**ORGANIZACION Y ARQUITECTURA DE COMPUTADORAS**

**SEGUNDO TERMINO 2013 - 2014**

**PRIMER PROYECTO**

**GRUPO # 2**

**JAIME CASTELLS PEREZ**

**JAVIER RON ARTEAGA**

**PROJECT WRITE-UP**

En esta primera entrega de proyecto se presentaran las funcionalidades requeridas en la tarea publicada en el sistema SIDWeb bajo el nombre de “[First Project – Assembly Language](https://www.sidweb.espol.edu.ec/courses/16126/assignments/20790)”.

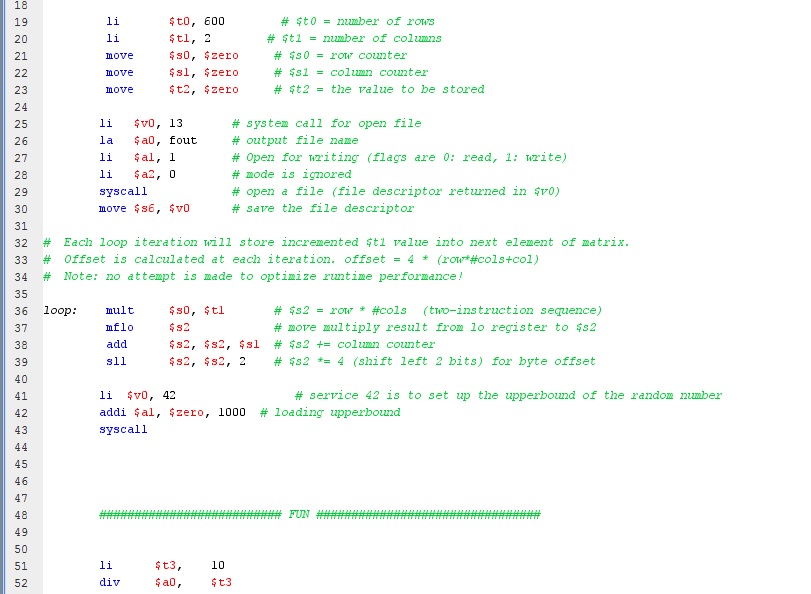
La tarea consiste en 5 partes descritas a continuación con sus respectivas respuestas y soluciones:

**PARTE A:**

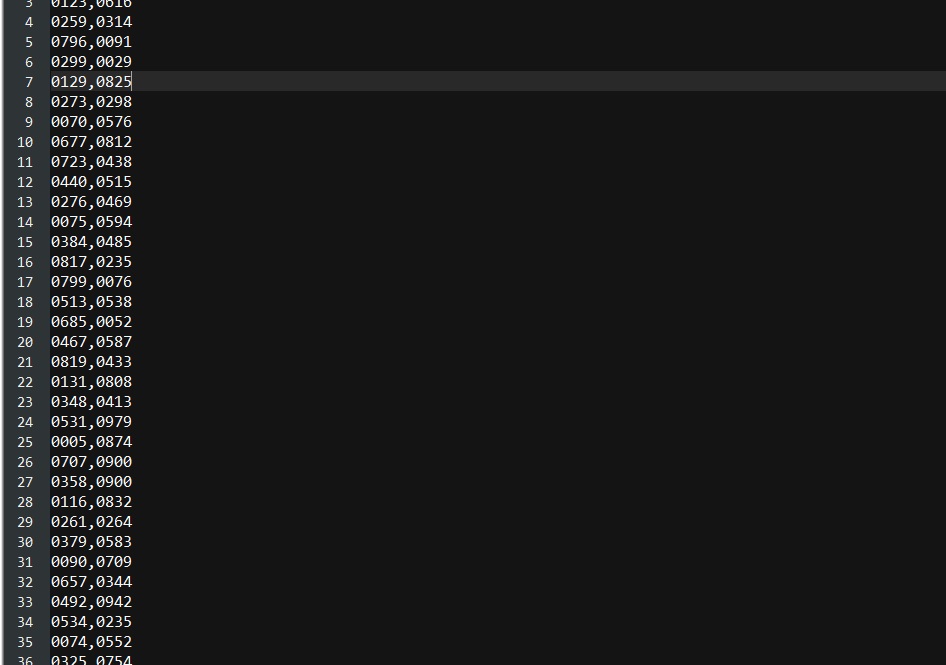
Usando el generador de números aleatorios del simulador MARS, crear un arreglo de tamaño 600x2 y llenarlo con números mayores a 0, y menores a 1000; y guardar esta información en un archivo llamado “randomMIPS.txt”.

