VNiVERSiDAD
D SALAMANCA
Facvltad D Ciencias

Procesador RISC: DLX

Asignatura: Arquitectura de Computadores

Práctica: Práctica 2. DLX

Autores: De La Peña Ramos, Jaime
Servate Hernández, Javier

Año académico: 2018 – 2019

Índice

1. Descripción de la práctica 3

2. Tabla de cambios.....5

3. Planteamiento versión no optimizada – “sinOptimizar.s”.....6

4. Resultados versión no optimizada – “sinOptimizar.s”.....9

5. Planteamiento versión optimizada – “conOptimizacion.s”....12

6. Resultados versión optimizada – “conOptimizacion.s” 14

7. Conclusiones finales.....16

1.Descripción de la práctica

El objetivo de la práctica es el desarrollo y posterior optimización de un código en lenguaje ensamblador que realice el siguiente cálculo.

$$M = (A \times B) \bullet \frac{1}{|A + B|} + C \bullet \alpha$$

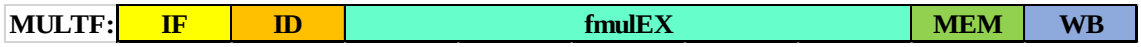
A continuación, se procede a describir cada uno de los elementos de la expresión anterior:

- A, B, C y M matrices 3x3
- α número real
- | A | determinante de la matriz
- "×" , "+" producto y suma de matrices
- "•" producto de matriz por escalar

Tanto los datos de entrada, como los datos de salida en DLX deben ser los siguientes y no puede ser modificado el orden de estos

```
A:          .float      1.500000,     2.750000,     3.257000
            .float      21.002000,    2.658000,     2.157000
            .float      56.251000,    3.154000,     3.255000

B:          .float      3.500000,     3.500000,     2.500000
            .float      4.500000,     4.500000,     6.500000
            .float      5.500000,     1.500000,     2.500000

C:          .float      15.000000,    17.000000,    14.000000
            .float      17.000000,    15.000000,    17.000000
            .float      14.000000,    17.000000,    15.000000

Alfa:       .float      1.235

M:          .float      0.0,0.0,0.0
            .float      0.0,0.0,0.0
            .float      0.0,0.0,0.0
```

Los ciclos de reloj de las operaciones utilizadas son los siguientes:



2.Tabla de cambios

Fecha	Modificación	Autor/es
13/04/2019	Realizada versión sin optimizar V 1.0 (389 ciclos)	Jaime de la Peña
19/04/2019	Revisión versión sin optimizar V 2.0 (389 ciclos)	Jaime de la Peña y Javier Servate
21/04/2019	Comienzo versión optimizada V 1.0 (377 ciclos)	Jaime de la Peña
22/04/2019	Continuación versión optimizada V 2.0 (335 ciclos)	Jaime de la Peña
23/04/2019	Continuación versión optimizada V 3.0 (326 ciclos)	Jaime de la Peña
24/04/2019	Continuación versión optimizada V 4.0 (323 ciclos)	Jaime de la Peña y Javier Servate
25/04/2019	Continuación versión optimizada V 5.0 (307 ciclos)	Jaime de la Peña
26/04/2019	Continuación versión optimizada V 6.0 (302 ciclos)	Jaime de la Peña
27/04/2019	Continuación versión optimizada V 7.0 (295 ciclos)	Jaime de la Peña
29/04/2019	Continuación versión optimizada V 8.0 (294 ciclos)	Jaime de la Peña
01/05/2019	Continuación versión optimizada V 9.0 (285 ciclos)	Jaime de la Peña
08/05/2019	Continuación versión optimizada V 10.0 (255 ciclos)	Jaime de la Peña
09/05/2019	Elaboración del guión	Jaime de la Peña

3.Planteamiento versión no optimizada – “sinOptimizar.s”

Para la realización de esta primera versión, cuyo fichero tendrá el nombre “sinOptimizar.s” se han seguido los siguientes pasos. Por último, añadir, que todas las operaciones que se realizarán a lo largo del proceso se han realizado a mano en una hoja que será presentada al profesor el día de la defensa de esta práctica:

1. En primer lugar, se carga en los registros del F0 al F17 los elementos de las matrices A y B.

2. A continuación, se suman ambas matrices, el método es ir sumando los elementos que se encuentren en la misma fila y columna de las matrices, así $(A+B)_{11} = A_{11} + B_{11}$.

