Project #2

Michael Schoen, Abdirahman Osman, Illya Starikov

Due Date: November 8th, 2016

For our project, we have decided to implement a memory matching game. The game will start off by enabling a single light. If the user correctly presses said light, the user will move onto the next level and enable a light sequence. For every successive level, an additional light have to be kept in mind (i.e. for the n^{th} level, there will be n lights that will have to be pressed).

This will continue for 15 games, where upon a successful finish, a special light sequence and a song will asynchronously play. From here, the user can choose to play another game if they so choose.

1 Explanation

Below we will take a more in-depth look into the implementation of our project.

1.1 Random Number Generation

Because we want our game to be different every round, we have to have some form of random number generation. Initially, we had tried to implement a *linear congruential generator*; unfortunately, we were unsuccessful. We instead went with a different implementation.

Both implementations require a seed value (and to have different results, we need a different seed value) — we "cheat" to get this. During our input, we not only check to see if the user initiated a game, but we also increment the register the seed is stored in. This way, there is only a $^1/_{255}$ chance the user will get the same game[1].

[1]Although this is a significant value (0.39%), it serves well for demo purposes.

Failed Attempt

The basic summary is as follows: beginning with a seed value (named X_n), a new linear, psuedo-random number can be calculated via:

$$X_{n+1} = (aX_n + c) \mod m \tag{1}$$

where the follow conditions hold

1. $a, X_n, c, m \in \mathbb{Z}^+$
2. $0 \leq X_n, c < m$
3. $0 < a < m$
4. $m \neq 0$

The issue we face is it is *highly* recommended that m be quite large (most popular implementations range from 2^{31} to 2^{48}); unfortunately, we only have 2^8 available to us. Because of this, we either get a cycle roughly every 10 iterations or the same number produced. As one might imagine, this is a game-breaking bug; we decided to go with something different.

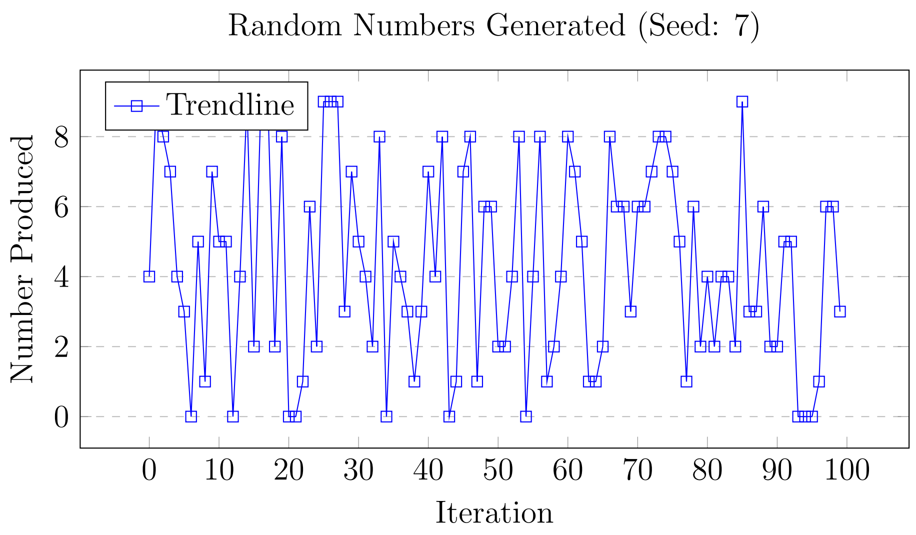
Our failed attempt for a linear congruential generator is as follows.

```
; Check writeup for how this works
; For now, the formula is

; X_n+1 = aX_n + c mod m

; For our purposes, a = 7, c = incrementor, m = 72
; for our purposes, incrementor can't be a multiple of 7
; X_n is stored in R7, incrementor in R6

; Result will be stored in A
RNG:
                MOV A, R7
                MOV B, #7D
                DIV AB
                MOV A, B
                JNZ SKIP1
                INC R6
SKIP1:          MOV A,  #7D
                MOV B, R7
                MUL AB                     ; A = aX_n

                ADD A, R6
                MOV B, #72D
                DIV AB

                MOV R7, B

                MOV A, B
                MOV B, #10D
                DIV AB

                MOV A, B
                RET
```

Successful Attempt

For our successful attempt, we ported a random number generating from an open source code base[2]. It similar to the linear congruential generator in the sense that it linearly produces a psuedo-random number; however, it is a different formula (one that can't be eloquently described in an equation). Below is a graph that shows the distribution of numbers up to 100 iterations.

