

4. Problemas

Problema 1. El fragmento de código que se proporciona a continuación:

1	lw	r1, 0x1ac												
2	lw	r2, 0xcclf												
3	add	r3, r0, r0												
4	mul	r4, r2, r1												
5	add	r3, r3, r4												
6	add	r5, r0, 0x1ac												
7	add	r6, r0, 0xcclf												
8	sub	r5, r5, #4												
9	sub	r6, r6, #4												
10	sw	(r5), r3												
11	sw	(r6), r4												

; r1 ← (0x1ac)
; r2 ← (0xcclf)
; r3 ← r0 + r0 (r3 ← 0)
; r4 ← r2 + r1
; r3 ← r3 + r4
; r5 ← r0 + 0x1ac
; r6 ← r0 + 0xcclf
; r5 ← r5 + 4
; r6 ← r6 + 4
; (r5) ← r3
; (r6) ← r4

se ejecuta en un procesador que puede captar y emitir cuatro instrucciones por ciclo. Indique el orden en que se emitirán las instrucciones para cada uno de los siguientes casos: (a) Una ventana de instrucciones centralizada con emisión ordenada; y (b) Una ventana de instrucciones centralizada con emisión desordenada.

NOTA: considere que hay una unidad funcional para la carga (que consume 2 ciclos), otra para el almacenamiento (1 ciclo), tres para la suma/resta(1 ciclo), y una para la multiplicación (4 ciclos).

Solución

(a) *Emisión ordenada.* En la Tabla 2 se muestran los ciclos en los que cada instrucción pasa por cada una de las etapas del cauce de la Figura 26.

Tabla 2. Ventana centralizada con emisión Ordenada

	Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	lw r1, 0x1ac	IF	ID	EX	EX										
2	lw r2, 0xcclf	IF	ID			EX	EX								
3	add r3, r0, r0	IF	ID			EX									
4	mul r4, r2, r1	IF	ID				EX								
5	add r3, r3, r4		IF	ID				EX	EX	EX	EX				
6	add r5, r0, 0x1ac		IF	ID								EX			
7	add r6, r0, 0xcclf		IF	ID								EX			
8	sub r5, r5, #4		IF	ID									EX		
9	sub r6, r6, #4			IF	ID								EX		
10	sw (r5), r3			IF	ID									EX	
11	sw (r6), r4				IF	ID									EX

En el ciclo (1) se captan las primeras cuatro instrucciones, 1-4, que pasan a la cola de instrucciones. Como no nos indican que dicha cola tenga un tamaño determinado

IRANADA

podemos suponer que tiene espacio suficiente para almacenar todas las instrucciones de la secuencia que se considera. Por tanto, en el ciclo (2) se captarán las cuatro instrucciones siguientes, 5-8, y en el ciclo (3) las últimas instrucciones de la secuencia, 9-11.

Igualmente, como no nos indican que haya limitación de tamaño en la ventana de instrucciones centralizada, también podemos suponer que tiene tamaño suficiente para contener las instrucciones de la secuencia considerada. Por ello, en el ciclo (2) se decodificarán las cuatro primeras instrucciones, en el ciclo (3) las cuatro siguientes, y en el ciclo (4) las tres últimas instrucciones.

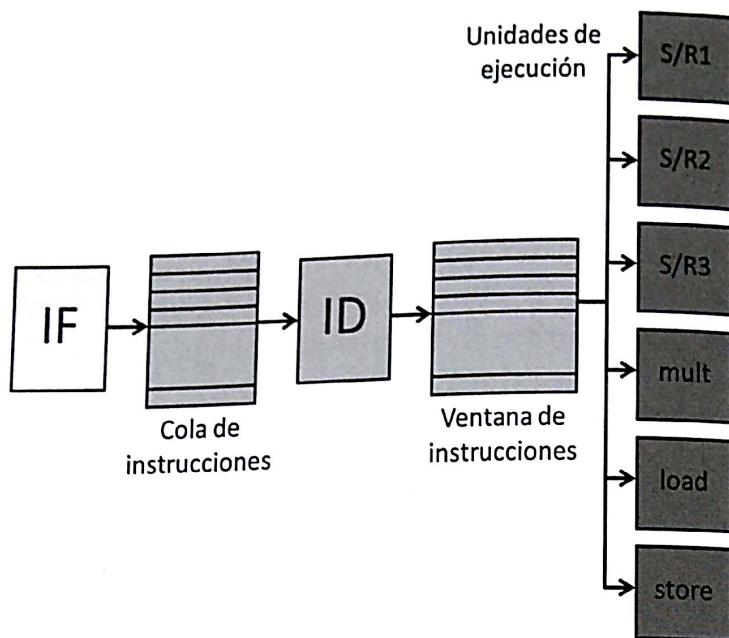


Figura 26. Esquema de etapas del procesador utilizadas en el problema

En cuanto a la ventana de instrucciones, el enunciado del problema no incluye ninguna indicación sobre si es alineada (hasta que no se emiten las instrucciones que se introducen tras su decodificación en un mismo ciclo, no se pueden emitir nuevas instrucciones) o no alineada (se puede emitir cualquier instrucción incluida en la ventana de instrucciones siempre que reúna las condiciones correspondientes de independencia, disponibilidad, etc.). Consideraremos que se trata de una ventana no alineada, que por otra parte es el caso más común.

Por tanto, en el ciclo (3) la instrucción 1 ya ha sido decodificada y puede emitirse desde la ventana de instrucciones. Aunque se pueden emitir cuatro instrucciones por ciclo, la instrucción 2 no puede emitirse hasta que no acabe de ejecutarse la instrucción 1, dado que solo hay una unidad de carga de datos de memoria: existe una dependencia estructural entre las instrucciones 1 y 2. Así, la instrucción 2 podría empezar a ejecutarse en el ciclo 5 y terminaría su ejecución en el ciclo 6. La instrucción 3 no tiene dependencias con la instrucción 2 y también podría emitirse junto con la instrucción 2. Hay que tener en cuenta que, puesto que la emisión es ordenada, aunque la instrucción 3 no tiene dependencias con la instrucción 2 y podría emitirse antes que ella, dado que la emisión es ordenada, debe esperar hasta que se emita la instrucción 2, que va delante en el código.

La instrucción 4 debe esperar a que termine la ejecución de la instrucción 2, puesto que uno de sus operandos es r2, que precisamente es el registro que carga la instrucción de acceso a memoria 2. Por lo tanto, se emitirá en el ciclo (7).

Como uno de los operandos de la instrucción 5 es r4, que es calculado precisamente por la instrucción 4, no puede emitirse hasta que no termine la ejecución de la instrucción 4 (que precisamente calcula ese valor). Por lo tanto, hasta el ciclo (11) no podría emitirse, al mismo tiempo que las instrucciones 6 y 7 que le siguen, que no dependen de la instrucción 5, y son independientes entre si. Al disponerse de tres unidades de suma/resta, se pueden emitir cada una de esas instrucciones a una unidad diferente y no habría problema de dependencia estructural. Las instrucciones 6 y 7, a pesar de no depender de la instrucción 4 no podrían emitirse antes que la instrucción 5 porque la emisión es ordenada.

Las instrucciones 8 y 9 dependen de las instrucciones 6 y 7, porque necesitan r5 y r6, respectivamente. Por lo tanto no pueden emitirse hasta que no termine la ejecución de las instrucciones 6 y 7, en el ciclo (12). Como hay tres unidades de suma/resta se pueden emitir las dos en el mismo ciclo.

Las instrucciones 10 y 11 también dependen de las instrucciones 8 y 9 y tendrían que esperar hasta el ciclo (13). No obstante, dado que solo hay una unidad de almacenamiento en memoria, primero se emite la instrucción 10 en el ciclo (13) y después la instrucción 11, en el ciclo (14).

(b) Emisión desordenada. En la Tabla 3 se muestran los ciclos en los que las instrucciones van ejecutando cada una de sus etapas.

Tabla 3. Ventana centralizada con emisión Desordenada

Instrucciones		1	2	3	4	5	6	7	8	9	10	11	12
1	lw r1, 0x1ac	IF	ID	EX	EX								
2	lw r2, 0xc1f	IF	ID			EX	EX						
3	add r3, r0, r0	IF	ID	EX									
4	mul r4, r2, r1	IF	ID										
5	add r3, r3, r4		IF	ID				EX	EX	EX	EX		
6	add r5, r0, 0x1ac		IF	ID	EX							EX	
7	add r6, r0, 0xc1f		IF	ID	EX								
8	sub r5, r5, #4		IF	ID		EX							
9	sub r6, r6, #4			IF	ID	EX							
10	sw (r5), r3			IF	ID								EX
11	sw (r6), r4			IF	ID							EX	

En cuanto a la captación y a la decodificación, la situación es la misma que en el caso de emisión desordenada. Del mismo modo, para emitirse la instrucción 2 debe esperar a la instrucción 1 como en el caso de la emisión ordenada dado que lo que determinaba esta espera era que solo había una unidad de carga de memoria. Sin embargo, la instrucción

3, que no depende ni de la instrucción 1 ni de la 2, puede emitirse al mismo tiempo que la instrucción 1.

La instrucción 4 debe esperar al ciclo (7), igual que en el caso de la emisión ordenada dado que necesita tener el valor de r2. Igualmente, la instrucción 5 tiene que esperar al ciclo (11), dado que necesita el valor de r4, calculado por la instrucción 4, como uno de sus operandos.

Sin embargo, las instrucciones 6 y 7 pueden emitirse en el ciclo (4), una vez han sido decodificadas en el ciclo (3). En efecto, no dependen de operandos que deben ser calculados en instrucciones previas, existen unidades de suma/resta suficientes, y se pueden emitir cuatro instrucciones por ciclo (en el ciclo (4) se emitirían tres instrucciones).

Las instrucciones 8 y 9 deben esperar a que acabe la ejecución de la 6 y la 7, respectivamente por lo que podrían emitirse en el ciclo (5). Igual que en el caso anterior, hay unidades funcionales suficientes y no se emitirían más de cuatro instrucciones por ciclo (se emitirían tres instrucciones, la 2, la 8, y la 9).

En cuanto a las instrucciones de almacenamiento, la instrucción 10 debe esperar al ciclo (12) para poderse emitir, dado que depende de r3, y ese valor solo estará disponible cuando acabe la ejecución de la instrucción 5. Sin embargo, la instrucción 11 podría emitirse en el ciclo (11) dado que el operando r4 del que depende está disponible al terminar la instrucción 4 en el ciclo anterior. En este caso, se ha evitado la colisión de las instrucciones para acceder a la única unidad de almacenamiento gracias a la emisión desordenada.

Como se puede observar, con la emisión desordenada la ejecución de las instrucciones se completa dos ciclos antes que con emisión ordenada.



Problema 2. Considerando el mismo fragmento de código utilizado en el Problema 1, y que el procesador dispone de las mismas unidades funcionales de suma/resta, multiplicación, carga, y almacenamiento de datos, con los mismos ciclos de retardo. Indique el tiempo que tardan en ejecutarse todas las instrucciones, si el procesador dispone de una estación de reserva, con capacidad para tres instrucciones, para cada una de sus unidades funcionales: (a) Suponiendo que el envío desde la estación de reserva es ordenado; y (b) Suponiendo que el envío desde la estación de reserva es desordenado.

Solución

(a) *Envío ordenado.* La Tabla 4 muestra la evolución temporal de las instrucciones, a través de las distintas etapas, hasta que termina su ejecución (Figura 27).

Dado que no nos indican que exista límite en el número de instrucciones que se pueden almacenar en la cola de instrucciones, suponemos que ésta tiene espacio para al menos las 11 instrucciones que se analizan y, por lo tanto, las instrucciones 1-4 se captan en el ciclo (1), las 5-8 en el ciclo (2), y las 9-11 en el ciclo (3).

Tabla 4. Estaciones de reserva con envío ordenado

	Instrucciones	Est.res.	1	2	3	4	5	6	7	8	9	10	11	12	13
1	lw r1,0x1ac	LD	IF	ID	EX	EX									
2	lw r2,0xc1f	LD	IF	ID				EX	EX						
3	add r3,r0,r0	ADD1	IF	ID	EX										
4	mul r4,r2,r1	MULT	IF	ID						EX	EX	EX	EX		
5	add r3,r3,r4	ADD1		IF	ID										
6	add r5,r0,0x1ac	ADD2		IF	ID	EX									
7	add r6,r0,0xc1f	ADD3		IF	ID	EX									EX
8	sub r5,r5,#4	ADD2		IF	ID		EX								
9	sub r6,r6,#4	ADD3			IF	ID	EX								
10	sw (r5),r3	ST				IF	ID								
11	sw (r6),r4	ST					IF	ID							EX

Al final del ciclo 2, las instrucciones 1-4 se habrán decodificado y emitido (aunque para en la gráfica se sigue manteniendo la notación ID para identificar ésta etapa, en realidad se realiza la decodificación y emisión a la estación de reserva correspondiente, ID/ISS).

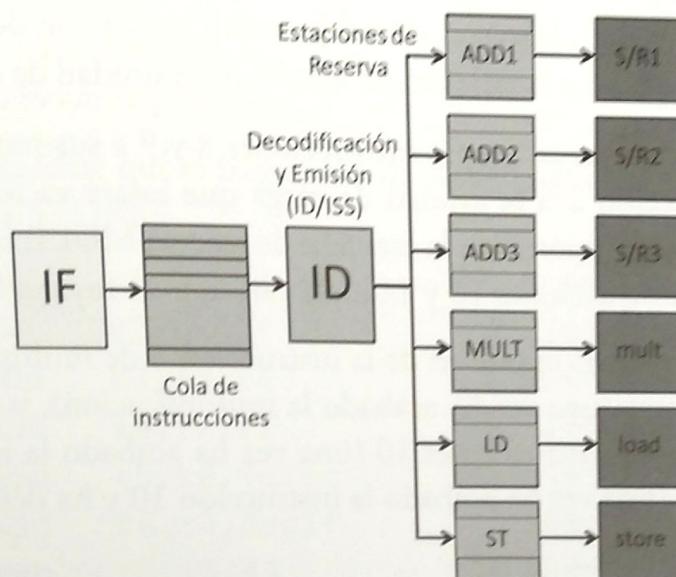


Figura 27. Esquema de etapas del procesador utilizadas en el problema

En la estación de reserva de la unidad de carga de datos (LD) estarán las instrucciones 1 y 2, en la estación de reserva de una de las unidades de suma/resta (ADD1) estará la instrucción 3, y en la estación de reserva de la unidad de multiplicación (MULT) estará la instrucción 4. Desde cada una de esas estaciones de reserva se realizará el envío de la instrucción a la unidad correspondiente.