3. El siguiente paso es calcular del determinante | A + B | resultante de la suma de las matrices de A y B. Para realizar esto se va a utilizar el método de cálculo de determinante mediante Sarrus, el cual tiene la siguiente expresión:

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}$$

Det (A) = | A | = $(a_{11} \cdot a_{22} \cdot a_{33} + a_{12} \cdot a_{23} \cdot a_{31} + a_{21} \cdot a_{32} \cdot a_{13})$ - - $(a_{13} \cdot a_{22} \cdot a_{31} + a_{12} \cdot a_{21} \cdot a_{33} + a_{23} \cdot a_{32} \cdot a_{11})$

4. Una vez calculado el determinante, se comprueba si este es 0 o no, en caso de ser 0 se realiza un salto para terminar el programa ya que no se debe realizar la división por ser esta una indeterminación. En caso de no ser 0 se realiza la división del determinante entre 1 de forma normal.

5. Tras calcular el determinante, se calcula la multiplicación de las matrices A y B, como ambas están cargadas en los registros iniciales simplemente se realiza la multiplicación de estas.

   Para ello se va a seguir la siguiente expresión. Sea A y B dos matrices y C el producto de ambas:

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix} \quad B = \begin{matrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{matrix} \quad C = \begin{matrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{matrix}$$

Para un elemento general: $c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + a_{i3} \cdot b_{3j} = \sum_{k=1}^{3} a_{ij} \cdot b_{kj}$

6. Tras esto, se realiza el producto de la matriz resultante del producto de A y B por el resultado obtenido de dividir el determinante de |A + B| entre 1. Es decir, a continuación, se realiza la siguiente expresión.

$$(A \; x \; B) \cdot \frac{1}{|\, A + B \,|}$$

7. A continuación, se carga en los registros F0 al F8 la matriz C y en el registro F9 el factor alfa.

8. Tras esto, se realiza el producto de una matriz, es decir C, por un escalar, es decir alfa. Por lo que la matriz resultante tendría la siguiente forma:

$$C \cdot \alpha = \begin{matrix} c_{11} \cdot \alpha & c_{12} \cdot \alpha & c_{13} \cdot \alpha \\ c_{21} \cdot \alpha & c_{22} \cdot \alpha & c_{23} \cdot \alpha \\ c_{31} \cdot \alpha & c_{32} \cdot \alpha & c_{33} \cdot \alpha \end{matrix}$$

9. A continuación, se suman las matrices resultantes de los pasos 6 y 8.

10. Tras esto, se carga en la matriz M los elementos uno por uno obtenidos de la suma de las matrices del paso 9.

11. Finalmente, finaliza la ejecución del programa.

4.Resultados versión no optimizada – “sinOptimizar.s”

