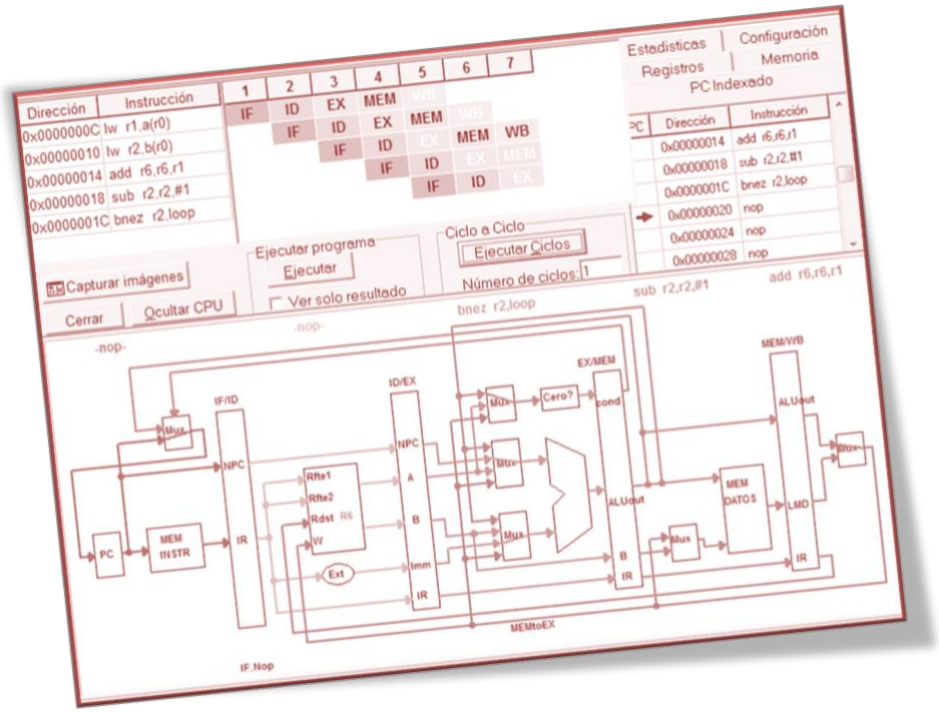

UD2: EVALUACIÓN DE PRESTACIONES DE UN PROCESADOR SEGMENTADO DLX

CURSO 2023-2024

ARQUITECTURA E INGENIERÍA DE COMPUTADORES:
Pau Micó

Gabriela Potenciano Carpintero.
Marcos del Amo Fernández.
Grado en Ingeniería Informática;
UPV Campus de Alcoy

ÍNDICE

1 Introducción

Una de las estrategias básicas de mejora de las prestaciones de un microprocesador consiste en la segmentación de la CPU. A partir de esta práctica se te propone que, mediante la adecuada configuración del microprocesador segmentado DLX, ejecutes una serie de programas básicos que te ayuden a evaluar las prestaciones del mismo.

En esta segunda práctica de Arquitectura e Ingeniería de Computadores, el foco está en comprender cómo operan y funcionan los microprocesadores segmentados. Para lograr estos objetivos, utilizaremos el simulador DLX y exploraremos distintas configuraciones utilizando ejemplos proporcionados específicamente para esta actividad.

1.1 Objetivos

- *aprender a utilizar un simulador de procesador segmentado llamado DLX*
- *entender la arquitectura de un procesador segmentado*
- *evaluar la mejora de prestaciones que supone el microprocesador segmentado DLX respecto a las de uno no segmentado*

2 Simulador DLX

DLXide es un simulador de procesador segmentado, basado en el set de intrucciones del procesador segmentado MIPS. En esta práctica vas a utilizar ensamblador como lenguaje de programación, por lo que sería muy interesante que repasaras lo aprendido en la asignatura de Estructura de Computadores. Además, también tienes un tutorial de ayuda en A Neophyte's Guide to DLX.

El fichero instalable para Windows del simulador DLX/MIPS puedes descargarlo desde aquí. Espera hasta llegar al punto de Ejecución Guiada para saber cómo funciona. Para conocer con algo más de detalle la estructura y el funcionamiento del microprocesador segmentado DLX, puedes acceder a los apuntes de la asignatura desde aquí. Además, te puedes descargar un resumen de las instrucciones ensamblador para una arquitectura MIPS desde aquí.

***Nota:** existe una versión del simulador DLX/MIPS (no probada, llamada openDLX) que es multiplataforma y de código abierto (Java) y que puedes descargarte desde aquí.*

2.1 Descripción del simulador

El principal objetivo de esta práctica es que el alumno sea capaz de escribir y ejecutar programas en el entorno DLXIDE. En este entorno se simula la ejecución de programas, escritos en ensamblador, en un procesador segmentado: el DLX.

