Práctica DLX Sucesión de Fibonacci

VNiVERSiDAD D SALAMANCA

Enrique Mesonero Ronco 52417500V
Guillermo Pascual Mangas 70911551K

Índice

Versión no optimizada

Para la realización de este ejercicio, se han utilizado los valores de entrada dados en el enunciado de la práctica. Además, se han añadido una serie de valores útiles para la realización del ejercicio, floats 0,1,10,4 (no optimizado) y floats 0,1,10,0.25,f_i, f_ia (optimizado, 0.25 en vez de 4 y f_i, f_ia para la carga/guardado de los valores de la sucesión de fibonacci).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; VARIABLES DE ENTRADA: NO MODIFICAR
; valor inicial para la secuencia (de 1.0 a 5.0)
valor_inicial:  .float  5
; Tamanho de la secuencia (multiplo de 5 minimo 10 maximo 30)
tamanho:        .word   10

;;;;; VARIABLES DE SALIDA: NO MODIFICAR ORDEN (TODAS FORMATO FLOAT)
vector:         .space  120
suma:           .float  0
; m11=vector[5], m12=vector[6]
; m21=vector[7], m22=vector[8]
M:              .float 0.0, 0.0
                .float 0.0, 0.0
detM:           .float 0.0
mediaM:         .float 0.0
; v11=m11/mediaM, v12=m12/mediaM
; v21=m21/mediaM, v22=m22/mediaM
V:              .float 0.0, 0.0
                .float 0.0, 0.0
detV:           .float 0.0
mediaV:         .float 0.0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;VALORES INICIALES
        f_0:            .float  0
        f_1:            .float  1
        f_10:           .float  10
        f_4:            .float  4
```

