Práctica 3

Procesadores vectoriales (I)

1 Introducción

El estudiante se familiarizó en el curso de *Arquitectura de Computadores* con los procesadores segmentados y en particular con el procesador DLX desarrollado por John L. Hennessy y David. A. Patterson. Una extensión natural de la segmentación del DLX consiste en introducir la arquitectura vectorial DLXV, manteniendo la arquitectura DLX como la parte escalar de la arquitectura DLXV y la compatibilidad con su juego de instrucciones.

En la presente práctica y la siguiente se analizará la bondad de la arquitectura vectorial DLXV y sus problemas mediante la ejecución de programas escritos en ensamblador DLXV en el simulador xdlxvsim. Se pretende que el estudiante conozca los fundamentos de las arquitecturas vectoriales y sea capaz de optimizar el código ensamblador para evitar riesgos y aumentar el rendimiento de la máquina vectorial.

2 La arquitectura DLXV. Revisión

La arquitectura de procesador vectorial que se va a utilizar se llama DLXV (ya que su parte escalar es la del DLX) y su parte vectorial es la extensión del DLX. Se supone al estudiante familiarizado con la arquitectura escalar DLX y los conceptos básicos de segmentación.

Las principales características de la arquitectura DLXV son:

1. **Registros vectoriales**. El DLXV posee 8 registros vectoriales y cada registro vectorial almacena 64 dobles palabras (una doble palabra consta de 64 bits). Cada registro vectorial tiene dos puertos de lectura y uno de escritura. Además de estos registros existen 3 registros especiales VLR, MVL y VMR. El primero marca la longitud del vector en la operación vectorial (la longitud puede cambiar). El segundo es fijo e indica el valor máximo de VLR (para DLXV será 64). El tercero es la máscara vectorial booleana de longitud MVL que se utilizará para hacer operaciones con instrucciones condicionales y matrices dispersas.
2. **Unidades funcionales vectoriales**. Cada unidad está completamente segmentada y puede iniciar una nueva operación en cada ciclo de reloj (después de una cierta latencia). Se necesita una unidad de control para detectar los conflictos estructurales y las dependencias de datos. El DLXV posee varios tipos de unidades funcionales que son: Unidad de carga/almacenamiento vectorial, suma/resta en punto flotante, multiplicación en punto flotante, división en punto flotante, unidad de operaciones enteras y unidad de operaciones lógicas. El siguiente cuadro define las latencias iniciales de las unidades funcionales.

Operación	Latencia
Carga vectorial	12
Suma vectorial	6
Multiplicación vectorial	7
División vectorial	20

Las instrucciones vectoriales, cuando entran en la unidad segmentada no se distinguen de las demás, de hecho las etapas IF e ID de la máquina DLX son idénticas. Sólo cuando van a emitirse (pasar a EX) se detecta que son instrucciones vectoriales y, en tal caso, pasan a una etapa de identificación de instrucciones vectoriales: IV

Para que una instrucción vectorial pueda pasar de la etapa ID a la IV deben comprobarse las dependencias con los datos escalares o registros en coma flotante. Una vez libres de dependencias estos datos son leídos y permanecen asociados a la instrucción vectorial, la cual pasa a la etapa IV.

En la etapa IV pueden estar un máximo de N instrucciones vectoriales en colada, si llega una instrucción vectorial N+1 entonces esta deberá esperar en ID, deteniendo por tanto toda la segmentación. El valor de N es la longitud de la cola de instrucciones vectoriales y es un parámetro de la máquina vectorial. En el caso de DLXV será de 5.

Las instrucciones vectoriales que se encuentran en IV son emitidas por orden, es decir; la etapa IV se comporta como una cola FIFO, de forma que una instrucción vectorial no puede adelantar a otra en esta etapa. Por tanto, la instrucción vectorial que puede emitirse potencialmente es aquella que llegó antes a la etapa IV. Para que sea posible emitir una instrucción vectorial deben comprobarse:

1. Dependencias estructurales; con la unidad de operación vectorial necesaria y con los puertos de lectura/escritura.
2. Dependencias de datos.