Un procesador segmentado es un procesador en el que en la ejecución de las instrucciones se han aplicado técnicas de segmentación. El diseño del procesador se realiza con el objetivo de que en cada ciclo de reloj se finalice (inicie) la ejecución de una instrucción. El problema es que la ejecución de una instrucción por cada ciclo no siempre se puede conseguir debido a problemas de dependencias (o riesgos) de datos, estructurales y/o de control. Las dependencias de datos y de control no son predecibles durante el diseño del procesador y dependen de las características del programa. El significado y tipo de cada uno de estos problemas se desarrollan en clase de teoría. Para la realización de esta práctica no es necesario su conocimiento previo, aunque el simulador utilizado permite analizar la influencia de las mismas sobre las prestaciones del computador simulado.

2.2 Características principales

En este caso, las características del simulador son:

- *Antememorias de datos y de instrucción separadas*
- *Banco de registros enteros. No existe banco de registros en coma flotante*
- *Instrucciones del DLX soportadas son las que operan con los registros enteros del simulador a excepción de las de multiplicación*
- *Visualización y modificación del contenido los registros en hexadecimal y decimal*
- *Visualización del contenido de la memoria en hexadecimal y decimal por palabras o por bytes*
- *Ejecución de los programas en ensamblador*

- *Simulación ciclo a ciclo de los programas en ensamblador, mostrando la etapa de ejecución en la que se encuentra cada instrucción y la configuración de los elementos de la CPU*

- *Posibilidad de utilizar diferentes técnicas para resolver los riesgos de control*

- *Ejecución con o sin mejora para dependencias de datos*

- *Memoria: el simulador sigue el mismo convenio de memoria que el procesador donde se ejecute. Como en prácticas todos los computadores tienen procesadores tipo 80x86 nos encontramos que siguen un convenio Little Endian (MSB en el byte de dirección superior o byte impar y LSB en el byte de dirección inferior o byte par)*

- *Registros: el registro r0, que es de solo lectura y siempre contiene el valor 0*

- *Instrucciones: su utilización se puede consultar en la ayuda. De todas formas hay que saber que la instrucción de carga en un registro del valor almacenado en una variable (load word, lw) se utiliza de la siguiente forma:*

  ```
  lw r1, a(r0)
  ```

  donde se estaría cargando en el registro r1 el valor guardado en la dirección de memoria de la variable 'a' (a la que obligatoriamente se le suma el valor del registro r0, que ya sabes que vale 0)

- *Etapas: las etapas en las que se divide el simulador DLXIDE son:*

  - *IF (Instruction Fetch) o búsqueda de la instrucción*

  - *ID/OF (Instruction Decodification) o decodificación de la instrucción y obtención de los operandos (Operand Fetch)*

  - *EX (Execution) o ejecución de la operación en la ALU*

  - *MEM (Memory) etapa en la que la instrucción simplemente espera un ciclo sin hacer nada. Esta etapa se incluye en la ruta de datos del microprocesador para resolver por hardware el riesgo estructural que supone el acceso para escritura sobre el banco de registros de dos instrucciones simultáneamente: una instrucción LOAD (que supone un retraso de un ciclo debido a la latencia de la memoria), seguida de cualquier otra instrucción que escriba en registros*

  - *WB (Write Back) o escritura del resultado en el buffer de entrada de la etapa de ejecución (anticipación o forwarding mediante cortocircuito)*

***NOTA:** El programa cuenta con un sistema de ayuda sobre el funcionamiento de los diferentes elementos que componen el simulador y una referencia del ensamblador del DLX donde se describen tanto el juego de instrucciones del DLX como las directivas del ensamblador.*

2.3 Riesgos en la segmentación

Uno de los objetivos de la práctica es evaluar las mejoras que se pueden introducir en el DLX segmentado para evitar la inserción de ciclos de parada debidos a los distintos riesgos que pueden aparecer en el mismo. Estos riesgos pueden ser:

- *Riesgos Estructurales. En la ejecución normal del DLX no se presentan riesgos estructurales, por lo que el simulador no cuenta con ningún modo de operación especial que simule una mejora de los mismos*
- *Riesgos por Dependencia de Datos. Que dependen del programa de ejecución. En el DLX aparecen cuando una operación lee un dato que una instrucción anterior está escribiendo de forma simultánea (RAW – Read After Write). Sólo aparecen si la ejecución de las dos instrucciones dependientes es muy próxima (aparecen separadas por uno o dos ciclos). Para reducir el impacto de los riesgos por dependencia de datos se pueden plantear dos estrategias:*
    - *Detención de la segmentación mediante la inserción de burbujas: para evitar las ocasiones en que se producen los riesgos el compilador se encarga automáticamente de separar las instrucciones que provocan dichos riesgos mediante la inserción de ciclos de espera o burbujas*
    - *Anticipación o Forwarding: reduciendo por HW (cortocircuitos) la distancia a la que tienen que estar dos operaciones para que aparezca el riesgo, reduciendo de esta forma el número de ciclos de parada que se deben insertar en el caso de la aparición de uno de estos riesgos*
- *Riesgos de Control. Se presentan siempre que hay un salto condicional. Se tratan de resolver mediante una serie de técnicas entre las que cabe citar:*
    - *Inserción de burbujas de retardo (delay slot): es imprescindible la utilización de un compilador que conozca el número de slots o burbujas de retardo que se utilizarán y que coloque en ellos instrucciones útiles (el impacto del riesgo de control será menor cuanto más instrucciones útiles insertemos en los slots de retardo)*
    - *Predicción sobre el salto: donde se intenta adivinar qué instrucción se ejecutará en base a un histórico de los saltos anteriores*

