FACULTAD DE CIENCIAS

GRADO EN INGENIERIA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Informe de la segunda práctica obligatoria:

Implementación y optimización de un algoritmo en ensamblador DLX

PB1

PABLO SANTOS BLÁZQUEZ

JUAN JOSÉ LÓPEZ GÓMEZ

Salamanca, 9 de mayo de 2022

Contenido

1 – Introducción.

El DLX es un microprocesador RISC diseñado por John Hennessy y David A. Patterson, los diseñadores principales de la arquitectura MIPS y de Berkeley RISC (respectivamente), los dos ejemplos de la arquitectura RISC.

El DLX es básicamente un MIPS revisado y simplificado con una arquitectura simple de carga/almacenamiento de 32 bits. Está pensado principalmente para propósitos educativos y se utiliza ampliamente en cursos de nivel universitario sobre arquitectura de computadores, como nosotros estamos haciendo.

2 – Objetivo de la práctica.

El objetivo de la práctica es el desarrollo y optimización de un código que realice el siguiente cálculo:

$$M = V(a_1, a_2, a_3, a_4) \times \frac{(a_2/a_5) + (a_4/a_5)}{(a_1/a_5) + (a_3/a_5)}$$

$$a_1, a_2, a_3, a_4, a_5$$

$$checkA, \quad checkM$$

Siendo:

- M matriz 4x4: $\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$
- $checkM = m_{11} + m_{12} + m_{13} + m_{14} + m_{21} + \cdots + m_{34} + m_{41} + m_{42} + m_{43} + m_{44}$ (suma de todos los elementos de M)
- a_x media aritmética de *listaX*
    - La media se podrá hacer sobre 0, 5, 10, 15 y 20 elementos de las listas en el orden dado. Esto se indicará en la variable *tamano*.
- $checkA = a1 * a2 * a3 * a4 * a5$ (producto de las medias)
- $V(a1, a2, a3, a4)$ $la\ matriz\ de\ Vandermonde$:

$$V(a_1, a_2, a_3, a_4) = \begin{bmatrix} 1 & a_1 & {a_1}^2 & {a_1}^3 \\ 1 & a_2 & {a_2}^2 & {a_2}^3 \\ 1 & a_3 & {a_3}^2 & {a_3}^3 \\ 1 & a_4 & {a_4}^2 & {a_4}^3 \end{bmatrix}$$

Se entregarán dos versiones que realicen correctamente los cálculos expuestos. Una de ellas debe ser optimizada empleando las técnicas habituales: uso de registros adicionales, reordenación de código, desenrollamiento de bucles, etc.

3 –Versión no optimizada.

3.1 – Desarrollo.

Para esta versión nos centramos únicamente en conseguir realizar los cálculos de forma correcta, por lo que seguimos un desarrollo secuencial de los pasos que haríamos si tuviéramos que realizar estos cálculos de forma manual.

1) Configuración inicial.

   Primero cargamos el valor de tamano en r3 y comprobamos que no sea 0. Después cargamos en r2 el valor 4 que usaremos para ir iterando sobre las listas. Por último guardamos en r1 y r4 el último valor que habrá que sumar a la variable lista para poder ir restando 4 sobre la misma, es decir, calculamos: (tamano * 4) – 4.

2) Calcular la media de las listas.

   Recorremos las listas mediante bucles sumando el valor de las mismas para después dividir entre tamano y obtener así las medias. Una vez calculado el valor, almacenamos el resultado en la variable a_x correspondiente.

3) Comprobaciones de divisores.

   Comprobamos que el valor del divisor del factor de multiplicación no sea una raíz del mismo, es decir, que no sea 0. Para esto comprobamos que a_5 no sea 0 y que $a_1 + a_3$ no sea 0.

4) Realizar los cálculos.

   Una vez tenemos todos los datos necesarios para realizar los cálculos empezamos a ejecutarlos, calculando primero checkA y el valor del factor de multiplicación.

   Acto seguido se calculan los cuadrados y los cubos de los a_x para después poder multiplicarlos por el factor de multiplicación y obtener la matriz M. Una vez calculada la matriz M se calcula el valor de checkM.

