LABORATORIO DE ARQUITECTURA DE COMPUTADORES, GUÍA DEL ALUMNO

SESIÓN 4: Coma flotante en procesadores segmentados

Objetivo: Analizar el impacto que produce en un cauce segmentado la operación con datos en coma flotante. Diseñar algoritmos simples con este tipo de datos en el DLX y optimizarlos.

LEER ATENTAMENTE ESTA GUÍA

Repaso del repertorio

Como puede consultarse en los diferentes documentos de consulta del Repertorio del DLX (ver Práctica 1), este procesador :

1. Tiene 32 registros de coma flotante (*Floating Point* en ingles --> FP) F0 a F31 de 32 bits que pueden funcionar como registros individuales de precisión simple (SP) F0, F1, F2, ..., F31 o en parejas de **D**oble **P**recisión (64 bits, DP) en cuyo caso se referencian F0, F2, F4...F30. El formato es el IEEE754.
2. Para transferir datos de un registro de coma fija o enteros a FP: MOVFP2I, MOVI2FP ("move FP to I(nteger)") y para transferir datos entre registros de FP existen: MOVF, MOVD
3. Para acceder a datos FP en memoria existen las versiones de Load/Store correspondientes: LF, LD, SF, SD
4. El registro de estado de FP es un registro especial que se utiliza para evaluar condiciones de salto. Las transferencias desde/a este registro se hacen a/desde GPRs.

Además reproducimos a continuación un resumen de operaciones en FP de precisión doble (DP). En negrita las que usaremos con **toda probabilidad**:

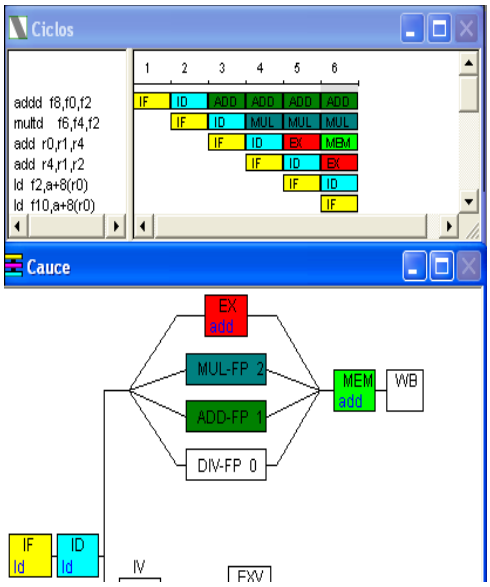
ADDD, SUBD	Suma, Resta en DP
MULTD, DIVD	Multiplica, Divide en DP
CVTD2I, CVTI2D	Conversión: CVTx2y convierte de tipo x a y. I=entero D=DP
ltD, gtD,leD.eqD.neD	Compara registros y pone bit de comparación en registro de estado FP.
BFPT, BFPF	Test (True/False) de Bit de comparación en el registro de estado FP y salto (desplazamiento de 16 bits desde PC+4)

NOTA: En el simulador DLX observamos los registros FP y la memoria con datos en coma flotante con los iconos rotulados "FP" y subíndices S o D (simple/doble precisión).

Como usaremos datos de CF doble precisión, todos ellos **ocuparán 8 bytes** en contraste con los enteros de 4 bytes. Tener cuidado con esto a la hora de declarar variables en la zona de datos.

Introducción a Coma Flotante

Las operaciones con datos representados en coma flotante toman más tiempo que las operaciones con enteros, ya que hay que hacer sumas de exponentes y mantisas, ajustes varios como alineación de mantisas, normalización, redondeos, etc. Por lo tanto, su paso por la etapa de ejecución EX toma más de un ciclo. Esto, es la LATENCIA de una operación de coma flotante puede ser superior a un ciclo.



Aunque parezca obvio, es importante destacar que las unidades de proceso en CF (suma, multiplicación, división) son diferentes a la "ALU" de enteros, por lo que es posible que dos o más instrucciones coincidan en la fase EX, pero cada una utilizando distintas unidades.

Por ejemplo, una suma entera y una suma en CF pueden ambas estar en la fase EX en el mismo ciclo, pues cada una ocupa unidades funcionales distintas; lo mismo puede ocurrir entre una suma y una multiplicación en CF.

Esto puede observarse claramente en WinDLXV donde el cauce está representado con esta diversificación en la etapa de proceso de datos: EX para los enteros, E1, E1.. o A1, A2.. para CF. En la figura se muestra una secuencia en la que tres unidades funcionales se utilizan simultaneamente.