Evidentemente la implementación de estas técnicas va a suponer un mayor coste en el diseño y la producción de la unidad, por lo que es necesario asegurar que se incrementen claramente las prestaciones de la máquina.

A continuación, y una vez entendidos los conceptos y definiciones relativos al procesamiento segmentado, se te propone la ejecución guiada de algunos programas de ejemplo.

3 SESIÓN 1: Ejecución de ejemplos

3.1 Cálculos y desarrollo individual

Una vez ejecutadas las simulaciones, contesta a las siguientes cuestiones:

1. ***Comenta que realiza cada una de las instrucciones del programa de ejemplo (`Ejemplo1.s`)***

1. .data 0 nos indica en la dirección de memoria en la que inicia el programa.

2. a: Word 33= inicializa la variable a con el valor 33

3. b: Word 33= inicializa la variable b con el valor 2355

4. .text 12 nos indica en la dirección de memoria en la que inicia la etiqueta text.

5. lw r1,a(r0) = se va al registro r1 y carga 33

6. lw r2,b(r0) = se va al registro r2 y carga 2355

7. loop: indica el inicio del bucle de instrucciones

8. add r6,r6,r1 = inicializa r6 y le suma r1 que vale a.

9. sub r2,r2,#1= a r2 se le resta 1 por cada iteración del bucle

10. bnez r2,loop = salta a la etiqueta loop cuando r2 sea distinto de 0.

11. Nop,nop,nop = Realiza tres ciclos sin operar.

12. Add r10,r6,r0 =una vez salimos del bucle sumamos el valor de r6 y r0 en el registro r10.

13. trap #0= esta instrucción nos finaliza el programa generando una excepción, devuelve el control al sistema operativo.

2. ***Obtén el Speed-Up y el valor de CPI del programa para la configuración básica que aparece por defecto***

-Ciclos sin segmentar = *5*2 + 3*4*2355 + 4*4 + 1*5 = 28286 ciclos*

-Ciclos segmentado= 18853 ciclos (del simulador)

-Speed-up = 28286 / 18852 = 1,50 (50% de mejora)

-CPI = (CP/NI) = 18852 / 7072 = 2,66 ciclos por instrucción

3. ***Carga, ensambla y ejecuta los demás ficheros de ejemplos suministrados***

4. ***Para cada uno de los ejemplos: indica qué se supone que tiene que hacer el código y comprueba el resultado obtenido (número de instrucciones ejecutadas, CPI, speed-up, porcentaje de utilización de la CPU, productividad o IPC y valores finales en los registros…). Además, también deberás indicar si algún ejemplo NO se ejecuta bien***

Ejemplo 2

Ejemplo 2.s almacena en R10 la multiplicación de los valores en a y b.

● Número de instrucciones ejecutadas: 43

● Númerodeciclosejecutadas:78

● Número de ciclos de parada: 31

● Valor registro R2: 301440

● Valor registro R3: 1

● Valor registro R5: 301440

-Ciclos sin segmentar = *1 (inst sw)*5 + 2 (inst lw) *5 + (43-3) inst * 4 = 175 ciclos*

-*Ciclos* segmentado *= 78 ciclos (obtenidos del simulador)*

-*Speed-up = 175/ 78 = 2.243 (124.35% de mejora)*

-*CPI = (CP/NI) = 78/43 =1.82 ciclos por instrucción*

-Utilización CPU (%) = ((78-31)/78) = 0,6 = 60%

-IPC = CPI^(-1) = 0,55 instrucciones por ciclo

Ejemplo 3

Ejemplo 3.s hace uso del bucle *(hasta que el valor de R4 se*a *igual a 0) para realizar la suma de dos vectores almacenándolo en otro vector y así cargarlo en diferentes registros.*

● *Número de instrucciones ejecutadas: 93*

● *Número de ciclos ejecutados:220*

- *Número de ciclos de parada: 123*
- *Valor registro R4: 40*
- *Valor registro R5: 10*
- *Valor registro R6: 2*
- *Valor registro R7: 12*
- *ValorregistroR10:12*
- *ValorregistroR11:10*
- *ValorregistroR12:15*
- *ValorregistroR13:11*
- *ValorregistroR14:18*
- *ValorregistroR15:12*
- *ValorregistroR16:21*
- *ValorregistroR17:13*
- *ValorregistroR18:24*
- *ValorregistroR19:14*
- *ValorregistroR20:0*

