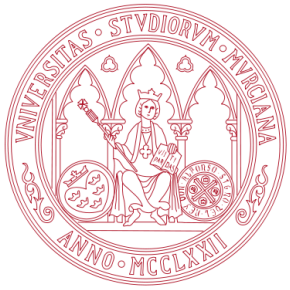

Universidad de Murcia

Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Ampliación de Estructura de Computadores

Práctica 2: Análisis de prestaciones de un cauce de instrucciones segmentado

Convocatoria de Febrero de 2025

CURSO 2024/25

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Boletines de prácticas

B2.1. Sesión 1: Análisis de la influencia de los riesgos de datos y control

B2.1.1. Objetivos

- Conocer el manejo del entorno DLXide que simula la arquitectura DLX segmentada.
- Realizar y ejecutar programas en lenguaje ensamblador DLX.
- Analizar la influencia de los riesgos de control y los riesgos de datos en el rendimiento de un procesador segmentado.

B2.1.2. El simulador de la arquitectura DLX (DLXide)

El entorno de simulación DLXide es capaz de simular ciclo a ciclo la ejecución de instrucciones de la arquitectura DLX segmentada, así como el avance de las mismas por la ruta de datos de la máquina. Soporta todas las intrucciones del DLX que operan sobre el banco de registros entero (a excepción de las de multiplicación). Implementa cachés de instrucciones y de datos separadas. Los registros se escriben en el primer semiciclo de reloj y se leen en el segundo semiciclo de reloj. Los riesgos de control pueden resolverse aplicando diversas estrategias: inserción de ciclos de parada (*stalls*), predicción estática como no tomado (*predict-not-taken*), que se verá en el boletín siguiente, y salto retardado (*delay-slot*), que no se estudia en la asignatura por ser un mecanismo en desuso. Los riesgos de datos pueden resolverse insertando ciclos de parada o aplicando la técnica de adelantamiento hardware (*forwarding*), que se analizará también en el siguiente boletín. Nótese que la ruta de datos simulada varía dependiendo de las estrategias empleadas para el tratamiento de los riesgos de control.

B2.1.3. Desarrollo

El siguiente fragmento de código DLX realiza la siguiente operación vectorial: $\vec{Z} = a + \vec{X} + \vec{Y}$

```
; z = a + x + y
; Tamaño de los vectores: 16 palabras
; Vector x
    .data
x:  .word 0,1,2,3,4,5,6,7,8,9
    .word 10,11,12,13,14,15

; Vector y
y:  .word 100,100,100,100,100,100,100,100
    .word 100,100,100,100,100,100,100,100

; Vector z
; 16 elementos son 64 bytes.
z:  .space 64

; escalar a
a:  .word -10

    .text
start:
   add r1,r0,x
   add r4,r1,#64    ; 16*4
   add r2,r0,y
   add r3,r0,z
   lw r10,a(r0)
```

```
loop:
   lw r12,0(r1)
   add r12,r10,r12
   lw r14,0(r2)
   add r14,r12,r14
   sw 0(r3),r14
   add r1,r1,#4
   add r2,r2,#4
   add r3,r3,#4
   seq r5,r4,r1
   beqz r5,loop
   trap #0      ; Fin de programa
```

El código fuente de este programa se encuentra en el mismo directorio del simulador, con el nombre `apxpy.s`.

Paso 1: Ejecutad el simulador DLXide con lo que nos aparecerá la ventana principal del programa. Para configurar el simulador, debemos acceder al menú "Simulador", opción "Configuración DLX". El programa abrirá una ventana de diálogo mostrando las estrategias disponibles para resolver los riesgos de datos y de control. Por defecto, se insertan ciclos de parada en ambos casos.

Paso 2: Cargad el fichero `apxpy.s`. Para ello, acceded al menú "Archivos", opción "Abrir", y seleccionad el fichero. Tras cargar un código fuente, éste debe ensamblarse (menú "Simulador", opción "Ensamblar" ó F8). En el caso de que hubiese errores, se mostrarían en la parte inferior de la ventana del programa. Tras corregirlos hay que volver a ensamblarlo. Cuando el programa se ensambla sin errores, se almacena en la memoria de la máquina simulada, informando al usuario. Comprobad que el código cargado corresponde al bucle mostrado anteriormente.

