Asignación: práctica de laboratorio

Criterios

- Análisis: Para cada punto, explicación de las respuestas aportadas y contraste con los resultados prácticos obtenidos.
- Corrección: Los supuestos son correctos, valores obtenidos justificados y coherentes
- Código: Se incluye el código utilizado para los experimentos de modo que éstos son reproducibles, demostrando así el rigor de la evidencia que se aporta

La práctica consta de una asignación de lectura y estudio previo y de 10 ejercicios o cuestiones. Debe abordarse cada uno de ellos respondiendo toda las preguntas razonadamente.

El resultado de todo ello será utilizado en la Actividad de Evaluación por Pares de Blackboard

Nota: En la guía se sugiere agregar código para realizar los experimentos incrementalmente. Es decir, en la misma fuente, puedes agregar etiquetas para definir el comienzo de la ejecución donde quieras, especificando al simulador que empiece en iniB o en xxx o en iniC o la etiqueta deseada, aprovechando valores cargados anteriormente en registros si se desea.

GUÍA DE ESTUDIO Y PRÁCTICAS

COMPUTADOR DLXV: una máquina vectorial

Los procesadores vectoriales y la arquitectural DLXV se encuentra en los siguientes libros:

- Ortega J., Anguita, M.,Prieto A. "Arquitectura de Computadores" Thomson, 2005. Capítulo 6: Procesadores vectoriales. Pág. 278 en adelante

- H-P: John Hennessy and David A. Patterson. “Arquitectura de computadores. Un enfoque cuantitativo” Mc Graw Hill. 1Ra Ed. 1993. Cap 7.2: Arquitectura Vectorial Básica (páginas 379 en adelante).

Es necesario leer al menos uno de ellos y realizar las actividades consultando esta bibliografía para comprender los conceptos que se estudian.

Repertorio vectorial

La siguiente tabla resume las instrucciones DLXV relacionadas con vectores. Fuente: HP p.383.

Instrucción	Operandos	Función
ADDV	V1,V2,V3	Suma los elementos de V2 y V3 y pone el resultado en V1.
ADDSV	V1,F0,V2	Suma F0 a cada elemento de V2 y pone el resultado en V1.
SUBV	V1,V2,V3	Resta los elementos de V3 a V2 y pone el resultado en V1.
SUBVS	V1,V2,F0	Resta F0 a los elementos de V2 y pone el resultado en V1.
SUBSV	V1,F0,V2	Resta a F0 los elementos de V2 y pone el resultado en V1.
MULTV	V1,V2,V3	Multiplica los elementos de V2 y V3; resultado en V1.
MULTSV	V1,F0,V2	Multiplica los elementos de V2 por F0; resultado en V1.
DIVV	V1,V2,V3	Divide los elementos de V2 por los de V3, resultados a V1
DIVVS	V1,V2,F0	Divide los elementos de V2 por F0, resultados a V1.
DIVSV	V1,F0,V2	Divide F0 por los elementos de V2, resultados a V1.
LV	V1,A(R1)	Carga el registro vectorial V1 desde la posición M(A+R1).
SV	A(R1),V1	Almacena el registro vectorial V1 a partir de la posición M(A+R1).
LVWS	V1,R1,R2	Carga V1 desde la dirección M(R1) con separación R2 bytes entre elementos.
SVWS	R1,R2,V1	Almacena V1 desde la dirección M(Rv21 r12) con separación R2 bytes entre elementos.

LVI	V1,R1,V2	Carga V1 con un vector cuyos elementos están en la dirección M(R1+V2(i)).
SVI	R1,V2,V1	Almacena V1 en las posiciones indicadas por la dirección M(R1+V2(i)).
CVI	V1,R2	Si VM(i)=1 v1(k++)=i*r2; i++ O sea: pone en v1 en secuencia los índices de aquellos elementos=1 en VM INCORRECTO lo que pone el H-P: ~~Crea un vector de índices, esto es, almacena en V2 los valores 0,R1,2·R1,3·R1,...,63·R~~1.
S_V	V1,V2	Compara (EQ, NE, GT, LT, GE, LE) los operandos; si la condición se cumple, pone a 1 el bit correspondiente en el registro de mascara vectorial VM y, si no se cumple, lo pone a 0. Ej: SNEV V1,V2 if V1(i) != V2(i) : VM(i) ← 1 else VM(i) ← 0
S_SV	F0,V1	Lo mismo pero usa el escalar F0 para la comparación. Ej: SNESV F0,V1
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
CVM		Pone todo el registro de mascara vectorial (VM) a 1
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
POP	R1,VM	Cuenta el número de 1's en el registro de mascara vectorial (VM) y almacena el resultado en R1.

Características importantes de DLXV

DLXV es la extensión de DLX como máquina vectorial. Ver en bibliografía su descripción básica. Debido a que para los vectores **no se utiliza cache** sino memoria principal, los accesos a memoria para recuperar datos que van a los registros vectoriales pueden realizarse **simultáneamente** con el acceso a datos escalares sin que se registren paradas.

