**David Flores Barbero 70907575R**

**Francisco Blázquez Matías 70919093L**

**DOCUMENTACIÓN DLX**

**Listado de código fuente:**

-dlx.S : Versión del programa sin optimización.

-dlxOptimizado.S: Versión del programa con la optimización.

**Explicación del trabajo realizado:**

Se ha partido realizado en un primer momento un programa que realizase el cálculo de posiciones aleatorias en un vector para ir introduciendo en los valores. Esta primera versión será denominada dlx.S y como parámetros de entrada tenemos los siguientes:

m: word 32768

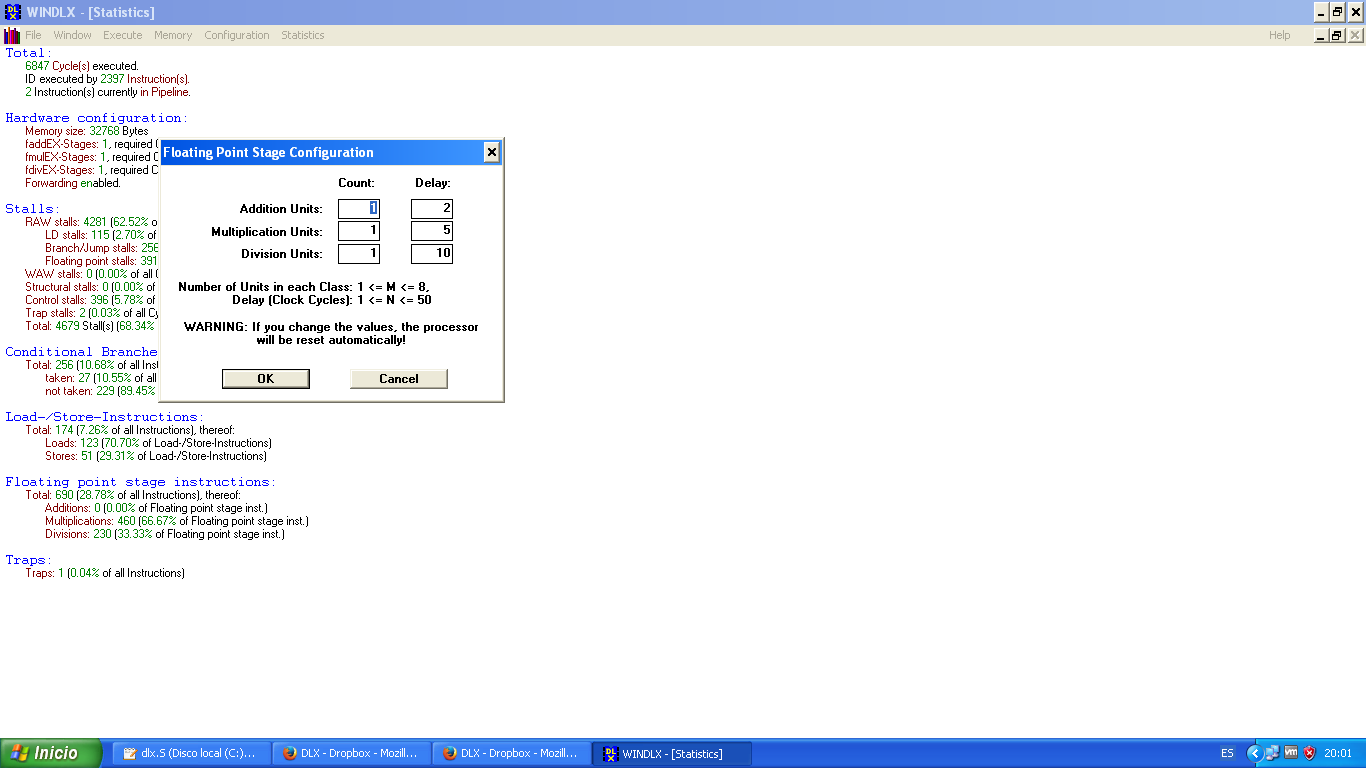
a: word 1103515245

c: word 12345

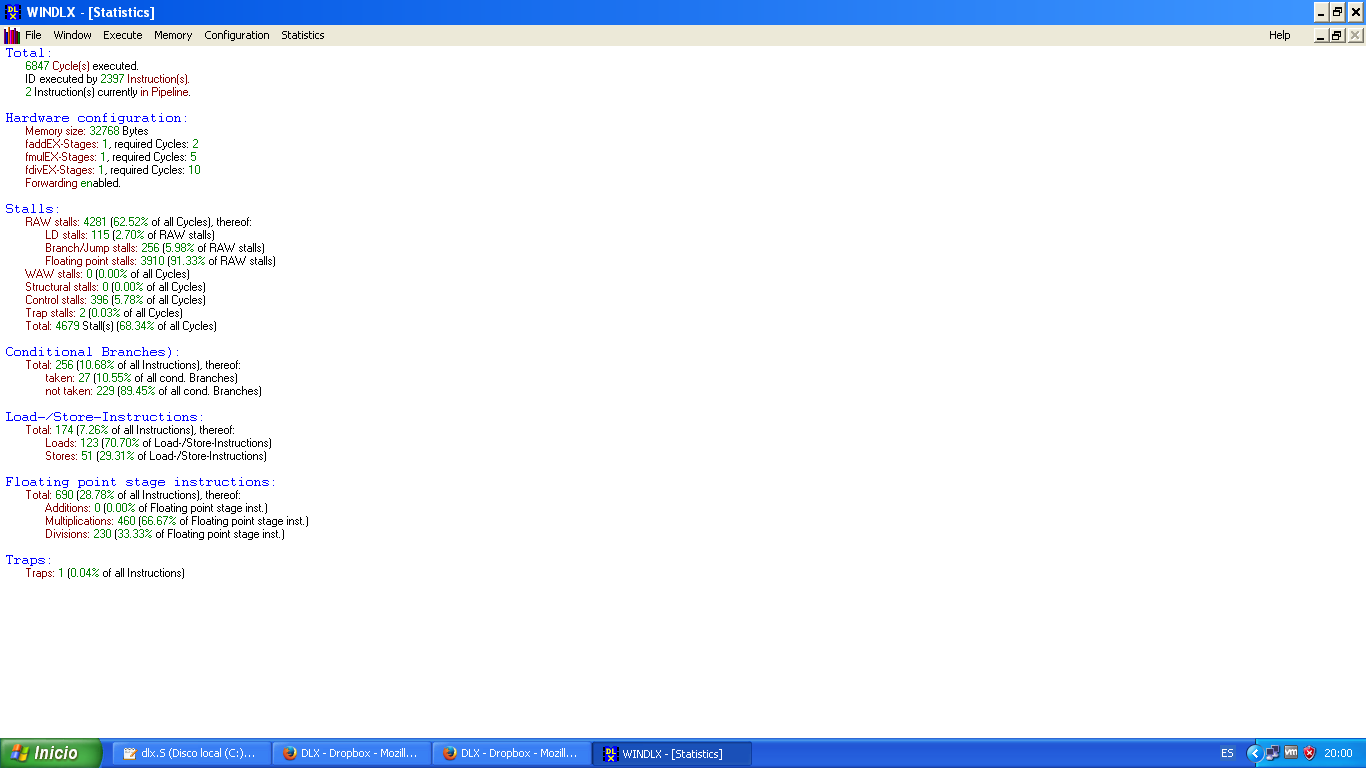
semilla : word 2

nmax: word 25

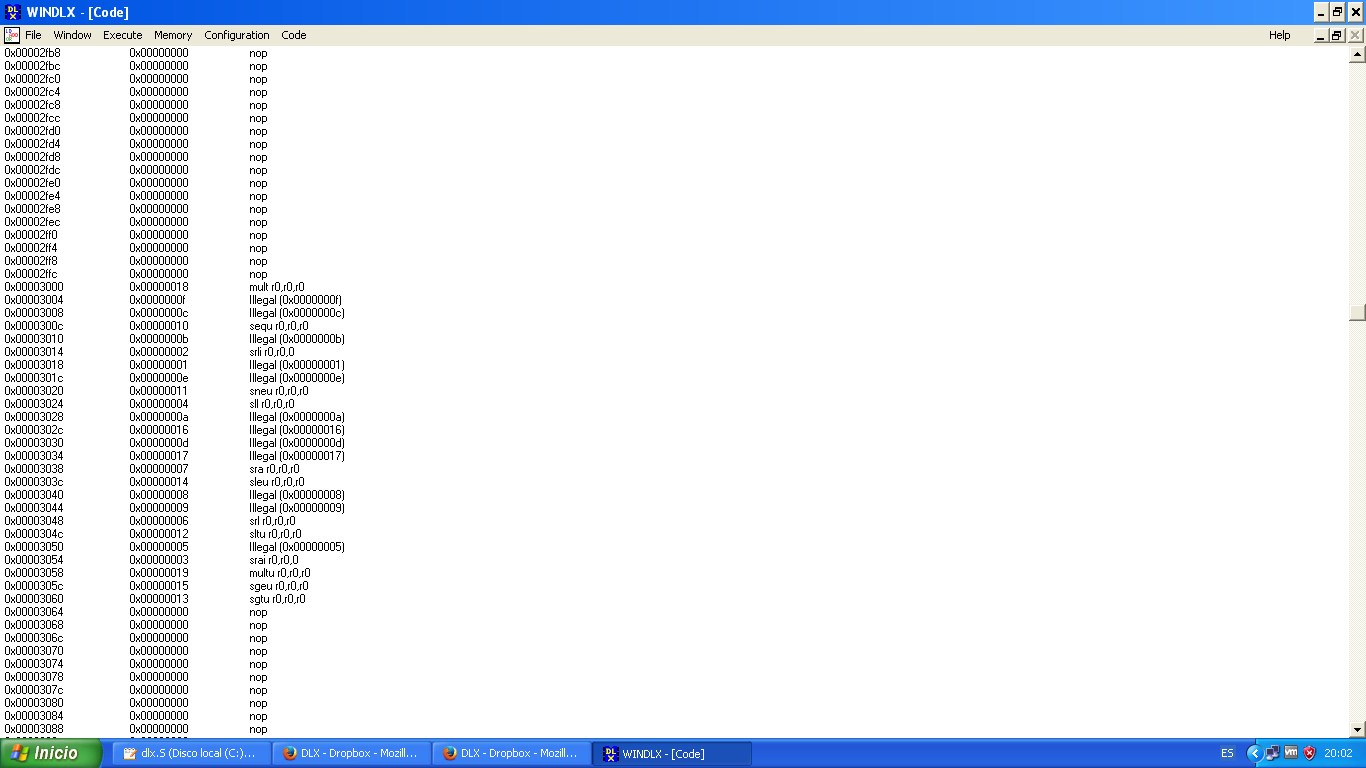
Y con la configuración de:



Obtendremos unas estadísticas al ejecutar con estos valores:



Y en el que podemos observar el vector colocado correctamente:

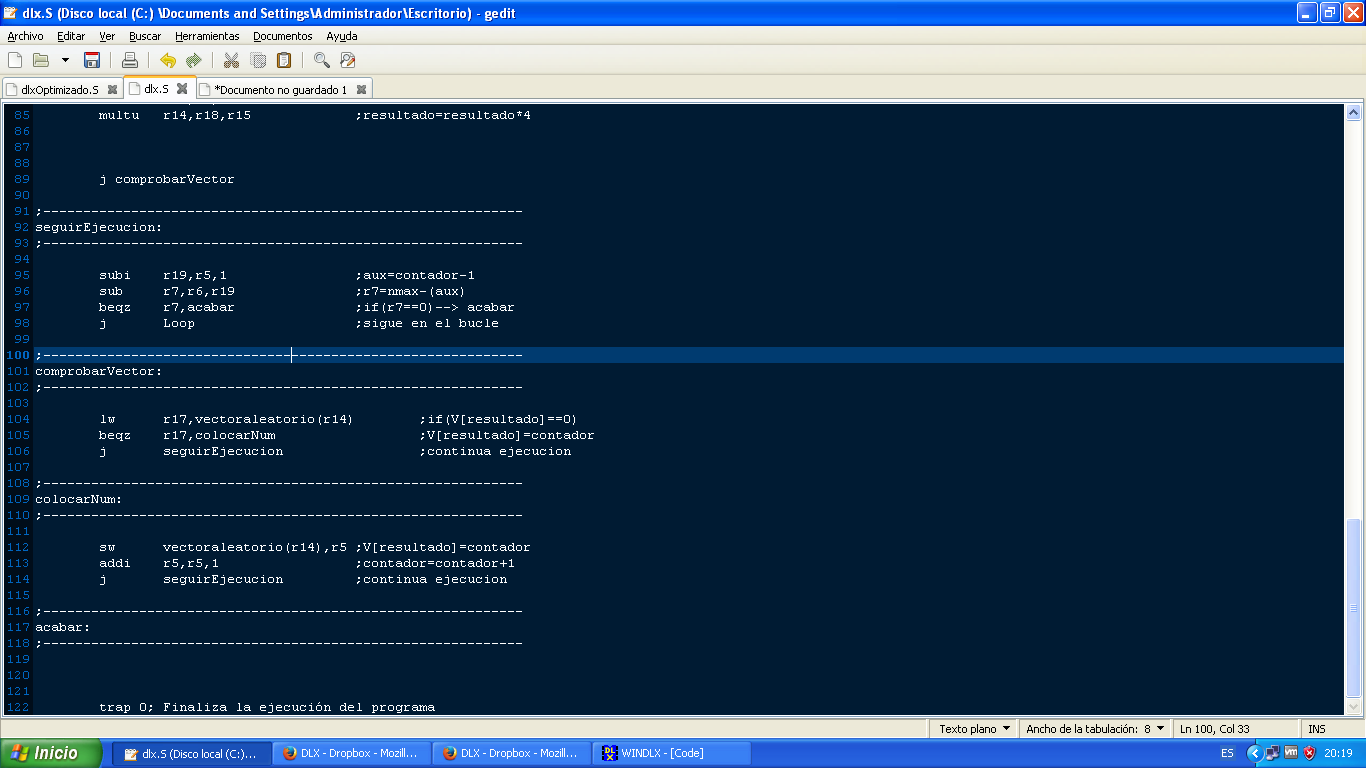


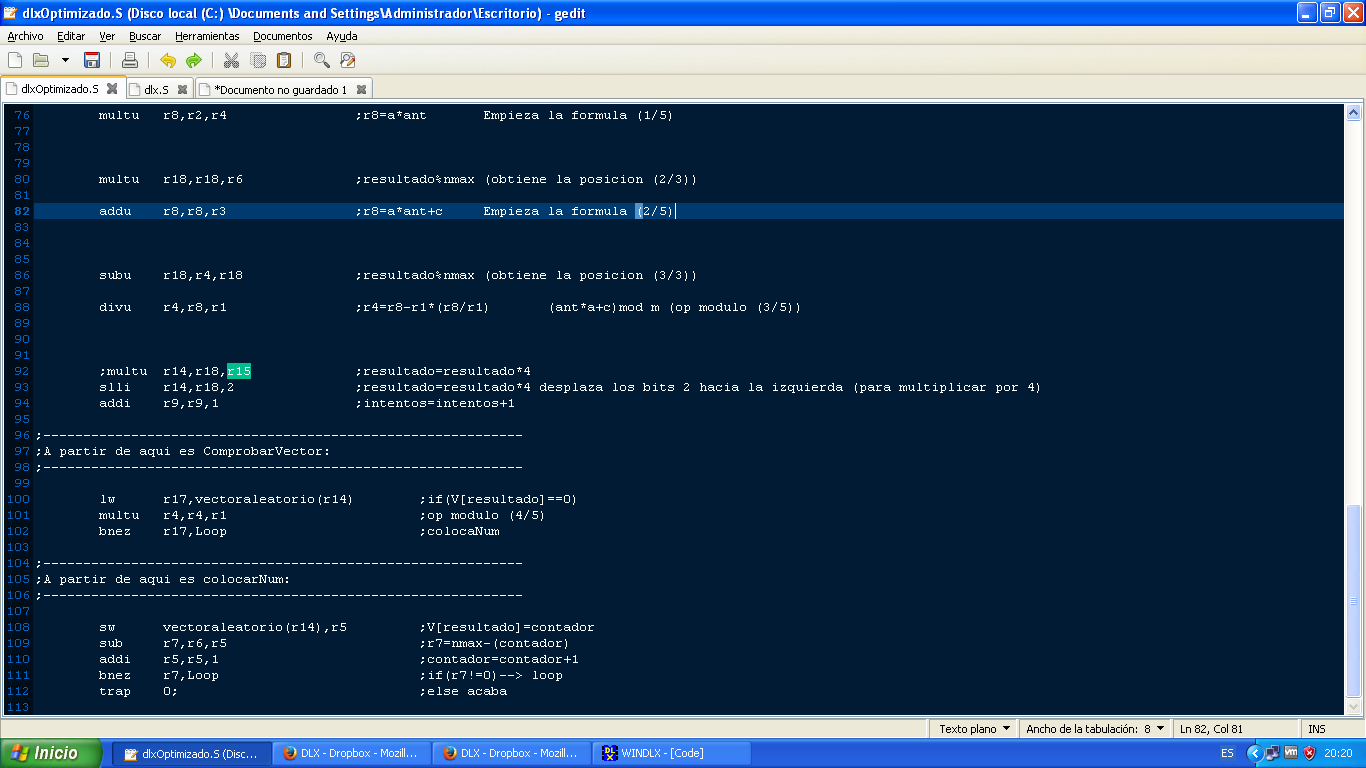
Cuyos resultados están en hexadecimal.