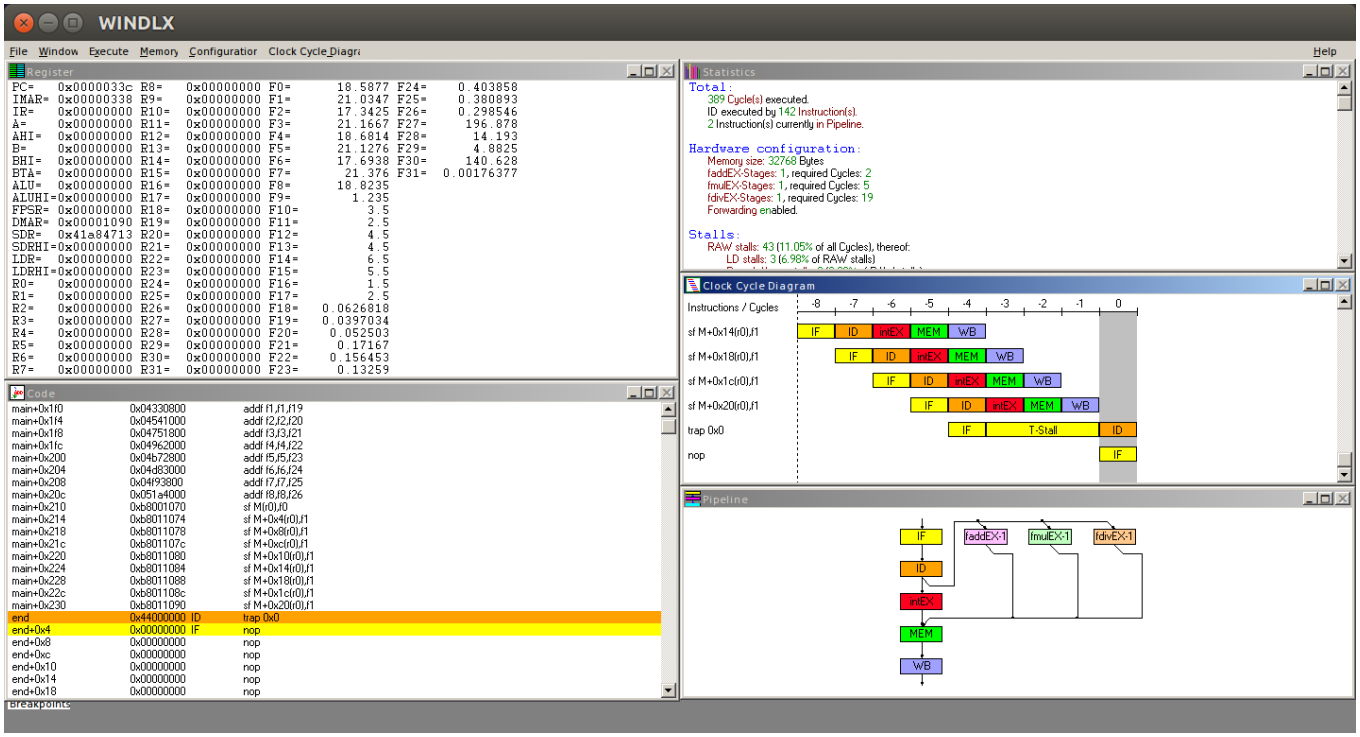


Figura 1

En la **Figura 1** puede verse una visión de la ejecución terminada del programa “sinOptimizar.s” donde puede verse el contenido de los registros, las estadísticas del programa, los ciclos de reloj, etc.

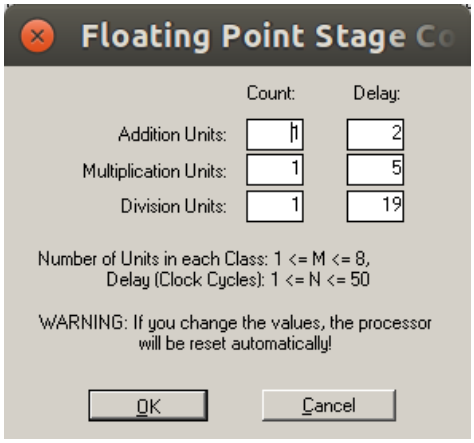


Figura 2

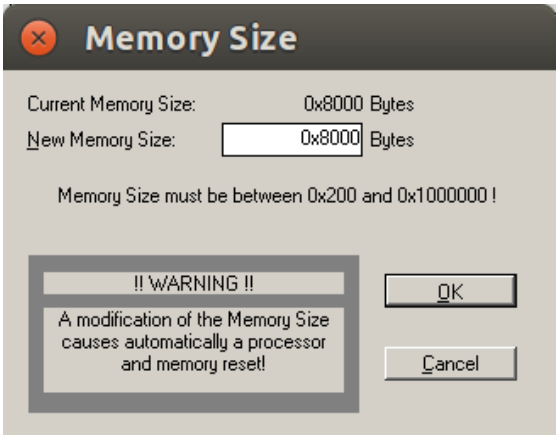


Figura 3

En la **Figura 2** y **Figura 3** puede verse la configuración empleada en el simulador DLX, esta configuración es obligatoria y hay que utilizarla en la ejecución de ambas versiones, la versión optimizada y la no optimizada.