Tanto las UF's de procesamiento como las de carga y almacenamiento están totalmente **segmentadas**, por lo tanto varios elementos del vector se encuentran en el cauce del operador simultáneamente, en distintos estados de procesamiento.

El bus de datos entre memoria y las UFs permite transferir una sola palabra doble por ciclo, tras una latencia configurable y de valor 12 ciclos por defecto. Por eso trasferir un vector de 10 componentes toma 12+10 ciclos = 22 ciclos, y transferir uno de solamente una componente, 13 ciclos. Durante estos ciclos es posible utilizar el cauce de los enteros y el de coma flotante y acceder a memoria caché (datos e instrucciones) **simultáneamente** y sin penalizaciones.

Operaciones vectoriales independientes pueden solaparse y, como en el caso de los escalares, instrucciones sin dependencias pueden emitirse una cada ciclo de reloj. En cambio, si existen dependencias de datos **RAW** entre una y la siguiente, hay que añadir una penalización debido a que **no existen circuitos de adelantamiento** como en el caso del cauce de los escalares, y es necesario

escribir y a continuación leer los operandos del banco de Registros vectoriales (**WB**). En el simulador DLXVSIM esta penalización es de solo 1 ciclo, sin embargo, en el libro de H-P menciona que son 4 ciclos. También hay penalización si hay riesgo **estructural** al utilizar la misma UF por parte de dos vectores.

Para la práctica se utilizará el simulador DLXVSIM (WinDLXV accumulates too many errors).

Funcionamiento

En DLXVSIM podemos visualizar los registros **vectoriales** (icono “V”) y registros **especiales** (menú Simulador-Examinar) nos permitirá ver el registro de **longitud** vectorial y el de **máscaras** vectorial. Notar que éste último tiene doble longitud, 64 bits, por lo que **vm** es la parte más significativa y **vmlo** los 32 bits menos significativos .

Habilita en las **estadísticas** que te muestre **operaciones pendientes**; allí verás paso a paso cuántos ciclos faltan para completar una instrucción que lleva varios ciclos como las de proceso en CF y las vectoriales. Para simulaciones sucesivas hay que inicializar las estadísticas y así borrar los resultados anteriores.

En DLXV, las unidades funcionales vectoriales están totalmente segmentadas así que la "velocidad de iniciación" (como se menciona en el libro de H-P) es 1 ciclo y los tiempos de arranque (lo que llamamos latencias) son por defecto los siguientes:

Operación	Latencias
Suma vectorial	6
Multiplicación vectorial	7
División vectorial	19
Carga vectorial	12

En el menú de configuración pueden alterarse estos valores para realizar experimentos.

La memoria, organizada en bancos tiene latencia de 12 ciclos (de ahí que la carga vectorial tenga una penalización de arranque de 12 ciclos). Posteriormente, se transmiten las componentes una tras otra desde cada banco en un ciclo de reloj, como se ha explicado antes. Si las componentes no están en módulos consecutivos y hay que cargar más de un valor desde un mismo banco de memoria, lógicamente, ocurrirán retardos. Inicialmente hay suficientes módulos en la configuración para que esto no ocurra en la mayoría de casos.

A continuación comprobaremos estas latencias y los efectos de los riesgos de datos y estructurales

1.- Latencia de carga/almacenamiento

El siguiente código solo carga el registro vectorial V0 con un vector X de hasta 64 componentes de doble precisión. Inicialmente cargaremos 1 componente para observar latencias

```
                    .data
x:          .double 1, 1, 2, 3, 4, 5, 6, 7, 8, 9
            .double 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
            .double 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
            .double 30, 31, 32, 33, 34, 35, 36, 37, 38, 39
            .double 40, 41, 42, 43, 44, 45, 46, 47, 48, 49
            .double 50, 51, 52, 53, 54, 55, 56, 57, 58, 59
            .double 60, 61, 62, 63
z:          .space 64*8
long:  .word  1
       .text
ini:   lw     r1, long(r0)
       movi2s vlr, r1
       lv     v0, x(r0)
       trap #6
```

a) Ejecuta paso a paso y tras el tercer ciclo, observa que aparece en las estadísticas la instrucción de carga y cuántos ciclos le tomará completarse: asegúrate de comprender este dato.

b) Cambia la longitud del vector y realiza cualquier otra prueba que desees para ver diferencias.

2.- Latencia de Unidades Funcionales

A continuación comprueba los efectos en los ciclos ante la existencia o no de riesgos de datos RAW y estructurales. Para ello, **añade el siguiente código al anterior** utilizando etiquetas para dirigir la ejecucion al punto deseado. Se utilizan vectores de solo un elemento para observar mejor la latencia de las UFs, puedes aumentar la longitud posteriormente si o deseas para mayor experimentación.