**Optimizaciones:**

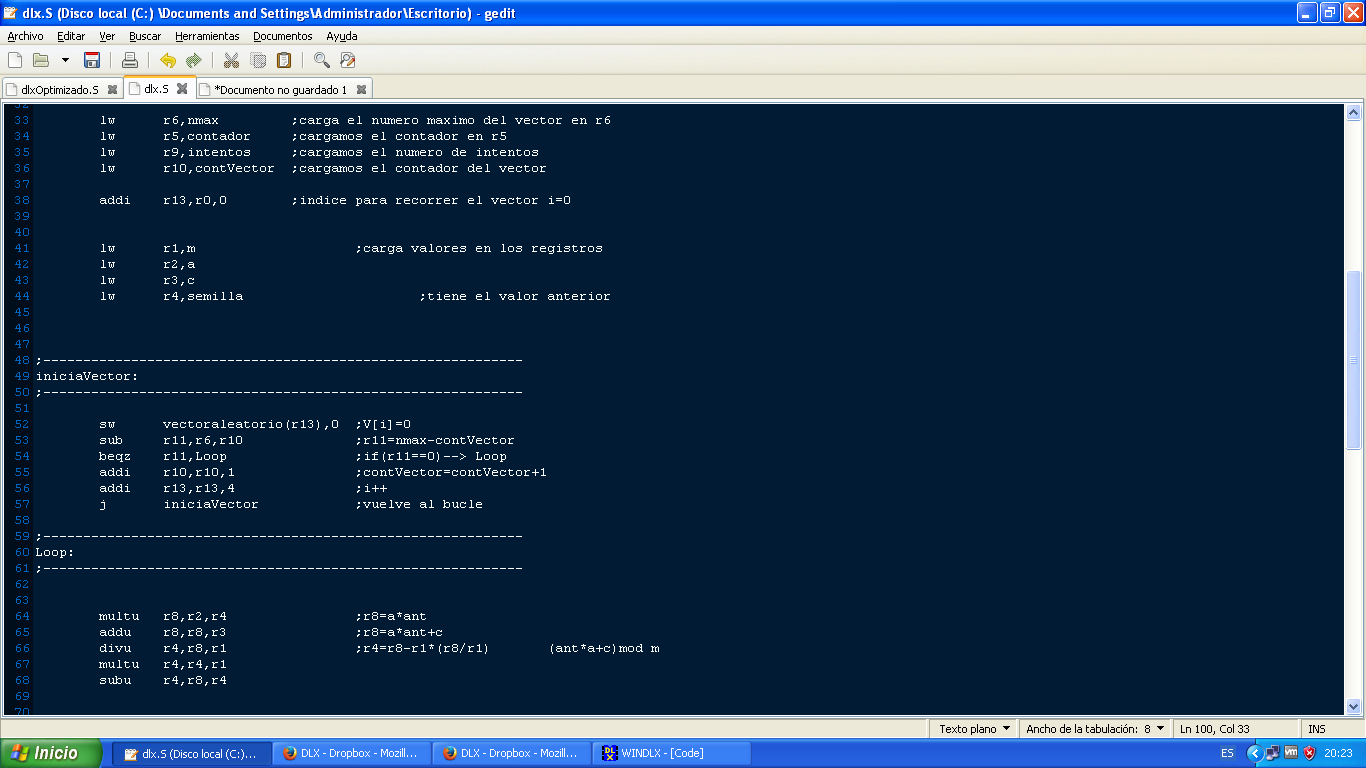
Lo primero que hicimos es una reducción de las instrucciones, quitando la parte del código de seguir ejecución y reordenando la misma para que esa parte quedara colocada y evitásemos así un salto innecesario.

