Andrei Brinza X9842586N

Álvaro García García 45138930G

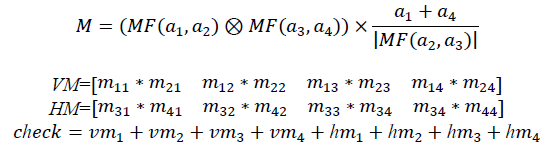
ARQUITECTURA DE COMPUTADORES

18 de mayo del 2021

# PRÁCTICA DLX: OPTIMIZACIÓN

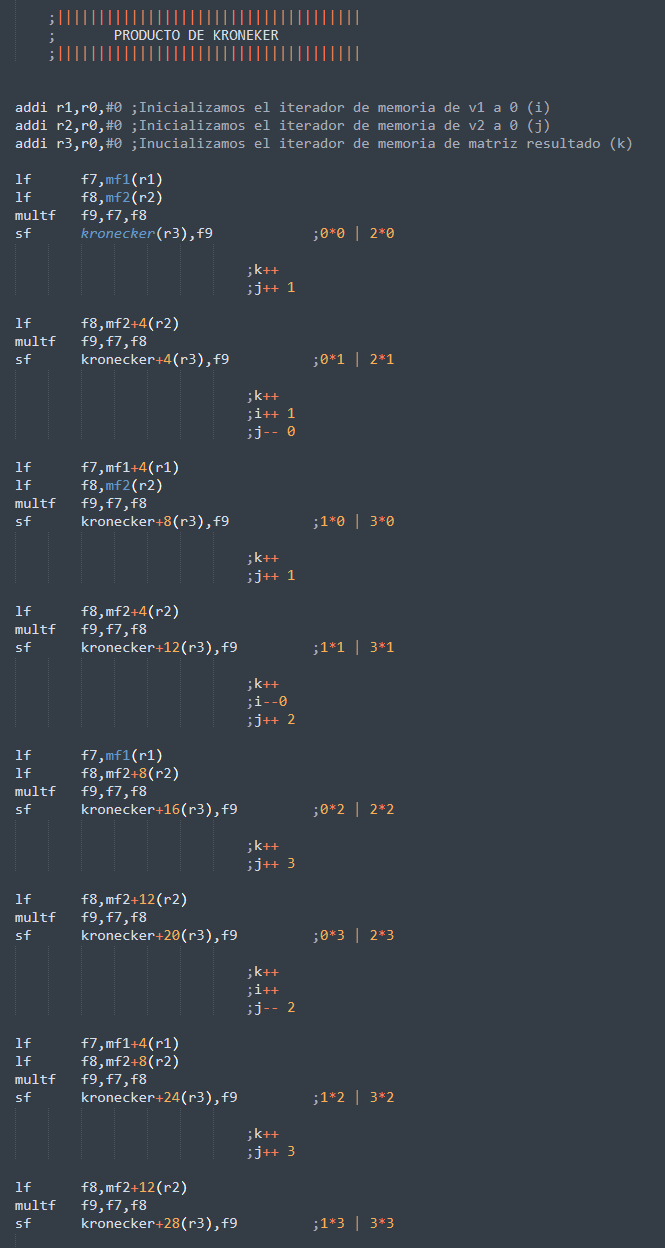
# Introducción

La práctica consistía en implementar y optimizar un código que tenía que realizar una serie de operaciones matemáticas

