2. SEGMENTACIÓN ENCAUZADA AVANZADA Y PARALELISMO DE INSTRUCCIONES: EJERCICIOS Y CUESTIONES

2.1. Completa tus conocimientos del MIPS64 (una versión de MIPS). Debes aprender qué registros tiene, tipos de datos, modos de direccionamiento y operaciones soportadas por el juego de instrucciones. Puedes encontrar esta información en Hennessy y Patterson [2002, pp. 130 y ss.] (el libro sólo utiliza un subconjunto del juego de instrucciones, pero es suficiente para nosotros). Especialmente interesante es la figura 2.31 en p. 137. Debes familiarizarte con el significado de expresiones como GPR, FPR, DP, SP y los nombres en ensamblador de las instrucciones. Por ejemplo, ¿qué diferencia hay entre DMUL, DMULU, MUL.D y MUL.S?

2.2. En una CPU encauzada la dirección de salto es conocida en la tercera etapa y la condición de salto una etapa después. Completa la siguiente tabla con las correspondientes penalizaciones, en ciclos, del salto/bifurcación.

Tratamiento de saltos	Penalización salto (incondicional)	Penalización bifurcación no efectiva (condicional)	Penalización bifurcación efectiva (condicional)
Parar instrucciones posteriores (desde que se descodifica hasta que se resuelve el salto o bifurcación) e ignorarlas			
Predecir salto no efectivo			
Predecir salto efectivo			

2.3. Supongamos que los saltos incondicionales son un 5% de todas las instrucciones mientras que los condicionales (bifurcaciones) son un 20%. Además los condicionales provocan salto un 60% de las veces. La dirección de salto se conoce al final del segundo ciclo en los saltos incondicionales, en tanto que las bifurcaciones se resuelven al final del tercer ciclo. Suponiendo que únicamente la primera etapa del cauce puede estar activa desde que un salto/bifurcación entra en el cauce hasta que se resuelve, calcula cuánto aumenta el CPI en relación a una máquina sin riesgos de control. (Ignorar otros posibles riesgos).

2.4. En R4000 una instrucción de suma en punto flotante ha finalizado la etapa RF. ¿Con cuántos ciclos de antelación ha tenido que ser emitida una instrucción de multiplicación para que la instrucción de suma tenga que ser detenida (no emitida) para evitar colisionar con ella? Ídem en el caso de una instrucción de división emitida con antelación. Utiliza la información, sobre el R4000, de la transparencia "Cauce del MIPS R4000, p. 5" [fichero *1InterruptMulticiclo.doc*].

2.5. Las interrupciones de entrada/salida pueden ser tratadas de un modo más sencillo que las internas con retorno. ¿Cómo?

2.6. Típicamente una rutina de tratamiento de una interrupción interna necesita conocer los valores de entrada de la instrucción interruptora. En casos así es prácticamente imprescindible que la interrupción sea precisa. ¿Por qué?

2.7. 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, es decir, que no tenga dependencias *loop-carried*.

2.8. Completa la siguiente tabla indicando, por cada fragmento de código MIPS64: a) las dependencias de datos que, en su caso, existen o pueden existir; b) si una planificación dinámica es, puede ser o no es suficiente para permitir que la ejecución de las dos instrucciones del fragmento se haga *fuera de orden*, y por qué. Supuestos: la bifurcación no es retardada y los accesos a memoria están alineados.

Fragmento de código	**Dependencias de datos**	**¿Es posible la ejecución fuera de orden si hay planificación dinámica?**
DADDI R1,R1,#4 LD R2, 7(R1)		
DADD R3,R1,R2 SD R2, 7(R1)		
SD R2, 7(R1) S.D F2,200(R7)		
BEZ R1,lab SD R1, 7(R1)		

2.9. Considera el siguiente fragmento de código:

```
loop:  LD      R1, 0(R2)
       DADDI   R1, R1, #1
       SD      R1, 0(R2)
       DADDI   R2, R2, #8
       DSUB    R4, R3, R2
       BNEZ    R4, loop
```

Se sabe que el valor inicial de R3 es R2+792 y que todos los accesos a caché son aciertos.