En el ciclo (3) se enviará la instrucción 1 a la unidad de carga para que empiece a ejecutarse. La instrucción 2 se mantendrá en la estación de reserva hasta que la instrucción 1 termine y la unidad de carga quede libre. La instrucción 3 también puede emitirse a la unidad de suma 1 dado que no depende de las instrucciones anteriores. La instrucción 4, sin embargo, debe esperar en la estación de reserva de la multiplicación hasta que termine de ejecutarse la instrucción 2.

Al final del ciclo (3), las instrucciones 5-8 se habrán decodificado y han pasado a las estaciones de reserva correspondientes. La forma de emitir instrucciones a las tres estaciones de reserva, ADD1, ADD2, ADD3, de las distintas unidades de suma depende del algoritmo que implemente la etapa de decodificación/emisión (ID en la Tabla 4). Aquí se ha considerado que se van asignando instrucciones a las estaciones menos cargadas, y si hay varias estaciones con la misma carga se asigna a aquella que tenga instrucciones de las que dependa la instrucción a emitir (ordenadamente según el índice de la estación de reserva, si existieran varias en esa situación o si las estaciones de reserva están vacías). Según esto, cuando se van a emitir las cuatro instrucciones, las tres estaciones de reserva están vacías y la instrucción 5 se asigna a ADD1, la 6 a ADD2 y la 7 a ADD3. La instrucción 8 se asignaría a la estación de reserva ADD2 porque las tres instrucciones tienen una instrucción y precisamente la instrucción 8 depende de la asignada a ADD2 (tiene como operando el resultado que produce la instrucción 6).

En el ciclo (4) podrán enviarse las instrucciones 6 y 7 a los correspondientes sumadores/restadores. Al final del ciclo (4) se habrán decodificado las instrucciones 9-11. Dado que en las estaciones de reserva ADD1 y ADD2 quedan una instrucción en cada una (respectivamente la 5 y la 8), la instrucción 9 pasaría a la estación de reserva ADD3 y las instrucciones 10 y 11 pasan a la estación de reserva de la unidad de almacenamiento ST.

En el ciclo (5) se pueden enviar las instrucciones 8 y 9 a sus respectivos sumadores/restadores y la instrucción 2 a la unidad de carga que estará ya libre. Al final de este ciclo (5) tenemos la instrucción 4 en la estación de reserva MULT, la 5 en la estación de reserva ADD1, y las instrucciones 10 y 11 en la estación de reserva ST.

En el ciclo (7) empieza la ejecución de la instrucción 4 de multiplicación, en el (11) la instrucción 5 de suma (una vez ha acabado la multiplicación), y en los ciclos (12) y (13) respectivamente, las instrucciones 10 (una vez ha acabado la instrucción 5, de la que depende) y la 11 (una vez ha acabado la instrucción 10 y ha dejado libre la unidad de almacenamiento).

Por lo tanto, en este caso se necesitan 13 ciclos para ejecutar todas las instrucciones. Si lo comparamos con el problema anterior, donde se utiliza ventana centralizada, se tarda un ciclo más que en el caso de la emisión desordenada, y un ciclo menos que la emisión ordenada. En este caso, se puede ver que, aunque dentro de cada estación de reserva el envío es ordenado, las instrucciones sí pueden ejecutarse desordenadamente debido a que se pueden enviar desde varias estaciones de reserva que no están sincronizadas entre sí.

(b) Envío desordenado. La Tabla 5 muestra la situación correspondiente al envío desordenado desde cada estación de reserva. Solo en la estación de reserva ST se produce una situación en la que hay más de una instrucción esperando, pudiendo enviarse la instrucción más reciente y la más antigua no. Ése es el único cambio que se observa en la Tabla 5.

Tabla 5. Estaciones de reserva con envío desordenado

	Instrucciones	Est.res.	1	2	3	4	5	6	7	8	9	10	11	12
1	lw r1,0x1ac	LD	IF	ID	EX	EX								
2	lw r2,0xc1f	LD	IF	ID			EX	EX						
3	add r3,r0,r0	ADD1	IF	ID	EX									
4	mul r4,r2,r1	MULT	IF	ID						EX	EX	EX	EX	
5	add r3,r3,r4	ADD1		IF	ID									
6	add r5,r0,0x1ac	ADD2		IF	ID	EX								EX
7	add r6,r0,0xc1f	ADD3		IF	ID	EX								
8	sub r5,r5,#4	ADD2		IF	ID		EX							
9	sub r6,r6,#4	ADD3			IF	ID	EX							
10	sw (r5),r3	ST				IF	ID							
11	sw (r6),r4	ST				IF	ID							EX

Como se puede ver, se necesitan 12 ciclos para terminar la ejecución de todas las instrucciones, como en el caso de la emisión desordenada con ventana de instrucciones. De hecho, las Tablas 3 y 5 coinciden en cuanto a la distribución de las instrucciones en etapas y ciclos.

Problema 3. Considere que el fragmento de código siguiente, correspondiente a una arquitectura LOAD/STORE:

1	lw	r1,0x10a	; r1←M(0x10a)
2	lw	r2,0xc1f	; r2←M(0xc1f)
3	add	r3,r3,r4	; r3←r3+r4
4	mul	r4,r2,r1	; r4←r2*r1
5	add	r5,r0,0x1ac	; r5←r0+0x1ac
6	add	r6,r0,0xc1f	; r6←r0+0xc1f
7	sub	r5,r5,#4	; r5←r5+4
8	sub	r6,r6,#4	; r6←r6+4
9	sw	0(r5),r3	; m(r5)←r3
10	sw	0(r6),r4	; m(r5)←r4

se ejecuta en un procesador superescalar que es capaz de captar 4 instrucciones/ciclo; de decodificar 2 instrucciones/ciclo; de emitir (utilizando una ventana de instrucciones con emisión no alineada) 2 instrucciones/ciclo; de escribir hasta 2 resultados/ciclo en los bancos de registros correspondientes (registros de reorden, o registros de la arquitectura según el caso); y de completar (retirando instrucciones del ROB) hasta 2 instrucciones/ciclo.

Indique el número de ciclos que tardaría en ejecutarse el programa suponiendo: (a) Emisión ordenada; y (b) Emisión desordenada. (c) Si el procesador funciona a 1 GHz, indique cuál es su velocidad pico y determine el valor del número de ciclos por instrucción promedio (CPI) para cada tipo de emisión.

NOTA: Considere que tiene una unidad funcional de carga (con 2 ciclos de retardo); una de almacenamiento (con 1 ciclo de retardo); dos unidades de suma/resta (con 1 ciclo de retardo); y una de multiplicación (con 3 ciclos de retardo); y que no hay limitaciones para el número de líneas de la cola de instrucciones, ventana de instrucciones, buffer de reorden, puertos de lectura/escritura etc.

Solución

Para responder a cada uno de los apartados se construyen las Tablas 6 y 7 que describen las etapas por las que van pasando cada una de las instrucciones.

(a) En este caso la emisión es ordenada y, por lo tanto, el comienzo de la ejecución de una instrucción que es posterior a otra en el código siempre se producirá en un ciclo posterior (o como muy pronto en el mismo ciclo) al comienzo de la ejecución de la primera. El final de la ejecución, en cambio, podría no respetar ese orden.

Tabla 6. Ventana centralizada con emisión Ordenada

	Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	lw r1, 0x1ac	IF	ID	EX	EX	ROB	WB								
2	lw r2, 0xc1f	IF	ID			EX	EX	ROB	WB						
3	add r3, r3, r4	IF		ID		EX	ROB		WB						
4	mul r4, r2, r1	IF		ID				EX	EX	EX	ROB	WB			
5	add r5, r0, 0x1ac		IF		ID			EX	ROB			WB			
6	add r6, r0, 0xc1f		IF		ID				EX	ROB			WB		
7	sub r5, r5, #4		IF			ID			EX	ROB			WB		
8	sub r6, r6, #4		IF			ID				EX	ROB			WB	
9	sw 0(r5), r3			IF			ID			EX				WB	
10	sw 0(r6), r4			IF			ID				EX				WB

En la Tabla 6 se observa que las instrucciones se captan (IF) de cuatro en cuatro en ciclos consecutivos dado que nos dicen que no hay limitaciones en el tamaño de la cola de instrucciones. También se captan de dos en dos en ciclos consecutivos, al no haber limitaciones en la ventana de instrucciones.

A partir del ciclo (3) se puede emitir la instrucción 1, pero la instrucción 2 debe esperar que termine la ejecución de la 1 dado que las dos utilizan la unidad de carga de datos de memoria y sólo se dispone de una de esas unidades. La instrucción 3 no depende de las anteriores pero no se puede emitir antes que la instrucción 2. Las instrucciones 2 y 3 se pueden emitir en el mismo ciclo dado que utilizan unidades funcionales que están disponibles y se pueden emitir hasta dos instrucciones por ciclo. La instrucción 4 depende de la instrucción 2, y no puede emitirse hasta que termine la ejecución de ésta. En el ciclo (7) se pueden emitir las instrucciones 4 y 5 dado que no existe dependencia entre ellas, las unidades de cálculo que necesitan están disponibles y se pueden emitir hasta dos instrucciones por ciclo. Igualmente, las instrucciones 6 y 7 pueden emitirse en el ciclo (8). La instrucción 7 depende de la 5, pero en ese ciclo (8) ya ha terminado su ejecución.

En el ciclo (9) se pueden emitir las instrucciones 8 y 9. Las instrucciones 3 y 5 de las que depende la instrucción 9 ya han producido sus resultados y las unidades que necesitan están disponibles. Finalmente, la instrucción 10 se emite en el ciclo (10). Esta instrucción 10 nunca se podría emitir al mismo tiempo que la instrucción 9 dado que depende de la instrucción 8 y además, solo se dispone de una unidad de almacenamiento en memoria. No obstante, aquí tampoco se podría emitir al mismo tiempo que la 9 aunque hubiera varias unidades de almacenamiento y no tuviera dependencias, dado que en el ciclo (9) ya se emite el número máximo de dos instrucciones.

En la Tabla 6 también se pueden ver los ciclos en los que se producen las escrituras de los resultados en el ROB (dado que no se pueden escribir más de dos por ciclo). No obstante, en ningún caso se producen colisiones por esta causa. Las instrucciones de almacenamiento de registros en memoria, al no tener que escribir resultados en el banco de registros, no tienen que almacenarlos temporalmente en el ROB. Por eso estas instrucciones "no pasan" por la etapa ROB.

Las instrucciones se retiran de dos en dos, de forma ordenada. Como se puede ver, excepto en las instrucciones 1, 2, y 4, en todas las instrucciones deben esperarse algunos ciclos desde que se termina de escribir el resultado en el buffer de reorden (etapa ROB) y se retira la instrucción (etapa WB).

(b) La Tabla 7 describe la evolución de las instrucciones en los distintos ciclos y etapas para la emisión desordenada.

Tabla 7. Ventana centralizada con emisión Desordenada

Instrucciones		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1 lw r1,0x1ac		IF	ID	EX	EX	ROB	WB								
2 lw r2,0xc1f		IF	ID			EX	EX	ROB	WB						
3 add r3,r3,r4		IF		ID	EX	ROB			WB						
4 mul r4,r2,r1		IF		ID				EX	EX	EX	ROB	WB			
5 add r5,r0,0x1ac			IF		ID	EX	ROB					WB			
6 add r6,r0,0xc1f			IF		ID		EX	ROB				WB			
7 sub r5,r5,#4			IF			ID	EX		ROB			WB			
8 sub r6,r6,#4			IF			ID		EX	ROB			WB			
9 sw 0(r5),r3				IF			ID		EX			WB			
10 sw 0(r6),r4				IF			ID			EX					WB

A pesar de que se permite la emisión desordenada, en este caso se observa que no se ha ganado ningún ciclo con respecto al caso de la emisión ordenada. Ahora puede adelantarse la emisión de la instrucción 3 a la de la instrucción 2 y la emisión de las instrucciones 5 6 y 7 a la de la instrucción 4. No obstante, la colisión entre las instrucciones 1 y 2 por el acceso a la unidad de carga, y la dependencia de la instrucción 4 con respecto a la de la instrucción 2, hacen que esta instrucción 4 finalice (se retire del ROB) en el mismo ciclo (11) que en el caso de emisión ordenada. Como en el caso de la emisión ordenada se retiraban, al máximo ritmo posible (dos instrucciones por ciclo)

las instrucciones que quedaban después de la instrucción 4, en el caso de la emisión desordenada no se ganará nada. Lo que ocurre es que, entre que se termina la etapa ROB o la de ejecución en las instrucciones de almacenamiento, las instrucciones esperan más ciclos hasta que se pueden retirar.

En este caso, a pesar de que se pueden emitir dos instrucciones por ciclo, las instrucciones de almacenamiento, 9 y 10, no se pueden emitir en el mismo ciclo dado que hay colisión entre ellas al no existir más que una unidad de almacenamiento. Otro efecto a reseñar es que al terminar la ejecución de la instrucción 7 hay una colisión en la escritura en el ROB (solo se pueden escribir dos datos por ciclo en el ROB según indica el enunciado). Esto hace que la unidad de suma/resta donde se ejecuta la instrucción 7 se mantiene ocupada. Sin embargo, esto no causa más problemas dado que hay otra unidad de suma/resta libre donde se puede ejecutar la instrucción 8. El retraso en la emisión de la instrucción 8 se debe a que no se pueden emitir más de dos instrucciones por ciclo (las instrucciones 6 y 7).

(c) Teniendo en cuenta que el procesador puede finalizar dos instrucciones por ciclo y que utiliza un reloj de 1 GHz, la velocidad pico del procesador se puede calcular a partir de:

$$GIPS = \frac{2(inst/ciclo) \times 10^9 (ciclos/segundo)}{10^9} = 2$$

Es decir puede alcanzar hasta 2 GIPS. En cuanto al cálculo del CPI medio, como en los dos casos de emisión ordenada y desordenada considerada se ha tardado el mismo tiempo (14 ciclos), en los dos casos se tendrá el mismo valor. Para calcularlo se utiliza la expresión del tiempo de CPU que, dado que el tiempo de ciclo es de 1 nseg. (la inversa de 1 GHz, es decir 10^{-9} seg.), es igual a 14 nseg. Así:

$$T_{CPU} = NI \times CPI \times T_{ciclo} = 10 inst \times CPI \times 10^{-9} (seg/ciclo) = 14 \times 10^{-9} seg$$

y, despejando se tiene que $CPI=1.4$ ciclos/instrucción. Es decir, se tarda más de un ciclo por instrucción por lo que, a pesar de que en algunos ciclos se terminan más de una instrucción, por término medio, no se pone de manifiesto el carácter superescalar del procesador (CPI debería de ser menor que 1).