Paso 3: Cread la ventana de simulación para ejecutar el programa (menú "Simulador", opción "Ejecutar"). Se abrirá una nueva ventana en la que se mostrará el estado interno de la máquina DLX así como el cronograma de ejecución. Si además se pulsa en el botón "Ver CPU" se mostrará el camino de datos de la arquitectura simulada.

El simulador permite la ejecución del programa ciclo a ciclo, avanzar varios ciclos o ejecutarlo completamente (hasta encontrar una instrucción `trap #0`). Tras cada ciclo de reloj se actualiza el cronograma. Cuando se inserta un ciclo de parada, las fases en las que se mantienen las mismas instrucciones se reflejan en cursiva. También se actualiza la ruta de datos de la máquina. Cada nueva instrucción buscada recibe un color que se utiliza durante todo su recorrido por el cauce. Cuando en una de las etapas no hay ninguna instrucción, se visualiza el texto -NOP-[1]. En la parte inferior de la ruta de datos aparecen también algunas señales de control que se activan para insertar ciclos de parada, abortar instrucciones en curso o son indicativas de que se está aplicando la técnica de *forwarding*:

- *IF.stall*: Mantiene la instrucción que está en la fase IF en la misma fase durante el próximo ciclo de reloj, pasando a la fase ID el equivalente a una instrucción `nop` (burbuja).
- *ID.stall*: Mantiene la instrucción que está en la fase ID en la misma fase durante el próximo ciclo de reloj, pasando a la fase EX el equivalente a una instrucción `nop` (burbuja).
- *ID.nop, EX.nop, MEM.nop*: Convierte la instrucción que se encuentra en la fase correspondiente en una instrucción `nop`.
- *WBtoMEM, WBtoEX, WBtoID, MEMtoEX, MEMtoID*: Aplica un adelantamiento (*forwarding*) entre las etapas indicadas, mandando las señales de control adecuadas a los multiplexores.

[1] No confundáis esta indicación con la instrucción NOP real.

El objetivo de este paso 3 consiste en ejecutar el programa ciclo a ciclo para la *primera iteración* del bucle, observando el avance de las instrucciones a lo largo de la unidad segmentada, así como la inserción de ciclos de parada cuando se detectan riesgos. En concreto, se pide **especificar detalladamente** aquellas instrucciones que insertan ciclos de parada en el procesador, así como el número de instrucciones ejecutadas y el número de ciclos de parada totales para la primera iteración del bucle.

Finalmente, se ha de calcular manualmente el número de instrucciones totales ejecutadas y el número de ciclos de parada totales para la ejecución del programa.

Ejecutad el programa completamente y comprobad que se ha almacenado en la dirección definida por la etiqueta `z` el vector con el resultado del programa. Anotad el número de instrucciones ejecutadas, el tiempo transcurrido (en ciclos) y el número de ciclos de parada.

¿Coinciden los resultados con los que se han calculado manualmente anteriormente? Calculad el CPI obtenido.

Paso 4: El objetivo de este paso es hacer cambios en el código de manera que, manteniendo la corrección del programa, se reduzca en lo posible el número de ciclos de parada. Para ello tendrás que

reordernar las instrucciones de manera que las instrucciones con dependencias se alejen las unas de las otras lo suficiente para no introducir ciclos de parada. En concreto, se pide **especificar detalladamente** el código del bucle del programa `apxpy.s`, determinar las instrucciones que insertan ciclos de parada en el procesador, así como el número de instrucciones ejecutadas y el número de ciclos de parada totales para la primera iteración del bucle.

Ejecutad el programa completamente y anotad el número de instrucciones ejecutadas, el tiempo en ciclos transcurrido y el número de ciclos de parada. Calculad el CPI.