**SOLUCION:**

Se creó un archivo generarAleatorios.asm, que cumple con la funcionalidad pedida.

  
Ilustración 1: generarAleatorios.asm

Del cual se obtuvieron los resultados. Se generaron archivos aleatorios, y cada dígito, uno a uno, se lo transforma en strings para ser guardados en un archivo de texto. Este archivo se llama randomMIPS.txt.

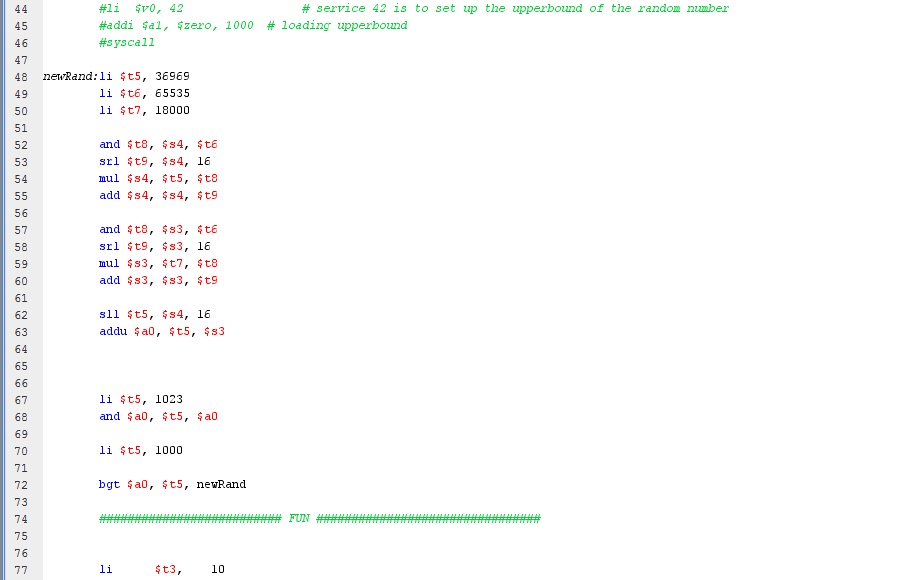
  
Ilustración 2: randomMIPS.txt

**PARTE B:**

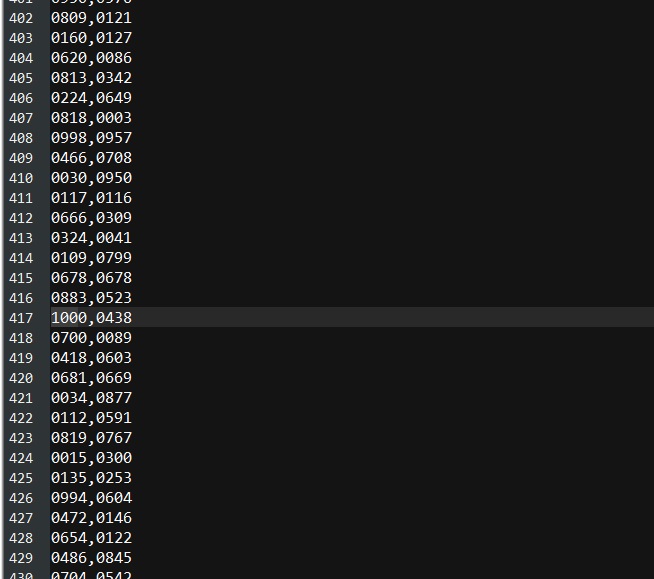
Para esta parte se requirió algo similar a la la parte A, con la excepción de que en lugar de usar el generador proveído por MIPS, se implemente un generador de aleatorios de tipo “Multiply-With-Carry”, inventado por George Marsaglia.

**SOLUCION:**

Se modificó el archivo de la parte A, para reemplazar el generador de aleatorios. El archivo es marsaglia.asm.

  
Ilustración 3: marsaglia.asm

Con esto, se obtuvieron los resultados, codificando el algoritmo que se dio en el Sidweb. Se usó el código anterior para basarse en este. El resultado se ve en randomMarsaglia.txt.

  
Ilustración 4: randomMarsaglia.txt

**PARTE C:**

En esta parte se pide graficar los resultados de las 2 partes anteriores, y analizar dichas gráficas comparándolas una con otra.

**SOLUCION:**

Podemos ver que con ambos generadores se logra una concentración algo uniforme dentro de todo el espacio usado; y es difícil notar alguna diferencia marcada entre los dos métodos.