-Ciclos sin segmentar = *13 (inst lw) *5 + 1 (inst sw)*5 + (93-14) inst * 4 = 386* ciclos

-Ciclos segmentado= 220 *ciclos (obtenidos del simulador)*

-*Speed-up* = 386/ 220 = *1.75 (75% de mejora)*

-*CPI = (C/NI)* = 220/93 =2.37 *ciclos por instrucción*

-Utilización CPU (%) = ((220-123)/220) = 0,44 = 44%

-IPC = CPI^(-1) = 0,42 instrucciones por ciclo

Ejemplo 4

Ejemplo4.s no se puede ejecutar debido a que el valor que se le intenta añadir en la 6a instrucción (66896) al registro r7, es demasiado grande. Como no se puede ejecutar, no podremos sacar el Speed-up, CPI y demás datos.

4 SESIÓN 2: Mejora de las prestaciones del DLX

En esta SEGUNDA SESIÓN de la práctica vas a tratar de mejorar las prestaciones del DLX trabajando sobre uno de los programas de ejemplo. Para ello:

- *carga el fichero* `Ejemplo1.s`*. Este programa multiplica dos números enteros y almacena el resultado en un registro*
- *elimina todas las operaciones 'nop' que aparecen en el código y asegúrate que en la ventana de configuración del simulador sólo está marcada la opción Insertar ciclos de parada. Los valores a multiplicar serán* `a=2355` *y* `b=33`
- *ejecuta el programa (comprueba que se ejcuta correctamente) y obtén el tiempo de CPU (en ciclos) y el CPI*

Resultado de ejecución:

Ciclos ejecutados = 18849

Instrucciones ejecutadas = 7069

Ciclos de parada = 11776

Cortocircuitos = 0

CPI =18849/7069 = 2,66

Tiempo de CPU = 18849 ciclos.

4.1 Riesgos por dependencia de datos: inserción de burbujas

Vamos ahora a probar dos métodos para mejorar las prestaciones del procesador segmentado resolviendo riesgos por dependencia de datos que aparecen en el programa. Primero vamos a optimizar el código, tal y como lo haría un compilador. Identificamos las dependencias de datos existentes en el programa observando la que aparece sobre el registro r2 del bucle:

```
loop:
add r6, r6, r1
sub r2, r2, #1
bnez r2, loop
```

Esta dependencia de datos genera dos ciclos de espera para cada vuelta del bucle, que podrían quedarse en uno reordenando las instrucciones del bucle de la siguiente forma:

```
loop:
sub r2, r2, #1
add r6, r6, r1
bnez r2, loop
```

- *modifica el ejemplo y vuelve a obtener el valor de CPI. A partir de este momento nos quedaremos con la versión del bucle que sea más rápida.*

Resultado de ejecución:

Ciclos ejecutados = 16495

Instrucciones ejecutadas = 7069

Ciclos de Parada = 9422

Cortocircuitos = 0

CPI = 16495/7069 = 2,33

Tiempo de CPU = 16495 ciclos.

El valor de CPI ha disminuido de 2.66 a 2.33, lo que indica una mejora en la eficiencia del procesador. Esto sugiere que la versión del bucle optimizado ha reducido los ciclos de espera debido a las dependencias de datos, lo que lleva a una ejecución más rápida del programa.

4.2 Riesgos por dependencia de datos: anticipación

La otra técnica para solucionar los problemas de dependencia de datos es modificando directamente el HW del sistema utilizando la técnica de los cortocircuitos.

- *vuelve a la ventana de cargar y modificar la configuración de modo que también esté marcada la casilla Forwarding que permite habilitar los cortocircuitos en el simulador*
- *ve a la ventana de ejecución y comprueba que el circuito ha cambiado*
- *ejecuta ciclo a ciclo algunas instrucciones para ver como funcionan los cortocircuitos añadidos*
- *ejecuta de nuevo el programa y obtén el CPI*

Ejecutando el programa tenemos que:

Ciclos ejecutados = 14139

Instrucciones ejecutadas = 7069

Ciclos de parada = 7066

Cortocircuitos = 2356

CPI = 14139/7069 =2.0001

Ahora, el CPI se ha reducido a 2.0001, lo que indica una significativa mejora en el rendimiento en comparación con los valores anteriores (2.66 y 2.33).Los ciclos de parada se han reducido considerablemente a 7066, lo que sugiere que los cortocircuitos han permitido al procesador anticipar y resolver las dependencias de datos de manera más eficiente, disminuyendo así el tiempo de espera.

