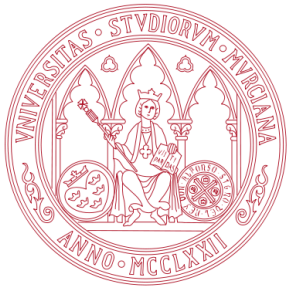


Universidad de Murcia

Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Ampliación de Estructura de Computadores

Práctica 3: Análisis de prestaciones de segmentación avanzada

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

B3.1. Sesión 1: Técnicas de adelantamiento y predición estática de saltos

B3.1.1. Objetivos

- Conocer y evaluar las técnicas de adelantamiento y predicción estática de saltos estudiadas en clase. En particular se va a evaluar el comportamiento de estas técnicas usando el entorno DLXide.
- Analizar la influencia de las técnicas de adelantamiento y predicción estática de saltos en los ciclos de parada y el rendimiento de un procesador segmentado.

B3.1.2. Desarrollo

En esta sesión usaremos el mismo código que en el boletín anterior, el programa `apxpy.s`:

```
; 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
```

Paso 1: Comenzaremos analizando la técnica de *adelantamiento o forwarding*. Para ello ejecutad el simulador DLXide y configurarlo (“Simulador”, opción “Configuración DLX”) para que resuelva los

riesgos de control con ciclos de parada, pero que los riesgos de datos se resuelvan mediante la técnica de *adelantamiento o forwarding*. Ejecutad el programa ciclo a ciclo para la *primera iteración* del bucle, observando cómo y cuándo se van aplicando los adelantamientos. En concreto, se pide **especificar detalladamente** el código del bucle del programa `apxpy.s` señalando los adelantamientos, 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. Anotad el número de instrucciones ejecutadas, el tiempo en ciclos y el número de ciclos de parada. Calculad el CPI resultante.

Paso 2: Ahora vamos a analizar la influencia de la predicción estática de saltos en el procesador segmentado. Para ello modificad la configuración del simulador para que los riesgos de control se resuelvan mediante la estrategia de predicción *predict-not-taken*, pero introduciendo ciclos de parada para resolver los riesgos de datos. Ejecutad el programa ciclo a ciclo para la *primera iteración* del bucle, observando cuándo el salto es efectivo, y cómo se cancelan las instrucciones introducidas incorrecta-

mente en el cauce. Específica las instrucciones canceladas en la primera iteración del bucle cuando se produce el salto.

Determina el número de instrucciones ejecutadas y el número de ciclos de parada totales para la primera iteración del bucle.