A. Supón que no hay cortocircuito y que las instrucciones de bifurcación no son retardadas y actualizan el PC en MEM. Además, cada vez que se detecta una bifurcación se detienen las instrucciones posteriores hasta que se conoce la instrucción por la que seguir. Dibuja un diagrama instrucciones-tiempo correspondiente a la ejecución en MIPS64 del fragmento de código considerado. Dibuja sólo la parte necesaria para conocer cuantos ciclos requiere la ejecución completa del bucle.

B. Responde a la misma cuestión (diagrama y número de ciclos) suponiendo que hay cortocircuito y se predice salto no efectivo.

C. Con los mismos supuestos del apartado anterior excepto que los saltos y bifurcaciones son retardados un ciclo, planifica las instrucciones del bucle (sólo se permite cambiar instrucciones de sitio y modificar operandos en las instrucciones). Una vez optimizado el código dibuja un diagrama instrucciones-tiempo y calcula el número total de ciclos que supone la ejecución del bucle.

D. Con los supuestos del apartado C, muestra el bucle desenrollado dos veces sin planificar y planificado y calcula el número total de ciclos en cada caso.

2.10. Considera el siguiente fragmento de código:

```
loop:  L.D     F0, 0(R2)     ; i1
       L.D     F4, 0(R3)     ; i2
       MUL.D   F0, F0, F4    ; i3
       ADD.D   F2, F0, F2    ; i4
       DADDUI  R2, R2, #8    ; i5
       DADDUI  R3, R3, #8    ; i6
       DSUBU   R5, R4, R2    ; i7
       BNEZ    R5, loop      ; i8
```

Se supone que el valor inicial de R4 es R2+792 y que, en caso de que a varias instrucciones les toque llegar al mismo tiempo a WB, tiene prioridad la primera instrucción en el orden secuencial, mientras la otra es detenida en ID y se emite cuando el riesgo desaparece.

A. Muestra el diagrama instrucciones-tiempo correspondiente a la ejecución del código considerado en MIPS FP sin cortocircuito. Supón que las bifurcaciones se tratan deteniendo las instrucciones posteriores hasta conocer el destino. ¿Cuántos ciclos supone la ejecución completa del bucle?

B. Responde a la misma cuestión suponiendo que hay cortocircuito y se predice salto no efectivo.

C. Suponiendo cortocircuito, elabora el grafo de dependencias de datos que hay dentro de una iteración del bucle (en cada arco debe especificarse el tipo de dependencia y la separación mínima en ciclos).

D. Prepara el grafo anterior para planificar el bloque básico: a) suprime los arcos que unan instrucciones que el compilador pueda cambiar de orden retocando una de ellas (por ejemplo, el compilador puede cambiar de orden i1 e i5 retocando i1); b) aunque no haya dependencia de datos, todos los nodos deben preceder a i8; c) suprime todo arco que una dos nodos unidos por un camino de mayor o igual longitud que la del arco.

E. Planifica el grafo del apartado anterior mediante el algoritmo de Warren.

F. A partir del resultado anterior y suponiendo que las bifurcaciones son retardadas un ciclo, ¿qué instrucción colocarás en el hueco de retardo? Una vez ocupado el hueco de retardo, dibuja el diagrama instrucciones-tiempo del código planificado y calcula el número total de ciclos que supone la ejecución del bucle.

G. Con cortocircuito y bifurcaciones retardadas un ciclo, muestra el bucle desenrollado dos veces sin planificar y planificado y calcula el número total de ciclos en cada caso.

2.11. Escribe el código resultante del encauzamiento software del bucle

```
Loop: L.D      F0, 0 (R1)
      ADD.D    F4, F0, F2
      S.D      F4, 0 (R1)
      DADDUI   R1, R1, #-8
      BNE      R1, R2, Loop
```

sabiendo que las bifurcaciones tienen un hueco de retardo y que el patrón repetitivo está formado por S.D, ADD.D y L.D. En concreto, el patrón repetitivo está formado por la instrucción S.D que almacena el nuevo elemento V[i] del vector, la instrucción ADD.D que calcula el nuevo valor del elemento V[i-1] y la instrucción L.D que carga el antiguo valor de V[i-2] (i=R1/8). Incluye el código de inicio y el de finalización.

