EE 337: Microprocessors Laboratory (Spring 2021)
Indian Institute of Technology Bombay

Project: 20 points — Date: March 8, 2021

Matching Pairs Game on Pt-51

1. [20 points] In this project, you will be writing a game program for a memory matching game. It is similar to the Concentration card game `https://en.wikipedia.org/wiki/Concentration_(card_game)`. The player will type on a keyboard connected to the Pt-51 board via UART.

   - When the game starts, the LCD displays the following message on its two lines, where `XX` is either blank or a positive integer.

     ```
     ********   0
     ********   L:XX
     ```

   - The `0` on the first line indicates the number of moves the player has made in a game. It should increment after every move.
   - The `L:XX` on the second line will be used to show the lowest number of moves used to finish the game so far. In the beginning, `XX` will be blank. After a player finishes the first game, `XX` will have the number of moves used by the player to finish the game. After two or more games have been finished by the player, `XX` has the minimum number of moves used by the player to complete a game.
   - The 16 `*` characters hide two `1`s, two `2`s, two `3`s, two `4`s, two `5`s, two `6`s, two `7`s, two `8`s.
   - The goal of the player is to uncover all the numbers while matching them with already uncovered numbers, using as few moves as possible. At the end of a game, the LCD might look like the following:

     ```
     15274623   26
     64185783   L:
     ```

   - A player uncovers a covered position by pressing an alphabet key. The mapping from LCD positions to the alphabet key is shown below.

     ```
     qwertyui
     asdfghjk
     ```

     Note that the keys are taken from the first two rows of a QWERTY keyboard.
   - If the player presses a key corresponding to an already uncovered position, this key press should be ignored. The number of moves is not incremented.
   - If the player presses a key corresponding to a covered position, the number at the position is revealed on the LCD. The number of moves is incremented on the first line of the LCD.
     – If the newly uncovered number is the first one to be uncovered, then it remains uncovered.
     – If the newly uncovered number matches an existing uncovered number, then the newly uncovered number remains uncovered.
     – If the newly uncovered number does not match an existing uncovered number, then the newly uncovered number is replaced by `*` after a 3 second delay.

- When all the positions have been uncovered, the current game ends and a new game begins. The number of moves is used to update the `XX` on the second line.
- To make the game interesting, the locations of the hidden numbers need to change from game to game. We associate a unique integer from 0 to 15 with each of the 16 positions as shown below.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

- We can use a linear feedback shift register (LFSR) to generate pseudorandomness. Consider a LFSR with 4 registers (each containing a bit). We can use a single nibble to store the current state of the LFSR. If the current state of the 8 bits is (b_3, b_2, b_1, b_0), the next state is

$$(b_3 \oplus b_0, b_3, b_2, b_1),$$

  where $\oplus$ represents bitwise XOR. With this next-state rule, the LFSR will cycle through all the non-zero 4-bit states with a period of 15. This is an example of a maximal-length LFSR `https://en.wikipedia.org/wiki/Linear-feedback_shift_register`.
- Initialize the LFSR state (b_3, b_2, b_1, b_0) to a non-zero value (this is done only once when the program loads for the first time).
  - Let B_0 be the decimal integer corresponding to $b_3b_2b_1b_0$.
  - Place the first `1` at the location B_0. For example, if $b_3b_2b_1b_0 = 0101$ then place the `1` at location 5 (the sixth location on the first line of the LCD). If $b_3b_2b_1b_0 = 1010$ then place the `1` at location 12 (the fifth location on the second line of the LCD).
  - Calculate the next state of the LFSR. Let the next state be $b'_3b'_2b'_1b'_0$.
  - Let B_1 be the decimal integer corresponding to $b'_3b'_2b'_1b'_0$.
  - Place the second `1` at the location $B_0 + B_1 \bmod 16$. As B_1 is an LFSR state, it is non-zero. Hence the second `1` will be placed in a location which is different from the first `1`.
  - Calculate the next state of the LFSR. Let B_2 be the decimal integer corresponding to this state.
  - Place the first `2` at the location $B_0 + B_2 \bmod 16$.
  - Calculate the next state of the LFSR. Let B_3 be the decimal integer corresponding to this state.
  - Place the second `2` at the location $B_0 + B_3 \bmod 16$.
  - And so on, until the second `8` is placed.
  - The characters are placed at B_0 and $B_0 + B_i \bmod 16$ for $i = 1, 2, \dots, 15$. As the B_i's are distinct integers in the set $\{1, 2, \dots, 15\}$, the locations of the characters will be distinct.
- Preserve the last state of the LFSR for the next game. This state will be used as the B_0 in the next game.