*Ilustración 5: Puntos generados por generarAleatorios.asm*

*Ilustración 6: Puntos generados por marsaglia.asm*

**PARTE D:**

Para esta sección se pide generar las siguientes configuraciones específicas de clusters sintéticos usando los generadores de aleatorios implementados previamente y, la funcionalidad implementada para imprimir datos en archivos:

* 1. Dos dimensiones – Dos clusters
  2. Tres dimensiones – Dos clusters
  3. Cuatro dimensiones – Dos clusters
  4. Dos dimensiones – Tres clusters
  5. Tres dimensiones – Tres clusters
  6. Cuatro dimensiones – Tres clusters
  7. Dos dimensiones – Cuatro clusters
  8. Tres dimensiones – Cuatro clusters
  9. Cuatro dimensiones – Cuatro clusters
  10. Dos dimensiones – Cinco clusters
  11. Tres dimensiones – Cinco clusters
  12. Cuatro dimensiones – Cinco clusters

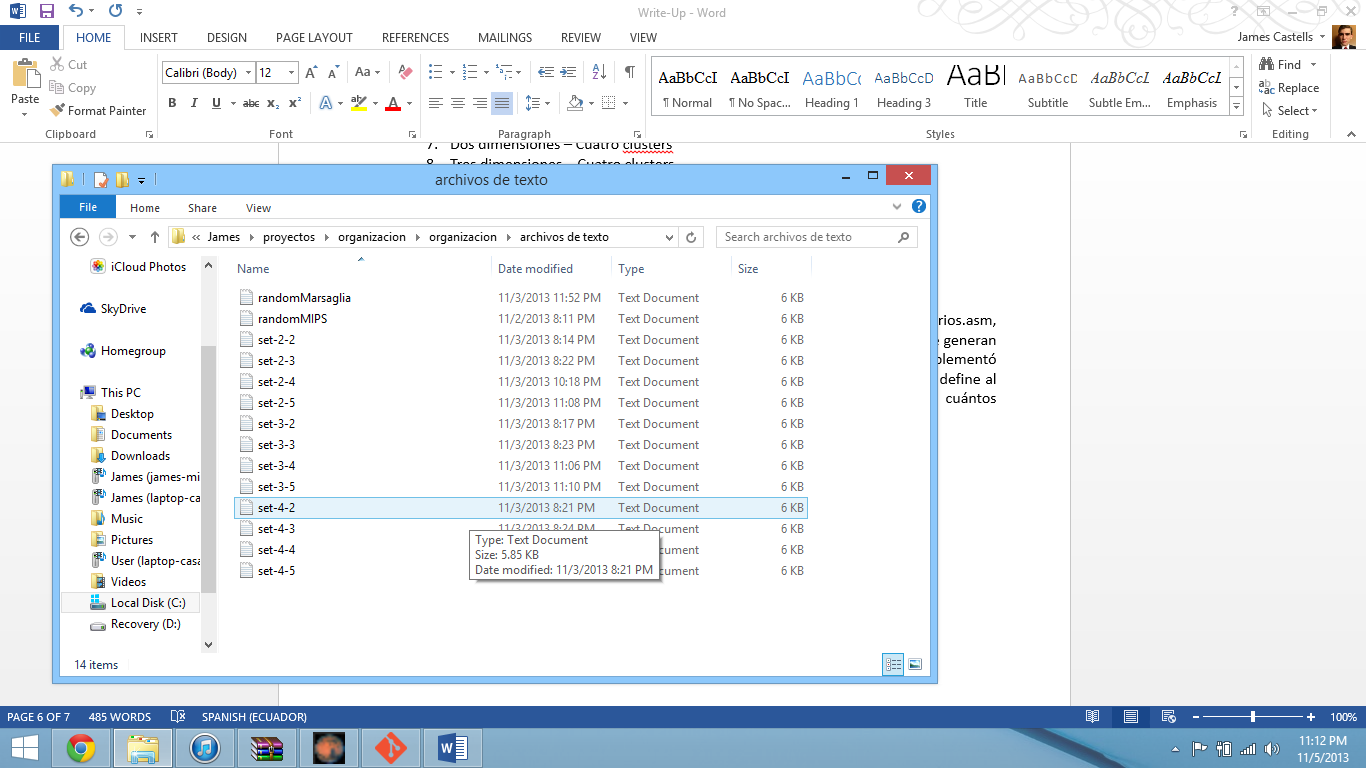
**SOLUCION**:

Primero se implementó el código generarAleatoriosClusters.asm, basado en generarAleatorios.asm, que genera clusters de puntos, es decir, se genera un punto aleatorio, y a partir de este se generan puntos cercanos a este con un rango mucho menor de aleatoriedad. Después de esto se implementó funcionalidad para crear tantos clusters con tantas dimensiones sean necesarias. Esto se define al principio del programa. Finalmente, se implementó que el usuario pueda especificar cuántos clusters y dimensiones quiere.

