




## PRÁCTICA DLXV-1: INTRODUCCIÓN A COMPUTADORES VECTORIALES






### Objetivos de la práctica

	<p>En esta práctica veremos el computador vectorial DLXV, similar al ya conocido DLX, pero con arquitectura vectorial.</p> <p>Emplearemos el simulador <b>WinDLXV</b> que podrás encontrar en el campus virtual), de manejo muy similar al WinDLX. El objetivo es comparar el funcionamiento de un computador escalar con el de uno vectorial, y asimilar algunos elementos básicos de la programación de computadores vectoriales:</p> <ul style="list-style-type: none"> <li>• Uso de instrucciones vectoriales para sustituir bucles de instrucciones escalares</li> <li>• Manejo del registro de longitud de vector, VLR</li> <li>• Conocimiento de las latencias en instrucciones vectoriales.</li> </ul> <p>Siguiendo las directrices de Bolonia sobre el fomento del pensamiento creativo, algunas preguntas de esta práctica serán de respuesta abierta. En estas preguntas no se espera una respuesta concreta, y por tanto no hay una única respuesta correcta.</p>
---	---








### Ejercicio 1: Instalar el simulador






	<p>Es necesario utilizar una cuenta de administrador para instalar el programa; utiliza el nombre de usuario y contraseña que te proporcione el profesor.</p>
	<p>Se proporciona un fichero ejecutable que instalará el simulador WinDLX. Ejecútalo y sigue las instrucciones, utilizando las opciones por defecto.</p>

## Ejercicio 2: Arrancar y configurar el simulador








	<p>Arranca el simulador utilizando el icono que ha dejado instalado.</p>
	<p>Observa el nuevo diseño del programa. Debe llamarte la atención el nuevo diagrama del cauce, que está bifurcado en dos: la parte superior corresponde a la <b>unidad escalar</b>, y la parte inferior representa la <b>unidad vectorial</b>. Además, ambas unidades aparecen segmentadas. En el DLX las ALUs no estaban segmentadas. Esta opción es configurable en el DLXV, y vamos a desactivarla:</p>
	<p>Ve al menú Configuración/Arquitectura. Donde dice “unidades funcionales escalares”, desactiva las dos casillas llamadas “segmentación”.</p>
	<p>¿Qué cambio se ha producido en el diagrama del cauce? Ha pasado de tener un camino dividido en etapas (en multiplicación FP y suma/resta FP) a tener dos caminos sin división en etapas (al igual que en DIV-FP 0).</p>
	<p>¿Qué consecuencias puede tener este cambio, a la hora de ejecutar un programa? Al efectuar este cambio y quitar los caminos segmentados tenemos el problema de que, al tener operaciones de multiplicación o suma/resta, no podemos comenzar con la siguiente instrucción hasta que no se haya completado la actual, cosa que, sin efectuar el cambio (y dejando la segmentación) sí que podíamos hacer.</p>









### Ejercicio 3: Cargar, estudiar y ejecutar un programa escalar

	Carga en el simulador el fichero “edaxpy.s” que viene con el programa (en la carpeta “programas” del WinDLXV).
	Abre el mismo fichero en un Bloc de Notas (o en tu editor favorito de ficheros de texto plano) y echa un vistazo al código.
	El programa, como puedes ver, no se diferencia de los programas del DLX escalar. El lenguaje es el mismo y las instrucciones son también las mismas. <b>Se trata de un programa completamente escalar</b> , que no hace uso de las instrucciones vectoriales, y podría ejecutarse igualmente en el simulador WinDLX que ya conocemos.
	<p>Copia el código del programa a partir de la directiva .text que marca el comienzo del código ejecutable. <b>Añade un comentario a cada instrucción</b>, indicando lo que hace. Puedes ayudarte de los anexos sobre instrucciones y registros del DLX que se dieron en la práctica DLX-1.</p> <pre>.text main:  addi r1,r0,x    ;r1 contiene la dirección de x       addi r2,r0,y    ;r2 contiene la dirección       addi r3,r0,z    ;r3 contiene la dirección de z       ld f0,a(r0)     ;f0 escalar       addi r4,r1,512  ;64 elementos son 512 bytes loop:  ld f2,0(r1)     ;carga el contenido de r2 en f2       multd f2,f0,f2  ;multiplica f2 por f0 y lo deja en f2       ld f4,0(r2)     ;carga el contenido de r2 en f4       addd f4,f2,f4    ;suma f2 y f4 y lo deja en f4       sd 0(r3),f4     ;guarda el contenido de f4 en r3       addi r1,r1,8     ;suma 8 al contenido de r1 y lo deja en r1 para que apunte al siguiente elemento       addi r2,r2,8     ;suma 8 al contenido de r2 y lo deja en r2 para que apunte al siguiente elemento       addi r3,r3,8     ;suma 8 al contenido de r3 y lo deja en r3 para que apunte al siguiente elemento       sub r5,r4,r1     ;resta r1 y r4 y lo deja en r5       bnez r5,loop    ;salta a la etiqueta loop si r5 no es igual a cero       nop             ;Hueco de retardo        trap 6          ;Fin del programa</pre>
	<p>Resumiendo: ¿Qué hace el programa en cada iteración del bucle?</p> <p>Va iterando elemento a elemento en los vectores haciendo la multiplicación: <math>Z = aX * Y</math> (siendo X e Y vectores) dejando el resultado en el correspondiente elemento del vector Z.</p>
	<p>¿Qué hace el programa completo?</p> <p>El programa completo realiza la operación <math>Z = aX * Y</math> siendo Z, X e Y vectores, y "a" un escalar que multiplica a X.</p>
	Realiza la simulación ciclo a ciclo (tecla F7) hasta encontrar un acceso a memoria