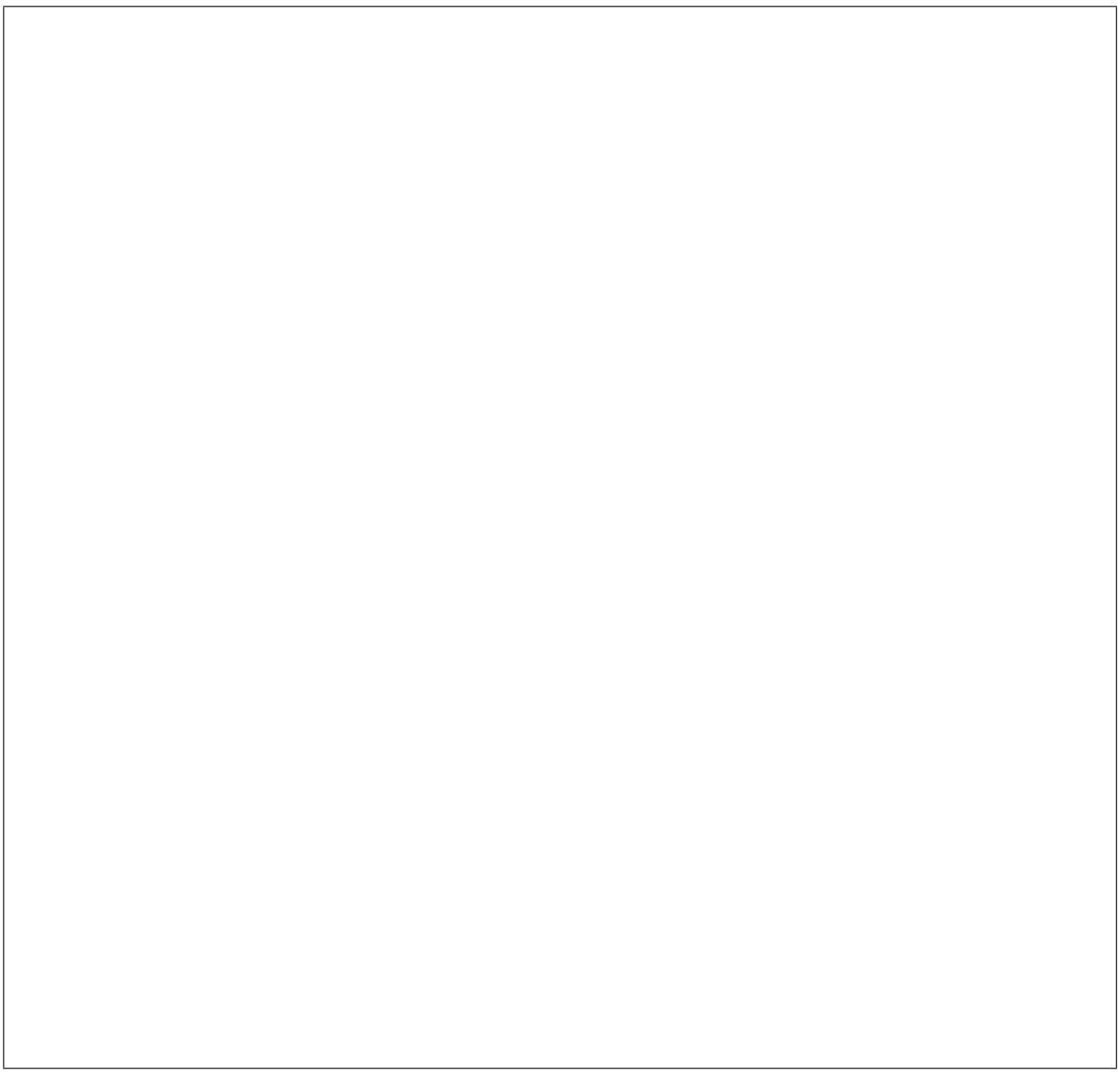
¿Cuantos ciclos de parada se producen cada vez que se ejecuta la instrucción de salto en cada una de las iteraciones del bucle?

Ahora ejecutad el programa completamente y anotad el número de instrucciones ejecutadas, el tiempo total (en ciclos) y el número de ciclos de parada. Calculad el CPI obtenido.

Paso 3: Veremos ahora la combinación de ambas técnicas de reducción de ciclos de parada para datos y control. Seleccionando las estrategia de predicción *predict-not-taken* junto con la técnica de adelantamiento, ejecutad el programa completamente y anotad el número de instrucciones ejecutadas, el tiempo total (en ciclos) y el número de ciclos de parada. Calculad el CPI obtenido.

¿En qué medida mejora el CPI respecto al paso 1 (adelantamientos)? ¿Y respecto al paso 2 (predicción estática de saltos)?

Paso 4: Por último, seleccionando las estrategia de predicción *predict-not-taken* junto con la técnica de adelantamiento (configuración anterior), modificad el código para reducir la penalización por riesgos de datos. En concreto, se pide **especificar detalladamente** el código del bucle del programa `apxpy.s` señalando los adelantamientos, 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 transcurrido y el número de ciclos de parada. Calculad el CPI obtenido.

¿Qué mejoras obtienes con respecto al paso 3 (con el código original)?