4.3 Riesgos de control

Cuando existe un riesgo de control la operación por defecto es insertar ciclos de parada (detención de la segmentación). Sin embargo existen otras alternativas algunas de las cuales están disponibles en el simulador. A continuación se citan las técnicas disponibles y qué es necesario hacer para que el programa funcione correctamente (además de modificar la unidad de control de la CPU):

- ***Predict Not Taken**: ante una instrucción de salto se supone que NO se va a saltar. Con esta técnica no es necesario modificar el HW de la CPU o los programas. El problema es que si la probabilidad de saltar es alta (como en el caso del ejemplo) esta configuración proporciona una escasa mejora*

- ***Delay Slot 3**: las tres instrucciones que se colocan detrás de una instrucción de salto condicional se ejecutarán SIEMPRE. Será necesario que el programador (o el compilador si es inteligente) modifique el programa para usar esta técnica. Es decir, el programador deberá reordenar el código escribiendo 3 instrucciones independientes de la de salto (en el peor de los casos, 3 instrucciones de nop). El compilador debe ser consciente que la CPU utiliza un delay de 3 slots.*

- ***Delay Slot 1**: la instrucción que se coloca detrás de una instrucción de salto condicional se ejecuta SIEMPRE. Será necesario que el programador (o el compilador si es inteligente) modifique el programa para usar esta técnica. Es decir, el programador deberá reordenar el código escribiendo 1 instrucción independientes de la de salto (en el peor de los casos, 1 instrucción de nop). El compilador debe ser consciente que la CPU utiliza un delay de 1 slot.*

En las técnicas de salto retardado con delay slot las prestaciones aumentan en el caso de encontrar instrucciones útiles que rellenen esos slots. Hay que tener en cuenta que si no se consigue poner ninguna instrucción útil en los mismos no se consigue mejorar el tiempo de ejecución de los programas. Estas técnicas se pueden complementar o no con la de forwarding. En principio deshabilitamos el forwarding para comprobar el efecto de las mejoras en los saltos:

- *cambia la configuración del simulador deshabilitando el forwarding y selecciona la técnica de Predict Not Taken.*

- *observa la ejecución paso a paso de los primeros ciclos del bucle antes de ejecutar hasta el final para obtener los resultados de ejecución*

Resultados de ejecución:

Ciclos ejecutados = 16492

Instrucciones Ejecutadas 7069

Ciclos de parada = 9419

Cortocircuitos = 0

CPI = 16492 / 7069 = 2,33

- *cambia la configuración del simulador deshabilitando la opción de Predict Not taken y habilitando al de Delay Slot 3*

- *antes de ejecutar el programa tendrás que cambiar el código ensamblador añadiendo tres instrucciones tras el salto. Analizando las instrucciones del programa puedes comprobar que sólo se puede poner una instrucción útil en el slot (tras el salto). Modifica el código escribiéndola tras el salto y añade dos instrucciones de NOP (ya que DS=3). El bucle quedará como sigue:*

```
loop:
sub r2, r2, #1
bnez r2, loop
add r6, r6, r1
nop
nop
```

- *observa la ejecución paso a paso de los primeros ciclos del bucle antes de ejecutarlo hasta el final para obtener los resultados de ejecución:*

Resultados de ejecución:

Ciclos ejecutados = 16495

Instrucciones ejecutadas = 11779

Ciclos de parada = 4712

Cortocircuitos = 0

CPI = 16495/11779 = 1,4

- *cambia la configuración del simulador deshabilitando la opción de Delay Slot 3 y habilita la de Delay Slot 1. En este caso sólo tendrás que añadir una instrucción detrás del salto, o sea que ya puedes quitar las dos instrucciones de NOP añadidas anteriormente*

- *observa la ejecución paso a paso de los primeros ciclos del bucle antes de ejecutarlo hasta el final para obtener los resultados de ejecución*

Resultados de ejecución:

Ciclos ejecutados = 14142

Instrucciones ejecutadas = 7069

Ciclos de parada = 7069

Cortocircuitos = 0

CPI = 2,0005

4.4 Cálculos y desarrollo individual

En este apartado vas a trabajar sobre el código del `Ejemplo2.s` que es un programa que multiplica dos números enteros y almacena el resultado final en el registro R6. En este caso todas las simulaciones se deben justificar adecuadamente en el apartado de conclusiones.

1. *obtén el tiempo de CPU, CPI y Speed-Up aplicando todas las posibles combinaciones de configuración del simulador (ver **Tabla 1**). Hazlo para los siguientes valores de las variables: **a** (128, 1) y **b** (1530). Calcula la aceleración o speed-up respecto al microprocesador SIN segmentar. El número de instrucciones ejecutadas en el micro NO segmentado lo puedes obtener realizando una ejecución básica en el DLX. Para calcular el número de ciclos toma un CPI=5 para las instrucciones LOAD/STORE y un CPI=4 para el resto*

Tabla 1. *Posibles combinaciones que presenta la configuración del simulador son las siguientes*

Solución Riesgos de Control	***Solución Riesgos Dependencia Datos***
Insertar ciclos de parada	*SIN anticipación*
Ejecución especulativa	*SIN anticipación*
Delay slot = 1 - Instrucción nop	*SIN anticipación*
Delay slot = 1 - Reordenación del código	*SIN anticipación*
Delay slot = 3 - Instrucciones nop	*SIN anticipación*
Delay slot = 3 - Reordenación del código	*SIN anticipación*
Insertar ciclos de parada	*CON anticipación*
Ejecución especulativa	*CON anticipación*