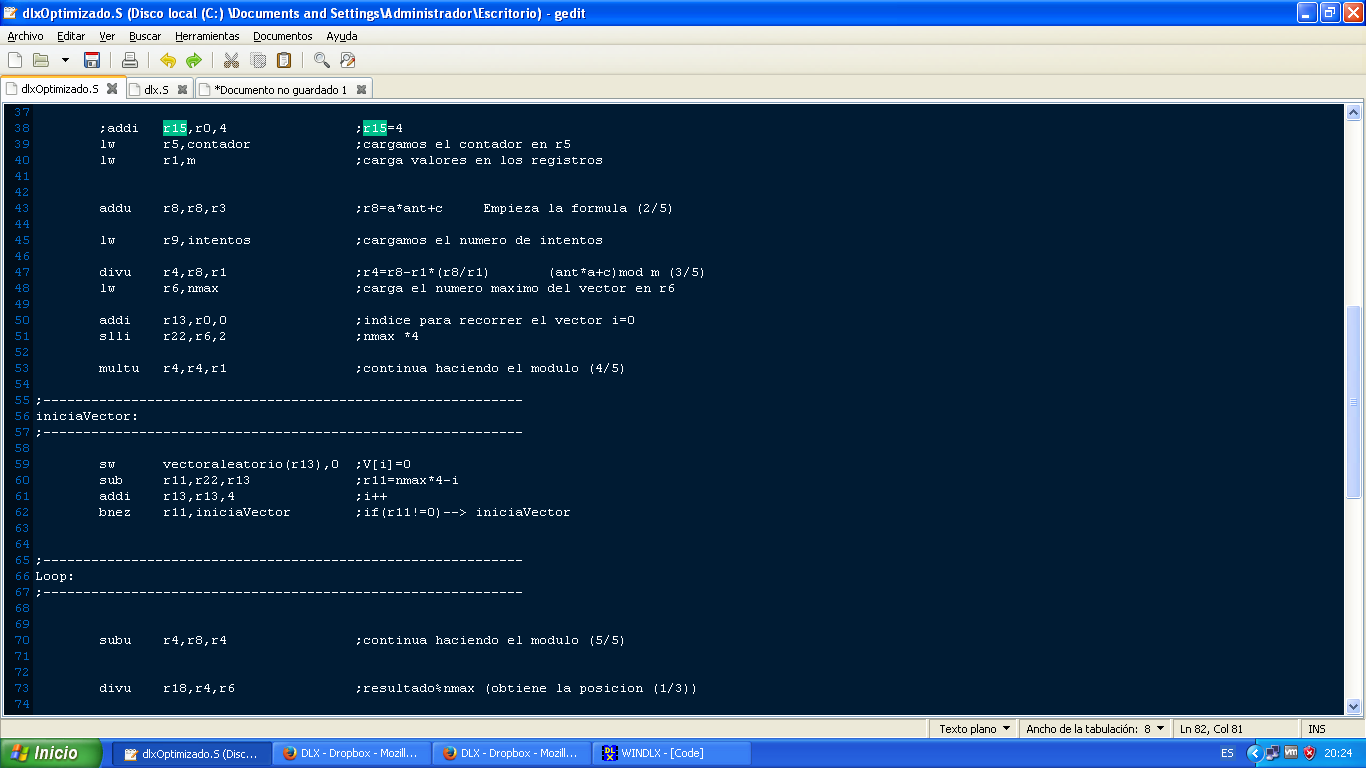
Lo segundo que hicimos fue una reducción de los saltos que hacía el programa, invirtiendo la lógica que habíamos usado en los condicionales y haciendo así que solo se saltase en aquellas ocasiones en las que fuera necesario. Podemos comprobar un ejemplo de lo mismo en las siguientes dos imágenes.





Se redujo el uso de contadores auxiliares innecesarios como en el caso de hacer la inicialización del vector en el cual se pasó de tener un contador auxiliar externo a utilizar las propias posiciones del vector para hacer la comprobación del bucle.