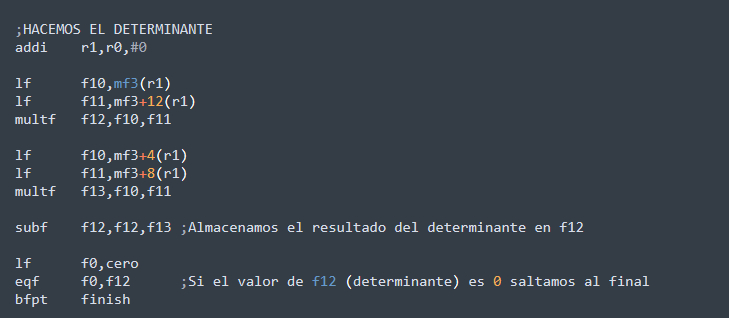


# Primera versión sin optimizar

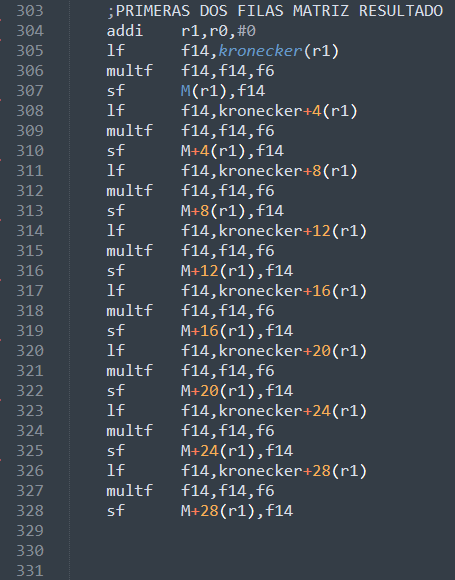
En la primera versión del código simplemente escribimos las instrucciones de forma que no usáramos ningún bucle para la multiplicación de matrices, en vez de eso usamos registros correspondientes a cada posición de la matriz. Cargamos en los registros float correspondientes y después realizamos las operaciones de multiplicación.



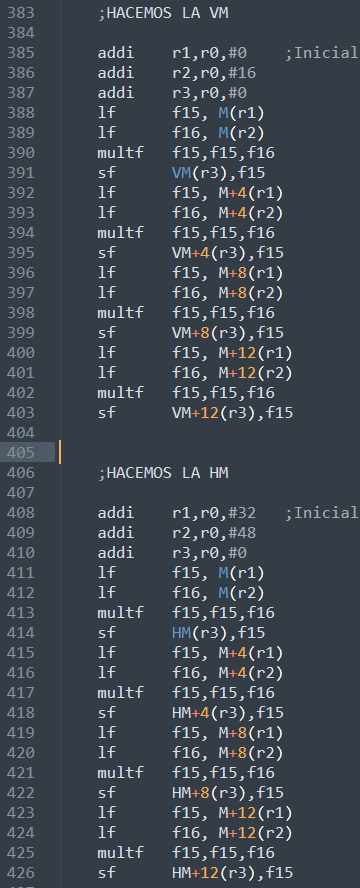
A continuación después de realizar el producto de kronecker procedemos a hacer la siguiente parte de la multiplicación. Aquí hacemos el determinante de la matriz comprobando que no hay división por cero y en caso de haberla acabamos con la ejecución.



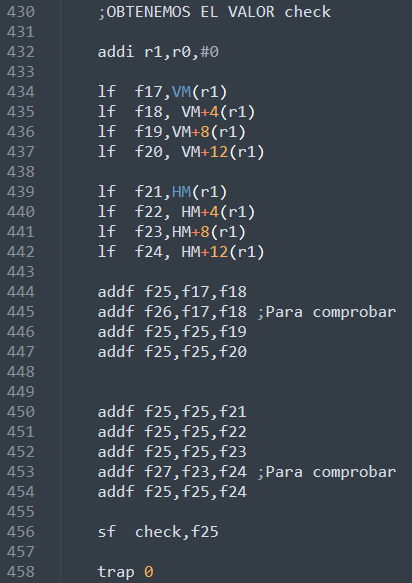
Después de tener las dos partes de la multiplicación se procede a calcular la matriz resultado M (4x4)



Una vez calculada la matriz M debemos comprobar los resultados de forma rápida , para ello usamos las matrices VM y HM que son sumas parciales de los elementos de la matriz M donde VM suma los elementos de las dos primeras filas y HM las dos últimas filas.



Finalmente calculamos el valor del check que será la suma de los elementos de las matrices VM y HM.

