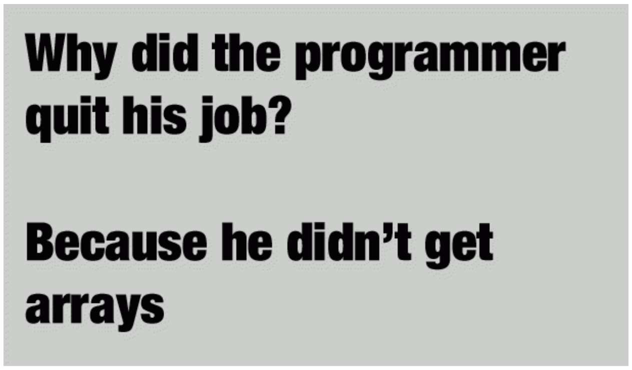
# CS 218 – Assignment #4

Purpose: Become familiar with the MIPS Instruction Set, and the MIPS functions, MIPS standard calling convention, and indexing for multiple dimension arrays.
Points: 80

## Assignment:



**Why did the programmer quit his job?**

**Because he didn't get arrays**

Write a simple assembly language function to simulate the rolling of two dice. The provided main calls the following functions:

- Write a value returning MIPS function, ***getInput()***, that will read a number between between 1 and 100,000. *Note*, A MIPS function returns its integer result in $v0.
- Write a value returning MIPS function, ***random()***, to generate a random number using a lagged Fibonacci generator[1].
- Write a void MIPS function, ***throwDice()***, to simulate the rolling of two dice ***n*** times. Each die can show an integer value from 1 to 6, so the sum of the values would vary from 2 to 12. The results should be stored in a two-dimension array. The first roll represents the row and the second roll represents the column. This function will call the ***random()*** function.
- Write a void MIPS function, ***results()***, to calculate sums, percentages, and display the two-dimensional matrix showing the results. The numbers should be printed in a two-dimensional format (see example output). All numbers in the table must be right justified (i.e., lined up on right side). Print the eleven possible different totals followed by the percent each combination occurred. For the final percentages, the sums should be computed as integers, then converted to float (single precision), divided by the roll count (also converted to single precision), and multiplied by 100.00. Due to the required formatting, this is likely the most difficult function.

## Array Implementation:

In assembly, multi-dimension arrays are implemented as a large single dimension array. The formula for calculating two-dimensional array indexing is:

```
addr(row,col) = baseAddr + (rowIndex * colSize + colIndex) * data_size
```

You must use the formula to access matrix elements. **No score** will be provided for submissions that do not use this formula. Refer to Chapter 5.6 2.2 for more information.

1 For more information, refer to: http://en.wikipedia.org/wiki/Lagged_Fibonacci_generator

**Submission:**

- All source files must assemble and execute with QtSpim/SPIM MIPS simulator.
- Submit source file
  - Submit a copy of the program source file via the on-line submission
- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time.
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

**Program Header Block**

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
#  Name: <your name>
#  NSHE ID: <your id>
#  Section: <section>
#  Assignment: <assignment number>
#  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a reduction of points.

**Scoring Rubric**

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

| Criteria | Weight | Summary |
|---|---|---|
| Assemble | - | Failure to assemble will result in a score of 0. |
| Program Header | 3% | Must include header block in the required format (see above). |
| General Comments | 7% | Must include an appropriate level of program documentation. |
| Program Functionality (and on-time) | 90% | Program must meet the functional requirements as outlined in the assignment. Must be submitted on time for full score. |

**Lagged Fibonacci Generator**

The Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13... where the next number in the sequence is the sum of the previous two numbers. A lagged Fibonacci sequence generates the next number as the sum of two prior numbers. For example:

$$S_n = (S_{n-j} + S_{n-k}) \bmod 2^{16}$$

The initial **S** values are generated by a special initialization routine. To simplify this assignment, a complete **S** table is provided (see provided main). Additionally, we will set the initial value of **jptr** to 16 and **kptr** to 4. As such, the algorithm for the Lagged Fibonacci Generator is:

```
itmp = sTable(jptr) + sTable(kptr)
sTable(jptr)  =  itmp mod 2^16
jptr = jptr - 1
kptr = kptr - 1
if ( jptr < 0 ) jptr = 16
if ( kptr < 0 ) kptr = 16
randDice = ( itmp / 100 ) mod 6
```

*Note*, algorithm assumes array index's start at 0.

**Example Output:**

The following is the example output for an input of 10,000 rolls. *Note*, each 'roll' consists of two dice.

```
MIPS Assignment #4
Program to Simulate Rolling Two Dice.


Dice Rolls Simulation Input Routine

Enter Number of Dice Rolls to Simulate: 10000


************************************

Rolls: 10000

  ------- ------- ------- ------- ------- -------
 |   275 |   265 |   283 |   297 |   277 |   298 |
 |   302 |   265 |   269 |   260 |   264 |   284 |
 |   295 |   275 |   289 |   275 |   273 |   271 |
 |   277 |   287 |   296 |   290 |   264 |   277 |
 |   280 |   265 |   287 |   245 |   276 |   261 |
 |   267 |   295 |   272 |   271 |   308 |   265 |
  ------- ------- ------- ------- ------- -------



Percentages:
    2:    2.75000000
    3:    5.67000008
    4:    8.42999935
    5:    11.18000031
    6:    13.93000031
    7:    16.64999962
    8:    14.29000092
    9:    10.52000046
    10:   8.23999977
    11:   5.69000006
    12:   2.64999986
```