3.2 – Estadísticas.

Tras la ejecución con tamano=20 obtenemos las siguientes estadísticas:

ESTADÍSTICAS	
Total : 864 ciclos ejecutados	
Nº de ciclos:	864
Nº de instrucciones ejecutadas (IDs):	545
Stalls	
RAW stalls:	174 (20.14% del total de ciclos)
LD stalls:	7 (4.02% de los RAW stalls)
Branch/Jump stalls:	1 (0.57% de los RAW stalls)
Floating point stalls:	166 (95.40% de los RAW stalls)

WAW stalls:	0 (0% del total de ciclos)
Structural stalls:	29 (3.36% del total de ciclos)
Control stalls:	104 (12.04% del total de ciclos)
Trap stalls:	4 (0.46% del total de ciclos)
Total	311 (36% del total de ciclos)
Conditional Branches	
Total:	113 (20.73% del total de instrucciones)
Tomados:	104 (92.04% de las veces)
No tomados:	9 (7.96% de las veces)
Instrucciones Load/Store	
Total:	140 (25.70% del total de instrucciones)
Loads:	117 (83.57% de estas instrucciones)
Stores:	23 (16.43% de estas instrucciones)
Instrucciones de punto flotante	
Total:	151 (27.71% del total de instrucciones)
Sumas:	120 (79.47% de estas instrucciones)
Multiplicaciones:	25 (16.56% de estas instrucciones)
Divisiones:	6 (3.97% de estas instrucciones)
Traps	
Traps:	1 (0.18% del total de instrucciones)

4 – Versión optimizada.

4.1 – Desarrollo.

Una vez desarrollada la primera versión y con la experiencia aprendida del funcionamiento de DLX comenzamos a desarrollar la versión optimizada.

Lo primero que hicimos fue simplificar las fórmulas matemáticas:

- Medias: observamos que no era necesario realizar una división por cada una de las listas para obtener los resultados, si no que podíamos realizar una única división 1/tamano y después multiplicar el resultado de esta por la suma de los valores de cada lista.

  Esto nos permite obtener el resultado de los a_x tan solo 5 ciclos después de que se haya realizado la última suma en cada una de ellas.

- Factor de multiplicación: la operación simplificada es: $\frac{a_2+a_4}{a_1+a_3}$.

  Con esta simplificación conseguimos reducir el número de divisiones a realizar. Habrá que comprobar igualmente que a_5 no sea 0, ya que produciría una indeterminación.

Una vez simplificadas las operaciones decidimos investigar la forma óptima de repartir las operaciones:

- Sumas: debemos dejar una instrucción entre suma y suma.
- Multiplicaciones: debemos dejar un margen de 4 instrucciones entre multiplicación y multiplicación.
- Divisiones: debemos dejar un margen de 16 instrucciones entre la división y la obtención de su resultado.

Con estas observaciones empezamos a desarrollar el código:

1) Carga de tamano y comprobación.

   Para minimizar el número de saltos decidimos implementar un código distinto para cada una de los posibles valores de tamano, dejando el valor 20 como última opción para no realizar ningún salto cuando se tenga este valor. Si tamano tuviera otro valor, únicamente se realizaría un salto.

A partir de aquí, analizaremos concretamente el caso de tamano = 20. Para los demás casos se ha realizado el mismo análisis y desarrollo, completando los ciclos vacíos de Stall con instrucciones nop.

2) Carga de lista1.

   Cargamos el valor de tamano en un registro float (f25) y cargamos los valores intercalando las sumas de los mismos.

3) Carga de lista2.

   Realizamos la última suma de lista1 y calculamos el valor de a_1. Mientras se obtiene el valor de a_1 se sigue cargando y calculando la suma de los valores de a_2, y mientras se carga y suma lista2, se van intercalando las multiplicaciones necesarias para calcular a_1^2 y a_1^3.

4) Carga de lista3.

   Realizamos la última suma de lista2 y calculamos el valor de a_2. Mientras se obtiene el valor de a_2 se sigue cargando y calculando la suma de los valores de a_3, y mientras se carga y suma lista3, se van intercalando las multiplicaciones necesarias para calcular a_2^2, a_2^3 y la primera multiplicación de checkA: $a_1 \cdot a_2$.