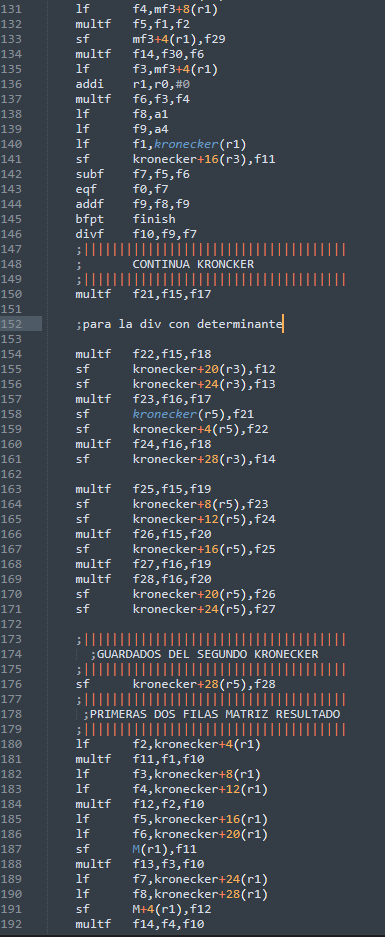


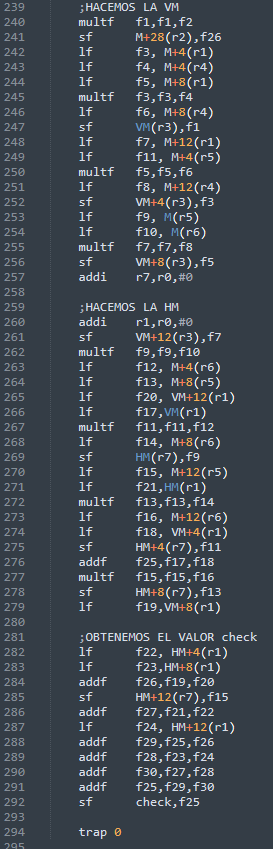
|  |  |
| --- | --- |
| **ESTADÍSTICAS** |  |
| **Total** |  |
| **Nº de ciclos:** | 527 |
| **Nº de instrucciones ejecutadas (IDs):** | 220 |
| **Stalls** |  |
| **RAW stalls:** | 298 (56.55%) |
| **Branch/Jump stalls:** | 0 (0.0%) |
| **Floating point stalls:** | 253 (85.00%) |
| **WAW stalls:** | 0 (0.0%) |
| **Structural stalls:** | 5 (0.95%) |
| **Trap stalls:** | 4 (0.76%) |
| **Total:** | 307 Stalls (58.25%) |
| **Conditional Branches** |  |
| **Total:** | 4 (1.82%) |
| **Tomados:** | 0 (0.0%) |
| **No tomados:** | 4 (100.0%) |
| **Instrucciones Load/Store** |  |
| **Total:** | 132 (60.00%) |
| **Loads:** | 77 (59.23%) |
| **Stores:** | 53 (40.77%) |
| **Instrucciones de punto floatante** |  |
| **Total:** | 60 (28.30%) |
| **Sumas:** | 11 (18.33%) |
| **Multiplicaciones:** | 45 (75.00%) |
| **Divisiones:** | 4 (6.67%) |
| **Traps** |  |
| **Traps:** | 1 (0.45%) |

En la primera versión la estadísticas obtenemos una cantidad considerable de stalls lo cual nos crea un total de 527 ciclos. La gran parte de los stalls son RAW stalls ya que las instrucciones están dispuestas secuencialmente y sin ordenar por tanto se producen muchas lecturas antes de las escrituras.

# Segunda versión optimizada

En la segunda versión del código procedemos a aplicar técnicas de optimización para poder aumentar el paralelismo y reducir los ciclos de nuestro código.





|  |  |
| --- | --- |
| **ESTADÍSTICAS** |  |
| **Total** |  |
| **Nº de ciclos:** | 282 |
| **Nº de instrucciones ejecutadas (IDs):** | 200 |
| **Stalls** |  |
| **RAW stalls:** | 18 (6.28) |
| **Branch/Jump stalls:** | 0 (0.00%) |
| **Floating point stalls:** | 17 (94.44%) |
| **WAW stalls:** | 0 (0.00%) |
| **Structural stalls:** | 56(19.86%) |
| **Trap stalls:** | 4 (1.42%) |
| **Total:** | 78 (27.66) |
| **Conditional Branches** |  |
| **Total:** | 4 (2.00%) |
| **Tomados:** | 0 (0.00%) |
| **No tomados:** | 4 (100.00%) |
| **Instrucciones Load/Store** |  |
| **Total:** | 118 (59.00%) |
| **Loads:** | 65 (55.08%) |
| **Stores:** | 53 (44.92%) |
| **Instrucciones de punto floatante** |  |
| **Total:** | 58 (29.00%) |
| **Sumas:** | 9 (15.52%) |
| **Multiplicaciones:** | 45 (77.60%) |
| **Divisiones:** | 4 (7.00%) |
| **Traps** |  |
| **Traps:** | 1 (0.50%) |

En las estadísticas de la versión optimizada vemos una considerable mejora en los ciclos debido a la reducción de stalls.

Para ello lo primero que hacemos es usar distintos registros para cada una de las operaciones , esto eliminaría la mayoría de RAW stalls junto al reordenamiento de las instrucciones de carga.

Además de esto intentamos aprovechar los ciclos en los que realizamos operaciones de división y multiplicación que son las que más que ciclos consumen y metemos instrucciones entre estas operaciones para aprovechar el paralelismo y reducir los stalls estructurales. A pesar de esto aún tenemos una cantidad considerable de stalls estructurales ya que las multiplicaciones no tenían suficiente separación en el tiempo.

# 4. Conclusión

Hemos conseguido optimizar el código aplicando una serie de procedimientos para reordenar el código de forma coherente. Aun así no hemos conseguido el máximo valor de optimización debido a que hemos desaprovechado algunos ciclos de multiplicaciones y en comprobaciones.

# 