Problema 4. Para la secuencia de instrucciones del problema 3, incremente hasta un máximo de cuatro, el número de instrucciones que se pueden emitir y finalizar por ciclo, el número de escrituras por ciclo, y el número de unidades de carga de datos de memoria, para reducir el tiempo de procesamiento de la secuencia al mínimo posible (a) en el caso de la emisión ordenada, y (b) en el caso de la emisión desordenada. Determine el valor de la velocidad pico y del CPI promedio en estos casos considerando el mismo ciclo de reloj que en el problema 3.

Solución

Para resolver el problema se modifican las Tablas 6 y 7 considerando que no existen limitaciones (hasta un valor máximo de cuatro) en el número de instrucciones que se pueden decodificar, emitir o retirar. Tampoco existen limitaciones (hasta un valor máximo de cuatro) en el número de escrituras en el buffer de reorden y se tienen dos unidades de carga en lugar de una (en este problema podemos saber que no hace falta incrementar más el número de estas unidades ya que solo hay dos instrucciones de carga de memoria). Así, se obtendrán las Tablas 8 y 9 para la emisión ordenada y desordenada, respectivamente.

(a) En la Tabla 8 se observa que, dado que hay dos unidades de carga de memoria se pueden emitir las instrucciones 1 y 2 al mismo tiempo. Esto permite adelantar la emisión de la instrucción de multiplicación que, a su vez permite adelantar la emisión de las instrucciones que la siguen. La instrucción 10 no se puede emitir en el ciclo (7), junto con la instrucción 9 porque está esperando el dato que proporciona la instrucción 4, no porque haya una única unidad de almacenamiento en memoria.

Como se puede ver, se produce una reducción de cuatro ciclos respecto al caso de la emisión ordenada del problema 3. En realidad no habría que emitir más de tres instrucciones por ciclo, ni escribir en el ROB más de dos resultados por ciclo. No obstante, si en lugar de finalizar cuatro instrucciones por ciclo se pudiesen finalizar siete instrucciones por ciclo, se reduciría un ciclo más. No obstante, terminar un número de instrucciones tan elevado tendría un coste hardware considerable, y posiblemente se aprovecharía en pocas ocasiones.

Tabla 8 Ventana centralizada con emisión ordenada

Instrucciones		1	2	3	4	5	6	7	8	9	10
1	lw r1,0x1ac	IF	ID	EX	EX	ROB	WB				
2	lw r2,0xc1f	IF	ID	EX	EX	ROB	WB				
3	add r3,r3,r4	IF	ID	EX	ROB		WB				
4	mul r4,r2,r1	IF	ID			EX	EX	EX	ROB	WB	
5	add r5,r0,0x1ac	IF	ID			EX	ROB			WB	
6	add r6,r0,0xc1f	IF	ID			EX	ROB			WB	
7	sub r5,r5,#4	IF	ID				EX	ROB		WB	
8	sub r6,r6,#4	IF	ID				EX	ROB			WB
9	sw 0(r5),r3			IF	ID			EX			WB
10	sw 0(r6),r4			IF	ID				EX		WB

(b) En la Tabla 9 se puede observar el efecto de la emisión desordenada. Ahora se puede adelantar la emisión de las instrucciones 5, 6, 7, 8 y 9 con respecto a lo que ocurre en la Tabla 8. Aunque también se está suponiendo que se pueden escribir cuatro resultados por ciclo en el ROB (si se supusiera que se pueden escribir sólo dos resultados por ciclo en el ROB se retrasaría la escritura de las instrucciones 5 y 6 y también se retrasaría la emisión de las instrucciones 7 y 8, pero esto no afectaría al número de ciclos final).

Tabla 9. Ventana centralizada con emisión desordenada

Instrucciones		1	2	3	4	5	6	7	8	9	10
1	lw r1,0x1ac	IF	ID	EX	EX	ROB	WB				
2	lw r2,0xc1f	IF	ID	EX	EX	ROB	WB				
3	add r3,r3,r4	IF	ID	EX	ROB		WB				
4	mul r4,r2,r1	IF	ID			EX	EX	EX	ROB	WB	
5	add r5,r0,0x1ac		IF	ID	EX	ROB				WB	
6	add r6,r0,0xc1f		IF	ID	EX	ROB				WB	
7	sub r5,r5,#4		IF	ID		EX	ROB			WB	
8	sub r6,r6,#4		IF	ID		EX	ROB				WB
9	sw 0(r5),r3			IF	ID		EX				WB
10	sw 0(r6),r4			IF	ID			EX			WB

Como se puede ver, tampoco aquí se observan tiempos diferentes entre la emisión ordenada y la desordenada. Esto es debido a que, por la dependencia entre las instrucciones 2, 4 y 10, el tiempo mínimo que tardan en completarse estas tres instrucciones es 9 ciclos, que es precisamente el tiempo que se obtiene ya en el caso de la emisión ordenada (salvo el ciclo que debe esperarse porque solo pueden retirarse cuatro instrucciones, problema que también se presenta en la emisión desordenada).

Tabla 10. Ventana centralizada con emisión ordenada y multiplicación de 1 ciclo

Instrucciones		1	2	3	4	5	6	7	8	9
1	lw r1,0x1ac	IF	ID	EX	EX	ROB	WB			
2	lw r2,0xc1f	IF	ID	EX	EX	ROB	WB			
3	add r3,r3,r4	IF	ID	EX	ROB		WB			
4	mul r4,r2,r1	IF	ID			EX	ROB	WB		
5	add r5,r0,0x1ac		IF	ID		EX	ROB	WB		
6	add r6,r0,0xc1f		IF	ID		EX	ROB	WB		
7	sub r5,r5,#4		IF	ID			EX	ROB	WB	
8	sub r6,r6,#4		IF	ID			EX	ROB	WB	
9	lw 0(r5),r3			IF	ID			EX	WB	
10	lw 0(r6),r4			IF	ID				EX	WB

Tabla 11. Ventana centralizada con emisión desordenada y multiplicación de 1 ciclo

Instrucciones		1	2	3	4	5	6	7	8
1	lw r1,0x1ac	IF	ID	EX	EX	ROB	WB		
2	lw r2,0xc1f	IF	ID	EX	EX	ROB	WB		
3	add r3,r3,r4	IF	ID	EX	ROB		WB		
4	mul r4,r2,r1	IF	ID			EX	ROB	WB	
5	add r5,r0,0x1ac		IF	ID	EX	ROB		WB	
6	add r6,r0,0xc1f		IF	ID	EX	ROB		WB	
7	sub r5,r5,#4		IF	ID		EX	ROB	WB	
8	sub r6,r6,#4		IF	ID		EX	ROB		WB
9	sw 0(r5),r3			IF	ID		EX		WB
10	sw 0(r6),r4			IF	ID			EX	WB

Para conseguir diferencias entre la emisión ordenada y desordenada tendríamos que reducir el tiempo de la multiplicación a un ciclo. Tal y como se observa en las Tablas 10 y 11.

Como se puede observar, se gana un ciclo. Si hubiera dos unidades de almacenamiento en memoria otra vez se igualarían los tiempos, ahora por la limitación a cuatro instrucciones que pueden retirarse como máximo.

En cuanto a la última pregunta del problema, dado que el procesador puede finalizar cuatro instrucciones por ciclo y que utiliza un reloj de 1 GHz, la velocidad pico del procesador se puede calcular a partir de:

$$GIPS = \frac{4(\text{inst/ciclo}) \times 10^9 (\text{ciclos/segundo})}{10^9} = 4$$

Así, ahora se podrían alcanzar hasta 4 GIPS dado que se ha doblado el número de instrucciones que pueden finalizarse por ciclo respecto al problema 3.

Respecto al cálculo del *CPI* medio, como en los dos casos de emisión ordenada y desordenada considerada se ha tardado de nuevo el mismo tiempo (10 ciclos), en los dos casos se tendrá el mismo valor. Así, utilizando la expresión del tiempo de CPU:

$$T_{CPU} = NI \times CPI \times T_{ciclo} = 10 \text{ inst} \times CPI \times 10^{-9} (\text{seg/ciclo}) = 10 \times 10^{-9} \text{ seg}$$

y, despejando, se tiene que *CPI*=1 ciclos/instrucción. Es decir, se tarda un ciclo por instrucción que correspondería al caso de un procesador segmentado a pleno rendimiento. Como sabemos que durante los primeros cinco ciclos no se termina ninguna instrucción (es lo que se denomina tiempo de latencia de inicio en un procesador segmentado), el procesador sí pone de manifiesto su carácter superescalar dado que debe ser capaz de terminar más de una instrucción por ciclo una vez termina la primera instrucción transcurrido el tiempo de latencia de inicio (a partir del ciclo 6).



Problema 5. Suponga que las cuatro instrucciones siguientes

multd	f4, f1, f2	; f4=f1*f2 (ciclo 2)
addd	f2, f4, f1	; f2=f4+f1 (ciclo 2)
subd	f4, f4, f1	; f4=f4-f1 (ciclo 3)
addd	f5, f2, f3	; f5=f2+f3 (ciclo 3)

se han decodificado en los ciclos indicados entre paréntesis, introduciéndose en una estación de reserva común para todas las unidades funcionales de coma flotante. Teniendo en cuenta que el procesador superescalar dispone de un ROB para implementar la finalización ordenada, y que la emisión es desordenada. Indique (a) cómo evolucionaría el ROB para esas instrucciones; (b) en qué momento empieza y termina la ejecución de

las instrucciones; y (c) cuáles son los valores que quedan en los registros de la arquitectura al terminar, si inicialmente $f_1=3.0$, $f_2=2.0$, y $f_3=1.0$.

NOTA: la multiplicación tarda 4 ciclos, y la suma y la resta 2 ciclos; hay tantas unidades funcionales como sea necesario para que no haya riesgos estructurales; y se pueden enviar, retirar, etc. dos instrucciones por ciclo como máximo)

Solución

Para resolver el problema se parte de la Tabla 12, donde se indican los ciclos en los que (1) las instrucciones se terminan de decodificar (ID) y han pasado a la estación de reserva, (2) comienza y termina la ejecución de la operación correspondiente a la instrucción (EX), (3) el resultado de operación se almacena en el ROB (ROB), y (4) el momento en que después de retirar la instrucción del ROB, los resultados se han almacenado en el banco de registros de la arquitectura (WB).

Las instrucciones 2 y 3 deben esperar que termine la ejecución de la instrucción 1 dado que dependen de f_4 . La instrucción 4 debe esperar que termine la ejecución de la instrucción 2.

Tabla 12. Ciclos y etapas del procesamiento de la secuencia de instrucciones del problema

Instrucciones		2	3	4	5	6	7	8	9	10	11	12
1	multd f_4, f_1, f_2	ID	EX	EX	EX	EX	ROB	WB				
2	addd f_2, f_4, f_1	ID					EX	EX	ROB	WB		
3	subd f_4, f_4, f_1		ID				EX	EX	ROB	WB		
4	addd f_5, f_2, f_3		ID						EX	EX	ROB	WB

(a) El ROB empieza a llenarse al final del ciclo (2), después de haberse decodificado las dos primeras instrucciones.

Final de (2)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
0	1	f_4	-	0	-	0
1	2	f_2	-	0	-	0

Al finalizar el tercer ciclo se introducen en el ROB las dos instrucciones restantes, y también habrá empezado la multiplicación.

Final de (3)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
0	1	f_4	-	0	mult	0
1	2	f_2	-	0	-	0
2	3	f_4	-	0	-	0
3	4	f_5	-	0	-	0

Hasta el ciclo (7) no ocurre nada en el ROB (en relación con las instrucciones que indica el problema). Como se muestra a continuación, se habrá terminado la multiplicación en el ciclo (6), y el resultado se almacena durante el ciclo (7) en el campo

de valor del registro 0 del ROB, el bit de OK pasará a 1. También se ha iniciado durante el ciclo (7) la ejecución de las instrucciones 2, y 3. Por tanto, al final del ciclo (7) el estado del ROB será el siguiente:

Final de (7)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
0	1	f4	6.0	1	Mult	0
1	2	f2	-	0	Add	0
2	3	f4	-	0	Sub	0
3	4	f5	-	0	-	0

En el ciclo (8) se siguen ejecutando las instrucciones 2 y 3 (terminan al final de dicho ciclo) y se retira la primer instrucción del ROB. Por lo tanto, el estado del ROB es:

Final de (8)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
1	2	f2	-	0	Add	0
2	3	f4	-	0	Sub	0
3	4	f5	-	0	-	0

Al final del ciclo (9), los resultados de las instrucciones 2 y 3 estarán almacenado en los campos de *Valor* de las líneas 2 y 3 del ROB, cuyos campos de *OK* estarán al valor 1 (dato válido), y habrá empezado la ejecución de la instrucción 4:

Final de (9)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
1	2	f2	9.0	1	Add	0
2	3	f4	3.0	1	Sub	0
3	4	f5	-	0	Add	0

Al final del ciclo (10) se habrán retirado las instrucciones 2 y 3 y se escribirán sus resultados en los registros f2 y f4. La instrucción 4 sigue ejecutándose hasta el final del ciclo (10):

Final de 10

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
3	4	f5	-	0	Add	0

Al finalizar el ciclo (11), el resultado de la instrucción 4 se habrá almacenado en el registro 3 del ROB.

Final de (11)

#	Instrucción	Reg.Dest.	Valor	OK	Unidad	Flush
3	4	f5	10.0	1	Add	0

En el ciclo (12), se retirará la instrucción y se escribirá el resultado en f5.

(b) Teniendo en cuenta la Tabla 12 se observa que la ejecución de las instrucciones

empieza en el ciclo 3 y acaba en el ciclo 10. La Tabla 12 nos permite ver también

en que momento empieza y termina la ejecución de cada una de las instrucciones. El procesamiento de las instrucciones termina en el ciclo (12), una vez se ha retirado del ROB la instrucción 4 y se escriben los resultados en f5.