B3.2. Sesión 2: Predicción de Saltos

B3.2.1. Objetivos

- Conocer las técnicas de predicción de saltos estática y dinámica estudiadas en clase.
- Entender los conceptos de contadores saturados y correlación de saltos.
- Ser capaz de entender e implementar un predictor de salto sencillo.
- Evaluar diversos factores que mejoran la predicción como la reducción de *aliasing* o la correlación de saltos.

B3.2.2. El simulador ChampSim

Para el desarrollo de esta sesión utilizaremos el simulador *ChampSim* que se ha utilizado recientemente en competiciones relacionadas con la predicción (en particular, en técnicas de prebúsqueda, como se verá en el boletín 4). Este simulador permite desarrollar de manera muy sencilla diferentes técnicas de predicción de salto o prebúsqueda, entre otras.

La compilación del simulador es muy sencilla y simplemente hay que escribir el comando "make".

Una vez complilado el simulador tendremos el ejecutable en el directorio "bin", bajo el nombre de "champsim".

Para ejecutarlo, lo podemos hacer mediante el script "run_champsim.sh" indicándole el nombre de la aplicación que queremos ejecutar. Las aplicaciones, que en realidad son trazas obtenidas a partir de las aplicaciones y que son la entrada del simulador, se encuentran en el directorio "traces". Estas cuatro aplicaciones pertenecen al conjunto de aplicaciones SPEC CPU 2017. Un ejemplo de comando para la ejecución es:

```
./run_champsim.sh traces/605.mcf.xz
```

Al ejecutar dicho comando el simulador nos mostrará estadísticas de la ejecución de la aplicación.

B3.2.3. Desarrollo

Paso 1: Ejecutad la aplicación `mcf` tal y como se ha descrito en la sección anterior y observad la salida del simulador. Buscad la línea donde se indica la tasa de acierto del predictor ("Branch Prediction Accuracy") y el IPC (inverso del CPI) y escribe los resultados.

Mirad ahora el fichero donde se describe el predictor de saltos ("branch/branch_predictor.cc") y abridlo con un editor de texto. La función "predict_branch" le dice al procesador qué predicción tomar dependiendo del PC (program counter) de la instrucción de salto. ¿Qué mecanismo de predicción estamos usando?

Paso 2: Cambiad ahora la predicción para que devuelva siempre tomado, compila de nuevo con "make" y ejecuta el simulador con la aplicación `mcf`. ¿Qué tasa de acierto se obtiene ahora? ¿Tiene sentido este resultado? ¿La mayoría de los saltos de la aplicación se toman o no se toman?

¿Y qué IPC se obiene ahora? En esta práctica simulamos un procesador más avanzado del visto en clase ¿Tendría sentido este resultado en el procesador DLX visto en clase, en el que la dirección de destino del salto se obtiene a la vez de que la condición del salto? ¿Por qué?

Realiza la misma prueba con la aplicación `gcc`. ¿Ahora la mayoría de los saltos son tomados o no tomados? Habréis observado por tanto que dependiendo de la aplicación, la predicción estática realizada no obtiene siempre la mayoría de los saltos bien predichos.

Paso 3: Vamos ahora a adaptar dinámicamente nuestra predicción dependiendo de como se vaya comportando la aplicación. Comenzaremos con un contador saturado de 2 bits para todos los saltos. Este contador se puede defirnir como una variable global puede tomar los valores 0, 1, 2 y 3. El contador se inicializa a 2 (función "initialize_branch_predictor"), se incrementa si el salto se toma y es menor que 3 y se decrementa si el salto no se toma y es mayor que 0 (función "last_branch_result") y se predice tomado si el valor del contador es 2 o 3 y no tomado si el valor es 0 o 1 (función "predict_branch").

Una vez implementado compilad de nuevo y ejecutad las aplicaciones `mcf` y `gcc`. ¿Qué tasa de predicción se obtiene ahora para `mcf` y `gcc`? ¿Hemos mejorado con respecto a la predicción estática? ¿Por qué?