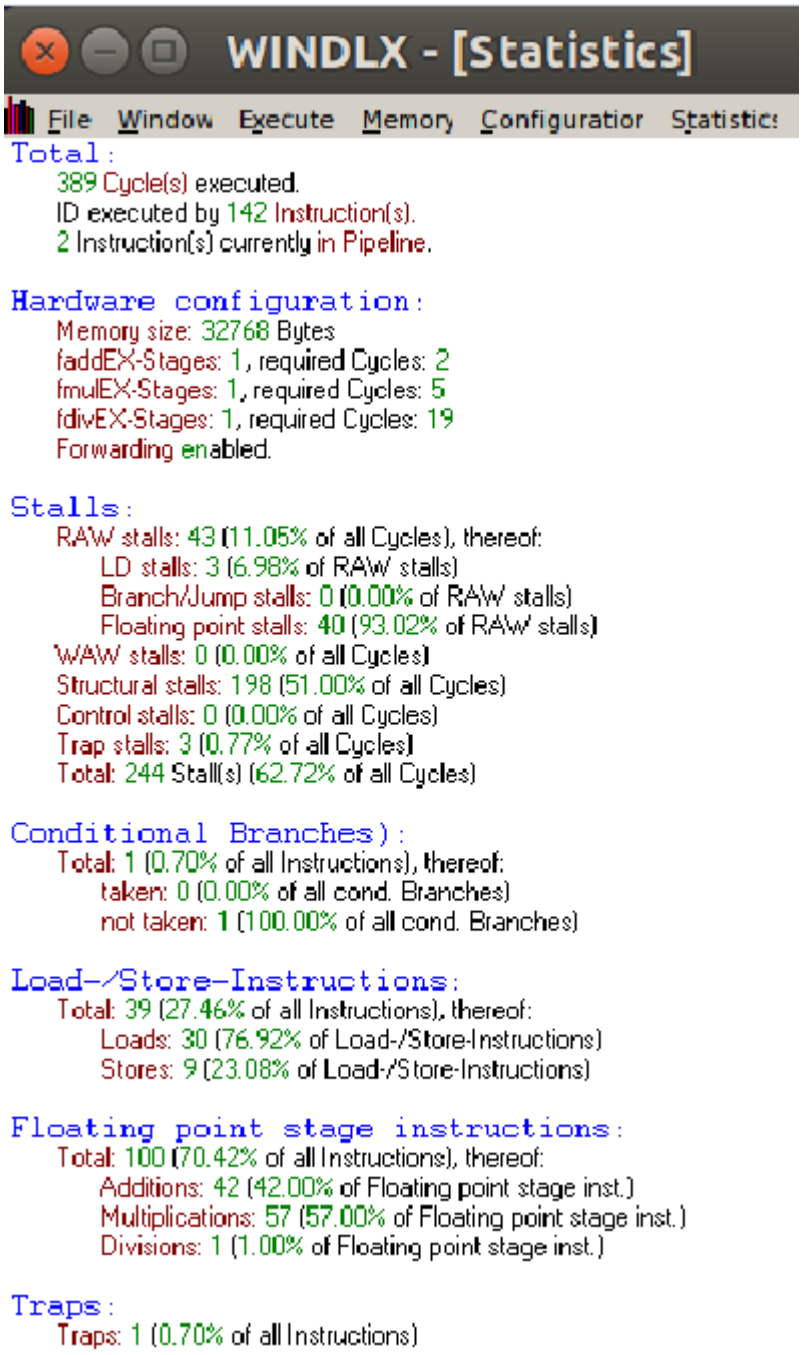


Figura 4

Finalmente, en la **Figura 4** se muestra un resumen de las estadísticas de esta versión del programa, se recuerda que es la versión no optimizada.