Delay slot = 1 - Instrucción nop[1]	*CON anticipación*
Delay slot = 1 - Reordenación del código	*CON anticipación*
Delay slot = 3 - Instrucciones nop	*CON anticipación*
Delay slot = 3 - Reordenación del código	*CON anticipación*

Para la presentación de los resultados de las simulaciones se propone el siguiente formato (la tabla tendrá 26 entradas, la primera para los valores obtenidos con el micro NO segmentado y los diferentes valores de las variables a y b. Las restantes 24 entradas corresponden a cada una de las 12 posibles configuraciones del simulador y para cada uno de los valores de la variable b (12 + 12):

Configuración. DLX	***Valores variables***	***Número instrucciones.***	***T_{CPU} (ciclos)***	***CPI***	***Speed up***	***¿Resultado correcto?***
NO segmentado	*a = 128* *b = 1530*	*43*	*175*	*4,06976744≈4*	*1*	*Sí*
NO segmentado	*a = 1* *b = 1530*	*15*	*63*	*4,2≈4*	*1*	*Sí*
Segmentado. ***NO forwarding*** ***Ciclos parada***	*a = 128* *b = 1530*	*43*	*78*	*1,81395349*	*2,24358974*	*Sí*
Segmentado. ***NO forwarding*** ***Ciclos parada***	*a = 1* *b = 1530*	*15*	*29*	*1,93333333*	*2,17241379*	*Sí*

[1] *La introducción de una instrucción nop en el slot es para comprobar que no se obtiene una mejora real. Para resolver los riesgos de control por delay slot y obtener mejora en la ejecución del programa el programador debe asegurar que las intrucciones del slot son instrucciones válidas de programa (de dentro del bucle) y que el resultado final es correcto*

Segmentado. ***NO forwarding.*** ***Ejecución especulativa***	*a = 128* *b = 1530*	*43*	*73*	*1,69767442*	*2,39726027*	*Sí*
Segmentado. ***NO forwarding.*** ***Ejecución especulativa***	*a = 1* *b = 1530*	*15*	*24*	*1,6*	*2,625*	*Sí*
Segmentado. ***NO forwarding.*** ***Delay slot = 1 – Instrucción nop***	*a = 128* *b = 1530*	*52*	*68*	*1,30769231*	*3,1121751*	*Sí*
Segmentado. ***NO forwarding.*** ***Delay slot = 1 – Instrucción nop***	*a = 1* *b = 1530*	*17*	*26*	*1,52941176*	*2,74615385*	*Sí*
Segmentado. ***NO forwarding.*** ***Delay slot = 1 - Reordenación del código***	*a = 128* *b = 1530*	*43*	*69*	*1,60465116*	*2,53623188*	*si*
Segmentado. ***NO forwarding.*** ***Delay slot = 1 – Reordenación del código***	*a = 1* *b = 1530*	*15*	*27*	*1,8*	*2,33333333*	*Sí*

Segmentado. ***NO forwarding.*** ***Delay slot = 3 – Instrucción nop***	*a = 128* *b = 1530*	*70*	*78*	*1,11428571*	*3,6523554*	*Sí*
Segmentado. ***NO forwarding.*** ***Delay slot = 3 – Instrucción nop***	*a = 1* *b = 1530*	*21*	*29*	*1,38095238*	*3,04137931*	*Sí*
Segmentado. ***NO forwarding.*** ***Delay slot = 3 – Reordenación del Código***	*a = 128* *b = 1530*	*47*	*57*	*1,21276596*	*3,35577315*	*No*
Segmentado. ***NO forwarding.*** ***Delay slot = 3– Reordenación del código***	*a = 1* *b = 1530*	*15*	*24*	*1,6*	*2,625*	*No*
Segmentado. ***SI forwarding*** ***Ciclos parada***	*a = 128* *b = 1530*	*43*	*74*	*1,72093023*	*2,36486486*	*Sí*
Segmentado. ***SI forwarding*** ***Ciclos parada***	*a = 1* *b = 1530*	*15*	*25*	*1,66666667*	*2,52*	*Sí*

Segmentado. ***SI forwarding*** ***Ejecución especulativa***	*a = 128* *b = 1530*	*43*	*68*	*1,58139535*	*2,57352941*	*Sí*
Segmentado. ***SI forwarding*** ***Ejecución especulativa***	*a = 1* *b = 1530*	*15*	*19*	*1,26666667*	*3,31578947*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 1 – Instrucción nop***	*a = 128* *b = 1530*	*52*	*56*	*1,07692308*	*3,77906977*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 1 – Instrucción nop***	*a = 1* *b = 1530*	*17*	*21*	*1,23529412*	*3,4*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 1 – Reordenación del código***	*a = 128* *b = 1530*	*43*	*47*	*1,09302326*	*3,72340426*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 1 – Reordenación del código***	*a = 1* *b = 1530*	*15*	*19*	*1,26666667*	*3,31578947*	*Sí*