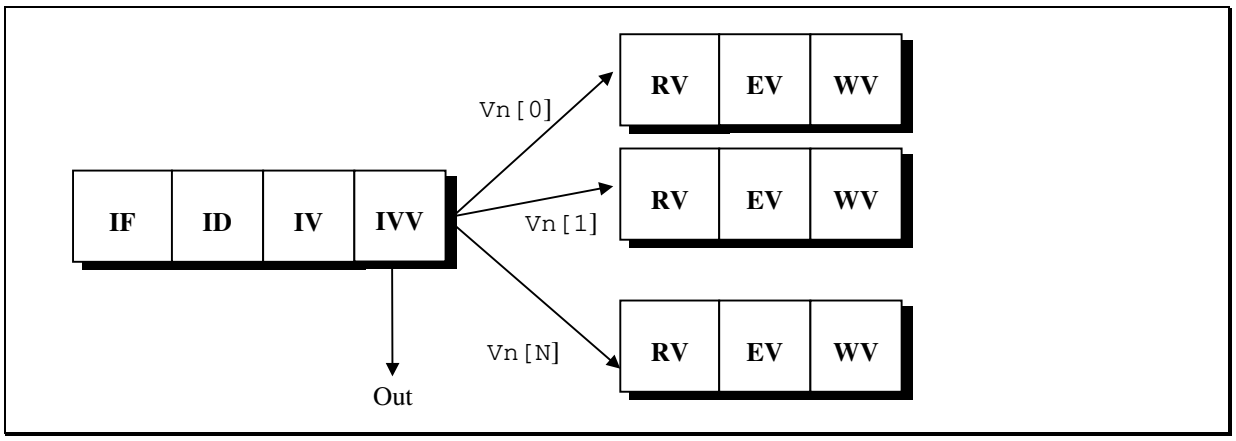
Si se emite con éxito una operación vectorial, se intentará emitir la siguiente de la etapa IV y así sucesivamente hasta que no quede ninguna instrucción en IV o bien se produzca alguna dependencia y no pueda emitirse la correspondiente instrucción vectorial. Una vez emitida una instrucción vectorial, ésta pasa a una etapa IVV. En realidad esta etapa contiene las instrucciones vectoriales que se encuentran ejecutándose. En cada ciclo de reloj, por cada una de las instrucciones que hay en IVV se lanza una operación de un elemento individual del vector. Dicha operación consta de tres etapas RV, EV y WV. Por supuesto, en estas etapas no es necesario comprobar ninguna dependencia, ya que éstas ya han sido comprobadas al emitir la instrucción vectorial.

Este esquema es el mismo para las operaciones de carga/almacenamiento vectorial, en las cuales en la etapa EV se lleva a cabo el acceso a memoria. La dirección efectiva la calcula la propia unidad de carga/almacenamiento vectorial, la cual deberá distinguir entre los tres tipos de acceso: secuencial, con *stride* y con vector de índice.

En la etapa de escritura hay que tener en cuenta si el elemento que se va a escribir resulta enmascarado por el `VMR` asociado a la instrucción y en tal caso inhibir dicha escritura.

Cuando se alcanza el último elemento del vector, definido por el valor `VLR` asociado a la instrucción vectorial; entonces dicha instrucción termina su ejecución y es eliminada de la etapa IVV.

El esquema de fases de una instrucción vectorial es, por tanto, el siguiente:



Podemos observar que la instrucción vectorial termina en la etapa IVV, mientras que las instrucciones de operaciones sobre cada elemento del vector terminan en WV. Se generan un número igual a `VLR` de operaciones elementales por cada instrucción vectorial.