[2] https://www.pjrc.com/tech/8051/



1.2 Light Show

When programming the lights, we notice there to be 9 lights; which just so happens to be the exact size of register A and the carry bit C in the `sfr`. So we decided to map every bit in A and C to a particular light. For simplicity, we used a 1_2 to represent an 'on' state and a 0_2 for the 'off' state.

Our mapping is as follows: we start the MSB of A, which represents the top-left light. The proceeding two bits make up the top-most row of the lighting board. The fourth bit represents the middle row, *first* light. We continue with this pattern until we come to the last light. The last light is represented by the carry bit C. Fig. 1 summarizes this lighting schematic.

As an example, $\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}$ $\boxed{1}$ would enable only the top and bottom lights, leaving the middle row blank. $\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}$ $\boxed{1}$ enables the left and right columns. $\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}$ $\boxed{1}$ makes an X.

As mentioned, to represented an 'on' state, we used a 1_2; however, the lights are active low. So, the first thing we do is `CPL A` and `CPL C` to ensure we don't have opposite affect. From there, we rotate through C to enable individual light. Finally, we `CPL` A and C again to bring the data back to its original state.

The individual lighting effects were done through trial and error of what looked aesthetically pleasing. There are three light patterns, one for losing, advancing to the next level, and the winning light sequence. The winning light sequence has five parts, eloquently named:

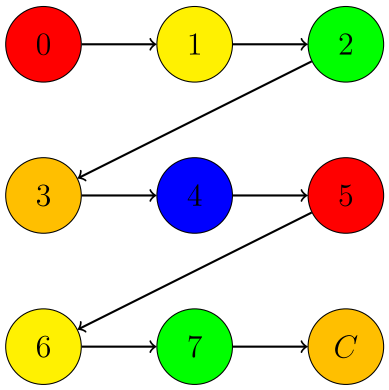


Figure 1 – The lighting schematic for "lights" subroutine.

1. Light Spiral Part I
2. Light Spiral Part II
3. Full Light Spiral
4. Os
5. Bring the Thundaaa

1.3 Music

We know the frequency of the Philips P89LPC932A1 to be 7.373 MHz, with 2 cycles per machine cycle. Therefore,

$$\frac{2\text{ cycles}}{\text{machine cycle}} \cdot \frac{1\text{ Period}}{7.373\,\text{MHz}} = 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} \tag{2}$$

We use this calculation as the base of our music.

E5	$f = 659.255\,\text{Hz} \implies T = 1516\,\mu\text{s}$ $1516\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 5589\,\text{mc}$ $5589\,\text{mc} \div 4 = 1398\,\text{mc}$ $1398\,\text{mc} \implies$ 699 iterations (with `DJNZ`)
F5	$f = 698.456\,\text{Hz} \implies T = 1431\,\mu\text{s}$ $1431\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 5275\,\text{mc}$ $5589\,\text{mc} \div 4 = 1318\,\text{mc}$ $1318\,\text{mc} \implies$ 569 iterations (with `DJNZ`)
G5	$f = 783.991\,\text{Hz} \implies T = 1275.5\,\mu\text{s}$ $1275.5\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 4702\,\text{mc}$ $4702\,\text{mc} \div 4 = 1176\,\text{mc}$ $1176\,\text{mc} \implies$ 588 iterations (with `DJNZ`)
D5	$f = 587.330\,\text{Hz} \implies T = 1702.6\,\mu\text{s}$ $1702.6\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 6277\,\text{mc}$ $7045\,\text{mc} \div 4 = 1570\,\text{mc}$ $1570\,\text{mc} \implies$ 785 iterations (with `DJNZ`)
C5	$f = 523.251\,\text{Hz} \implies T = 1911.1\,\mu\text{s}$ $1911.1\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 7045\,\text{mc}$ $7045\,\text{mc} \div 4 = 1760\,\text{mc}$ $1760\,\text{mc} \implies$ 880 iterations (with `DJNZ`)
Flat D5	$f = 554.365\,\text{Hz} \implies T = 1803.8\,\mu\text{s}$ $1803.8\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 6650\,\text{mc}$ $6650\,\text{mc} \div 4 = 1662\,\text{mc}$ $1662\,\text{mc} \implies$ 831 iterations (with `DJNZ`)

2 Future Work

3 Work Effort

- Michael Schoen
  - Programmed binary counter.

    - Programmed game logic.

- Osman Abdirahman
    - Programmed initial beep.
    - Programmed song implementation.
- Illya Starikov
    - Programmed random number renerator.
    - Programmed light sequence.