Segmentado. ***SI forwarding*** ***Delay slot = 3 – Instrucción nop***	*a = 128* *b = 1530*	*70*	*74*	*1,05714286*	*3,84978001*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 3 – Instrucción nop***	*a = 1* *b = 1530*	*21*	*25*	*1,19047619*	*3,528*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 3 – Reordenación del Código***	*a = 128* *b = 1530*	*47*	*51*	*1,08510638*	*3,75057*	*No*
Segmentado. ***SI forwarding*** ***Delay slot = 3 – Reordenación del código***	*a = 1* *b = 1530*	*19*	*23*	*1,21052632*	*3,46956522*	*No*

Como la reordenación del código básica para DS = 3 tanto para a = 128 y a = 1, vemos que no nos da el resultado esperado pasamos a una segunda reordenación. Con la misma intención de no utilizar instrucciones NOP.

Utilizando el siguiente código:

```
.data 0
a:      .word 128
b:      .word 1530
c:      .word 0

.text 12
        lw r1,a(r0)
```

```
        lw r2,b(r0)
        addi r4,r0,#-1
        add r5,r0,r0
giro:
        beqz r3,giro
        add r4,r4,#1
        srli r1,r1,#1
        andi r3,r1,#1
        sll r2,r2,r4
        add r5,r5,r2
        bnez r1,giro
        add r4,r0,r0
        add r6,r5,r0
        sw c(r0),r6
        trap #0
```

Obtenemos el siguiente resultado:

Configuración. DLX	**Valores variables**	**Número instrucciones.**	**T_{CPU} (ciclos)**	**CPI**	**Speed up**	**¿Resultado correcto?**
Segmentado. **No forwarding** **Delay slot = 3 – Reordenación del Código**	a = 128 b = 1530	43	81	1,88372093	2,16049383	Si

Segmentado. ***No forwarding*** ***Delay slot = 3*** ***–*** ***Reordenación del código***	*a = 1* *b = 1530*			*#¡DIV/0!*	*#¡DIV/0!*	*No*
Segmentado. ***SI forwarding*** ***Delay slot = 3*** ***–*** ***Reordenación del Código***	*a = 128* *b = 1530*	*43*	*47*	*1,09302326*	*3,72340426*	*Si*
Segmentado. ***SI forwarding*** ***Delay slot = 3*** ***–*** ***Reordenación del código***	*a = 1* *b = 1530*			*#¡DIV/0!*	*#¡DIV/0!*	*No*

Como seguimos sin tener el resultado esperado para a=1, pasamos a probar a reordenar el código con 1 nop:

```
.data 0
a:      .word 1
b:      .word 1530
c:      .word 0

.text 12
        lw r1,a(r0)
        lw r2,b(r0)
        addi r4,r0,#-1
        add r5,r0,r0
giro:   andi r3,r1,#1
```

beqz r3,giro

nop

add r4,r4,#1

srli r1,r1,#1

sll r2,r2,r4

add r5,r5,r2

bnez r1,giro

add r4,r0,r0

add r6,r5,r0

sw c(r0),r6

trap #0

Utilizando dos instrucciones de programa y una de nop en el slot.

Configuración. DLX	***Valores variables***	***Número instrucciones.***	***T_{CPU} (ciclos)***	***CPI***	***Speed up***	***¿Resultado correcto?***
Segmentado. ***No forwarding*** ***Delay slot = 3 – Reordenación del Código y 1 NOP***	*a = 128* *b = 1530*	*51*	*90*	*1,76470588*	*2,30620155*	*Sí*
Segmentado. ***No forwarding*** ***Delay slot = 3 – Reordenación del código y 1 NOP***	*a = 1* *b = 1530*	*16*	*27*	*1,6875*	*2,48888889*	*Sí*

Segmentado. ***SI forwarding*** ***Delay slot = 3 – Reordenación del Código y 1 NOP***	*a = 128* *b = 1530*	*51*	*55*	*1,07843137*	*3,77378436*	*Sí*
Segmentado. ***SI forwarding*** ***Delay slot = 3 – Reordenación del código y 1 NOP***	*a = 1* *b = 1530*	*16*	*20*	*1,25*	*3,36*	*Sí*

Nota: *la columna de 'Resultado correcto' se utiliza para verificar que el resultado del programa es, efectivamente, el que debe ser, aún habiendo reordenado el código. Es decir que, en el caso de reordenación, el alumno deberá comprobar que el valor del registro en el que se obtiene el resultado final es el mismo que el obtenido al ejecutar el programa original. Si, por ejemplo, reordenamos código para un DS=3 pero el resultado no es el correcto, deberemos indicarlo en la columna e introducir un nuevo registro en la tabla repitiendo la ejecución con un DS=3 no tan óptimo (con dos instrucciones de programa más una de nop en el slot). Se intenta conseguir la máxima mejora asegurando que el programa funciona correctamente.*

Para el micro no segmentado 3 instrucciones LOAD/STORE.