5.Planteamiento versión optimizada – "conOptimizacion.s"

Para la realización de esta segunda versión, la versión optimizada,cuyo fichero tendrá el nombre "conOptimizacion.s" se han seguido los siguientes pasos:

1. En primer lugar, es necesario dejar constancia que la forma de realizar la operación es la misma que en la versión anterior con la diferencia que se van a mezclar los pasos para sacar el mayor rendimiento posible

2. La siguiente forma de optimización es combinar las distintas operaciones que hay que realizar (carga de elementos, sumas, restas, multiplicaciones…) De forma que las operaciones sean los más independientemente posibles unas de otras, porque en caso de que no se haga eso se dará una parada conocida como *RAW_STALL*, por ejemplo, no es óptimo realizar el siguiente cálculo: **OP 1**: a = b · c y la operación siguiente sea **OP 2**: d = a + 5, porque **OP 2** tiene que esperar a que se finalice la **OP 1**. Por lo que se va a tratar de realizar las operaciones lo más independiente posibles

3. Otra forma de optimización que se va a realizar es evitar las paradas conocidas como *STRUCTURAL_STALL* estas paradas se deben a los recursos disponibles y los procesos que utilizan estos recursos, es decir, en un momento de tiempo X solo puede haber una operación utilizando, por ejemplo, la operación de multiplicación multf, así en este momento X y hasta que termine la multiplicación no debería haber más operaciones que pidan este recurso ya que deberán esperar a que la anterior operación que lo esté usando termine.

```
; PASO 1: SUMA DE MATRICES A + B Y CÁLCULO DEL DETERMINANTE MEDIANTE LAPLACE

            lf  f4,A+16
            lf  f13,B+16
            lf  f8,A+32
            addf f22, f4, f13; a22
            multf f29, f4, f13 ; EMPIEZA LAS OPERACIONES DE MULTIPLICACION A X B
            lf  f17,B+32
            lf  f5,A+20
            lf  f14,B+20
            addf f26, f8, f17 ; a33
            lf  f7,A+28
```

Figura 5

En la **Figura 5** puede verse un ejemplo de lo realizado en los pasos 2 y 4, es decir, las operaciones similares se intentan separar lo máximo posible para generar las menores paradas de tipo *STRUCTURAL_STALL* y, por otro lado, también se reorganiza el código para que las operaciones sean lo más independientes posibles y que no se produzcan esperas de tipo *RAW_STALL*.

4. Otra forma en la cual también se ha optimizado el programa es en las paradas producidas por la espera de un dato, es decir las *LD_STALL*, *Load Stall*, estas paradas son producidas porque hay una instrucción que necesita de un dato, pero dicho dato aún no ha sido cargado en ningún registro, por tanto, es necesario esperar a que esté disponible.

5. Adicionalmente, se añade que no se podría mejorar las paradas producidas por la operación "trap", estas paradas son conocidas como *TRAP_STALL* y aparentemente son las únicas que no se podrían mejorar.

6. Por último, como ya se ha explicado anteriormente, hay muchas operaciones que implican la utilización de recurso de la multiplicación, por tanto en vez de realizar operaciones de multiplicaciones se ha pensado en utilizar la división, ya que se divide el resultado de multiplicar una matriz por 1, el cual es la misma matriz y después dividirlo por el determinante de A + B, de esta forma se ahorra utilizar multiplicaciones y se meten las divisiones de tal forma que se evite los *RAW_STALL*, *STRUCTURAL_STALL* y *WAW_STALL*.

6.Resultados versión optimizada – “conOptimizacion.s”