Paso 4: En el caso anterior estámos tratando todos los saltos por igual y el comportamiento de un salto puede afectar a la predicción de otro. Muchos de los saltos tienden a tener un patrón mayoritariamente tomado o no tomado (por ejemplo en bucles). Sería recomendable por tanto tener un contador por cada salto. Como el número de saltos depende de la aplicación vamos a simplificar y a usar 64 contadores independientemente de la aplicación. Cada salto usará un contador en la medida de lo posible. Para saber qué contador usa cada salto vamos a definir una función *hash* que dada la dirección del salto nos diga cuál de los 64 contadores debe usar. Una función simple es PC % 64.

Cambiad el contador anterior por un array de 64 entradas, y actualizad el código en base a lo descrito anteriormente. Ejecutad `gcc` y `mcf`. ¿Qué resultados se obtienen ahora? ¿Mejora ahora la predicción?

¿Seríais capaces de calcular qué tamaño tiene la tabla de predicción?

Paso 5: Podríamos pensar a la vista de los resultados anteriores que ya hemos solucionado el problema de la predicción de una forma muy simple, pero sin embargo no es así. Ejecutad ahora las aplicaciones `lbm` y `wrf`, y comprobad que la tasa de acierto para estas aplicaciones no es tan alta. Vamos a estudiar primero si el problema se debe al aliasing, es decir dos saltos distintos están usando el mismo contador ya que aunque su PC es distinto, su PC % 64 es el mismo. Para ello vamos a incrementar el número de contadores en potencias de dos hasta 1024, es decir, 64, 128, 256, 512 y 1024 entradas en la tabla de contadores (adapta convenientemente la función hash). Anota las 10 tasas de acierto en la siguiente tabla.

Tamaño	lbm	wrf
64		
128		
256		
512		
1024		

¿Cuál de las dos aplicaciones (`lbm` o `wrf`) obtiene un mayor beneficio al ir aumentando el tamaño de la tabla de predicción? ¿Cuántas entradas de la tabla de predicción crees que son suficientes para `lbm` y cuántas entradas serían suficientes para `wrf`?

¿Cómo crees que el *aliasing* está afectando a ambas aplicaciones?

¿Por qué llega un momento en que la tasa de acierto satura?

Paso 6: Por último vamos a intentar mejorar aún más la predicción mediante la técnica de correlación. Nuestra decisión ahora no va a depender solo del PC del salto sino también de cómo se haya comportado los últimos saltos independientemente de su PC. Para ello necesitamos la historia global de comportamiento de los saltos. La historia es una ristra de 0s y 1s donde 0 indica que el salto no fue tomado y 1 que sí lo fue. La historia se puede actualizar en cada salto de forma muy sencilla:

```
historia_global = historia_global << 1; // Desplazamos 1 bit a la izquierda
if (taken) historia_global++; // Añadimos un 1 a la derecha si el salto fue
                              tomado, si no se deja en 0
```

A la hora de ver qué contador usar, usamos una mezcla del PC y la historia global. Una de las mejores formas de hacer esto es mediante el operador XOR: `PC ^ historia_global`. Una vez hecho esto, buscamos la entrada de la tabla igual que en el paso anterior con el módulo (%) del número de entradas de la tabla. Ejemplo para 1024 entradas sería: `(PC ^ historia_global) % 1024`.

Implementa este predictor, conocido como **gshare**, y compara los resultados para `lbm` y `wrf` en el apartado anterior. ¿Qué resultados se obtienen ahora? ¿Se ha mejorado la predicción?

Paso 7: Calcula ahora la mejora en IPC del predictor gshare para las 4 aplicaciones evaluadas respecto a la predicción estática siempre no tomado.

Si tuvierais que sacar la media, ¿cuál usaríais para el IPC? ¿Cuál sería la mejora media para las 4 aplicaciones?

B3.3. Sesión 3: Segmentación de Punto Flotante

B3.3.1. Objetivos

- Ser capaz de analizar y evaluar el diseño de un cauce segmentado para instrucciones de punto flotante utilizando un simulador gráfico.
- Conocer la influencia de los riesgos de datos, estructurales y de control en las operaciones de punto flotante.

