3. SEGMENTACIÓN AVANZADA Y PARALELISMO DE INSTRUCCIONES: EJERCICIOS Y CUESTIONES

3.1. A) Enumera las dependencias, antidependencias y dependencias de salida que aparecen en el siguiente bucle e indica si se dan en la misma o entre distintas iteraciones.

```
for (i=1; i<100; i=i+1)
{       a[i] = b[i] + c[i];       /* S1 */
        b[i] = a[i] + d[i];       /* S2 */
        a[i+1] = a[i] + e[i];     /* S3 */
}
```

B) Escribe un bucle equivalente que sea paralelo

3.2. Supongamos las latencias para DLX que aparecen en la transparencia 5 del fichero ILP1.pdf. Desenrolla el siguiente bucle una vez de modo que, debidamente planificado, se ejecute sin ciclos de espera. Aunque el bucle no es paralelo se puede planificar sin ciclos de espera.

```
loop:   LD      F0, 0(R1)
        LD      F4, 0(R2)
        MULTD   F0, F0, F4
        ADDD    F2, F0, F2
        SUBI    R1, R1, 8
        SUBI    R2, R2, 8
        BNEQZ   R1, loop
```

Se sabe que el número de pasadas por el bucle es par y que, inicialmente, F2 y F10 valen cero.

3.3. En un procesador superencauzado se tiene un BTB (*branch-target buffer*) exclusivamente para bifurcaciones. Cuando la instrucción apuntada por el PC es una bifurcación, la probabilidad de acertar en el buffer es del 90% y la penalización por fallar es de 3 ciclos. El porcentaje de acierto en las predicciones es del 90% y la penalización por predicción errónea es de 4 ciclos. El 15% de las instrucciones ejecutadas son bifurcaciones. Se desea conocer la mejora de velocidad del procesador con BTB frente a un procesador con una penalización de las bifurcaciones fija de 2 ciclos. Supóngase que el CPI sin contar las paradas por bifurcaciones es 1.

3.4. A) En el código de la transparencia 9 del fichero ILP4PlanificacCompil.pdf, hay una falsa dependencia, ¿cómo la eliminarías? ¿Cómo quedará después el grafo de dependencia de datos? Aplica el algoritmo de Warren para planificar el bloque básico para un procesador escalar. B) ¿Cómo se planificaría ese bloque básico para un superescalar capaz de emitir dos instrucciones cualesquiera en cada ciclo? ¿Podría ejecutarse el bloque básico en menos ciclos aumentando el número de instrucciones que puede emitir el procesador en cada ciclo?

3.5. En el hardware para el *scoreboard* las unidades funcionales se organizan en grupos que comparten los buses que llevan datos del banco de registros a las entradas de las unidades funcionales y el bus que lleva el resultado de cualquier unidad funcional del grupo al banco de registros. A) Supongamos que tenemos en un mismo grupo dos unidades funcionales de punto flotante de multiplicación. Explica qué parada se producirá en la ejecución del siguiente fragmento de código

```
ADDD    F0, F2, F4
MULTD   F6, F0, F10
MULTD   F8, F0, F10
```

que no se produciría con el hardware y algoritmo de Tomasulo.

B) En cambio, en la ejecución del siguiente fragmento de código, suponiendo que MULTD gasta 4 ciclos en EX, se producirá una parada con el método de Tomasulo que no se producirá con el *scoreboard*. Explícala.

```
MULTD   F0, F2, F4
NOP
NOP
LD      F6, 24(R8)
```

3.6. Consideremos que el siguiente bucle, llamado SAXPY, se ejecuta 200 veces en un procesador, con cortocircuito para las operaciones enteras, en el que se ignora el retardo de las bifurcaciones.

```
foo:  LD      F2, 0(R1)     ; carga X(i)
      MULTD   F4, F2, F0    ; multiplica a·X(i)
      LD      F6, 0(R2)     ; carga Y(i)
      ADDD    F6,F4,F6      ; suma a·X(i)+Y(i)
      SD      0(R2), F6     ; almacena Y(i)
      ADDI    R1,R1,8       ; actualiza índice de X
      ADDI    R2,R2,8       ; actualiza índice de Y
      SGTI    R3, R1, done  ; Si R1> done, R3 ← 1
      BEQZ    R3, foo       ; salto
```

A) **Scoreboard**. Supuestos: a) una instrucción que tiene como operando fuente el destino de otra puede leer el operando en el mismo ciclo en que la otra escribe el resultado; b) en el mismo ciclo en que una instrucción termina WR se puede emitir otra instrucción que necesite la misma unidad funcional; c) sólo hay una unidad funcional entera; d) una multiplicación FP requiere 6 ciclos y una suma FP, 4.