(c) A partir de la evolución del ROB (el orden en que se han retirado las instrucciones y se almacenan los resultados en el banco de registros) al retirar cada instrucción se tiene:

Instrucción 1: $f1=3.0; f2=2.0; f3=1.0; f4=3.0 \times 2.0 = 6.0$

Instrucción 2: $f1=3.0; f2=6.0 + 3.0 = 9.0; f3=1.0; f4=6.0$

Instrucción 3: $f1=3.0; f2=9.0; f3=1.0; f4=6.0 - 3.0 = 3.0$

Instrucción 4: $f1=3.0; f2=9.0; f3=1.0; f4=3.0; f5=9.0 + 1.0 = 10.0$

Por lo tanto, al final los registros quedan con los valores: $f1=3.0; f2=9.0; f3=1.0; f4=3.0; f5=10.0$



Problema 6. Un procesador superescalar de 64 bits es capaz de captar (IF), decodificar (ID), emitir (ISS), y finalizar (WB), tres instrucciones por ciclo. Dispone de una ventana de instrucciones centralizada y un buffer de reordenamiento (ROB) para realizar el renombramiento de registros y la finalización ordenada de instrucciones. Además, el procesador permite los adelantamientos de *stores* por *loads* no especulativos, pero no permite los adelantamientos especulativos.

Indique, para la secuencia de instrucciones siguiente:

(1)	lf	f1,0(r1)	; f1=[r1+0]
(2)	lf	f2,8(r1)	; f2=[r1+8]
(3)	addf	f3,f2,f1	; f3=f2+f1
(4)	sf	16(r1),f3	; [r1+16]=f3
(5)	lf	f4,24(r1)	; f4=[r1+24]
(6)	mulf	f5,f4,f1	; f5=f4*f1
(7)	sf	8(r1),f5	; [r1+8]=f5
(8)	addi	r2,r1,#32	; r2=r1+32
(9)	lf	f6,0(r2)	; f6=[r2+0]
(10)	addf	f7,f6,f3	; f7=f6+f3
(11)	sf	8(r2),f7	; [r2+8]=f7

(a) ¿En cuántos ciclos se completa el procesamiento de las instrucciones suponiendo emisión ordenada?

(b) ¿En cuántos ciclos se completa el procesamiento si la emisión es desordenada?

(c) ¿Y si es desordenada y se permite adelantamiento especulativo de stores por loads?

NOTA: El procesador dispone de una unidad de carga con un retardo de 2 ciclos, una de almacenamiento con retardo de 1 ciclo, dos ALUs con 1 ciclo de retardo, un multiplicador de coma flotante con 3 ciclos de retardo y dos sumadores de coma flotante

con 2 ciclos de retardo. Hay espacio suficiente para las instrucciones de la secuencia indicada en la cola de instrucciones, la ventana de instrucciones, y el ROB.

Solución

(a) La Tabla 13 muestra el procesamiento de la secuencia de instrucciones, a través de las distintas etapas del procesador, en el caso de emisión ordenada y adelantamiento no especulativo.

Como se puede ver en la Tabla 13, entre las instrucciones 1 y 2 hay una dependencia estructural dado que ambas necesitan la unidad de carga de memoria y solo hay una. Por eso la instrucción 2 debe esperar al ciclo (5), una vez que la instrucción 1 ha terminado su ejecución en el ciclo (4). La instrucción 3 depende de las instrucciones 1 y 2, y tiene que esperar a que termine la ejecución de la instrucción 2 para empezar la ejecución. La instrucción 4 también depende de la instrucción 3.

Tabla 13. Procesamiento de las instrucciones de la secuencia con emisión ordenada y adelantamiento no especulativo

	Instrucciones	1	2	3	4	5	6	7	8	9	10
1	lf f1,0(r1)	IF	ID	EX	EX	ROB	WB				
2	lf f2,8(r1)	IF	ID			EX	EX	ROB	WB		
3	addf f3,f2,f1	IF	ID					EX	EX	ROB	
4	sf 16(r1),f3		IF	ID						EX	WB
5	lf f4,24(r1)		IF	ID						EX	EX
6	mulf f5,f4,f1		IF	ID							
7	sf 8(r1),f5			IF	ID						
8	addi r2,r1,#32			IF	ID						
9	lf f6,0(r2)			IF	ID						
10	addf f7,f6,f3				IF	ID					
11	sf 8(r2),f7				IF	ID					

Tabla 13. Procesamiento de las instrucciones de la secuencia con emisión ordenada y adelantamiento no especulativo (**continuación**)

	Instrucciones	11	12	13	14	15	16	17	18	19	20
1	lf f1,0(r1)										
2	lf f2,8(r1)										
3	addf f3,f2,f1										
4	sf 16(r1),f3										
5	lf f4,24(r1)	ROB	WB								
6	mulf f5,f4,f1	EX	EX	EX	ROB	WB					
7	sf 8(r1),f5				EX	WB					
8	addi r2,r1,#32				EX	ROB	WB				
9	lf f6,0(r2)					EX	EX	ROB	WB		
10	addf f7,f6,f3						EX	EX	ROB	WB	
11	sf 8(r2),f7							EX	WB		

La instrucción 5 puede emitirse al mismo tiempo que la instrucción 4. La instrucción 4 escribe el resultado de la instrucción 3 en memoria y la instrucción 5 lee desde memoria, pero las direcciones de memoria a la que acceden son diferentes (es decir

$r1+16$ y $r1+24$) y no hay problema de dependencia de datos RAW. La instrucción 6 depende de la 5 y la 7 de la 6, por lo que deben empezar su ejecución una después de que acabe la otra. La instrucción 8 no depende de las anteriores pero como la emisión es ordenada se emitirá al mismo tiempo que la instrucción 7.

Finalmente, la instrucción 9 depende de la 8, la instrucción 10 de la 9, y la 11 de la 10. Por lo tanto, cada instrucción solo puede emitirse y empezar a ejecutarse cuando acaba la instrucción anterior.

Como muestra la Tabla 13 la secuencia de instrucciones tarda 20 ciclos en ejecutarse. Si tenemos en cuenta que un procesador superescalar es un procesador segmentado, podemos estimar el número de instrucciones que terminan de ejecutarse por ciclo a partir de la expresión:

$$T_{CPU}(\text{ciclos}) = T_{latencia}(\text{ciclos}) + (N - 1) \times CPI$$

y sustituyendo, se tiene:

$$20 = T_{latencia}(\text{ciclos}) + (N - 1) \times CPI = 6 + 10 \times CPI$$

y al despejar:

$$CPI = \frac{20 - 6}{10} = 1.4$$

Por lo tanto, dado que se necesita, por término medio, más de un ciclo por instrucción, el procesador superescalar funciona muy por debajo de su velocidad pico, que le permitiría finalizar tres instrucciones por ciclo, y por lo tanto tener un $CPI=0.33$. Funciona, incluso con un rendimiento peor que un procesador segmentado no superescalar.

(b) La Tabla 14 muestra el procesamiento de las instrucciones con emisión desordenada. Las instrucciones 1 a 4 se procesan de igual forma que en la Tabla 13 debido a las dependencias RAW y estructurales que existen entre ellas.

Tabla 14. Procesamiento de las instrucciones de la secuencia con emisión desordenada y adelantamiento no especulativo

Instrucciones		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	lf f1,0(r1)	IF ID		EX	EX	ROB	WB								
2	lf f2,8(r1)	IF ID				EX	EX	ROB	WB						
3	lddf f3,f2,f1	IF ID						EX	EX	ROB					
4	lf 16(r1),f3	IF ID								EX	WB				
5	lf f4,24(r1)	IF ID						EX	EX	ROB	WB				
6	mulf f5,f4,f1	IF ID								EX	EX	EX	ROB	WB	
7	sf 8(r1),f5			IF ID									EX	WB	
8	addi r2,r1,#32			IF ID	EX	ROB								WB	
9	af f6,0(r2)			IF ID					EX	EX	ROB				WB
10	addf f7,f6,f3			IF ID							EX	EX	ROB	WB	
11	sf 8(r2),f7			IF ID									EX	WB	

La instrucción 5 podría adelantar a la instrucción 4. Se trata de una instrucción de almacenamiento, pero las direcciones a las que acceden son diferentes ($r1+16$ y $r1+24$) y no habrá dependencia RAW. Se podría adelantar más ciclos, pero la unidad de carga de memoria se encuentra ocupada ejecutando la instrucción 1 y la instrucción 2.

Adelantar la carga de memoria correspondiente a la instrucción 5 permite adelantar las instrucciones 6 y 7, entre las que hay dependencias (la 6 depende de la 5 y la 7 de la 6).

El comienzo de la ejecución de la instrucción 8 puede adelantarse al ciclo 5 dado que tiene sus datos disponibles, la unidad funcional que necesita está libre (una de las ALUs) y tenemos emisión desordenada. Esto permite adelantar la instrucción 9 de carga de memoria que podría ejecutarse antes que la instrucción 7, de almacenamiento, que la precede. No se trataría de un adelantamiento especulativo dado que la instrucción 7 tendría calculada la dirección de acceso a partir del ciclo (4), y la instrucción 9 a partir del ciclo (6), una vez termina la ejecución de la instrucción 8. Por tanto se conocerían las direcciones y serían direcciones diferentes (una es $r1+8$ y otra es $r2=r1+32$).

Las instrucciones 9, 10 y 11 también tienen que emitirse una tras otra debido a las dependencias entre ellas. La finalización (WB) de la instrucción 9 debe retrasarse un ciclo dado que no se pueden retirar más de tres instrucciones por ciclo.

En cuanto al valor de *CPI* promedio que se consigue, se tiene que:

$$14 = T_{latencia}(\text{ciclos}) + (N - 1) \times CPI = 6 + 10 \times CPI$$

✓ despejando:

$$CPI = \frac{14 - 6}{10} = 0.8$$

Aquí sí se pone de manifiesto un comportamiento superescalar, ya que por término medio se tiene menos de un ciclo por instrucción. Es decir, se termina más de una instrucción por ciclo: $1 / 0.8 = 1.25$. Aunque sigue siendo menor que el máximo de tres instrucciones finalizadas por ciclo.

(c) No hay cambio respecto al caso en el que solo se permite adelantamiento no especulativo de *stores* por *loads* que se ha analizado en el caso (b) anterior. Esto se debe a que, en el análisis que se ha realizado no se tiene ningún adelantamiento no especulativo.

Problema 7. Un procesador superescalares capaz de captar, decodificar y retirar hasta dos instrucciones por ciclo, con una única ventana de instrucciones con emisión desordenada y un buffer de reordenamiento (ROB) para el renombrado y la finalización ordenada.

El procesador utiliza un predictor de saltos dinámico de dos bits que se consulta al captar las instrucciones de salto condicional. Si la dirección de memoria de la instrucción

de salto se encuentra en un BTB (Branch Target Buffer) que tiene el procesador, y la predicción es saltar, se cambia el valor del PC por la dirección de destino del salto (que también estará almacenada en el BTB) para que se emplee a captar instrucciones desde ahí en el siguiente ciclo.

Cuando la instrucción de salto entra en el buffer de reordenamiento (ROB) tras la decodificación, se marca con un bit `pred=1` si el predictor decidió tomar el salto o `pred=0` en caso contrario. Posteriormente, cuando se resuelve la condición del salto en la etapa de ejecución, se comprueba si la predicción se realizó con éxito, y en caso contrario, se marcan con `flush=1` todas aquellas instrucciones que se han introducido en el cauce de forma especulativa para que no actualicen los registros al ser retiradas, y se fija PC para que apunte a la instrucción siguiente a la de salto (la dirección de la instrucción de salto está guardada en el BTB) y se capten instrucciones a partir de ahí en el siguiente ciclo.

Como la primera vez que se capta una instrucción de salto condicional no se encontrará información de ella en el BTB, no se podrá predecir su comportamiento hasta la etapa de decodificación, en la que se emplea un predictor estático que predice "saltar" en caso de instrucciones de salto condicional a direcciones más bajas que la dirección de la instrucción (saltos hacia atrás) y "no saltar" para las instrucciones de salto condicional a direcciones mayores que la dirección de la instrucción (saltos hacia delante), y crea una entrada en el BTB en la que se guarda la dirección de la instrucción de salto, la de destino del salto y los dos bits de historia (11 si finalmente se produce el salto ó 00 si no se produce).

Para el código siguiente:

1	lw	r1, it	; r1=(it)
3	lf	f1, a	; f1=(a)
4	lf	f2, b	; f2=(b)
5	bucle: addf	f2, f2, f1	; f2=f2+f1
6	subi	r1, r1, #1	; r1=r1-1
7	bnez	r1, bucle	; si r1!=0 no salta
8	sw	b, f2	; (b)=f2
9	trap	#0	; fin del programa

- (a) Realice una traza de la ejecución del programa en el procesador suponiendo que el contenido de `it` es igual a 2. ¿Cuántos ciclos tarda en ejecutarse el programa?
- (b) Estime la penalización (en ciclos) que se produce en el procesamiento del salto tanto si acierta como si falla el predictor del procesador.
- (c) ¿Cuántos ciclos de penalización en total se sufrirían si el contenido de `it` fuera igual a 50?

NOTA: Suponga que hay tantas unidades de ejecución y de acceso a memoria como sea necesario para que no haya colisiones por riesgos estructurales, y una latencia de 2 ciclos

para las instrucciones de carga de memoria y las operaciones de suma en coma flotante, y de 1 ciclo para las instrucciones aritméticas con enteros y los almacenamientos. Las instrucciones que van a continuación de una instrucción de salto y se captan en el mismo ciclo que dicha instrucción no continúan su procesamiento si se predice "saltar".

Solución

(a) La traza de ejecución del programa para las dos iteraciones correspondientes al valor almacenado inicialmente en la dirección it se muestra en la Tabla 15.

Dado que se nos indica que hay unidades de acceso a memoria y de procesamiento suficientes para no ocasionar colisiones por riesgos estructurales, las instrucciones 1 y 2 que se captan en el ciclo (1) se pueden ejecutar simultáneamente y la instrucción 3 que se capta en el ciclo (2) puede empezar a ejecutarse en el ciclo (4) en otra unidad de carga de datos diferente. Se utilizan, por tanto, tres unidades de carga de datos de memoria.