NOTA para el caso D: la división por 0 arroja una excepción, así que f0 debe tener un valor !=0. Pero también, si el vector v1 es 0 se produce un error en el simulador, por ello, se aconseja que el código se añada al del anterior ejercicio, para que así haya valores distintos de 0 previamente cargados.

a) Dibuja diagramas de cauce y anota tus conclusiones.

b) Aumenta la longitud del vector y mide el efecto en las paradas que se registran entre instrucciones dependientes por datos o uso de UF y también en los ciclos totales.

iniA:
addi r4,r0,1
movi2s vlr,r4
addv v3,v1,v1
multv v4,v2,v2
trap #6
	iniB:
	addi r4,r0,1
	movi2s vlr,r4
	addv v3,v1,v1
	addv v4,v2,v2
	trap #6
		iniC:
		addi r4,r0,1
		movi2s vlr,r4
		addv v3,v2,v1
		multv v4,v3,v3
		trap #6
			iniD:
			addi r4,r0,1
			movi2s vlr,r4
			ld f0, a(r0)
			divvs v2,v0,f0
			trap #6

CASO A	CASO B	CASO C	CASO D

3.- Relación ente cauces escalares y cauce vectorial

En este experimento comprobamos el funcionamiento simultáneo de los cauces, introduciendo código escalar y observando las estadísticas, memoria de enteros, registros vectoriales etc. Escribe, compila y ejecuta los siguientes programas:

```
CASO A
ini2:
  addi r4,r0,64
  movi2s vlr,r4
  lv v0,x(r0)
  ld f0,x(r0)
  lw r1,suma(r0)
  multsv v1,f0,v0
  addi r7,r0,10
  xor  r1,r1,r1
loop: add r1,r1,r7
  subi r7,r7,1
  bnez r7,loop
  sw   suma(r0),r1
trap 6
```

```
CASO B
ini3:
  addi r4,r0,64
  movi2s vlr,r4
  lv v0,x(r0)
  add r1,r0,r0
  ld f0,x(r0)
  multsv v1,f0,v0
trap 6
```

a) ¿Qué instrucción es la última en completarse en cada caso? Explica lo que sucede

b) ¿Cuántos ciclos toma la ejecución en cada caso? Calcula los ciclos, identifica y explica las paradas

4.- DAPX

Z = aX + Y donde Z, X e Y son vectores, y a es un escalar. Retomamos el ejercicio de multiplicación de escalar por vector (DAPX) hecho en las prácticas en coma flotante para reescribirlo ahora con el repertorio vectorial. El programa que hiciste entonces has de realizarlo ahora utilizando las instrucciones vectoriales, manteniendo las declaraciones de los datos. Verás que desparece el bucle y las instrucciones de manipulación de índices y contadores. Halla la ganancia al usar la versión vectorial (hay que definir la misma latencia en todos los operadores para ambos casos y comparar una vez optimizadas ambas alternativas)

5.- Scatter

Analiza el siguiente programa para averiguar qué hace y conocer el uso del registro de máscara VM y algunas instrucciones importantes del repertorio. Examina los registros vectoriales, los registros especiales (VLR y VM) y la memoria donde se almacena el resultado.

a) Explica qué hace cada una de las siguientes instrucciones y para qué se necesita en el programa:

```
i.      snesv f0,v
ii.     cvi v1,r2
iii.    movi2s vlr,r1
iv.     pop r1,vm
```

b) ¿Qué hace el programa -en una frase-?

```
.data
a: .double 100, 100, 200, 33, 100, 0, 00, 0, 00, 0
   .double 00, 0, 00, 0, 00, 0, 00, 0, 00, 0
   .double  0, 00, 0, 00, 0, 00, 0, 00, 0, 100

c: .space 30*8

.text 2000
ini:
  addi r1,r0,30
  movi2s vlr,r1
  addi r2,r0,8
  lv v0,a(r0)
  subd f0,f0,f0
  snesv f0,v0
  cvi v1,r2
  pop r1,vm
  movi2s vlr,r1
  lvi v2,r10,v1
  sv c(r0),v2

  trap 6
```

6.- Trasponer una matriz

Dada una matriz de dimensión NxN, trasponerla (intercambiar filas por columnas). Parámetros de entrada: matriz de NxN (definiendo 100 datos 10x10) y N; Debe funcionar para cualquier N <=10. Utilizar N=4 y N=10 para los experimentos, en DLXVSim.

a) Utilizando el repertorio de la máquina vectorial con 12 ciclos latencia de memoria. Optimizar para minimizar ciclos. Comentar porqué y cómo se obtiene este recuento de ciclos.

b) Utilizando el repertorio de CF convencional y optimizando para un CPI=1.

c) Comparar ambas estrategias.

~~Notar que: la instrucción CVI no funciona en los simuladores, ha de utilizarse otra de las que permite cargar datos no contiguos.~~ Puede usarse CVI o LVWS o SVWS

Nota: **Definir datos significativos que permitan verificar el funcionamiento de los algoritmos, observando el resultado en una zona de memoria distinta a la de la matriz de entrada.**