Se pide el estado del scoreboard en el ciclo en que la bifurcación es emitida por segunda vez. Utiliza el formato visto en clase más una tabla en que se indique en qué ciclo o ciclos ha estado cada instrucción en cada etapa.

Supóngase que la bifurcación consume un ciclo en cada etapa e ignórense posibles conflictos en el acceso a los buses que conectan los registros con las unidades funcionales. ¿Cuántos ciclos consume cada pasada por el bucle?

B) **Tomasulo**. Supuestos: a) hay suficiente número de estaciones de reserva como para que ninguna instrucción no pueda ser emitida por falta de ellas; b) en el mismo ciclo que una instrucción termina WR puede comenzar la ejecución de otra que necesita el resultado de la primera; c) en las instrucciones *load* y *store*, la etapa EX calcula la dirección y accede a memoria en un solo ciclo; d) se dispone de una unidad funcional FP encauzada y una unidad funcional entera; e) una multiplicación FP requiere 7 ciclos y una suma FP, 5.

Muestra el estado del procesador en el ciclo en que la bifurcación se ejecuta por segunda vez. Utiliza el formato visto en clase más una tabla con los ciclos en que ha estado cada instrucción en cada etapa. ¿Cuántos ciclos consume cada pasada por el bucle?

C) **Superescalar**. Supuestos: a) se pueden emitir dos instrucciones cualesquiera en el mismo ciclo; b) las latencias son las que aparecen en la transparencia 5 del fichero ILP1.pdf; c) hay dos unidades funcionales enteras, una UF encauzada FP de suma y otra de multiplicación; d) existe cortocircuito; e) se ignoran los retardos de salto y de carga.

Desenrolla el código tres veces y planifícalo. Por cada instrucción indica el ciclo en que se emite y los ciclos en los que se ejecuta. ¿Cuántos ciclos de reloj consume ahora cada iteración del bucle original? ¿Cuál es la mejora en velocidad frente al código original en un procesador escalar?

D) **VLIW**. Supuestos: a) latencias de FP: las que aparecen en la transparencia 5 del fichero ILP1.pdf; b) formato de las instrucciones, el mismo que hemos visto en clase; c) ignorar retardos de salto.

Desenrolla el código tres veces y planifícalo. ¿Cuántos ciclos de reloj consume ahora cada iteración del bucle original?

E) **Especulación**. Supuestos: a) una única unidad funcional entera y las mismas unidades funcionales FP vistas en clase; b) latencias, las de la tabla anterior salvo que no hay retardo de carga; c) se dan por ciertos los supuestos del apartado B de este ejercicio.

Muestra el estado del procesador en el ciclo en que la bifurcación se emite por segunda vez. Utiliza el formato visto en clase y añade una tabla en que se indique en qué ciclos se ha producido el procesamiento de cada instrucción por cada etapa. Indica cuántos ciclos consume cada iteración del bucle.

F) Procesador **especulativo superescalar**. Supuestos: a) en cada ciclo se puede emitir una operación entera, una FP y una de acceso a memoria; b) las mismas latencias del apartado F; c) se aumenta la anchura del CDB para permitir que se escriban en él hasta tres resultados a la vez; d) se dan por ciertos los supuestos del apartado B.

Muestra el estado del procesador en el ciclo en que la bifurcación se emite por segunda vez (utiliza el mismo formato del apartado anterior). Indica cuántos ciclos consume cada iteración del bucle.

3.7. Escribe, para DLX superescalar, el código resultante del encauzamiento software del bucle

```
Loop:   LD    F0, 0 (R1)
        ADDD  F4, F0, F2
        SD    0 (R1), F4
        SUBI  R1, R1, 8
        BNEZ  R1, Loop
```

sabiendo que el patrón repetitivo está formado por SD, ADDD y LD. En concreto, el patrón repetitivo está formado por la instrucción SD que almacena el nuevo elemento V[i] del vector, la instrucción ADDD que calcula el nuevo valor del elemento V[i-1] y la instrucción LD que carga el antiguo valor de V[i-2] (i=R1/8). Incluye el código de inicio y el de finalización.

3.8. Tomemos el bucle del ejercicio anterior y supongamos que la latencia entre operaciones de punto flotante es 5 ciclos. El encauzamiento software del bucle resulta tener ahora una parada. Muestra como, el encauzamiento software del bucle puede funcionar sin paradas si, previamente, el bucle original se desenrolla una vez.