En los simuladores que utilizamos, la latencia es configurable por el usuario. Supongamos que definimos una latencia de **4 ciclos** (duración total de la etapa de proceso de datos) para las operaciones de suma y resta que son las que utilizaremos.

IMPORTANTE:

- En DLXVSim, en el menú "Simulador" es necesario cambiar la **configuración** poniendo **4** en la latencia (ciclos totales de la etapa de proceso). A continuación hay que **modificar el hardware** (también en el menú Simulador), que es lo mismo que se muestra al arrancar el programa. Esto guarda la nueva configuración.
- En WinDLXV en **Configuración / Arquitectura** hay que poner **3** ciclos de latencia (aquí se pone el exceso de ciclos en comparación con enteros) y **DESACTIVAR la segmentación** de la unidad de suma-resta en coma flotante.
- En DLXVSim los parones LOAD-ALU contabilizan como **paradas escalares** y los ALU-ALU y ALU-STORE en **FP**
- En WinDLXV se contabiliza erróneamente dos veces alguna parada RAW.

Al emplear más ciclos en la etapa de ejecución unas instrucciones que otras, el cauce se desequilibra pues tiene diferente longitud en las instrucciones de CF que en las de enteros. Esto da lugar a **Riegos Estructurales,** cuando dos instrucciones compiten por el mismo recurso, el cual puede ser la misma unidad funcional de ejecución (etapas EXn) o la memoria (ME).

También las dependencias de datos RAW pueden causar paradas de **más** de un ciclo, aún con adelantamiento de operandos, ya que los operadores de CF en la etapa EXn consumen varios ciclos en lugar de uno.

Otro efecto un poco sorprendente es que las instrucciones no se completan en el mismo orden secuencial en el que entraron en el cauce ya que unas emplean más ciclos que otras. Todo esto es lo que vas a ir comprobando a continuación.

Riesgos Estructurales

Veamos el comportamiento del cauce **en ausencia de riesgos RAW** utilizando las instrucciones propuestas as continuación y comparando siempre en los **dos** simuladores de que disponemos.

Declara algunos valores en la zona de datos (a: .double 1,2,3...). Asegúrate de que el adelantamiento de operandos esté activado. **Haz a mano un diagrama del cauce para cada uno de los casos siguientes según los estudias en ambos simuladores y asegúrate de que lo entiendes.**

CASO A	Conflicto estructural por el sumador en CF

`addd f4,f0,f2`
`addd f4,f0,f2`
`addd f6,f2,f0`	

CASO B	Conflicto estructural por ME

`addd f4,f0,f2`
`add r1,r2,r3`
`add r1,r2,r3`
`add r1,r2,r3`	

1. En el caso **A,** comprueba que se dan 3 parones **estructurales** en cada instrucción ALU excepto la primera, ya que la unidad de suma en coma flotante toma 4 ciclos en computar una suma y las tres instrucciones han de utilizarla, aunque no tengan dependencias de datos RAW.
2. En caso **B**, también se produce un parón **estructural** pero, en esta ocasión, el recurso conflictivo es la memoria (etapa ME). Explica por qué se llega a esta situación. Puedes sustituir las sumas enteras por cualquier instrucción como cargas(*) o almacenamientos y seguirás teniendo el mismo parón estructural en ME. Este efecto solo puede verse en WinDLXV (DLXVSim no considera este conflicto de paso simultáneo por ME). (*)Si utilizas cargas en CF utiliza registros CF distintos para observarlo correctamente en WindDLXV

Segmentación de Unidades Funcionales

Al igual que se segmenta la ejecución de una instrucción (Segmentación de Cauce) también puede segmentarse una unidad funcional multiciclo, como es el sumador en coma flotante, o la unidad de multiplicación o la de división. Esto permite que se eviten largas paradas si hay una secuencia de instrucciones de proceso en CF que requieran la misma unidad funcional. WinDLX permite segmentar la unidad de suma (menú configuración / arquitectura), el otro simulador no.

Repite en WinDLX el CASO **A** de las pruebas anteriores pero esta vez configura el sumador con segmentación (Configuración/Arquitectura) y explica lo que sucede.

A continuación, introduce una dependencia RAW entre las sumas que has usado y compara lo que sucede ahora. De nuevo haz un diagrama del cauce al lado y asegúrate de comprender lo que sucede:

SIN Dependencias	UF segmentada, ausencia de RAW

`addd f4,f0,f2`
`addd f4,f0,f2`
`addd f6,f2,f0`	

CON RAW	RAW en UF segmentada

`addd f4,f0,f2`
`addd _____`
`addd _____`	

En el primer caso pero se evitan las paradas por riesgo estructural al usar la misma UF y se mejora notablemente el rendimiento.