Tabla 15. Traza de procesamiento del código del problema para dos iteraciones, $it=2$

Instrucciones			1	2	3	4	5	6	7	8	9	10	11	12	13
1	lw	r1, it	IF	ID	EX	EX	ROB	WB							
2	lf	f1, a	IF	ID	EX	EX	ROB	WB							
3	lf	f2, b		IF	ID	EX	EX	ROB	WB						
4	addf	f2, f2, f1		IF	ID			EX	EX	ROB	WB				
5	subi	r1, r1, #1			IF	ID	EX	ROB			WB				
6	bnez	r1, bucle			IF	ID		EX	ROB			WB			
7	sf	b, f2				IF	---								
8	trap	#0				IF	---								
9	addf	f2, f2, f1					IF	ID		EX	EX	ROB	WB		
10	subi	r1, r1, #1					IF	ID	EX	ROB			WB		
11	bnez	r1, bucle						IF	ID	EX					
12	sf	b, r2						IF	---					WB	
13	addf	f2, f2, f1							IF	ID	flush	flush	flush	WB*	
14	subi	r1, r1, #1							IF	ID	flush	flush	flush	WB*	
15	bnez	r1, bucle								IF	---				
16	sf	b, r2								IF	---				
17	sf	b, f2									IF	ID	EX	WB	
18	trap	#0									IF	ID	EX	ROB	WB
Penalizaciones					P			P	P						

La instrucción 4 se capta en el ciclo (2) junto con la instrucción 3 pero debe esperar a que termine de ejecutarse la instrucción 3 para empezar su ejecución, dado que hay dependencia de datos entre las instrucciones 3 y 4 debido al uso del registro $f2$.

En cualquier caso, la instrucción 5 se puede empezar a ejecutar antes que la instrucción 4 dado que la emisión es desordenada. La instrucción de salto condicional (instrucción 6) se capta junto con la instrucción 5 y debería esperar un ciclo, hasta el ciclo (6), para empezar a ejecutarse, dado que depende de $r1$.

De todas formas, la predicción correspondiente a la instrucción 6 se realizaría cuando esté en la etapa ID dado que es la primera vez que se capta y no hay información de ella en el BTB. Como se trata de una instrucción de salto hacia atrás la predicción es "saltar", que coincidiría con lo que se determinaría en la etapa de ejecución EX, en el ciclo (6), puesto que r_1 es distinto de cero (r_1 era igual a 2 inicialmente, pero la instrucción 5 ha hecho $r_1=1$). Dado que la predicción es correcta, las instrucciones 9 y 10 que se captaron en el ciclo (5), tras la predicción realizada en el ciclo (4) por la instrucción 6, seguirán su procesamiento. Las instrucciones 7 y 8 que se captaron en el mismo ciclo (4), cuando se estaba haciendo la predicción, simplemente se desechan y no pasarían a decodificarse, ni al ROB. Se considera que la instrucción de salto no tiene etapa de escritura en el buffer de reordenamiento (ROB).

El comienzo de la ejecución de la instrucción 9 debe esperar un ciclo dado que tiene una dependencia RAW con la instrucción 4 por el uso de f_2 . No obstante, como la emisión es desordenada, la instrucción 10 puede emitirse para empezar su ejecución antes que la instrucción 9, en el ciclo (7).

La instrucción de salto se vuelve a captar en el ciclo (6) junto con la instrucción que le sigue de escritura en memoria (instrucciones 11 y 12). Ahora se tiene información en el BTB para la instrucción de salto, y por lo tanto ya se puede hacerla predicción en su etapa de captación. Dado que en la ejecución anterior de esta instrucción de salto condicional se produjo el salto, los dos bits de historia se cargaron con 11, y la predicción es "saltar". Por lo tanto, en el ciclo (7) se captarían las instrucciones 13 y 14. La instrucción 12, que se captó con la propia instrucción de salto 11, se desecha y no pasa ni a la etapa de decodificación ni se introduce en el ROB.

Sin embargo, cuando la instrucción de salto 11 llega a su etapa de ejecución, en el ciclo (8), ya la instrucción 10 se ha ejecutado en el ciclo (7) y ha hecho $r_1=0$. Por lo tanto no se verifica la condición de salto y la predicción ha sido errónea. A partir de la información almacenada en el BTB se determinará la dirección a partir de la que tiene que proseguir la ejecución: la siguiente a la dirección de la instrucción de salto (la instrucción de almacenamiento en memoria). Esta instrucción se captaría en el ciclo (9).

Las instrucciones que se han captado en los ciclos (7) y (8) se han introducido erróneamente en el cauce. Las instrucciones 13 y 14 se habrán introducido en el ROB y se marcan como flush para que no tengan efecto al retirarse del ROB. Las instrucciones 15 y 16, que solo se habían captado, se pueden desechar.

Se ha considerado que las instrucciones que están marcadas como flush realmente no cuentan como instrucciones retiradas dado que no producen ningún efecto (son las etapas marcadas con WB*). Simplemente se produce un salto en el puntero correspondiente del ROB. Así, en el ciclo (12) realmente se retiran dos instrucciones, la 11 y la 17.

(b) Las penalizaciones para las predicciones tienen que tener en cuenta que el tipo de predicción depende de que sea la primera vez que se capta la instrucción (predicción estática) o de que estén definidos los bits de historia en las siguientes iteraciones.

Para la predicción estática:

- $P_{est_OK} = 1$ ciclos, dado que la predicción se hace en la etapa ID
- $P_{est_fallo} = 3$ ciclos, dado que hasta el final de la etapa EX de la instrucción de salto no se ha evaluado la condición para determinar si la predicción ha sido incorrecta.

Para la predicción dinámica de dos bits:

- $P_{din_OK} = 0$ ciclos, dado que la predicción se hace en la etapa IF
- $P_{din_fallo} = 2$ ciclos, dado que hasta el final de la etapa EX de la instrucción de salto no se ha evaluado la condición para determinar si la predicción ha sido incorrecta. Como en este caso se ha captado la instrucción que actualiza r1 en el ciclo (5), anterior al ciclo en el que se capta la instrucción de salto condicional (ciclo (6)), la etapa EX de la instrucción de salto condicional no tiene que esperar un ciclo como ocurría en el fallo con la predicción estática.

La traza de la Tabla 15 corresponde a la ejecución del programa para $it=2$. Como se puede ver, se produce una predicción estática correcta, y una predicción dinámica incorrecta, por lo tanto la penalización total es $P = 1 + 2 = 3$ ciclos, tal y como está indicado en la Tabla 15.

(c) Para obtener la penalización en el caso de 50 iteraciones podemos tener en cuenta la siguiente expresión:

$$P(it) = P_{est_OK} + (it - 2) \times P_{din_OK} + P_{din_fallo} = 1 + (50-2) \times 0 + 2 = 3$$

Dado que cuando la predicción dinámica es correcta no hay penalización, solo hay penalización en la predicción estática correcta correspondiente a la primera iteración y en la predicción dinámica incorrecta de la última iteración. Así, independientemente del número de iteraciones, se tienen 3 ciclos de penalización.

Problema 8. En un programa, una instrucción de salto condicional (a una dirección de salto posterior) tiene el siguiente comportamiento en una ejecución de dicho programa:

NSSS NNSS NSSN SSSN

donde S indica que se produce el salto y N que no. Indique la penalización que se introduce si se utiliza: (a) Predicción fija (siempre se considera que se va a producir el salto); (b) Predicción estática (si el desplazamiento es negativo se toma y si es positivo); (c) Predicción dinámica con dos bits, inicialmente en el estado (11); y (d) Predicción dinámica con tres bits, inicialmente en el estado (111).

NOTA: La penalización por saltos incorrectamente predichos es de 4 ciclos y para los saltos correctamente predichos es 0 ciclos.

Solución

Para conocer la penalización en cada una de las alternativas se tiene en cuenta la predicción que hace cada uno de los esquemas que se considera, y se compara con lo que finalmente ocurre (se produce o no se produce el salto).

(a) Dado que en el esquema de predicción fija considerado siempre se predice que se produce el salto, habrá penalización en todos los casos que no se produce (es decir, aparece N en la secuencia que se analiza). Como en NSSS NNSS NSSN SSSN hay seis N, la penalización es 6 penalizaciones \times 4 ciclos/penalización = 24 ciclos.

(b) El esquema predice *saltar* si el salto condicional se produce a una dirección anterior (desplazamiento negativo) y *no saltar* si se produce a una posición posterior (desplazamiento positivo). Como en este caso la instrucción saltaría a una dirección posterior, se produciría una penalización si se produce el salto dado que la predicción sería *no saltar*. Como en NSSS NNSS NSSN SSSN hay diez S, la penalización es 10 penalizaciones \times 4 ciclos/penalización = 40 ciclos.

(c) En este procedimiento de salto, al ser dinámico, hay que tener en cuenta el estado de los dos bits de historia que utiliza el procedimiento de predicción. La Tabla 16 indica los cambios de bits de historia, las predicciones y las penalizaciones que se producen.

Tabla 16. Evolución de bits de historia y predicciones para la secuenciada

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Se Produce	N	S	S	S	N	N	S	S	N	S	S	N	S	S	S	N
Bits Historia	11	10	11	11	11	10	01	10	11	10	11	11	10	11	11	11
Se Predice	S	S	S	S	S	S	N	S	S	S	S	S	S	S	S	S
Penalización	X				X	X	X		X			X				X

La Figura 28 se muestra el diagrama de estados que utiliza el procedimiento para actualizar los bits de historia y realizar las predicciones. Inicialmente, los bits de historia se encuentran en el estado 11 y predicen *saltar*. Como se produce *no saltar* hay penalización y se produce la transición al estado 10 en el que la predicción sigue siendo *saltar*.

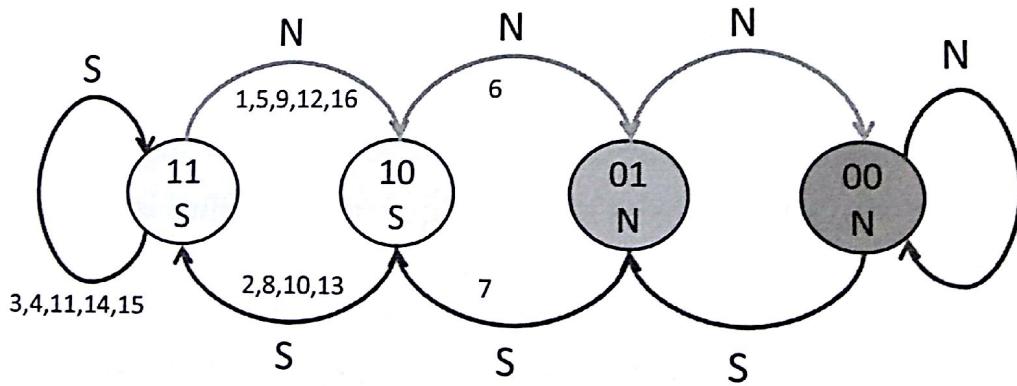


Figura 28. Diagrama de estados para la predicción dinámica de dos bits y transiciones para la secuenciada

En el diagrama de la Figura 28, esta transición se indica marcando el correspondiente arco del grafo con "1" (es la primera ejecución de la instrucción de salto). Así, se irán marcando las transiciones para las siguientes ejecuciones de la instrucción de salto (2,...,16). El estado de los bits de historia tras la ejecución número 16 de la instrucción de salto es 10. El número de penalizaciones es siete, y por lo tanto la penalización total es $7 \text{ penalizaciones} \times 4 \text{ ciclos/penalización} = 28 \text{ ciclos}$.

(d) Cuando se utiliza la predicción dinámica con tres bits, la predicción que se hace viene determinada por el dígito (0 o 1) que aparece más veces en los bits de historia. En nuestro caso consideraremos que si hay mayoría de unos se predice *saltar* y si hay mayoría de ceros se predice *no saltar*. Tras la predicción, se desplazan los bits de historia hacia la derecha, y se pierde el bit menos significativo, en tanto que como bit más significativo *se introduce un uno si se produce un salto* o *un cero si no se produce un salto*. En la Tabla 17 se muestra este comportamiento con la secuencia que indica el problema.

Como se puede observar en la Tabla 17, aquí casualmente las penalizaciones se producen en los mismos lugares para la predicción con tres y con dos bits de historia. Se tienen, por tanto siete penalizaciones, que dan lugar a un total de 28 ciclos de penalización.

Tabla 17. Evolución de bits de los tres bits de historia y predicciones para la secuencia dada

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Se Produce	N	S	S	S	N	N	S	S	N	S	S	N	S	S	S	N
Bits Historia	111	011	101	110	111	011	001	101	110	011	101	110	011	101	110	111
Se Predice	S	S	S	S	S	S	N	S	S	S	S	S	S	S	S	S
Penalización	X				X	X	X		X		X					X

Problema 9.

Para el bucle siguiente:

```

for (i=1;i<=5;i++)
{
    If (c=5) goto etiqueta;
    r=s+t;
    c=c+1;
    if (c>5) then goto etiqueta;
    r=r+t;
}
etiqueta:.....

```

un compilador ha generado el código ensamblador que se muestra en la Figura 29.

```

1      addi r1,r0,#5    ; r1=5 (iteraciones)
2      lw    r2,datoc   ; r2=c (datoc)=c
3      add  r4,r1,r0   ; r4=r1+r0 (r4=r1=5)
4      bucle: sub  r3,r1,r2   ; r3=5-c
5          beqz r3,etiqueta ; si r3=0(c=5) ir a etiqueta_1
6          addf f3,f2,f1   ; f3=f2+f1 (r=s+t)
7          addi r2,r2,#1   ; r2=r2+1 (c=c+1)
8          sub  r3,r2,r1   ; r3=c-5
9          bgtz r3,etiqueta ; si r3>0(c>5) ir a etiqueta_2
10         addf f3,f3,f1   ; f3=f3+f1 (r=r+t)
11         subi r4,r4,#1   ; r4=r4-1 (c=c+1)
12         bnez r4,etiqueta ; si r4!=0 ir a bucle_3
13     etiqueta:

```

Figura 29. Código generado por el compilador para el programa del problema

Indique cuál es la penalización efectiva debida a los saltos, en función del valor inicial de c (número entero), considerando que el procesador utiliza: (a) Predicción estática (si el desplazamiento es negativo se predice saltar y si es positivo se predice no saltar); (b) Predicción dinámica con un bit (1= predice Saltar; 0 = No Saltar; Se inicializa el bit de historia a 1); (c) Predicción dinámica con dos bits (11=predice Saltar; 10= Saltar; 01= No Saltar; 00= No Saltar; Se inicializan los bits de historia a 11). (d) ¿Cuál de los tres esquemas es más eficaz por término medio si hay un 50% de probabilidades de que c sea menor o igual a 0, un 25% de que sea mayor que 5; y un 25% de que sea cualquier número entre 1 y 5, siendo todos equiprobables?

NOTA: La penalización por saltos incorrectamente predichos es de 4 ciclos y para los saltos correctamente predichos es 0 ciclos.