```
;VALORES INICIALES
        f_0:            .float  0
        f_1:            .float  1
        f_10:           .float  10
        f_4:            .float  0.25
        f_i:            .float  0
        f_ia:           .float  0
                .text
                .global main
```

Se realiza la operación de fibonacci en un bucle, se realiza una comprobación para saber si son los 2 primeros valores para cargar 0 y el valor inicial, una vez acabada la sucesión, gracias a un valor r2, que es el índice de la sucesión, comprobamos que hemos acabado y saltamos al apartado de matrices, donde realizaremos la matriz, calculamos su determinante y su media, a partir del determinante calculamos la matriz V, su determinante y su media, y con esto acabamos el programa.

Cabe destacar, que se realiza una comprobación de división entre 0 (en caso de que detM sea 0), ya que es la unica división con posibilidad de ser 0.

ESTADÍSTICAS	
Total	
Número total de ciclos	Tam 10: 373 | Tam 30: 753
Número total de instrucciones ejecutadas (IDs)	Tam 10: 180 | Tam 30: 440
Stalls	
RAW Stalls	Tam 10: 76 (20.38% AC) | Tam 30: 96 (12.75% AC)
LD stalls	Tam 10: 0 (0% ARAW) | Tam 30: 0 (0% ARAW)
Branch/Jump stalls	Tam 10: 11 (14.47% ARAW) | Tam 30: 31 (32.29% ARAW)
Floating point stalls	Tam 10: 65 (85.53% ARAW) | Tam 30: 65 (67.71% ARAW)
WAW stalls	Tam 10: 0 (0% AC) | Tam 30: 0 (0% AC)
Structural stalls	Tam 10: 91 (24.40% AC) | Tam 30: 151 (20.05% AC)
Control stalls	Tam 10: 13 (3.48% AC) | Tam 30: 33 (4.38% AC)
Trap stalls	Tam 10: 2 (0.54% AC) | Tam 30: 2 (0.26% AC)
Total	Tam 10: 182 (48.79% AC) | Tam 30: 282 (37.45% AC)
Conditional Branches	
Total	Tam 10: 22 (12.22% AI) | Tam 30: 62 (14.09% AI)
Tomados	Tam 10: 2 (9.09% AB) | Tam 30: 2 (3.22% AB)
No tomados	Tam 10: 20 (90.91% AB) | Tam 30: 60 (96.77% AB)
Instrucciones Load/Store	
Total	Tam 10: 36 (20.00% AI) | Tam 30: 56 (12.73% AI)
Loads	Tam 10: 13 (36.11% ALS) | Tam 30: 13 (23.21% ALS)
Stores	Tam 10: 23 (64.00% ALS) | Tam 30: 43 (76.78% ALS)
Instrucciones de punto flotante	
Total	Tam 10: 55 (30.56% AI) | Tam 30:135 (30.68% AI)
Sumas	Tam 10: 45 (81.82% AF) | Tam 30: 125 (92.59% AF)
Multiplicaciones	Tam 10: 4 (7.27% AF) | Tam 30: 4 (2.96% AF)
Divisiones	Tam 10: 6 (10.91% AF) | Tam 30: 6 (4.44% AF)
Traps	
Traps	Tam 10: 1 (0.56% AI) | Tam 30: 1 (0.23% AI)

Versión optimizada

Para realizar la versión optimizada del programa DLX, se han realizado dos tipos de optimizaciones, una de ella, puramente matemática, la otra, centrada en el código (reordenación de código, renombre de registros, etc…)

Optimización matemática:
Para la optimización matemática, hemos decidido multiplicar por 0.25 aquellas operaciones que requieren dividir entre 4 (las medias de las matrices) y realizar una sola división de 1 entre detM, para calcular la matriz matrizV, multiplicando este resultado por los valores m11, m12, m21 y m22, esto con el objetivo de ahorrar ciclos reduciendo el número de divisiones que vamos a realizar (pasando de 6 divisiones a 1).

Optimización código ensamblador:

Para la optimización del código, se han realizado múltiples procesos:

- Primero de todo, se han cargado en el vector los valores vector[0] y vector[1] de forma manual, para evitar una de las comprobaciones y saltos.
- Segundo, se ha desenrollado el bucle de fibonacci, realizando cinco operaciones por bucle (ya que los valores del número de elementos deben ser múltiplos de cinco) en lugar de una.
- Para evitar un elevado número de stalls, lo que se ha hecho es colocar operaciones de load y save entre operaciones de suma, multiplicación y la división, aunque aún quedan un número elevado de stalls entre operaciones debido al gran número de multiplicaciones que deben realizarse para realizar los cálculos requeridos por el ejercicio de forma correcta.
- Otros pequeños cambios son cambiar la forma de calcular los valores de la sucesión de fibonacci (en vez de realizar sumas,

se ha cambiado por operaciones load/save), y la eliminación de la última comprobación para terminar el programa.

ESTADÍSTICAS	
Total	
Número total de ciclos	Tam 10: 211 | Tam 30: 455
Número total de instrucciones ejecutadas (IDs)	Tam 10: 148 | Tam 30: 352
Stalls	
RAW Stalls	Tam 10: 27 (12.80% AC) | Tam 30: 27 (5.93% AC)
LD stalls	Tam 10: 1 (3.70% ARAW) | Tam 30: 1 (3.70% ARAW)
Branch/Jump stalls	Tam 10: 0 (0% ARAW) | Tam 30: 0 (0% ARAW)
Floating point stalls	Tam 10: 26 (96.30% ARAW) | Tam 30: 26 (96.30% ARAW)
WAW stalls	Tam 10: 0 (0% AC) | Tam 30: 0 (0% AC)
Structural stalls	Tam 10: 12 (5.70% AC) | Tam 30: 12 (2.64% AC)
Control stalls	Tam 10: 2 (0.95% AC) | Tam 30: 6 (1.32% AC)
Trap stalls	Tam 10: 7 (3.32% AC) | Tam 30: 7 (1.54% AC)
Total	Tam 10: 48 (22.75% AC) | Tam 30: 52 (11.43% AC)
Conditional Branches	
Total	Tam 10: 11 (7.43% AI) | Tam 30: 31 (8.81% AI)
Tomados	Tam 10: 1 (9.09% AB) | Tam 30: 1 (3.22% AB)
No tomados	Tam 10: 10 (90.91% AB) | Tam 30: 30 (96.77% AB)
Instrucciones Load/Store	
Total	Tam 10: 65 (43.92% AI) | Tam 30: 145 (41.19% AI)
Loads	Tam 10: 32 (49.23% ALS) | Tam 30: 72 (49.66% ALS)
Stores	Tam 10: 33 (50.77% ALS) | Tam 30: 73 (50.34% ALS)
Instrucciones de punto flotante	
Total	Tam 10: 38 (25.68% AI) | Tam 30:78 (22.16% AI)
Sumas	Tam 10: 27 (71.05% AF) | Tam 30: 67 (86.00% AF)
Multiplicaciones	Tam 10: 10 (26.32% AF) | Tam 30: 10 (12.82% AF)
Divisiones	Tam 10: 1 (2.63% AF) | Tam 30: 1 (1.28% AF)
Traps	
Traps	Tam 10: 1 (0.68% AI) | Tam 30: 1 (0.28% AI)