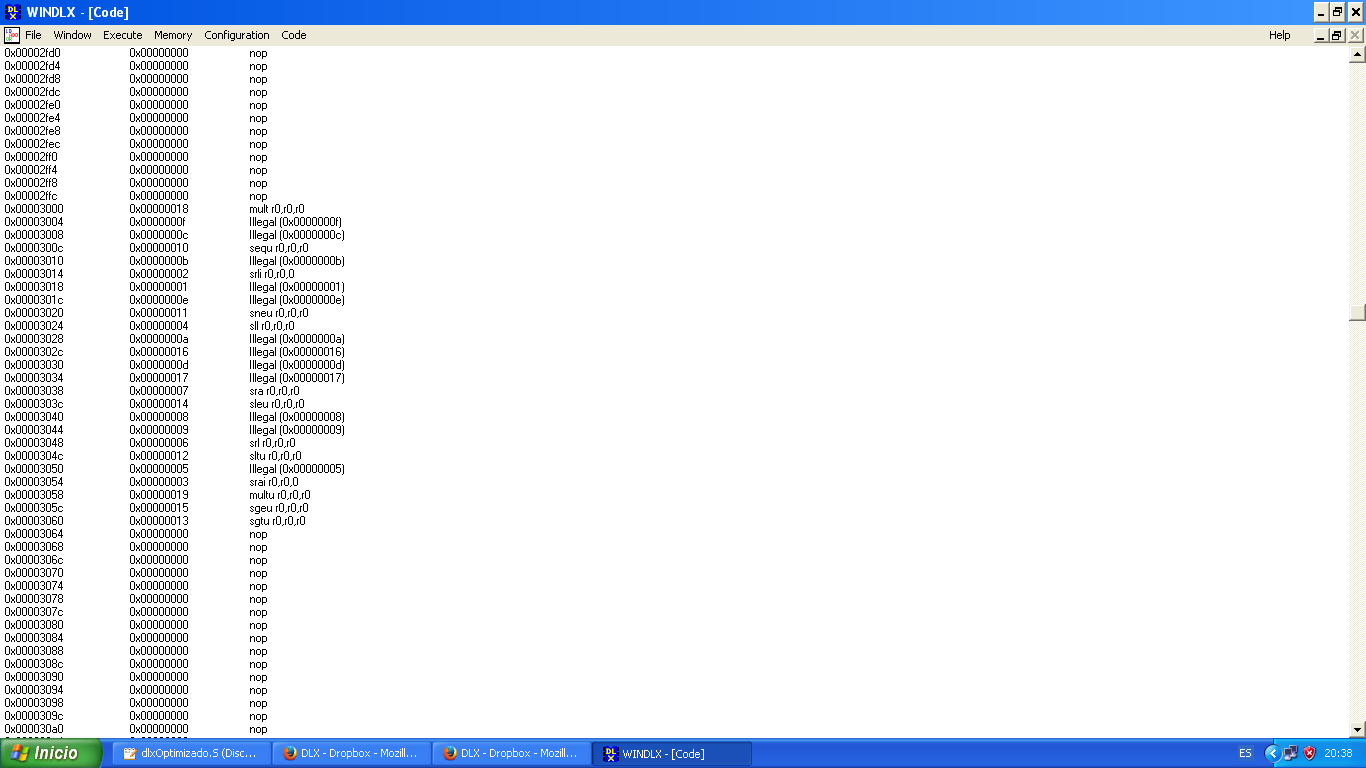


Se produjo una optimización también al sacar del bucle las operaciones de carga de los parámetros de entrada, puesto que estos parámetros solo van a ser leídos una vez a excepción del parámetro ant que guardará el resultado de la operación anterior pero cuyo resultado siempre va a residir en el registro 4. Por lo tanto no es necesaria la operación de carga de estos registros en cada iteración del bucle.

Se produjo una gran mejora también al recolocar instrucciones de manera que se evitases dependencias entre los elementos que estuvieran inmediatamente seguidos.