B3.3.2. Desarrollo

Para realizar esta sesión se usará el simulador Simula3MS1 (versión 5.01) que implementa un procesador MIPS segmentado y con ejecución en orden, similar al estudiado en clase. El manual del simulador está disponible en el aula virtual.

El programa `P3-codigo1.s` realiza la operación vectorial $\vec{Z} = \frac{a \times \vec{X} + \vec{Y}}{b}$, donde $\vec{Z}$, $\vec{X}$ e $\vec{Y}$ son vectores de cien elementos de doble precisión y a y b son constantes de simple precisión. Tenéis disponible el fichero en el mismo directorio del simulador. El código es el siguiente:

```
# Datos
.data
a: .float 10
b: .float 5
#vector x
x:  .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
x1: .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
x2: .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
x3: .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
x4: .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
#vector y
y:    .double 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
y1:   .double 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
y2:   .double 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
y3:   .double 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
y4:   .double 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
#vector z
z:    .space 800

# Código: z[i]=(a*x[i]+y[i])/b
.text
.globl main
main:
        la $t1, x            #carga la dir de x[0] en $t1
        la $t2, y            #carga la dir de y[0] en $t2
        la $t3, z            #carga la dir de z[0] en $t3
        la $t4, a            #carga la dir de "a" en $t4
        la $t5, b            #carga la dir de "b" en $t5
        addi $t6,$t1,800     #preparamos el contador para detectar el final
        lwc1 $f0, 0($t4)     #carga "a" en f0
        cvt.d.s $f0, $f0     #convertimos "a" en DP y lo guardamos en $f0:$f1
        lwc1 $f2, 0($t5)     #carga "b" en f2
        cvt.d.s $f2, $f2     #convertimos "b" en DP y lo guardamos en $f2:$f3

loop:
        lwc1 $f4,0($t1)      #f4=x[i] (parte alta)
        lwc1 $f5,4($t1)      #f5=x[i] (parte baja)
        lwc1 $f6,0($t2)      #f6=y[i]
        lwc1 $f7,4($t2)      #f7=x[i]
        mul.d $f8,$f0,$f4    #f8=a*x[i]
        add.d $f8,$f8,$f6    #f8=a*x[i]+y[i]
        div.d $f8,$f8,$f2    #f8=(a*x[i]+y[i])/b
        swc1 $f8, 0($t3)     #z[i]=(a*x[i]+y[i])/b (Parte alta)
```

```
swc1 $f9, 4($t3)     #z[i]=(a*x[i]+y[i])/b (Parte baja)
addi $t1,$t1,8       #actualizamos todos los índices
addi $t2,$t2,8
addi $t3,$t3,8
bne $t1, $t6, loop   #comprobamos si hemos terminado el bucle
addi $v0, $0, 10     #lamada para salir del programa
syscall
```

Paso 1: Ejecutad el simulador Simula3MSv5.01 (desde un terminal de Linux se puede ejecutar con el comando: "java -jar Simula3MSv5.01.jar") con lo que nos aparecerá la ventana principal del programa. Para configurar el simulador, debemos acceder al menú "Configuración" y seleccionar la opción "Camino de Datos/Segmentado/Básico/Con Adelantamiento". El programa abrirá una ventana de diálogo mostrando las opciones para las distintas unidades funcionales. Selecciona una unidad sumadora, otra multiplicadora y otra de división con latencias de 4, 7 y 10 ciclos, respectivamente, estando las dos primeras unidades segmentadas pero no la de división y pulsa el botón "Aceptar". De nuevo, se debe acceder al menú "Configuración" y seleccionar la opción "Salto/Un Hueco/Predecir no tomado".

Paso 2: Cargad el fichero `P3-codigo1.s`. Para ello, acceded al menú "Archivo", opción "Abrir", y seleccionad el fichero. Tras cargar un código fuente, éste debe ensamblarse (pulsando sobre el botón "Ensamblar"). 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, dejando la parte inferior de la ventana del programa en blanco. Comprobad que el código cargado corresponde al bucle mostrado anteriormente.