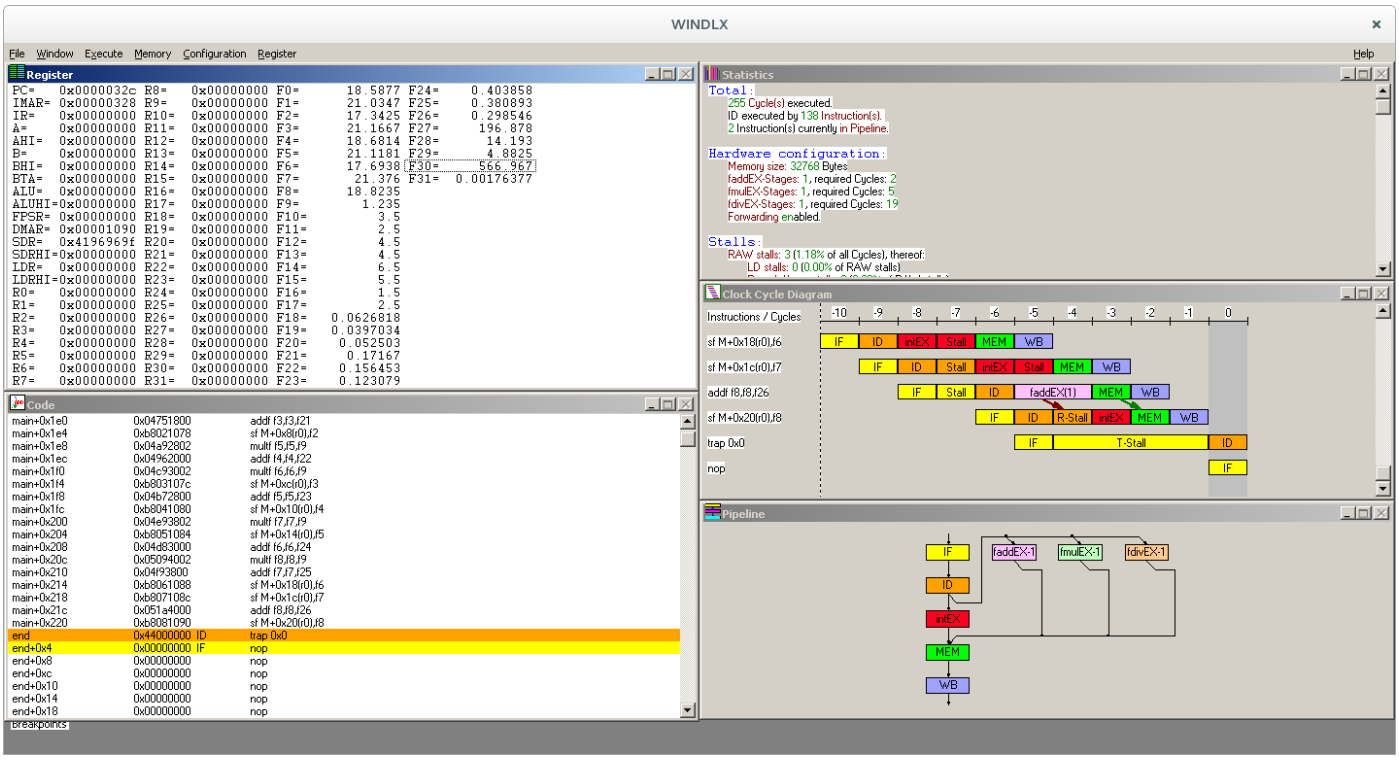


Figura 6

En la **Figura 6** puede verse una visión de la ejecución terminada del programa “conOptimizacion.s” donde puede verse el contenido de los registros, las estadísticas del programa, los ciclos de reloj, etc.

Figura 7

Finalmente, en la **Figura 7** se muestra un resumen de las estadísticas de esta versión del programa, se recuerda que es la versión optimizada.

7.Conclusiones finales

ESTADÍSTICAS	SIN OPTIMIZAR	CON OPTIMIZACIÓN
Total		
Nº de ciclos:	**389**	**255**
Nº de instrucciones ejecutadas (IDs):	**142**	**138**
Stalls		
RAW stalls:	**43**	**3**
LD stalls:	**3**	**0**
Branch/Jump stalls:	**0**	**0**
Floating point stalls:	**40**	**3**
WAW stalls:	**0**	**0**
Structural stalls:	**198**	**107**
Control stalls:	**0**	**0**
Trap stalls:	**3**	**4**
Total:	**244**	**114**
Conditional Branches		
Total:	**1**	**1**
Tomados:	**0**	**0**
No tomados:	**1**	**1**
Instrucciones Load/Store		
Total:	**39**	**39**
Loads:	**30**	**30**
Stores:	**9**	**9**
Instrucciones de punto flotante		
Total:	**100**	**96**
Sumas:	**42**	**41**
Multiplicaciones:	**57**	**48**
Divisiones:	**1**	**7**
Traps		
Traps:	**1**	**1**

Figura 8

Finalmente, en la **Figura 8**, se muestra una tabla comparativa mostrando las estadísticas de ambos programas, de esta forma se comprueba que se ha conseguido mejorar en 134 ciclos respecto al programa original.