2.12. Tomemos el bucle del ejercicio anterior, supongamos que el retardo de carga es de un ciclo y que la latencia entre un ADD.D y un S.D con dependencia de datos es de 5 ciclos. El encauzamiento software del bucle tendrá ahora una parada de 2 ciclos. Muestra que, esa parada desaparece si, previamente, el bucle original se desenrolla una vez. Fórmese el patrón repetitivo con los dos S.D, los dos ADD.D y los dos L.D de tres iteraciones consecutivas. Compara el total de ciclos del código resultante con el del bucle desenrollado y planificado.

2.13. Para que el algoritmo de Warren sea aplicable a procesadores superescalares basta con tener en cuenta que el contador de tiempo no se incrementa mientras haya instrucciones planificables para el tiempo actual.A) Planifica el bloque básico de la transparencia 8 del fichero 3StaticSch.pdf para un procesador superescalar capaz de emitir dos instrucciones cualesquiera en cada ciclo. Observa que el peso del arco adicional (i6,i9) ahora no es 1.

B) ¿Podría ejecutarse el bloque básico en menos ciclos aumentando el número de instrucciones que puede emitir el procesador en cada ciclo? ¿Por qué?

2.14. Supongamos las latencias para MIPS que aparecen en la transparencia 15 del fichero *3StaticSch.pdf*.

A) Desenrolla el siguiente bucle 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:  L.D      F0, 0(R1)
       L.D      F4, 0(R2)
       MUL.D    F0, F0, F4
       ADD.D    F2, F0, F2
       DADDUI   R1, R1, #-8
       DADDUI   R2, R2, #-8
       BNEZ     R1, loop
```

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

B) Planifica el bucle obtenido en el apartado anterior para MIPS superescalar. Compara el número de ciclos empleado por iteración en ambos casos.

2.15. 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.

2.16. A) Tal como puede verse en la página "Planificación dinámica: *scoreboard*-3" [fichero *6ScorTomas.pdf*] en MIPS con *scoreboard* las unidades de multiplicación FP comparten un mismo grupo de buses. Esto supone que ambas unidades no pueden acceder a la vez al banco de registros. ¿Qué parada se producirá en la ejecución del siguiente fragmento de código

```
ADD.D    F0, F2, F4
MUL.D    F6, F0, F10
MUL.D    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 MUL.D gasta 4 ciclos en EX, se producirá una parada con el método de Tomasulo que no se producirá con el *scoreboard*. Explícala.

```
MUL.D    F0, F2, F4
NOP
NOP
L.D      F6, 24(R8)
```

2.17. ¿Cuál es el efecto en cada uno de los factores que sirven para calcular TCPU, es decir, IE, CPI y T, de las instrucciones especulativas que se ejecutan correctamente? ¿Y el de las que se ejecutan incorrectamente? En este último caso, ¿la variación de qué factor se corresponde mejor con la variación de TCPU?

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

```
foo:  L.D        F2, 0(R1)      ; carga X(i)
      MUL.D      F4, F2, F0     ; multiplica a·X(i)
      L.D        F6, 0(R2)      ; carga Y(i)
      ADD.D      F6,F4,F6       ; suma a·X(i)+Y(i)
      S.D        F6, 0(R2)      ; almacena Y(i)
      DADDUI     R1,R1,#8       ; actualiza índice de X
      DADDUI     R2,R2,#8       ; actualiza índice de Y
      DSGTUI     R3, R1, done   ; Si R1> done, R3 ← 1. Si no, R3 ← 0
      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 15 del fichero *3StaticSch.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 15 del fichero *3StaticSch.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) si varias instrucciones han terminado y están en la cabecera del ROB se pueden retirar a la vez; d) por lo demás, se dan por ciertos los supuestos del apartado B.

Muestra el estado del procesador al final del 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 E; c) se aumenta la anchura del CDB para permitir que se escriban en él hasta tres resultados a la vez; d) por lo demás, 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.