Para simplificar el resto de experimentos **no segmentaremos** la unidad funcional, de modo que cámbiala de nuevo a NO-SEGMENTADA.

Análisis de riesgos RAW

Completa la tabla siguiente (estos datos varían según la latencia de las UFs de proceso en CF):

Productor	***Consumidor***	***Ciclos parada***	***Ejemplo***
FP ALU	FP ALU	3	addd **fx**,fa,fb / addd fa,fb,**fx**
FP ALU	Store		addd **fx**,fa,fb / Store **fx**
Load	FP ALU		ld **fx** / add fa,fb,**fx**
Load	Store		ld **fx** / sd **fx**

Recuerda que la secuencia Load-Store en WinDLXV y DLXVSim para enteros no funciona igual. Pero para el caso de coma flotante comprobarás que sí (lo que sugiere un fallo en WinDLXV).

Riesgos WAW

Un cauce que se descompensa en ejecución puede dar lugar a situaciones como la que vamos a estudiar ahora mediante los Casos 1 y 2 de la tabla a continuación.

- En el Caso 1: comprobarás que la segunda instrucción **se completa antes** que la primera; la carga de un dato de memoria emplea el mismo número de ciclos sea éste dato un entero o un CF. ¿Crees que ocurrirá lo mismo si reemplazamos la instrucción de carga por una ALU como add r1,r2,r3? ¡Claro que sí! --Compruébalo.
- En el Caso 2 las dos instrucciones mostradas escriben en **f4** con lo que se establece entre ellas una **Dependencia de datos WAW** (**W**rite **A**fter **W**rite) que no tendría ninguna importancia si el trayecto en el cauce fuera igual de largo en ambas instrucciones; pero no es así. Entonces reflexiona y explica lo que sucede. ¿Qué ocurriría si no se detectara y remediara esta situación en el procesador?.

CASO 1	**CASO 2**
`addd f4,f0,f2` `ld f6,a(r0)`	`addd **f4**,f0,f2` `ld **f4**,a(r0)`

Asegúrate de completar todos los cuadros anteriores en que hay que hacer diagramas y poner retardos, ya que forma parte de la materia a evaluar con toda seguridad.

Actividades

Para los programas propuestos siempre hay que hacer lo mismo:

a) Primero desarrollar una versión no-optimizada (provocando todos los parones posibles y desaprovechando el hueco de retardo con nop), señalando en un comentario dónde creemos que se producirán paradas y cuántas.

b) A continuación simulamos en los dos simuladores comprobando nuestras previsiones, ejecutando paso a paso al menos una iteración. Contrastamos también el total de ciclos previstos con los de los simuladores. Ajustamos lo necesario.

c) En una copia del programa, optimizamos reordenando y aprovechando el hueco con instrucción útil de modo que no queden paradas o éstas sean las mínimas, reclaculando los ciclos y la Ganancia

d) Al igual que antes, comprobamos el nuevo programa y nuestras previsiones, ajustando lo necesario.

Para las siguientes actividades cambiar las configuraciones de los simuladores para que trabajen con unidades de multiplicación y suma en coma flotante que tarden **2 ciclos en total** y que **no** estén segmentadas. Por supuesto, usar adelantamiento de resultados y también la opción de salto retardado.

1. Desarrolla un programa que, dados dos vectores a y b de números en CF de doble precisión, calcule lo siguiente: C(i) = a(i)·cte+b(i). La cte es un número real en doble precisión. Reserva el espacio necesario para el vector C. Se recomienda esquematizar el cauce SOLO al lado de las paradas para comprender qué sucede.

2. Utiliza el programa de la sesión 2 reverse.s (suma de dos vectores en otro vector en orden invertido) y transfórmalo para que procese vectores de datos en coma flotante.

3. Desarrolla un programa que calcule el producto escalar **PE** de dos vectores de hasta 20 componentes (PE = Σ a(i)·b(i)) y lo deje en una posición de memoria, es decir, una variable llamada PE. **No dejes de traer este programa para la próxima sesión, pues será un punto de partida.**

4. Transformar el programa ***condsuma.s*** de la sesión3 dedicada a Riesgos de Control para que los vectores de datos a sumar sean reales de doble precisión. Esto es, los vectores A, B, C y S han de ser números en CF de doble precisión.