3 El simulador xdlxvsim

El simulador está compilado para un entorno de X-windows bajo linux, de esta manera se deberán de arrancar las X en la partición de RedHat5.1 que ha sido utilizada en las practicas anteriores. Una vez se tenga el entorno de X se deberá ejecutar: `xdlxvsim`. Al arrancar la aplicación aparecerá una ventana con el menú principal:

Área del menú	**Botones de control**	**Botones rápidos**
File	Pipeline	Step
Execute	Clock Cycle Diagram	Run
Registers	Virtual FP Stages	
Memory	Floating Point Stages	
Configuration	Vectorial Stages	
Statistics	Vectorial Load Stages	
Breakpoints	Vectorial Registers	
About		

Menú File

Si seleccionamos el menú File tenemos las siguientes opciones:

- Reset DLXV → *Se ponen a cero todos los registros del procesador y también las estadísticas.*
- Reset ALL → *Igual que el anterior y además se limpia la memoria.*
- Load → *Se carga un programa en la memoria del DLXV desde un fichero ASCII.*
- Quit → *Sale del simulador xdlxvsim.*

Menú Execute

Dentro del menú Execute están las siguientes opciones:

- Single Cycle → *Ejecuta un ciclo de reloj a partir de la dirección que preguntará después.*
- Multiple Cycles → *Ejecuta un cierto número de ciclos a partir de una dirección.*
- Run → *Ejecuta a partir de la dirección dada hasta que encuentre una excepción.*
- Run To → *Ejecuta a partir de la dirección dada hasta otra dirección.*

Menú Registers

En el menú Registers se tienen las siguientes opciones:

- Internal → *Muestra los valores de los registros internos del procesador.*
- Special → *Son los valores de los registros especiales del procesador (PC, VLR, VM...)*
- Integer → *Muestra los valores de los registros enteros `r0 - r31 del DLX.`*
- Simple FP → *Muestra los valores de los registros en coma flotante de simple precisión `f0 - f31`*
- Double FP → *Muestra los valores de los registros en coma flotante de doble precisión `f0-f30`*
- Vectorial → *Muestra una ventana con 8 botones. Uno para cada registro vectorial.*

Menú Memory

Si seleccionamos el menú Memory vemos las siguientes opciones:

Display	→ *Con esta opción se puede ver el contenido de la parte de memoria que se desee.*
Change	→ *Se puede ver y cambiar el contenido de la parte de memoria que se desee.*
Symbols	→ *Con esta opción se pueden ver los símbolos asociados a direcciones de memoria*

Menú Configuration

Dentro del menú Configuration se tienen las siguientes opciones:

Memory	→ *Configura el tamaño de la memoria y el número de módulos de memoria.*
Floating Point	→ *Configura las unidades de operación de coma flotante.*
Vectorial	→ *Configura las unidades de operación vectoriales.*
Machine	→ *Configura las opciones de funcionamiento del procesador (planificación, etc..)*

Menú Statistics

En el menú Statistics tenemos las siguientes opciones:

Stalls	→ *Muestra estadísticas de paradas (vectoriales, enteras y de punto flotante).*
Operation Count	→ *Cuenta las operaciones ejecutadas.*
Conditional Branches	→ *Muestra las estadísticas sobre los saltos condicionales.*

Menú Breakpoints

Dentro del menú Breakpoints tenemos las siguientes opciones:

Display	→ *Muestra los puntos de ruptura establecidos.*
Set	→ *Establece un punto de ruptura en la dirección dada.*
Delete	→ *Elimina los puntos de ruptura.*

Botones rápidos: Step, Run

Los botones 'Step' y 'Run' realizan básicamente las mismas funciones que Execute - Single Step y Execute - Run, pero de una forma mas abreviada:

Botones de control

1. Pipeline: El botón *Pipeline* muestra una ventana con todas las etapas de la segmentación del procesador DLXV.
2. Clock Cycle Diagram: El botón *Clock Cycle Diagram* muestra una ventana con un esquema del diagrama de ciclos de reloj del procesador DLXV.
3. Virtual FP Stages: Muestra una ventana con el estado de las unidades virtuales en coma flotante.
4. Floating Point Stages: Muestra una ventana con el estado de los operadores en coma flotante.
5. Vectorial Stages: Esta ventana muestra el estado de cada una de las unidades de operación vectorial.
6. Vectorial Load Stages: Esta ventana muestra el estado de cada una de las unidades de carga/almacenamiento vectoriales.
7. Vectorial Registers: Esta ventana muestra información detallada sobre cada uno de los registros vectoriales.

4 Experimentando con el simulador vectorial

Una vez revisados los principales conceptos de la arquitectura vectorial DLXV se va a experimentar con diversos segmentos de código en ensamblador del DLXV para observar las principales características del procesador.

