.
> 4. The base case is to move 0 disks (in steps 1 and 3), that is, do nothing – which obviously doesn't violate the rules.

Now we can define the solution of n disks as H(n), according to the above rules:

$$H(n) = \begin{cases} 0 & ,n = 0 \\ 2H(n-1)+1 & ,n > 0 \end{cases}$$

Your mission:

1. Write the user program described below.
2. Write the keyboard interrupt service routine described below.

Details:

The assignment consists of three parts, **but you only need to do the first two parts**. You can finish this assignment by completing the attachment provided in part C.

A. The user program

The argument of tower of honoi, recorded as N , will be initialized with xFFFF and stored at X3FFF in memory.

Your user program, which starts at x3000, will continually (i.e. in an infinite loop) print your student id like: `PB12345678 PB12345678 PB12345678 PB12345678 PB12345678 PB12345678 ······` and wait for argument N to become a valid value. When N becomes a valid value, the program stops waiting and calls the HONOI subroutine to solve the problem. After HONOI subroutine solve the problem, your result should be displayed on the console in decimal format.

To ensure the output on the screen is not too fast to be seen by the naked eye, the user program should include a piece of code that will count down from 2500 (or any other numbers) after each word is output on the screen. A simple way to do this is with the following subroutine DELAY:

```
        ; code of delay
DELAY   ST R1, SaveR1
        LD R1, COUNT
REP     ADD R1, R1, #-1
        BRp REP
        LD R1, SaveR1
        RET
```

B. The keyboard interrupt service routine

The keyboard interrupt service routine, which starts at x1000, will examine the key typed to see if it is a decimal digit. If the character typed **IS NOT** a decimal digit, the interrupt service routine will, starting on a new line on the screen, print `"<the input character> is not a decimal digit."` The service routine would then print a line feed (x0A) to the screen, and finally terminate with an RTI.

If the character typed **IS** a decimal digit, the interrupt service routine will, starting on a new line on the screen, print " is a decimal digit.". Then you should save this decimal number to x3FFF.

For example, if the input key is 'K', the interrupt service routine will print: `K is not a decimal digit.` If the input key is ‘4’, the interrupt service routine will print: `4 is a decimal digit.` and change the value of x3FFF to 4.

The service routine would then print a line feed (x0A) to the screen, and finally terminate with an RTI.

Hint: Don't forget to save and restore any registers that you use in the interrupt service routine.

C. The operating system enabling code

Unfortunately, we have not installed Windows or Linux on the LC-3, so we provide you with STARTER CODE (in attachment) that enables interrupts. You MUST use the starter code for this assignment. The locations to write the user program and interrupt service routine are marked with comments.

The starter code does the following:

1. Initializes the interrupt vector table with the starting address of the interrupt service routine. The keyboard interrupt vector is x80 and the interrupt vector table begins at memory location x0100. The keyboard interrupt service routine begins at x1000. Therefore, we must initialize memory location x0180 with the value x1000.
2. Sets bit 14 of the KBSR to enable interrupts.
3. Pushes a PSR and PC to the system stack so that it can jump to the user program at x3000 using an RTI instruction.

Example:

```
PB12345678 PB12345678 PB12345678 PB12345678 PB12345678 PB12345678
h is not a decimal digit. //Input character 'h'
PB12345678 PB12345678 PB12345678 PB12345678 PB12345678 PB12345678
5 is a decimal digit. //Input character '5'
Tower of honoi needs 31 moves.



--- Halting the LC-3 ---
```

Notes and Suggestions:

1. Since the interrupt can be triggered at any point, the output of the interrupt service routine may show up anywhere.

2. Since your user program contains an infinite loop, you will have to press the "Pause" button in the simulator if you wish to stop the program.
3. Unlike previous labs, the PC will be initialized to x800 for this assignment because the first code that is executed will be in the operating system.
4. Please make sure that the "Ignore privileged mode" switch is ON. (Default configuration is OFF in LC-3 simulator)

Additional Requirements:

If you don’t comply with these requirements, the lab may be counted as an invalid work.

1. The report shall contain at least 3 parts: How do you work out the algorithm? How do you write the program? And how do you design your own test cases to ensure the program works fine?

2. Your submission be structured as shown below.

```
PB21******_Name.zip
├── PB21******_Name_report.pdf
└── lab5.asm
```