Solución

Para resolver el problema se empieza por determinar los valores de las penalizaciones según el valor de la variable c, en cada uno de los procedimientos de penalización considerados y para cada una de las instrucciones de salto condicional que hay en el código.

En concreto, se tienen tres instrucciones de salto, la 5, 9, y 12. Las instrucciones 5 y 9 son saltos condicionales hacia adelante (hacia direcciones de memoria mayores) y la instrucción 12 es una instrucción de salto condicional hacia atrás (hacia direcciones de memoria menores) que controla el bucle. En lo que sigue, denominaremos instrucciones de salto condicional 1, 2, y 3, respectivamente, a las instrucciones de salto 5, 9, y 12. Si la instrucción de salto condicional i ($i=1,2,3$) da lugar a un salto, lo notaremos como Si, y si no da lugar a salto, Ni.

Para determinar el perfil de salto/no salto de cada una de las instrucciones se tiene en cuenta que no habrá más de 5 iteraciones, dado que el registro r4 que se utiliza para controlar el bucle se hace r4=5 en la instrucción 3. En la Tabla 18 se muestra la secuencia de Saltos/No saltos para las distintas instrucciones, según los valores de c, indicando entre paréntesis los valores de c y r4 (control del bucle) en cada iteración, en el momento en que la instrucción de salto condicional correspondiente se va a ejecutar.

Tabla 18. Secuencias de Saltos/No Saltos según el valor de c y las iteraciones del bucle

Iteración i	1	2	3	4	5
$c \leq 0$	$(c \leq 0)$ $N1$	$(c \leq 1)$ $N1$	$(c \leq 2)$ $N1$	$(c \leq 3)$ $N1$	$(c \leq 4)$ $N1$
	$(c \leq 1)$ $N2$	$(c \leq 2)$ $N2$	$(c \leq 3)$ $N2$	$(c \leq 4)$ $N2$	$(c \leq 5)$ $N2$
	$(r4=4)$ $S3$	$(r4=3)$ $S3$	$(r4=2)$ $S3$	$(r4=1)$ $S3$	$(r4=0)$ $N3$
$c = 1$	$(c = 1)$ $N1$	$(c = 2)$ $N1$	$(c = 3)$ $N1$	$(c = 4)$ $N1$	$(c = 5)$ $N1$
	$(c = 2)$ $N2$	$(c = 3)$ $N2$	$(c = 4)$ $N2$	$(c = 5)$ $N2$	$(c = 5)$ $S1$
	$(r4=4)$ $S3$	$(r4=3)$ $S3$	$(r4=2)$ $S3$	$(r4=1)$ $S3$	
$c = 2$	$(c = 2)$ $N1$	$(c = 3)$ $N1$	$(c = 4)$ $N1$	$(c = 5)$ $N2$	$(c = 5)$ $S1$
	$(c = 3)$ $N2$	$(c = 4)$ $N2$	$(c = 5)$ $N2$		
	$(r4=4)$ $S3$	$(r4=3)$ $S3$	$(r4=2)$ $S3$		
$c = 3$	$(c = 3)$ $N1$	$(c = 4)$ $N1$	$(c = 5)$ $N2$	$(c = 5)$ $S1$	
	$(c = 4)$ $N2$	$(c = 5)$ $N2$			
	$(r4=4)$ $S3$	$(r4=3)$ $S3$			
$c = 4$	$(c = 4)$ $N1$				
	$(c = 5)$ $N2$	$(c = 5)$ $S1$			
	$(r4=4)$ $S3$				
$c = 5$	$(c = 5)$ $S1$				
$c > 5$	$(c > 5)$ $S2$				
	$(c > 6)$				

Se puede resumir la secuencia de saltos/no saltos de la Tabla 18 como se indica a continuación:

Si $c \leq 0$, la secuencia es $(N1N2S3)^4N1N2N3$, que es equivalente a:

$$N1N2S3N1N2S3N1N2S3N1N2S3N1N2N3$$

es decir, la secuencia para la instrucción de salto 1 es $N1N1N1N1N1 = (N1)^5$, para la instrucción de salto 2 es $N2N2N2N2N2 = (N2)^5$, y para la instrucción de salto 3 es $N3N3N3N3S3 = (N3)^5S3$

Si $1 \leq c \leq 5$, la secuencia es $(N1N2S3)^{5-c}S1$

Si $c > 5$, la secuencia de $N1S2$

Una vez determinadas las secuencias de saltos/no saltos que ocasionan las instrucciones en función del parámetro c podemos determinar la penalización para cada tipo de predictor, en cada caso.

(a) Para la predicción estática, las instrucciones de salto 1 y 2 saltan hacia adelante. Por lo tanto, para ellas la predicción es no saltar, y no se produce penalización si no se produce salto (N). En cambio, la instrucción 3 salta hacia atrás, y por tanto, no hay penalización si se produce el salto (S). Por tanto las penalizaciones son:

- Si $c \leq 0$, como la secuencia es $(N1N2S3)^c N1N2N3$, solo hay penalización debido al $N3$ último, es decir $P=4$.
- Si $1 \leq c \leq 5$, la secuencia es $(N1N2S3)^{c-1} S1$, solo hay penalización debido al $S1$ último, es decir $P=4$.
- Si $c > 5$, la secuencia es $N1S2$, y solo hay penalización debido al $S2$ último. Es decir $P=4$.

(b) En este caso la predicción es dinámica con un bit de historia, que se inicializa a 1. El diagrama de estados para este tipo de predicción se muestra en la Figura 30(a).

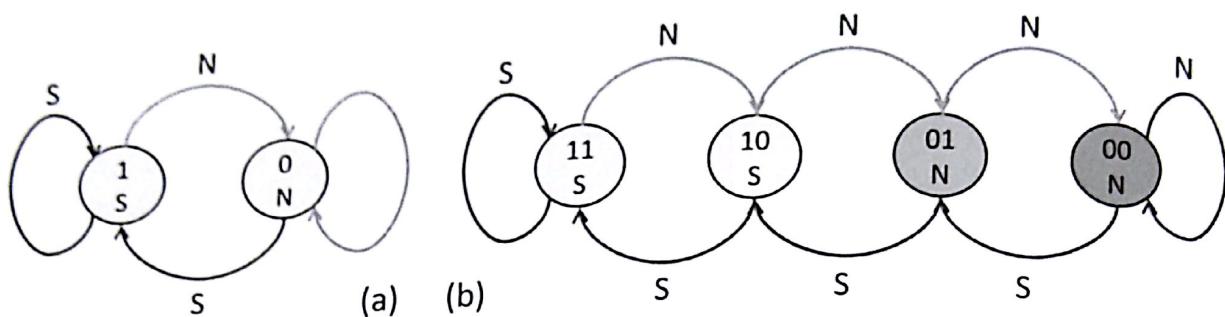


Figura 30. Diagrama de estados para la predicción dinámica con (a) un bit y (b) dos bits de historia

En este caso, hay que tener en cuenta la secuencia de instrucciones de salto de cada tipo, por separado:

- Si $c \leq 0$, la secuencia para la instrucción 1 es $(N1)^5$, se producirá una penalización para la primera ejecución ($N1$) dado que el bit de historia se inicializa a 1, y por lo tanto la predicción es saltar (según la Figura 30(a)), se produce la transición al estado 0, donde la predicción es no saltar y no se producirán más penalizaciones, por tanto, $P=4$ para $(N1)^5$. Para la instrucción 2, $(N2)^5$, y se tendrá lo mismo que en el caso anterior, $P=4$. En la instrucción 3, se tiene $(S3)^5 N3$, y la penalización se produce en el último $N3$ ya que para los $S3$ el estado se mantendrá en el valor 1 al que estaba inicializado. Es decir $P=4$ para la instrucción 3. Por lo tanto, la penalización total será igual a $P=3 \times 4=12$ ciclos.
- Si $1 \leq c < 5$, dado que secuencia es $(N1N2S3)^{c-1} S1$, se tiene $(N1)^{c-1} S1$ para la instrucción 1, y habrá dos penalizaciones (el primer $N1$ y el último $S1$) salvo en el caso de que $c=5$. Para la instrucción de salto 2 la secuencia de saltos/no saltos es $(N2)^{c-1}$ y solo habrá penalización en la primera ejecución (el primer $N2$). Finalmente, la instrucción de salto 3 da lugar a la secuencia $(S3)^{c-1}$ y no genera penalización dado que el bit de historia está inicializado a 1, empieza prediciendo saltar y se mantiene ahí. El total de penalizaciones para estos valores de c es $P=3 \times 4=12$.
- Si $c=5$, la secuencia es $S1$, y no habrá penalización porque solo se ejecuta la instrucción de salto 1 y ésta da lugar a salto.
- Si $c > 5$, la secuencia para la instrucción 1 es $N1$ y habrá una penalización en este caso, dado que el bit de historia se inicializa a 1, y la predicción es saltar. En el caso

de la instrucción 2, como se produce un salto (S2), no hay penalización. Por lo tanto, en este caso solo hay una penalización, y $P=4$.

(c) En este caso se razona utilizando las secuencias para cada instrucción, igual que se ha hecho en el caso (b), pero se tiene en cuenta el diagrama de estados de la Figura 30(b) para un predictor dinámico con dos bits de historia.

- Si $c \leq 0$, la secuencia para la instrucción 1 es $(N1)^5$, se producirá una penalización para la primera ejecución ($N1$) dado que el bit de historia se inicializa a 11 y la predicción es saltar (según la Figura 30(a)), se produce la transición al estado 10, donde la predicción también es saltar y se produce otra penalización. Despues se produce la transición al estado 01 donde la predicción es no saltar y no se producirán más penalizaciones, por lo que $P=8$ para $(N1)^5$. Para la instrucción 2, $(N2)^5$, y se tendrá lo mismo que en el caso anterior: $P=8$. En la instrucción 3, se tiene $(S3)^5 N3$, y la penalización se produce en el último $N3$ ya que para los $S3$ el estado se mantendrá en el valor 11 al que estaba inicializado. Es decir $P=4$ para la instrucción 3. Por lo tanto, la penalización total será igual a $P=5 \times 4 = 20$ ciclos.
- Si $1 \leq c \leq 5$, se tiene $(N1)^{5-c} S1$ para la instrucción 1. Si $c \leq 3$, tendremos más de dos $N1$ seguidos, y habrá dos penalizaciones debido al primer y segundo $N1$, y otra más debido al último $S1$. Para $c=4$ se tendrá $N1 S1$ y solo habrá penalización para el primer $N1$. Para $c=5$, se tiene $S1$ y no habrá penalización. Para la instrucción de salto 2 la secuencia de saltos/no saltos es $(N2)^{5-c}$. Si $c \leq 3$, tendremos más de dos $N2$ seguidos y habrá dos penalizaciones. Si $c=4$ habrá un $N2$, y por lo tanto una penalización. Finalmente, la instrucción de salto 3 da lugar a la secuencia $(S3)^{5-c}$ y no genera penalización dado que el bit de historia está inicializado a 1, empieza prediciendo saltar y se mantiene ahí. El total de penalizaciones para estos valores de c son los siguientes: si $c \leq 3$, $P=(3+2+0) \times 4 = 20$ ciclos; si $c=4$, $P=(1+1) \times 4 = 8$; y si $c=5$, $P=0$
- Si $c > 5$, la secuencia para la instrucción 1 es $N1$ y habrá una penalización en este caso dado que el bit de historia se inicializa a 1 y la predicción es saltar. En el caso de la instrucción 2, como se produce un salto (S2) no hay penalización. Por lo tanto, en este caso solo hay una penalización, y $P=4$.

En la Tabla 19 se resumen las penalizaciones para cada valor de c y los distintos tipos de procedimiento de predicción considerados.

Tabla 19. Penalizaciones según procedimiento de predicción y valor de c

	Estática	Dinámica (1 bit)	Dinámica (2 bits)
$c \leq 0$	4	12	20
$c=1$	4	12	20
$c=2$	4	12	20
$c=3$	4	12	20
$c=4$	4	12	8
$c=5$	4	0	0
$c > 5$	4	4	4

A partir de la Tabla 19 podemos determinar el mejor esquema según las probabilidades que tengamos de que c tome uno u otro valor. Tenemos que hay el 50% de probabilidades de que $c \leq 0$, el 25% de que $c > 5$, y el 25% de que $1 \leq c \leq 5$ siendo, en este caso, las probabilidades iguales para todos los valores de c , y por lo tanto iguales al 5%. Por lo tanto, las expresiones para la penalización media en cada esquema de predicción serán:

Caso (a). Dado que para cualquier valor de c la penalización de la predicción estática es 4, tenemos que $P_{estatica} = 4 \text{ ciclos}$.

Caso (b). La penalización media para la predicción dinámica con un bit de historia P_{Din1} se obtiene de la expresión:

$$\begin{aligned} P_{Din1} &= (p(c \leq 0) + p(c = 1) + p(c = 2) + p(c = 3) + p(c = 4)) \times 12 + \\ &\quad + p(c = 5) \times 0 + p(c > 5) * 4 = \\ &= (0.5 + 0.05 + 0.05 + 0.05 + 0.05) \times 12 + 0.25 \times 4 = 9.4 \text{ ciclos} \end{aligned}$$

Caso (c). En el caso de predicción dinámica con dos bits de historia, la penalización media P_{Din2} es:

$$\begin{aligned} P_{Din2} &= (p(c \leq 0) + p(c = 1) + p(c = 2) + p(c = 3)) \times 20 + (8 \times p(c = 4)) + \\ &\quad + p(c = 5) \times 0 + p(c > 5) * 4 = \\ &= (0.5 + 0.05 + 0.05 + 0.05) \times 20 + 0.05 \times 8 + 0.25 \times 4 = 14.4 \text{ ciclos} \end{aligned}$$

En este caso concreto, la mejor opción es la predicción estática, seguida por la dinámica de un bit de historia. La peor es la dinámica con dos bits de historia. Obviamente, si las probabilidades de que c tome los distintos valores cambian, el mejor esquema puede cambiar. Por ejemplo, si las probabilidades de que c tome los valores 4 ó 5 fuesen muy elevadas en relación con las demás, la predicción dinámica con dos bits de historia sería más competitiva. También sería interesante analizar qué cambios se producirían si en lugar de inicializar los bits de historia a 1 y a 11, se inicializan a 0 y a 00.

Problema 10. Indique cómo se podrían utilizar instrucciones con predicado para mejorar el paralelismo entre instrucciones (ILP) de la sentencia:

if ($A=0$) then $A=A-1$; else $A=A+1$;

NOTA: Puede utilizar las instrucciones con predicado que considere, para obtener el comportamiento más eficiente para este trozo de código.