Paso 3: Cread la ventana de simulación para ejecutar el programa (pulsar sobre el botón "Ejecutar"). Se abrirá una nueva ventana en la que se mostrará el camino de datos del procesador MIPS así como el diagrama multiciclo y monociclo en las distintas pestañas. Además, se observa el contenido de los Registros generales y de punto flotante, la Memoria (Segmento de datos) y el Código del programa ensamblado (Segmento de texto).

El simulador permite la ejecución del programa ciclo a ciclo (Botones Ciclo siguiente y Ciclo anterior) o ejecutarlo, mediante el botón "Ejecutar", completamente (hasta encontrar una instrucción `syscall`). Tras cada ciclo de reloj se actualiza el diagrama multiciclo y monociclo. Las distintas instrucciones que se encuentran en el camino de datos aparecen en distintos colores. Cuando se inserta un ciclo de parada, se inserta una instrucción nop en la etapa del camino de datos y aparece un simbolo de burbuja (*burb*) en el diagrama multiciclo.

El objetivo de este paso 3 consiste en ejecutar la primera iteración del programa ciclo a ciclo, utilizando para ello la vista "Diagrama Multiciclo". Observad el avance de las instrucciones a lo largo del cauce, 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 (NI), el número total de ciclos y el de ciclos de parada para la primera iteración del bucle.

Paso 4: Calcula 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 anotad el número de instrucciones ejecutadas, el número de total de ciclos y el número de ciclos de parada.

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

Paso 5: Trata de planificar el bucle, reordenando instrucciones como estimes conveniente con el fin de minimizar las detenciones que se producen, y calcula el CPI de esta nueva versión, así como la ganancia que se consigue respecto a la versión previa.

Paso 6: El programa `P3-codigo2.s` realiza la operación vectorial $\vec{Z} = a \times \vec{A} + b \times \vec{B} + c \times \vec{C} + d \times \vec{D}$, donde a, b, c y d son constantes de simple precisión y $\vec{A}$, $\vec{B}$, $\vec{C}$, $\vec{D}$ y $\vec{Z}$ son vectores de doble precisión de tamaño 10.

Utilizando las opciones apropiadas del simulador Simula3MS define una configuración con adelantamiento hardware, donde el salto se predice no tomado, con un ciclo de parada en el caso de ser tomado y que tiene las siguientes unidades funcionales:

- 1 unidad sumadora (no segmentada) de punto flotante con 4 ciclos de latencia.
- 1 unidad multiplicadora (no segmentada) de punto flotante con 7 ciclos de latencia.

- 1 unidad divisora (no segmentada) de punto flotante con 10 ciclos de latencia.

Ejecuta el programa completamente y anota el CPI obtenido, el total de ciclos de detención y su desglose en riesgos de datos y estructurales.

Paso 7: Compara los resultados con los que se obtienen al tener DOS unidades sumadoras y DOS unidades multiplicadoras sin segmentar y manteniendo las mismas latencias del apartado anterior. Anota el CPI obtenido, el total de ciclos de detención y su desglose en riesgos de datos y estructurales. Fíjate en cuál de los dos tipos de riesgos se ve mejorado al aumentar el nº de UFs.

Paso 8: Con las mismas latencias que en los apartados anteriores, incrementa el número de unidades funcionales de punto flotante al valor mínimo que consideres necesario para conseguir evitar los riesgos estructurales (sin reorganizar el código). ¿Cuántas UFs necesitamos de cada tipo? ¿Cuántas detenciones hay y de qué tipo son? ¿Qué CPI se obtiene?

Ahora vamos a segmentar tanto el sumador como el multiplicador de punto flotante y vamos a analizar cómo esta decisión influye en los resultados. Para ello calcula manualmente cuántos serían ahora los riesgos (estructurales y de datos). A partir de estos números ¿cuál sería el nuevo CPI? Justifica la respuesta. Verifica los resultados calculados manualmente con los que proporciona el simulador.