El algoritmo esta estructurado de la siguiente manera:

1. Se reserva espacio para guardar los centros de los clusters.
2. Se crean los centros de los clusters aleatoriamente y se los guarda en el espacio reservado. Cada vez que se crea un nuevo centro, se comprueba que no este muy cerca de ninguno de los creados anteriormente.
3. Se crean 1200 puntos aleatorios dentro de un rango establecido cerca de los centros antes creados. Después de crear cada punto, se lo imprime en el archivo y se genera el siguiente.

Un reto algo complicado en esta parte fue manejar la memoria de tal manera que se puedan encontrar los puntos almacenados previamente y extraer sus valores, para esto se implemento un sistema de indexado sencillo (aunque el código en MIPS haya sido un tanto complejo).

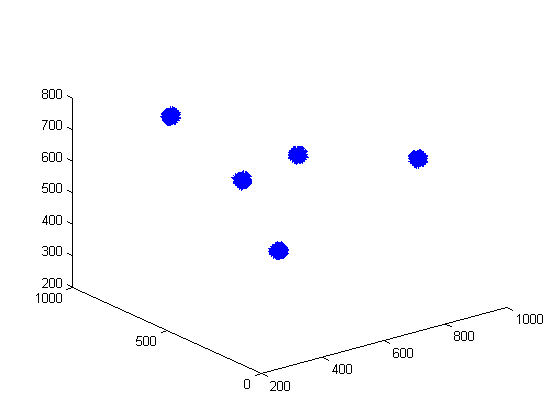
 *Ilustración 7: Los archivos con clusters generados.*