Solución

Una posible implementación de la sentencia indicada en el programa mediante instrucciones máquina típicas sería:

lw	R1, 0 (R3)	
bnez	R1, dir1	; Cargar A (cuya dirección está en R3)
subi	R1, R1, 1	; Saltar a dir1 si R1(=A) no es cero
j	dir2	; R1 = R1 -1
dir1:	addi R1, R1, 1	; Saltar a dir2
dir2:	sw 0 (R3), R1	; R1 = R1 +1
		; Almacenar R1 en A

Mediante la utilización de predicados se podría transformar el código como se indica a continuación:

lw	R1, 0 (R3)	
P1, P2	cmp R1, 0	; Cargar A (dirección en R3)
(P1)	subi R1, R1, 1	; Si R1=0, P1=1(P2=0), Y si no P2=1(P1=0)
(P2)	addi R1, R1, 1	; R1 = R1-1 si P1=1
	sw 0 (R3), R1	; R1 = R1+1 si P2=1
		; Almacenar R1 en A

De esta forma se evitan instrucciones de salto, pero existen dependencias de tipo WAR y WAW entre las instrucciones de suma y resta (las dos instrucciones leen R1 y escriben en R1). Estas dependencias pueden no ser resueltas por el hardware y ocasionar que se pierdan algunos ciclos. La situación se puede mejorar con el siguiente código:

lw	R1, 0 (R3)	
P1, P2	cmp R1, 0	; Cargar A (dirección en R3)
(P1)	subi R2, R1, 1	; Si R1=0, P1=1(P2=0), Y si no P2=1(P1=0)
(P2)	addi R4, R1, 1	; R2 = R1-1 si P1=1
(P1)	sw 0 (R3), R2	; R4 = R1+1 si P2=1
(P2)	sw 0 (R3), R4	; Almacenar R2 en A
		; Almacenar R4 en A

Así, utilizando los predicados también para las instrucciones de almacenamiento se pueden renombrar los operandos. Al desaparecer dependencias explícitas, el compilador puede reorganizar el código con más libertad y aprovechar mejor las posibilidades que ofrece la máquina.

Problema 11. Suponga un procesador en el que a todas las instrucciones se le puede anteponer un predicado, p, con dos valores posibles, 0 y 1 (o "true" y "false"):

(p) instrucción

Así, si la instrucción tiene predicado sólo se ejecutaría si el valor del predicado p es 1 (o "true"). Dicho valor habrá sido establecido por una instrucción de comparación con el formato

[p1, p2] cmp.cnd rx, ry

donde cnd es la condición que se comprueba entre rx y ry (lt, le, gt, ge, eq, ne). Si la condición es verdadera, p1=1 [y p2=0], y si es falsa, p1=0 [y p2=1].

(a) Escriba la secuencia de instrucciones máquina que implementarían el siguiente código sin utilizar ninguna instrucción de salto:

```

if ((x≥y) and (y≥0)) then x=x+y;
else if ((y<0) and (x<y)) then x = 2*y;

```

- (b) Optimice el código anterior para un procesador VLIW con dos slots de emisión, en el que las instrucciones de comparación sólo pueden colocarse en el primero de ellos.

NOTA: los datos x e y son de 32 bits y están en posiciones consecutivas de memoria y los predicados no se pueden suponer inicializados a ningún valor; y las latencias de las operaciones son de 1 ciclo para las sumas, restas y comparaciones, de tres ciclos para las multiplicaciones y de cuatro ciclos para las cargas de memoria).

Solución

(a) Para asignar correctamente los predicados y evitar instrucciones de salto, es conveniente considerar el correspondiente diagrama de flujo del código. La Figura 31 proporciona dicho diagrama, para cuya elaboración se ha tenido en cuenta que para cargar los predicados se utilizan instrucciones de comparación de dos operandos y se han indicado los predicados que habría que utilizar con el valor que deberían tomar para habilitar la ejecución del camino correspondiente del diagrama de flujo.

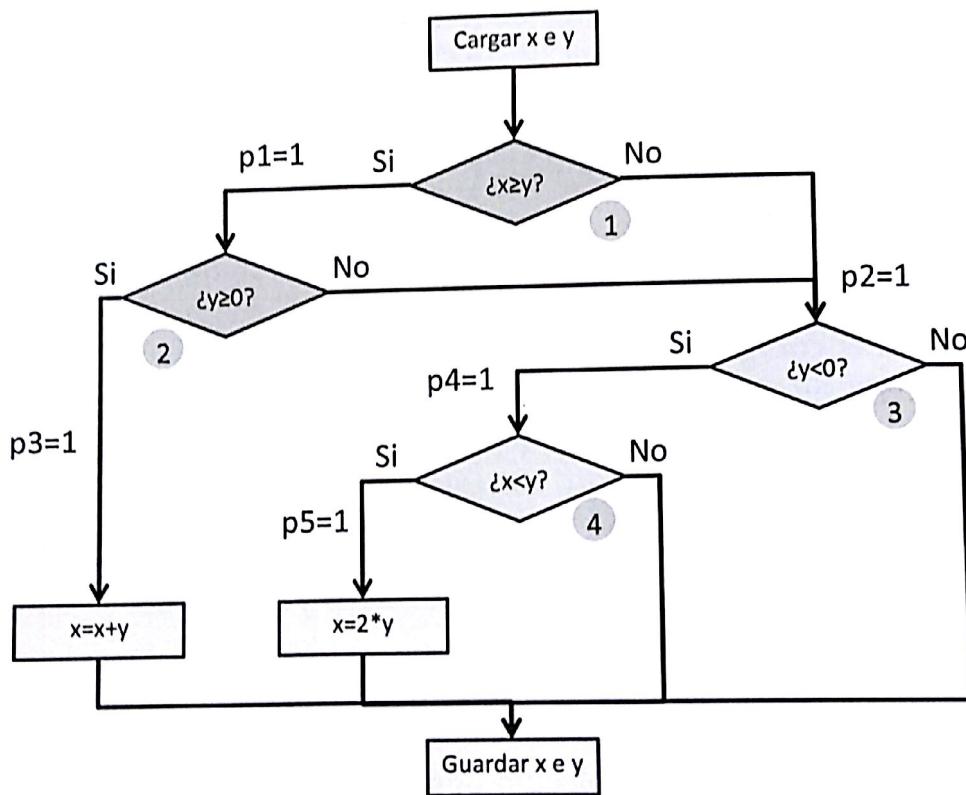


Figura 31. Diagrama de flujo para el código del programa

Las comparaciones marcadas con 1 y 2 en la Figura 31 corresponden a la condición del "if" del código y las marcadas con 3 y 4 a la alternativa del "else". Para que se ejecute $x=x+y$ es necesario que cumplan 1 y 2. Se puede utilizar un predicado p_1 que sólo si se verifican 1 y 2 termina siendo igual a 1. Si cualquiera de las dos comparaciones no se cumple, se pasaría a ejecutar la opción del "else". Se puede utilizar un predicado, p_2 , que termina siendo igual a 1 si no se cumple 1 ó 2.

El código a ejecutar sería el siguiente:

1		lw	r1,x	; r1 = x
2	p3	lw	r2,x	; r2 = y
3	p4	cmp.ne	r0,r0	; Inicializamos p3=0
4	p5	cmp.ne	r0,r0	; Inicializamos p4=0
5	p1, p2	cmp.ge	r0,r0	; Inicializamos p5=0
6	(p1) p3, p2	cmp.ge	r1,r2	; ¿x >= y?
7	(p2) p4	cmp.lt	r2,r0	; ¿y >= 0?
8	(p4) p5	cmp.lt	r2,r0	; ¿y < 0?
9	(p3)	add	r1,r2	; ¿x < y?
10	(p5)	slli	r3,r1,#1	; r3 = r1+r2
11		sw	x,r3	; r3 = 2*r1
12				; x = r3

En dicho código, las instrucciones 3, 4 y 5 aseguran que los predicados p3, p4, y p5 toman el valor 0. Si no se hiciera esto, no podríamos saber con certeza el valor que toman esos predicados para las instrucciones 9, 10, y 11 dado no se puede asegurar que las instrucciones 7, 8, y 9 (las instrucciones en las que se asigna valor a p3, p4, y p5) se ejecuten (la ejecución dependen de los valores de sus predicados).

Tras las instrucciones 6 y 7, p3 será igual a 1 si las condiciones de las instrucciones 6 y 7 se verifican las dos, y p2=1 si alguna de las dos condiciones de comparación no se cumple. Si p3 es igual a 1 se ejecutaría la instrucción 10, y si p2=1 se ejecutaría la instrucción 8. Si la condición se verifica para la comparación, se haría p4=1 y se ejecutaría 9, y si se verifica la condición tendríamos p5=1. Por tanto, solo si se verifican las dos condiciones tendríamos que p5=1, y entonces se podría ejecutar la instrucción 11. La instrucción 12 se ejecutaría siempre. En realidad podríamos ahorrarnos su ejecución si se introduce únicamente para los casos en los que p3 o p5 son iguales a 1. Teniendo en cuenta la ubicación de las instrucciones que se muestra en la Tabla 20 lo único que ocurriría es que habría dos instrucciones de almacenamiento, una en cada slot, que dependerían de los correspondientes predicados, p3, y p5. El interés de utilizarlas es que al ahorrarse accesos a memoria innecesarios no se ocuparía más ancho de banda de acceso a memoria que el imprescindible.

Tabla 20. Código VLIW para el diagrama de la Figura 31

slot 1		slot 2	
lw	r2,x	lw	r1,x
cmp.ne	r0,r0	nop	
cmp.ne	r0,r0	nop	
p5	cmp.ne	nop	
p1, p2	cmp.ge	nop	
(p1) p3, p2	cmp.ge	nop	
(p2) p4	cmp.lt	(p3) add	r3,r1,r2
(p4) p5	cmp.lt	nop	
	nop	(p5) slli	r3,r1,#1
	nop	sw	x,r3

Tal y como están distribuidas las operaciones en la Tabla 20, cuando se necesitan los datos r_1 y r_2 ya han pasado los cuatro ciclos de retardo de la instrucción de carga de memoria. Obviamente, la operación $y=2^x$ no se implementa como una multiplicación, sino como un desplazamiento a la izquierda.

Es posible reducir el número de predicados en este problema. De hecho, el mismo predicado p_2 se podría utilizar de forma similar a p_1 para la alternativa del "else". Así, solo si las comparaciones 3 y 4 se cumplen, p_2 acabaría siendo igual a 1 y se ejecuta $x=2^y$.

El código a ejecutar sería el siguiente:

	lw	r_1, x	; $r_1 = x$
	lw	r_2, x	; $r_2 = y$
(p1)	p_1, p_2	cmp.ge	r_1, r_2 ; $\{x \geq y\}$?
(p2)	p_1, p_2	cmp.ge	r_2, r_0 ; $\{y \geq 0\}$?
(p2)	p_2	cmp.lt	r_2, r_0 ; $\{y < 0\}$?
(p2)	p_2	cmp.lt	r_1, r_2 ; $\{x < y\}$?
(p1)		add	r_3, r_1, r_2 ; $r_3 = r_1 + r_2$
(p2)		slli	$r_3, r_1, \#1$; $r_3 = 2^r_1$
		sw	x, r_3 ; $x = r_3$

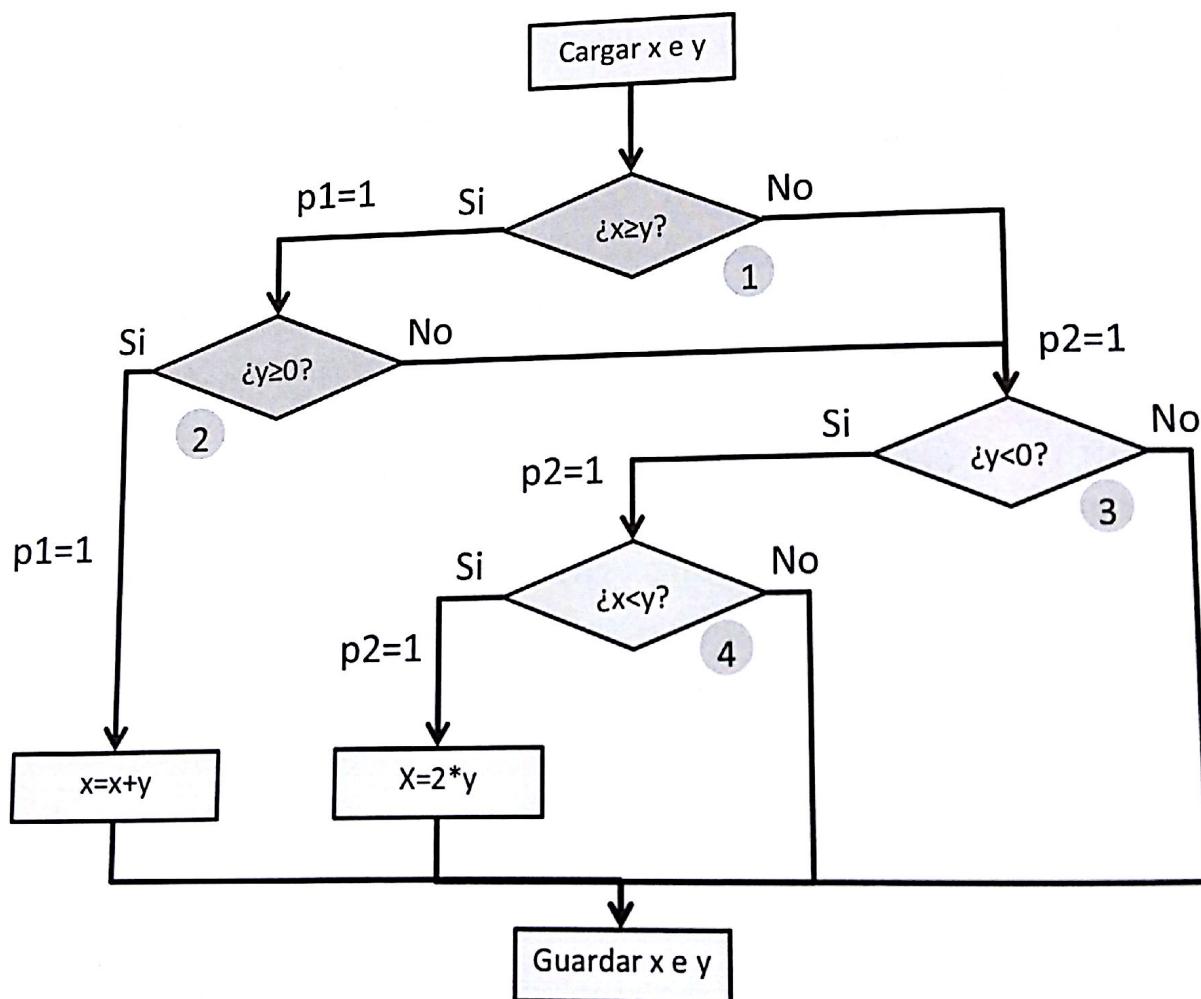


Figura 32. Diagrama de flujo para el código del programa con la nueva asignación de predicados.

