LABORATORIO DE ARQUITECTURA DE COMPUTADORES, GUÍA DEL ALUMNO

SESIÓN 3: Riesgos de control en DLX

Tras una visión mediante diapositivas y ejemplos de lo que implican los riesgos de control y las técnicas de salto retardado y predicción, en esta sesión estudiaremos algunas estrategias existentes en las herramientas de simulación disponibles para tratar los riesgos de control.

Salto retardado

El simulador **DlxvSim (DLXV3.1)** utiliza la técnica del Salto Retardado. En el simulador **WinDLXV 1.0** también se utiliza esta técnica si bien hay que seleccionarla en el menú de configuración.

Cuando describimos el cauce en la sección anterior, vimos que el PC se carga en la etapa **ID** en el caso de instrucciones de salto. Pues bien, en ese caso, y según muestra la figura a continuación, el cauce tiene un hueco de retardo (HR).

n	bnez **r1**, etiq	IF	***ID***	EX	MEM	WB		
n+1	Inst. util o NOP		IF	ID	EX	MEM	WB	
n+2 o Etiq	Inst. X			**IF**	ID	EX	MEM	WB

Con salto retardado, simplemente, se ejecuta **siempre** la instrucción en el/los HR, **se salte o no**. La idea es poner allí una instrucción que normalmente figura antes de la del salto ya que ésta se va a completar incluso tomando el salto.

Esto es así para saltos condicionales e incondicionales. Por lo tanto, si no hay ninguna instrucción útil que pueda moverse para aprovechar el HR, es necesario poner una NOP. Pero entonces no se aprovecha el ciclo del HR con una instrucción útil e igualmente se desperdicia un ciclo (en un bucle, un ciclo por vuelta).

Pregunta: ¿Qué pasará si la instrucción después del salto colocado al final de un bucle es "trap #6"? --Comprueba si lo que has supuesto sucede o no--

Predicción de Salto

En el simulador **WinDLXV 1.0** es posible seleccionar una de dos técnicas (Configuración/Saltos):

- Salto retardado, como hemos visto para el caso de DLXV3.1. o
- **Predicción de salto No Tomado**.

En el último caso, se hace la suposición de que el salto NO se va a producir y se carga en el cauce la instrucción inmediata a la del salto que es la que se ejecutaría si no se toma el salto. Cuando se evalúa la condición de salto, es decir cuando se toma la decisión de salto o no-salto, entonces:

- Si la predicción fue errónea y **SÍ** ha de realizarse el salto, se **anula** en el cauce la instrucción cargada en el HR y se busca la indicada por el salto, provocando las paradas correspondientes.
- Si la predicción es correcta y el salto NO se toma, se completan las instrucciones sin paradas.

Recuerda que al usar la técnica de predicción ya la instrucción en el HR NO se ejecutará SIEMPRE como sucede con el Salto retardado por lo que no tiene sentido usar NOPs al confeccionar el programa

Actividades

1. Para la versión optimizada del programa "**reverso.s**" (suma de vectores que aparece en orden invertido) de la práctica anterior, realiza las modificaciones necesarias para aprovechar el hueco de retardo con la técnica de **salto** retardado (sustituir el **NOP** por una instrucción útil manteniendo igualmente que no haya paradas por riesgos de datos RAW). Recalcula la ganancia teórica vs. el programa inicial no reordenado (con paradas y NOPs) para N=200.

2. Sean cuatro vectores A, B C y D, de N elementos enteros. Hacer un programa llamado **condsuma.s** basado en el esquema mostrado abajo que un calcule un vector suma S tal que:

   si D(i)=0 entonces S(i) = A(i)+C(i) y si D(i)!=0 entonces S(i) = A(i)+B(i).

   a) Para el caso de Salto retardado, diseñar el código provocando el mayor número de paradas posibles, señalando en qué instrucciones hay parada de datos RAW y cuántas (si hiciste correctamente los diagramas de la pág.2 de la guía de la sesión anterior, verás que en un caso hay dos paradas RAW). Desaprovechar también los huecos de retardo.

   IMPORTANTE: calcular teóricamente el número de ciclos que tardará en ejecutar el programa (en papel, es lo más importante del ejercicio).

   Para ello completa la declaración de la zona de datos y usa el siguiente algoritmo que corresponde a un código sin optimizar y que tarda **227** ciclos con 52 paradas en DLXV3.1 debe tardar exactamente eso y funcionar correctamente, si no, es que algo va mal.

   b) Explica cómo se obtienen esas 52 paradas y los 227 ciclos.

```
R9 <-- M(N)
R7<-- 0
loop:
  R1 <-- A(r7)
  R2 <--B(R7)
  R5 <--D(R7)
  if (R5 != 0) goto contin
  R2 <-- C(R7)
contin:
   R3 <--R2+R1
   M(S+R7) <-- R3
   R7 <--R7 +4
   R9 <--R9-1
    if (R9 != 0) goto loop
---fin--
```

```
A:      .word    .....
B:      .word    .....
D:      .word    0,  5,  0,  0,  6,  5,  9, 0,  0, 0, 2, 0, 3. 44, 87
C:      .word    ....
S:      .space   ...
N:      .word    15
```

   c) Optimizarlo y aprovechar los HR al máximo. Nota: para eliminar la parada doble es necesario insertar dos instrucciones, pero en DLXV3.1, al situar solo una ya no hay paradas: esto es un error de la herramienta (en WinDLX esto sí es coherente). Hallar ganancia (en papel) extrapolando los cálculos para N= 500 elementos (asumir 50% de ceros en D).

3. Partiendo del código original no-optimizado haz las modificaciones necesarias para ejecutar con **predicción de salto no tomado** en WinDLXV el mismo código del punto anterior, **comprobando que el programa se ejecuta correctamente.** Razona lo que sucede, cuántos ciclos se tarda y **compara el rendimiento** del uso de esta técnica con la anterior para este caso concreto, cualitativa y cuantitativamente.