	<p>¿Qué ocurre en este acceso a memoria, que sea diferente de lo que ocurría en el DLX escalar?  <b>No tengo referencia a cómo lo hace DLX escalar pero no debería haber diferencia significativa en este caso.</b></p>						
	<p>Lo que ocurre es debido a que la memoria está optimizada para descargar grandes cantidades de datos en un solo acceso, pero al descargar un solo dato a la vez no aprovechamos esta característica y tenemos que soportar una latencia grande. Más adelante en la práctica verás cómo se aprovecha este tipo de memoria.</p>						
	<p>Ejecuta hasta el final (tecla F4) y observa el cronograma.</p>						
	<p>¿Te llama la atención alguna peculiaridad de este cronograma?  <b>Lo que más me llama la atención es que cada multiplicación tarda 7 ciclos de reloj en ejecutarse y cuando acaba el bucle, tenemos una instrucción NOP que deja un ciclo de reloj como retardo en la ejecución del programa.</b></p>						
	<p>Anota las estadísticas:</p> <table border="1" data-bbox="284 1155 992 1270"> <tbody> <tr> <td>Instrucciones</td><td>647</td></tr> <tr> <td>CPI</td><td>2093</td></tr> <tr> <td>Riesgos de Datos (RAW/WAW/WAR)</td><td>1088</td></tr> </tbody> </table>	Instrucciones	647	CPI	2093	Riesgos de Datos (RAW/WAW/WAR)	1088
Instrucciones	647						
CPI	2093						
Riesgos de Datos (RAW/WAW/WAR)	1088						

## Ejercicio 4: Cargar, estudiar y ejecutar un programa vectorial

	Carga en el simulador el fichero “ <b>vdaxpy.s</b> ” que viene con el programa (en la carpeta “programas” del WinDLXV).
	Abre el mismo fichero en un Bloc de Notas (o en tu editor favorito de ficheros de texto plano) y echa un vistazo al código.
	<p>Este programa hace exactamente la misma función que el anterior, pero haciendo uso de las <b>instrucciones vectoriales</b>. Si las instrucciones escalares (las que ya conocemos de prácticas anteriores) operan con datos individuales, las instrucciones vectoriales operan con <b>vectores</b>, es decir, <b>secuencias de datos</b> numéricos. Así en una sola instrucción se pueden expresar muchas operaciones aritméticas.</p> <p>Esto es un ejemplo de arquitectura <b>SIMD: Single Instruction, Multiple Data</b>. Cada instrucción es capaz de operar sobre todo un conjunto de datos.</p>
	<p>Copia el código del programa a partir de la directiva .text que marca el comienzo del código ejecutable. <b>Añade un comentario a cada instrucción</b>, indicando lo que hace. Puedes ayudarte del manual dado con la práctica.</p> <pre>.text main:   addi r1,r0,x    ;r1 contiene la dirección de x         addi r2,r0,y    ;r2 contiene la dirección de y         addi r3,r0,z    ;r3 contiene la dirección de z         ld f0, a(r0)    ;f0 escalar         addi r4,r0,64   ;64 elementos         movi2s vlr,r4   ;copia el contenido de r4 en vlr         lv v0,0(r1)     ;carga el registro vectorial v0 desde la posición de r1         multsv v1,f0,v0 ;multiplica los elementos de v0 por f0 dejando el resultado en v2         lv v2,0(r2)     ;carga el registro vectorial v2 desde la posición de r2         addv v3,v1,v2   ;suma los elementos de v1 y v2 dejando el resultado en v3         sv 0(r3),v3     ;guarda el registro vectorial v3 desde la posición de r3          trap 6         ;Fin del programa</pre>
	<p>¿Qué es lo que más te llama la atención al comparar este programa con el del ejercicio anterior?</p> <p>Este cronograma es mucho más corto que el anterior y se pueden ver muchas más instrucciones por ciclo de reloj.</p>
	Realiza la simulación ciclo a ciclo (tecla F7) hasta encontrar un acceso a memoria. Para observar mejor su funcionamiento, identifica en la ventana llamada “Registros” cuáles son los datos que deben cambiar, y observa paso a paso cómo van cambiando.
	<p>¿Qué ocurre en este acceso a memoria, que sea diferente de lo que ocurría en el DLX escalar?</p> <p>No tengo referencia a cómo lo hace DLX escalar. Pero en este caso sí que debería cambiar debido a que estamos usando operaciones vectoriales.</p>