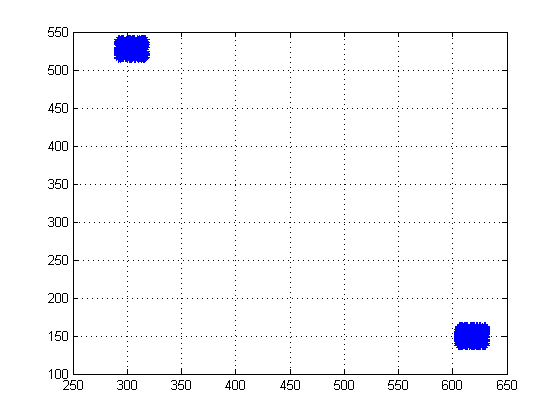
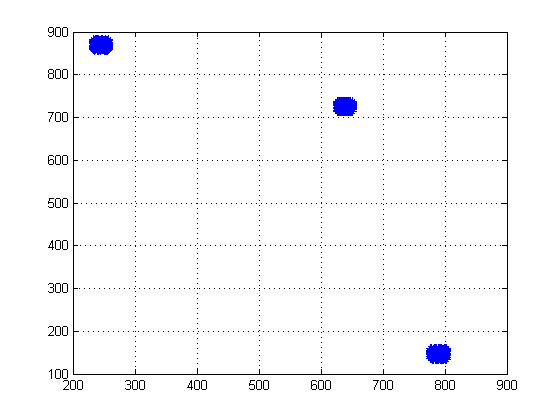
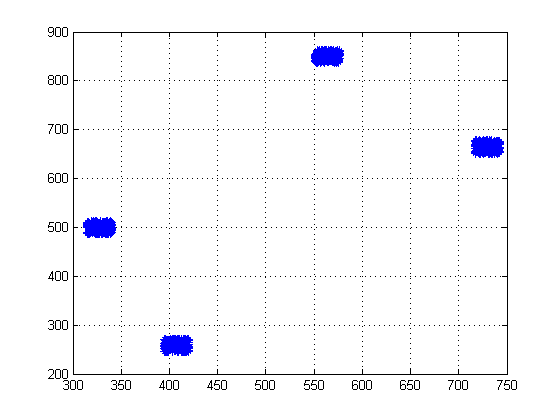
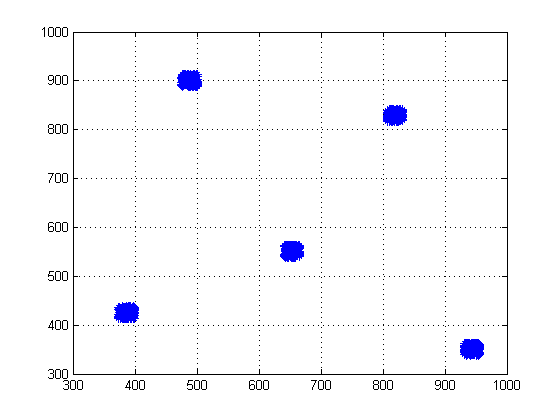
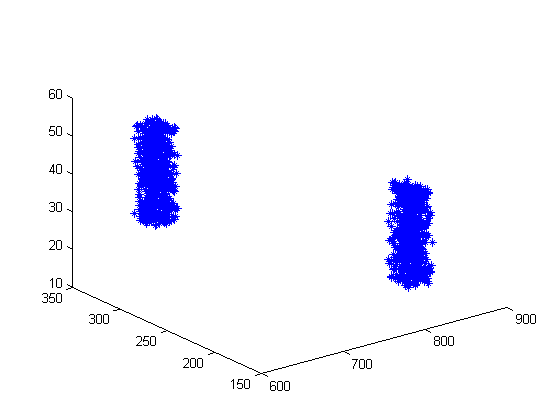
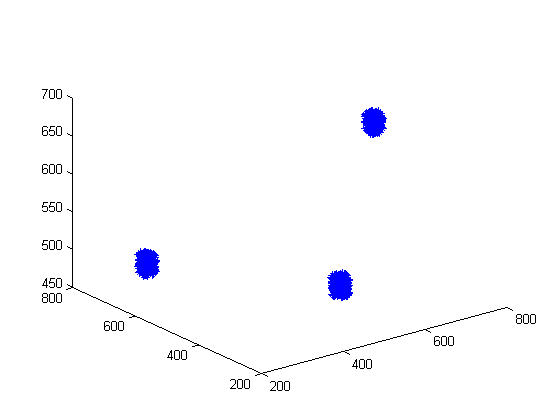
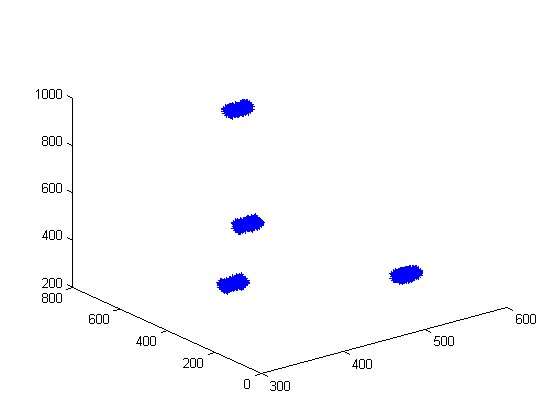
**PARTE E:**

En esta parte se pide graficar algunos de los datos obtenidos para observar los clusters generados.

**SOLUCION:**

Se pueden observar claramente que los archivos en 2D y 3D ofrecen clusters definidos:

*  
Ilustración 8: 3 dimensiones con 5 clusters.*

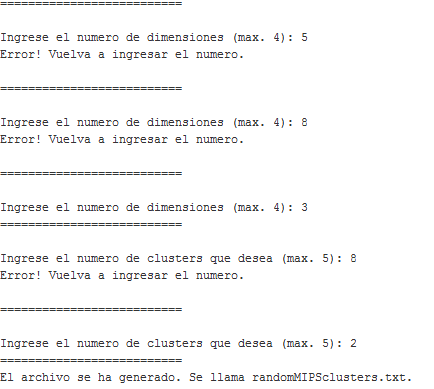
*  
Ilustración 9: 2 dimensiones con 2 clústers.  
Ilustración 10: 2 dimensiones con 3 clústers.  
Ilustración 11: 2 dimensiones con 4 clústers.  
Ilustración 12: 2 dimensiones con 5 clústers.  
Ilustración 13: 3 dimensiones con 2 clústers.  
Ilustración 14: 3 dimensiones con 3 clústers.  
Ilustración 15: 3 dimensiones con 4 clústers.*

**PARTE F:**

El programa debe tener buena interfaz para el usuario.

**SOLUCION:**

Al pedir números de clústers y dimensiones, se presenta información clara al usuario:

  
*Ilustración 16: Pedida de dimensiones y clusters al usuario.*