5) Carga de lista4.

   Realizamos la última suma de lista3 y calculamos el valor de a_3. Mientras se obtiene el valor de a_3 se sigue cargando y calculando la suma de los valores de a_4, y mientras se carga y suma lista4, se van intercalando las multiplicaciones necesarias para calcular a_3^2, a_3^3 y la segunda multiplicación de checkA: $a_1 \cdot a_2 \cdot a_3$.

6) Carga de lista5.

   Realizamos la última suma de lista4 y calculamos el valor de a_4. Mientras se obtiene el valor de a_4 se sigue cargando y calculando la suma de los valores de a_5. Una vez obtenido a_4 comienza la división del factor de multiplicación (no sin antes haber realizado las comprobaciones del divisor). Mientras se produce la división y se

carga y suma lista5, se van intercalando las multiplicaciones necesarias para calcular a_4^2, $a4^3$ y la tercera multiplicación de checkA: $a_1 \cdot a_2 \cdot a_3 \cdot a_4$.

7) Cálculos finales.

   Una vez obtenidos los valores necesarios para conseguir realizar las operaciones, empezamos a realizar todas las multiplicaciones de forma secuencial, dejando entre cada una de ellas 4 instrucciones, como anteriormente especificamos. Mientras se calculan los valores de M, se irán sumando los valores de checkM que estén disponibles y se irán almacenando en sus respectivas variables los valores que ya estén calculados.

Gracias al balanceo entre las operaciones de suma y la escritura de las variables, conseguimos que el procesador esté trabajando en todo momento.

4.2 – Estadísticas.

Tras la ejecución de la optimización con tamano=20 obtenemos las siguientes estadísticas:

ESTADÍSTICAS	
Total : 327 ciclos ejecutados	
Nº de ciclos:	327
Nº de instrucciones ejecutadas (IDs):	289
Stalls	
RAW stalls:	0 (0% del total de ciclos)
LD stalls:	0 (0% de los RAW stalls)
Branch/Jump stalls:	0 (0% de los RAW stalls)
Floating point stalls:	0 (0% de los RAW stalls)
WAW stalls:	0 (0% del total de ciclos)
Structural stalls:	0 (0% del total de ciclos)
Control stalls:	0 (0% del total de ciclos)
Trap stalls:	3 (0.92% del total de ciclos)
Total	3 (0.92% del total de ciclos)
Conditional Branches	
Total:	6 (2.08% del total de instrucciones)
Tomados:	0 (0% de las veces)
No tomados:	6 (100% de las veces)
Instrucciones Load/Store	
Total:	124 (42.91% del total de instrucciones)
Loads:	101 (81.45% de estas instrucciones)
Stores:	23 (18.55% de estas instrucciones)
Instrucciones de punto flotante	
Total:	148 (51.21% del total de instrucciones)
Sumas:	117 (79.05% de estas instrucciones)
Multiplicaciones:	29 (19.59% de estas instrucciones)

Divisiones:	2 (1.35% de estas instrucciones)
Traps	
Traps:	1 (0.35% del total de instrucciones)

5 – Conclusión.

La correcta planificación de las instrucciones es tan importante que , como ya hemos observado, puede llegar a suponer una gran mejora en el rendimiento, como en nuestro caso, que conseguimos obtener una mejora del 62.15%.

La mejor forma de mejorar este rendimiento es un buen planteamiento que consiga tener la pipeline completa en todo momento, como nosotros hemos conseguido. Es muy conveniente acompañar esta estrategia de una correcta simplificación matemática que encaje con el planteamiento anteriormente desarrollado.

6 – Referencias.

1 - Colaboradores de Wikipedia. (2021, 16 enero). DLX. Wikipedia, la enciclopedia libre. Recuperado 9 de mayo de 2022, de https://es.wikipedia.org/wiki/DLX

2 - Curso: ARQUITECTURA DE COMPUTADORES. (s. f.). Studium USAL. Recuperado 9 de mayo de 2022, de https://studium.usal.es/course/view.php?id=2106353