	Como se comentaba antes, la memoria está optimizada para descargar grandes cantidades de datos en un solo acceso. Al descargar un solo dato a la vez no aprovechamos esta característica, pero al descargar vectores mejora drásticamente el rendimiento.						
	Sigue ejecutando ciclo a ciclo hasta ver en funcionamiento una operación aritmética. Comprueba que, después de resolverse las dependencias de datos, existe un tiempo de latencia hasta que se emite el primer resultado de la operación, y a partir de ahí comienzan a emitirse resultados a uno por ciclo de reloj						
	Ejecuta hasta el final (tecla F4) y observa el cronograma.						
	<p>¿Te llama la atención alguna peculiaridad de este cronograma?</p> <p><b>Este cronograma es más corto que el anterior y podemos ver cómo las instrucciones vectoriales se mantienen y hacen varios ciclos de reloj por cada instrucción.</b></p>						
	<p>Anota las estadísticas:</p> <table border="1" data-bbox="284 1037 992 1160"> <tr> <td>Instrucciones</td><td>12</td></tr> <tr> <td>CPI</td><td>26250</td></tr> <tr> <td>Riesgos de Datos (RAW/WAW/WAR)</td><td>227</td></tr> </table>	Instrucciones	12	CPI	26250	Riesgos de Datos (RAW/WAW/WAR)	227
Instrucciones	12						
CPI	26250						
Riesgos de Datos (RAW/WAW/WAR)	227						
	<p>¿Te resulta raro el nuevo CPI? Hemos estudiado que un CPI alto corresponde normalmente a un mal rendimiento, así que era de esperar que el CPI bajase al mejorarlo. ¿Por qué es tan grande el CPI en este ejemplo?</p> <p><b>Porque se están realizando operaciones vectoriales con todos los elementos del vector a la vez, por lo tanto, tarda más ciclos de reloj en completarse la instrucción vectorial que otra instrucción escalar equivalente.</b></p>						
	<p>Comenta a qué se debe la mejora del rendimiento.</p> <p><b>La mejora en el rendimiento se debe a que las instrucciones vectoriales nos permiten operar con vectores completos y no elemento a elemento como se haría con las instrucciones escalares.</b></p>						
	<p>A la vista del cronograma y de los parones que se han producido, ¿crees que el rendimiento sigue siendo mejorable? ¿Por qué? (En esta pregunta se dará por buena cualquier respuesta que sea coherente, independientemente de que sea correcta o no)</p> <p>En esta práctica tenemos el mismo ejemplo con ambas instrucciones (escalares y vectoriales). Es bastante significativo el hecho de que con las instrucciones vectoriales tenemos muchas menos instrucciones y parones por dependencias en la ejecución del programa que en la versión de instrucciones escalares. Sin embargo, los CPI son mayores en la versión vectorial debido al número de operaciones por instrucción, que ocupa una cantidad mayor de ciclos de reloj. En cuanto al rendimiento seguro que puede seguir siendo mejorable si usamos entrelazado de memoria en los accesos a memoria, tanto de escritura como de lectura.</p>						