Observa el ejemplo 1 que se muestra a continuación. En el se realiza una resta vectorial bajo el control de la máscara vectorial VMR. Genera un fichero con el código propuesto y cárgalo en el simulador. Realiza una simulación paso a paso del programa desde la dirección `inicio` observando el cauce, la ejecución de instrucciones y la zona de memoria correspondiente utilizando las ventanas adecuadas del simulador.

```
; Ejemplo vectorial 1
; Modificación de la longitud del vector y ejecución condicional
; de instrucciones de carga/almacenamiento


        .data 0
        .global a
a:      .double 100, 0, 100, 0, 100, 0, 100, 0, 100, 0
        .double 100, 0, 100, 0, 100, 0, 100, 0, 100, 0
        .global b
b:      .double 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        .double 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
        .global cero
cero:   .double 0

inicio: add     r10,r0,a    ;r10 contiene la dirección del vector A
        add     r11,r0,b    ;r11 contiene la dirección del vector B
        add     r1,r0,#20   ;20 -> r1
        movi2s  vlr,r1      ;20 -> vlr Modificamos la longitud del vector
        lv      v1,r10      ;Carga vector A en V1
        ld      f0,cero     ;Carga un cero en coma flotante en F0
        snesv   f0,v1       ;Pone VM a 1 sí V1(i) distinto de F0
        lv      v2,r11      ;Carga en V2 los elementos de B que toquen
        subv    v1,v1,v2    ;Resta bajo el control de la mascara
        sv      r10,v1      ;Almacena el resultado en A

        trap    #0            ;Fin del programa vectorial
```

Comprueba las detenciones que se producen en el cauce ¿Porqué se producen estas detenciones?. Realiza una simulación del mismo código anterior pero esta vez con la opción de encadenamiento de las instrucciones vectoriales de la máquina. ¿Qué diferencia se produce en el número de detenciones?. ¿De que dependerán las detenciones en este segundo caso? Manteniendo la opción de encadenamiento de instrucciones vectoriales experimenta reconfigurando la profundidad de la segmentación para las etapas vectoriales que se utilizan en el anterior código.

Vamos ahora a seguir analizando la dependencia de las detenciones con la configuración de la máquina vectorial. Para ello ejecuta el siguiente código con la opción de encadenamiento.

```
; Ejemplo Vectorial 2

        .data 0
        .global a
a:      .double 3.14159265358979
        .global x
x:      .double 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        .double 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
        .global y
y:      .double 100, 100, 100, 100, 100, 100, 100, 100, 100, 100
        .double 100, 100, 100, 100, 100, 100, 100, 100, 100, 100

inicio: add     r10,r0,x      ;r10 contiene la dirección del vector X
        add     r11,r0,y      ;r11 contiene la dirección del vector Y
        add     r1,r0,#20     ;20 -> r1
        movi2s  vlr,r1        ;20 -> vlr
        ld      f0,a          ;Carga escalar a
        lv      v1,r11        ;Carga vector X
        lv      v2,r10        ;Carga vector Y
        multsv  v3,f0,v1      ;Multiplicación vector*escalar
        addv    v4,v1,v1      ;Suma vectores
        sv      r11,v1        ;Almacena el resultado

        trap    #0            ;Fin del programa
```

¿A que se deben las detenciones que se producen en el anterior código? ¿Cómo se podría reconfigurar el procesador para evitar estas detenciones?.

Para finalizar la experimentación se propone al alumno que escriba una rutina de ensamblador DLXV que realice la operación `Z=a*X+Y` con el menor número de ciclos posible siendo X e Y vectores de 200 elementos de números en coma flotante de doble precisión y `a` la constante π almacenada como número de doble precisión. La configuración del procesador DLXV será la indicada por defecto en el simulador. Los vectores X e Y estarán almacenados en las direcciones x e y. El vector Z lo almacenaremos a continuación de éstos y antes del código de instrucciones. Reservaremos el espacio para el vector Z con la directiva de compilación:

```
.space 1600.
```

5 Lecturas adicionales y agradecimientos

Los fundamentos de los procesadores vectoriales (y en particular del DLXV) están perfectamente documentados en *Arquitectura de Computadores. Un enfoque cualitativo*. de J. L. Hennessy y D. A. Patterson. De este libro se ha extraído prácticamente toda la información para realizar las dos prácticas de procesadores vectoriales de esta asignatura. Se recomienda una lectura detallada del capítulo 7 sobre procesadores vectoriales y DLXV.

