Lab 4: Flappy Bird

Professor Patt has played the game *Flappy Bird*, but was very annoyed because of the difficulty. As a computer man, he would like to make a slower version of this game. Your job is to help him to finish the basic part of this game and display it in LC3's console.

Implementation Details

- You are required to write in **LC-3 assembly language**.
- Your program should consist of three parts:
  - The system booting code, starting at 0x0200.
  - The interrupt service routine, starting at (   ).
  - The user program, starting at x3000.

  You can write more than one `.ORIG` and `.END` pairs in a single .asm file, in order to making your code distributed in different memory addresses.

User program

The user program prints continuously and infinitely. In the original game, a bird is flying from left to right, but you may fly from top to bottom. In the game, the bird is represented by 3 continuous chars(for example `aaa`). Without control, it will fall to left, but the user can make it fly to right 1-9 blocks (chars) by pressing corresponding numbers. By the way, the bird will change its appearance after the user presses `a-z`. You should print 20 chars each line, and use `.` as air.

Here is an example:

```
.....aaa............ User Input(you don't need to echo it)
....aaa.............
...aaa..............
............aaa..... <-9
...........aaa......
.................aaa <-8
................aaa.
...............aaa..
..............aaa...
.............aaa....
............aaa.....
...........aaa......
..........aaa.......
.........aaa........
........ddd......... <-d
.......ddd..........
......ddd...........
.....ddd............
....ddd.............
...ddd..............
..aaa............... <-a
.aaa................
aaa.................
aaa.................
ooo................. <-o
ooo.................
....ooo............. <-4
...ooo..............
```

Falling to ground (the leftmost side) won't end the game. Flying too high (right) is not allowed. Just put the bird on the rightmost side if it fly out of the screen.

Delay for a short time between two lines so that our eyes can keep up with the output. A simple way to delay is to count down from 256 (or any number) to 0, like:

```
DELAY    ST R1, SAVER1
         LD R1, COUNT
LOOP     ADD R1, R1, #-1
         BRnp LOOP
         LD L1, SAVER1
         RET
COUNT    .FILL #256
SAVER1   .BLKW #1
```

Set the initial position and appearance of the bird by yourself.

Interrupt service routine

Once someone hits the keyboard, the keyboard interrupt service should handle the input and do some corresponding work.

The detail is left for you to finish.

(Here are some default trap and interrupt service routines in LC-3, you can find them between x0000 and x2FFF in memory.)

System booting code

You may have already observed that the system booting code in the LC-3 simulator is located at x0200:

```
            .ORIG x0200
            LD  R6, OS_SP

            LD  R0, USER_PSR
            ADD R6, R6, #-1
            STR R0, R6, #0

            LD  R0, USER_PC
            ADD R6, R6, #-1
            STR R0, R6, #0

            RTI

OS_SP       .FILL x3000
USER_PSR    .FILL x8002
USER_PC     .FILL x3000
```

The original system booting code is executed in supervisor mode. It initializes the system stack at x3000 (`OS_SP`), and then pushes PSR (`USER_PSR`) and PC (`USER_PC`) for user mode, and then finally `RTI` to the user code at x3000 (`USER_PC`).

In order to make the keyboard interrupt enable, some operations are necessary. Please **re-write** the system booting code, furnishing the requirements for the keyboard interrupt. Here are some hints:

1. You should enable the keyboard interrupt. The interrupt enable bit is part of the device status register. For the keyboard, the device status register, i.e. KBSR, is located at (   ). Therefore, you should do (   ).
2. You should let the machine know how to handle the keyboard interrupt. The interrupt vector INTV for the keyboard is x80. Therefore, you should put the starting address of (   ) in the correct place in the interrupt vector table, that is (   ).

Limitation

- input char: `0-9` and `a-z`

Grading

Lab 4 takes **7%** of the total score, consisting of Check part (50%) and Report part (50%).

Check Part

- First upload your code to Learning in ZJU, then find a TA to check your code in person. TAs will first test the correctness of your program, then ask you some questions to make sure you understand what you code but not cheat.
- You can try again if you fail in checking, but there will be a penalty of -10% (of checking part) for each try.
- We strongly suggest you to make a thorough test by yourself before checking.
- We strongly suggest you to write enough comments in your code so that you will be aware of what's going on in your program and confident to answer TA's questions.

Report Part

- Report must be written **in English**, concise and carrying main ideas. Try to use the report to convince TAs that you complete the task by yourself.
- Your lab report should contains the following contents:
  - Algorithm. Flowchart or Pseudocode is prefered. The complexity of your algorithm will not affect your score.
  - Essential parts of your code with sufficient comments. Please only select the most important code phases and explain them.
  - Questions that TA asked you, and Answers.
- **No more than 3 A4 pages.** No template provided. Be sure to make it readable.

Penalty

- **Wrong Answer**: -10% of Check part each time.
- **Delay**: -20% of the corresponding part per day.
- **Cheating**: -100% of this lab. Additionly, -10% of the final score of this course. **Please note that uploading your answer to the Internet is also CHEATING!!!**