1. ***¿con qué configuración del simulador se obtiene de media una mejor relación CPI?***

   Se obtendría una mejor relación de CPI en la columna en la que el valor de dicho campo sea menor ya que ejecutaría más instrucciones en un ciclo. La configuración que obtiene mejor media en relación CPI es la de Segmentado, con forwarding, y DS = 3 – Instrucción nop. Obteniendo CPI = 1,05714286 para a = 128 y CPI = 1,19047619 para a =1.

2. ***¿con qué configuración se minimiza el tiempo de ejecución?***

   La configuración que minimice el tiempo de ejecución será la que tenga menos ciclos. Por lo que para a = 1 será:

Segmentado. ***Si forwarding*** ***Ejecución especulativa.***	*a = 1*	*15*	*19*	*1,26666667*	*3,31578947*	*Sí*

Que obtiene 19 ciclos

O también la configuración:

Segmentado. ***Si forwarding. DS = 1. Reordenación del código***	*a = 1*	*15*	*19*	*1,26666667*	*3,31578947*	*Sí*

Que también obtiene 19 ciclos

Y para a = 128 será:

Segmentado. ***SI forwarding*** ***Delay slot = 1 Reordenación del código***	*a = 128* *b = 1530*	*43*	*47*	*1,09302326*	*3,72340426*	*Sí*

Con 47 cíclos.

Teniendo esto en cuenta diremos que para ambos valores de a = coincide que la mejor configuración es Segmentado, con forwarding, DS = 1 con reordenación del código.

3. ***¿qué opción escogerías teniendo en cuenta los posibles costes?***

Para evitar los riesgos de datos elegiría el forwarding a pesar de que supone hardware adicionar.
En segundo lugar para evitar los riesgos de control elegiría el delay slot 1 , descartando por supuesto la ejecución especulativa ya que supone hardware adicional y la mejora que nos proporciona no es notable .También descartaría el delay slot 3 ya que en este código en concreto no se puede hacer un uso eficiente de el y es mucho más complicado sacar tres instrucciones del bucle.

4. ***¿qué configuración es la más próxima a alcanzar su límite de prestaciones teórico?***

Dado que el DLX está segmentado en 5 etapas (k=5) y una de ellas es la etapa MEM su límite teórico es 4. La configuración con un speed up más próximo a 4, sería la siguiente, que coincide con la de un menor CPI:

Segmentado. ***SI forwarding*** ***Delay slot = 3 – Instrucción nop***	*a = 128* *b = 1530*	*70*	*74*	*1,05714286*	*3,84978001*	*Sí*

Segmentado con forwarding, DS = 3 con instrucción nop.

5. ***desde el punto de vista de la programación, y después de haber realizado numerosas pruebas de configuración, ¿cuál sería la mejor estrategia para incrementar las prestaciones en un microprocesador segmentado?***

Deberíamos incrementar el número de etapas (k) ya que así la mejora máxima que podría obtener sería mayor. Le aplicaríamos también forwarding para eliminar los riesgos por dependencia de datos. Para evitar el riesgo de control, como bien hemos dicho antes aplicaríamos el Delay Slot 1 o el Delay Slot 3 si se pudiera reorganizar el código y ejecutarlo.

5 Conclusiones personales

Después de completar esta práctica, hemos adquirido una comprensión más sólida acerca de la importancia de los riesgos inherentes al uso de procesadores segmentados, así como de las diversas estrategias empleadas para mitigar dichos riesgos.

Asimismo, hemos experimentado una aproximación más práctica hacia la comprensión del mundo real al emplear el simulador, una herramienta previamente desconocida para nosotros pero que ha resultado fundamental para comprender la operatividad de los procesadores y las múltiples instrucciones que se ejecutan en estos sistemas.

Este ejercicio ha sido crucial para consolidar y aplicar los conocimientos adquiridos durante las sesiones teóricas. Nos ha permitido familiarizarnos con el uso del simulador, interpretar sus resultados y fortalecer nuestra comprensión en torno a estos conceptos fundamentales.

5 Bibliografía

- *J. Ortega, M. Anguita, A. Prieto, Arquitectura de Computadores, Ed.Thomson*
- *P. de Miguel, Fundamentos de Computadores, Ed. Paraninfo*
- *C. Hamacher, Z. Vranesic, S.G. Zaky, Computer Organization*
- *Pau Micó Tormos, Apuntes disponibles en Poliformat , UPV, 2023. Última consulta 19 de Noviembre de 2023.*
- *Jose Vicente Albert, Juan Jose Paneque, Pedro López, Simulador DLXIDE (desarrollado por DISCA UPV), DISCA - UPV, 1999.*