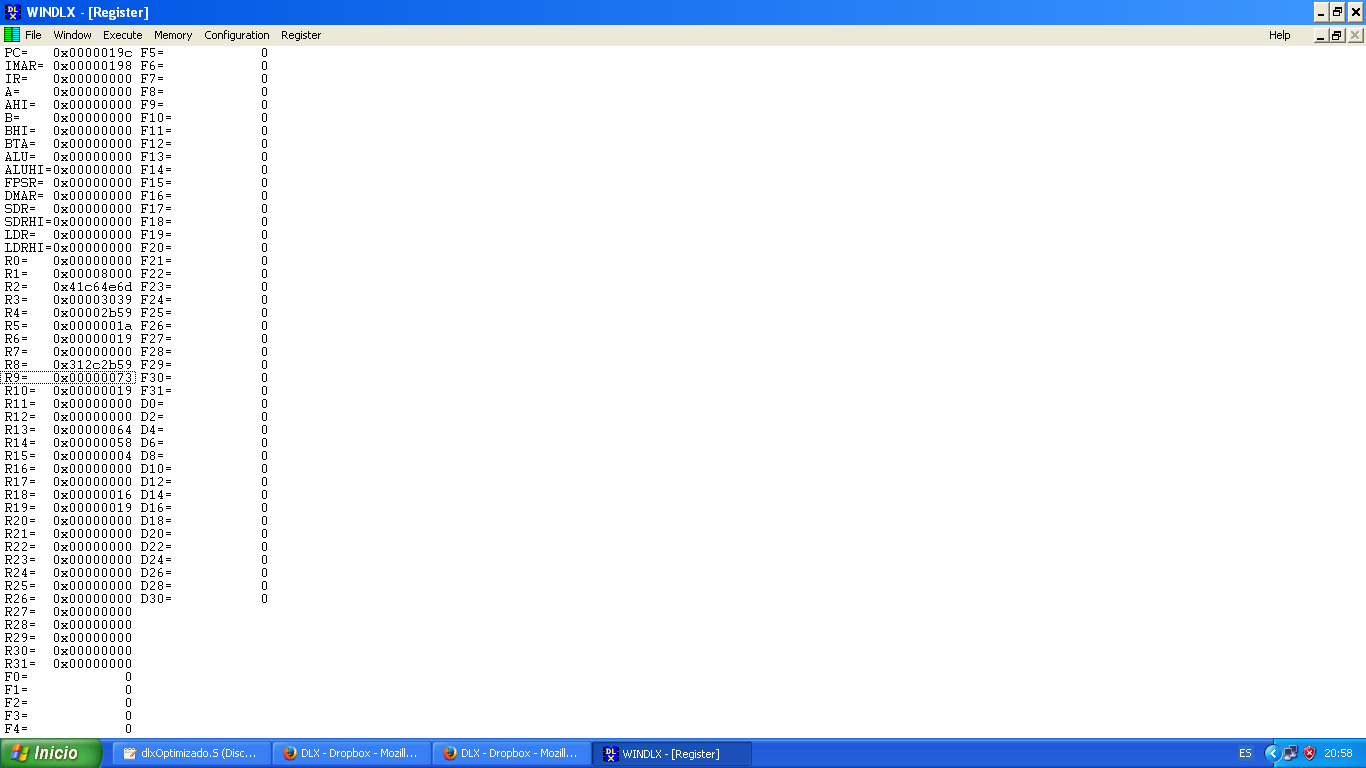
Y una de las mayores mejoras la obtuvimos de hacer un reconocimiento de las instrucciones tal que calculáramos en primera instancia antes de entrar al bucle el primer número aleatorio ya que, esto nos permitió poder intercalar una vez dentro del bucle operaciones propias de la generación del siguiente numero aleatorio con las propias de calcular el módulo de número actual. Esto es posible siempre y cuando se mantuvieran los valores del número hasta una vez calculado el numero aleatorio siguiente. Con esto conseguimos una gran mejora y una gran reducción de los ciclos del procesador debido a que era posible quitarnos muchas dependencias consecutivas entre multiplicaciones y divisiones que son las operaciones de mayor carga.

Y como consecuencia tenemos el siguiente vector:

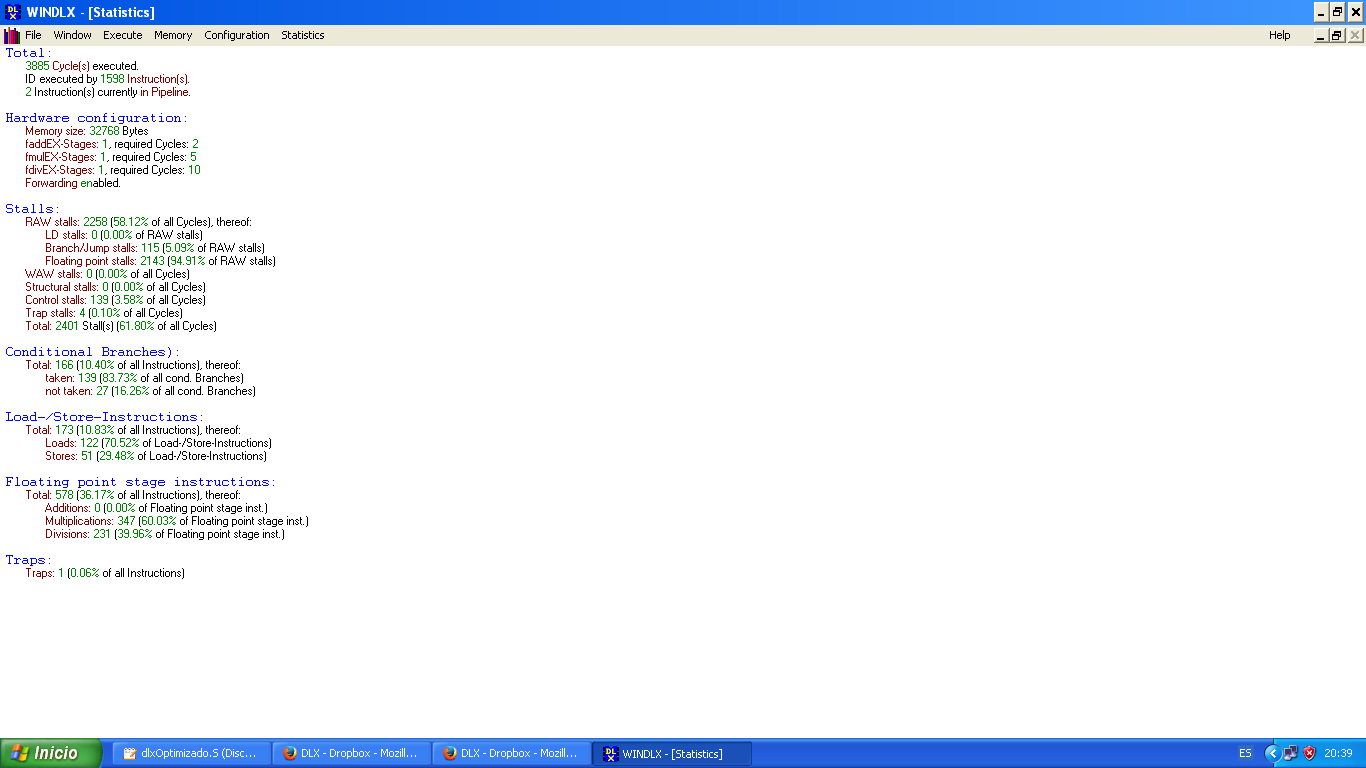


Que se comprueba fácilmente que es el mismo que se formaba en la versión sin optimizar.

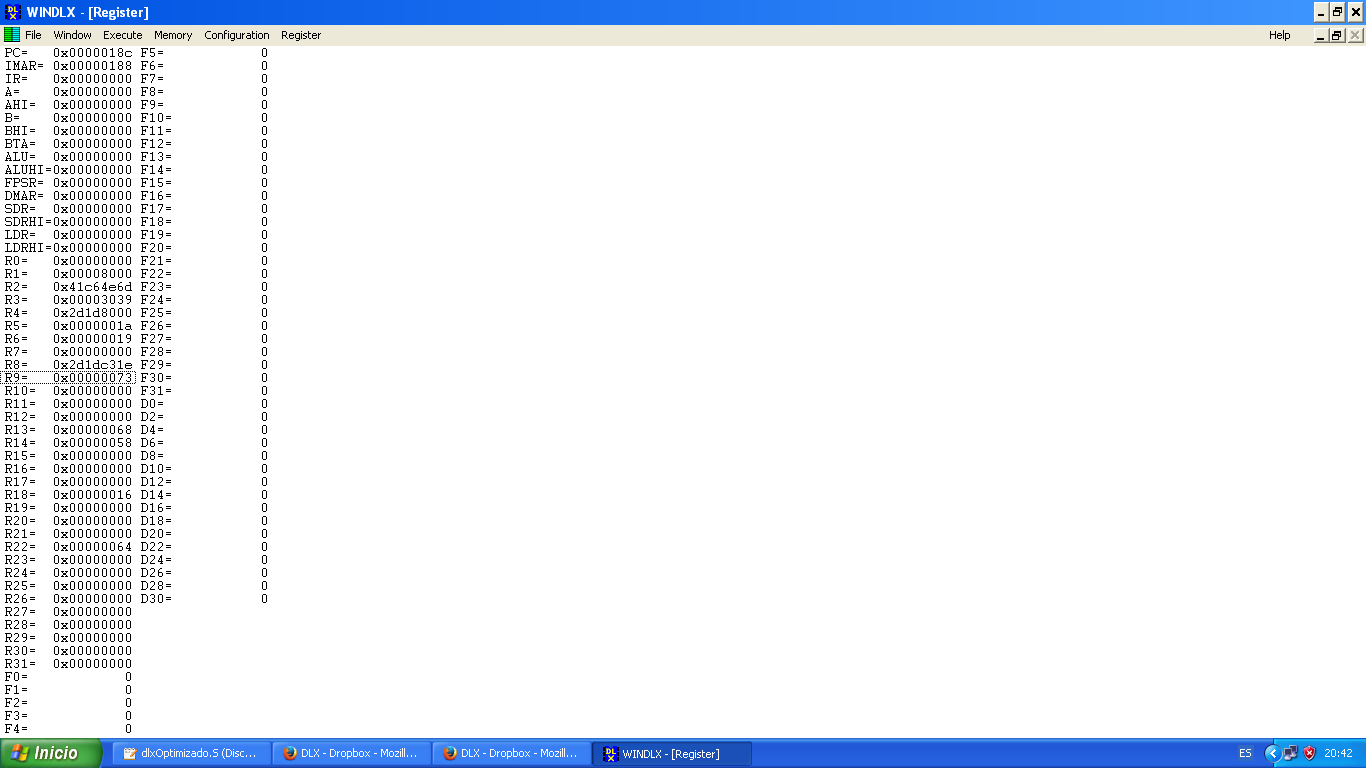
Cuyo número de intentos lo podemos comprobar que es 115 (73 en hexadecimal).



Y cuyas estadísticas son las siguientes:



Y como se puede comprobar se ha producido una mejora significativa en el rendimiento del programa siendo así que se ha conseguido una mejora de aproximadamente unos 3000 ciclos.



En cuanto al numero de intentos podemos comprobar como el registro r9 es que los lleva y se han producido unos 115 intentos.