El simulador xdlxvsim ha sido realizado por Gracia Sánchez Carpena como proyecto fin de carrera bajo la dirección del Dr. Pedro López Rodríguez, profesor titular de la Universidad Politécnica de Valencia. El profesor López ha desarrollado también el simulador de DLXV para entorno Windows y este está disponible en su página de *web* en la Universidad Politécnica de Valencia.

6 Resumen de instrucciones del DLXV

En DLXV la operación vectorial tiene el mismo nombre que en el DLX, pero añadiendo la letra "v". Estas son las instrucciones vectoriales del DLXV. Obviamente DLXV puede ejecutar también las instrucciones del DLX.

Instrucción	Función que realiza
`addv v1, v2, v3`	Suma elementos de V2 y V3, después pone cada resultado en V1.
`addsv v1, f0, v2`	Suma F0 a cada elemento de V2, después pone cada resultado en V1.
`subv v1, v2, v3`	Resta a los elemento de V2 los de V3 y pone cada resultado en V1
`subvs v1, v2, f0`	Resta a cada elemento de V2 el valor de F0 y pone cada resultado en V1
`subsv v1, f0, v2`	Resta a F0 cada elemento de V2 y pone el resultado en V1.
`multv v1, v2, v3`	Multiplica cada elemento de V2 y V3, después pone los resultados en V1.
`multsv v1, f0, v2`	Multiplica F0 por cada elemento de V2 y pone cada resultado en V1.
`divv v1, v2, v3`	Divide cada elemento de V2 y V3, después pone cada resultado en V1.
`divvs v1, v2, f0`	Divide cada elemento de V2 por F0 y pone cada resultado en V1.
`divsv v1, f0, v2`	Divide F0 por cada elemento de V2 y pone cada resultado en V1.
`lv v1, r1`	Carga V1 desde memoria comenzando en la dirección R1.
`sv r1, v1`	Almacena V1 en memoria comenzando en la dirección R1.
`lvws v1, r1, r2`	Carga V1 desde memoria comenzando en la dirección R1 con stride R2.
`svws r1, r2, v1`	Almacena V1 en memoria comenzando en la dirección R1 con stride R2.
`lvi v1, (r1+v2)`	Carga V1 desde memoria, desde las direcciones R1+V2(i). Es decir, V2 es un vector de índices.
`svi (r1+v2), v1`	Almacena V1 en memoria a las direcciones R1+V2(i). Es decir, V2 es un vector de índices.
`cvi v1, r1`	Crea un vector de índices almacenando en V1 los valores: 0, 1·R1, 2·R1, ..., 63·R1
`s_v v1, v2`	Compara (EQ, NE, GT, LT, GE, LE) los elementos de V1 y V2. Si la condición es cierta pone un 1 en el bit correspondiente del vector; en cualquier caso pone un 0. Pone el vector de bits resultante en el registro de máscara vectorial (vm).
`s_sv f0, v1`	Realiza la misma comparación que la instrucción anterior, pero utilizando un escalar (F0) como operando.
`pop r1, vm`	Cuenta los unos del registro de máscara vectorial y la almacena en R1.
`cvm`	Pone todo el registro de máscara vectorial a uno.
`movi2s vlr, r1`	Transfiere el contenido de R1 al registro de longitud vectorial (vlr).
`movs2i r1, vlr`	Transfiere el contenido del registro de longitud vectorial (vlr) a R1.
`movf2s vm, f0`	Transfiere el contenido de F0 al registro de máscara vectorial (vm).
`movs2f f0, vm`	Transfiere el contenido del registro de máscara vectorial (vm) a F0.
`snesv f0, v1`	Pone vm(i) a 1 si v1(i) ≠f0