La Tabla 21 muestra el correspondiente código VLIW para esa alternativa.

Tabla 21. Código VLIW para el diagrama de la Figura 32

slot 1		slot 2	
	lw r2, x		lw r1, x
p1, p2	cmp.ge r1, r2	nop	
(p1)	cmp.ge r2, r0	nop	
(p2)	cmp.ltr r2, r0	(p1) add r3, r1, r2	
(p2)	cmp.ltr r1, r2	nop	
	nop	(p2) slli r3, r1, #1	
	nop	sw x, r3	

En cualquier caso, a pesar de que se necesitan tres instrucciones VLIW menos, el retardo de la carga de memoria hará que se haya que esperar tres ciclos más para enviar la primera instrucción de comparación, que necesita los datos x e y. El único beneficio es la reducción en el número de predicados (es necesario que los predicados se lean a principio del ciclo de ejecución y se escriban al final).

Problema 12. En un procesador de 32 bits con arquitectura LOAD/STORE (los accesos a memoria se hacen mediante instrucciones de carga y almacenamiento de registros y las operaciones se hacen con datos en registros), todas las instrucciones pueden predicarse. Para establecer los valores de los predicados se utilizan instrucciones de comparación (cmp) con el formato

(p) p1, p2 cmp.cnd x, y

donde cnd es la condición que se comprueba entre x e y (lt, le, gt, ge, eq, ne). Si la condición es verdadera p1=1 y p2=0, y si es falsa, p1=0 y p2=1. Una instrucción sólo se ejecuta si el predicado p=1 (ese valor se habrá establecido en otra instrucción de comparación previa). Para el siguiente código de alto nivel, escriba un código máquina para el procesador descrito que no tenga ninguna instrucción de salto:

```
for (i=0; i<2; i++)
    if (a[i]==0) then c[i]=a[i]+b[i];
    else c[i]=a[i]+1;
```

NOTA: los predicados no se encuentran inicializados a ningún valor.

Solución

El código que indica el problema contiene un bucle con dos iteraciones, por tanto podemos evitar la instrucción de control del bucle desenrollándolo. En este caso sólo que habría que repetir el cuerpo del bucle dos veces. Obviamente, esta solución solo se podrá adoptar si el número de instrucciones del cuerpo del bucle no es muy elevado (el tamaño del código crecería de forma considerablemente). Para evitar las instrucciones de salto dentro del bucle utilizaremos predicados.

1		lw	r1, 0(a)	; se carga a[1] en r1
2	(p1)	p1, p2 cmp.eq	r1, 0	; ¿a[1]=0? p1 y p2 se cargan
3	(p1)	lw	r2, (b)	; se carga b[1] en r2 si p1=1
4	(p2)	add	r3, r1, r2	; r3=r1+r2 si p1=1
5	(p2)	add	r3, r1, #1	; r3=r1+1 si p2=1
6		sw	0(c), r3	; r3 se almacena en c[1]
7		lw	r1, 4(a)	; se carga a[2] en r1
8	(p1)	p1, p2 cmp.eq	r1, 0	; ¿a[1]=0? p1 y p2 se cargan
9	(p1)	lw	r2, 4(b)	; se carga b[2] en r2 si p1=1
10	(p2)	add	r3, r1, r2	; r3=r1+r2 si p1=1
11	(p2)	add	r3, r1, #1	; r3=r1+1 si p2=1
12		sw	4(c), r3	; r3 se almacena en c[2]

Las instrucciones 1-6 corresponden a la primera iteración y las 7-12 a la segunda. Dado que los datos son de 32 bits y se direccionan bytes, el desplazamiento que debe utilizarse con las direcciones a partir de las que empiezan los arrays (es decir a, b, y c) para acceder a los componentes utilizados en la segunda iteración es igual a 4, tal y como se observa en las instrucciones 7, 9, y 12.

Si el procesador no implementa renombrado de registros en hardware, se pueden reducir las dependencias entre las instrucciones sustituyendo los registros r1, r2 y r3 de las instrucciones 7 a 12, por r4, r5, y r6, respectivamente. Igualmente se puede hacer con los registros de predicado, y los p1 y p2 de las instrucciones 9, 10 y 11, se pueden sustituir por p3 y p4, respectivamente. Si se dispone de un procesador VLIW con dos slots y se puede emitir cualquier tipo de instrucción a cada slot, el código que se conseguiría es bastante compacto:

slot 1			slot 2		
	lw	r1, 0(a)	lw	r4, 4(a)	
p1, p2	cmp.eq	r1, 0	p3, p4	cmp.eq	r4, 0
(p1)	lw	r2, (b)	(p3)	lw	r5, 4(b)
(p2)	add	r3, r1, r2	(p3)	add	r6, r4, r5
(p2)	add	r3, r1, #1	(p4)	add	r6, r4, #1
	sw	0(c), r3	sw		4(c), r6

Problema 13. Un procesador VLIW emite dos operaciones por ciclo (es decir, tiene dos campos o "slots" en cada instrucción VLIW) y todas las operaciones pueden predicarse. Los valores de los predicados se utilizan instrucciones de comparación (cmp) con el formato

(p) p1[, p2] cmp.cnd x, y

donde cnd es la condición que se comprueba entre x e y (lt, le, gt, ge, eq, ne). Si la condición es verdadera p1=1 [y p2=0], y si es falsa, p1=0 [y p2=1]. Una instrucción sólo se ejecuta si el predicado p=1 (ese valor se habrá establecido en otra instrucción de comparación previa). ¿Cuál sería el código VLIW de la siguiente sentencia sin que haya operaciones de salto en las iteraciones del bucle, y considerando que las operaciones de comparación solo pueden aparecer en el primer slot, las de salto solo en el segundo, y el resto de operaciones en cualquiera de los dos slots?

```

for(i=0; i<n; i++)
    if (a[i]<0) then b[i]=a[i]+2;
    else b[i]=2*a[i];

```

NOTA: los predicados no se encuentran inicializados a ningún valor; tras las operaciones de carga de memoria se debe esperar un ciclo hasta que el dato esté disponible; y se implementa salto retardado de forma que la instrucción VLIW que se capte después de la instrucción de salto siempre se ejecuta.

Solución

Para empezar, se escribirá un código con instrucciones escalares que corresponderán a las operaciones que se codifican en los *slots* de las instrucciones VLIW. Un posible código con instrucciones típicas de un repertorio LOAD/STORE que se pueden predicar como se indica en el enunciado puede ser el siguiente:

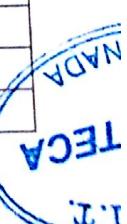
1	inic:	addi r1, r0, #n	; r1 ← n (r0=0)
2	bucle:	add r2, r0, r0	; r2 ← 0 (para recorrer a y b)
3		lw r3, a(r2)	; r3 apunta a a[i] (inic. a[0])
4		p1, p2 cmp.lt r3, r0	; p1=1 (p2=0) si r3<0
5	(p1)	addi r4, r3, #2	; r4=a[i]+2
6	(p2)	add r4, r3, r3	; r4=2*a[i]
7		sw b(r2), r4	; b[i] almacena r4
8		addi r2, r2, #4	; apuntar al siguiente a[] y b[]
9		subi r1, r1, #1	; una iteración menos
10		p3 cmp.gt r1, r0	; p3=1 mientras r1>0
11	(p3)	j bucle	; salta si r1>0 (quedan iter.)

Las instrucciones con predicado que se han utilizado en el programa anterior permiten eliminar las instrucciones de salto condicional en el cuerpo del bucle, e incluso evitar el uso de una instrucción de salto condicional para controlar el final de las iteraciones: se utiliza una instrucción de salto incondicional con predicado. No obstante, habría que ver si la implementación hardware hace que esto sea más eficiente (dos instrucciones) que un salto condicional.

A continuación, la Tabla 22 muestra la ubicación de las operaciones en los dos *slots* de las instrucciones VLIW teniendo en cuenta las dependencias entre las instrucciones y que las instrucciones de comparación solo pueden ubicarse en el primer *slot* y las de salto en el segundo.

Tabla 22. Código VLIW para la secuencia de instrucciones escalares del problema

Etiqueta	slot1	slot2
inic:	addi r1, r0, #n	add r2, r0, r0
bucle:	lw r3, a(r2)	nop
	p1, p2 cmp.lt r3, r0	nop
(p1)	addi r4, r3, #2	(p2) add r4, r3, r3
	sw b(r2), r4	subi r1, r1, #1
		addi r2, r2, #4
p3	cmp.gt r1, r0	(p3) j bucle
	nop	nop
	nop	



Como se puede observar en el código, se utilizan siete instrucciones VLIW en las que no se "llenan" tres *slots* (más los dos *slots* tras el salto retardado). Se ha tenido en cuenta las dependencias RAW entre las instrucciones 2 y 3, 3 y 4, 4 y 5, 4 y 6, 5 y 7, 6 y 7, 9 y 10, y 10 y 11.

Entre las instrucciones 7 y 8 hay una dependencia WAR. Aunque se podrían emitir al mismo tiempo (se pueden ubicar en el mismo *slot*) si se tiene en cuenta que las lecturas del registro r2 se harán en una etapa anterior (decodificación y acceso a operandos) a la escritura en r2 que realiza la instrucción 8, aquí se ha preferido ponerlas en instrucciones VLIW distintas dado que se puede adelantar la instrucción 9 que es independiente de la 7 y la 8, y emitir la instrucción 8 junto con la instrucción 10, de comparación.

También se han puesto las dos operaciones de comparación en el primer *slot* y la de salto en el segundo. Además, teniendo en cuenta que el procesador implementa salto retardado ejecutando siempre la instrucción que sigue a la instrucción de salto se ha introducido una instrucción VLIW con dos nop: si hubieran operaciones y no tuvieran que ejecutarse porque se produce el salto, se podrían producir errores en los resultados.

Al ejecutar este código VLIW hay que tener en cuenta los ciclos de latencia entre las instrucciones. Por lo tanto, los ciclos en los que se producen las emisiones se muestran en la Tabla 23.

Así, después de emitir la instrucción en la que se accede a memoria en el ciclo 2 hay que esperar un ciclo para que el dato que se pasa a r3 esté disponible, y después de la instrucción que contiene el salto incondicional y se emite en el ciclo 8, dejamos, como se ha dicho antes, un no-operar (nop) ya que el salto se implementa como salto retardado de un hueco.

Tabla 23. Ciclos en las que se emiten las instrucciones VLIW de la Tabla 22

ciclo	Etiq.	<i>slot1</i>		<i>slot2</i>	
1	inic:		addi r1, r0, #n		add r2, r0, r0
2	bucle:		lw r3, a(r2)		nop
3					
4		p1, p2	cmp.lt r3, r0		nop
5	(p1)		addi r4, r3, #2	(p2)	add r4, r3, r3
6			sw b(r2), r4		subi r1, r1, #1
7		p3	cmp.gt r1, r0		addi r2, r2, #4
8			nop	(p3)	j bucle
9			nop		nop

Se puede intentar reducir el tiempo que tarda en producirse la emisión de las instrucciones reorganizando el código, de forma que se ocupen los huecos que aparecen debido a los retardos (por supuesto, manteniendo las dependencias y las restricciones de ubicación de las operaciones en los distintos *slots*). Una alternativa se muestra en la Tabla 24.

Tabla 24. Ciclos de emisión de instrucciones VLIW con operaciones reordenadas

ciclo	Etq.		slot1		slot2	
1	inic:		addi	r1, r0, #n	add	r2, r0, r0
2	bucle:		lw	r3, a(r2)	subi	r1, r1, #1
3		p3	cmp.gt	r1, r0	addi	r2, r2, #4
4		p1, p2	cmp.lt	r3, r0	nop	
5	(p1)		addi	r4, r3, #2	(p2)	add
6			nop		(p3)	j bucle
7			sw	(b) r2, r4		nop

Como se puede ver en la Tabla 24 se ha conseguido introducir operaciones para hacer algo útil en el ciclo de espera entre la instrucción de acceso a memoria y la que necesita el dato. Además, después de la instrucción de salto, se ha introducido la operación de almacenamiento en memoria, que siempre se tiene que ejecutar (se produzca o no se produzca el salto).

De esta forma, se han ahorrado dos ciclos por iteración (en el caso de la Tabla 24 se tardarían 6 ciclos en lugar de los 8 ciclos que se tienen en la Tabla 24). Estos $2 \times n$ ciclos que se ahorran pueden suponer una reducción de tiempo considerable si el valor de n es elevado.

Problema 14. ¿Cómo transformaría la secuencia de instrucciones VLIW que se proporciona a continuación de forma que sólo se utilicen instrucciones de movimiento de datos condicionales y sin que haya ninguna instrucción de salto condicional.

Inst.	slot1		slot 2	
1	lw	r1, x(r2)	add	r10, r11, r12
2	nop		add	r13, r10, r14
3	beqz	r3, direc	nop	
4	lw	r4, 0(r3)	nop	
5	lw	r5, 0(r4)	nop	
direc:				

Solución

Para resolver el problema se van a considerar sólo las instrucciones que se incluyen en el campo de operación 1, ya que las del segundo slot son independientes de éstas y no afectarán a los cambios que vamos a realizar en el código. Por lo tanto, nos centraremos en eliminar los saltos condicionales de la siguiente secuencia de instrucciones:

```

1   lw      r1, x(r2)
2   nop
3   beqz  r3, direc
4   lw      r4, 0(r3)
5   lw      r5, 0(r4)
direc:

```

En esta secuencia, la instrucción de salto 3 se introduce para que no se ejecute la instrucción de carga 4 si $r3 = 0$. Por tanto, se puede suponer que la instrucción de salto tiene la función de evitar una violación del acceso a memoria. En la Figura 33 se ilustra el uso que se da a los registros $r3$ y $r4$ como punteros, en las instrucciones 4 y 5 de la secuencia anterior.

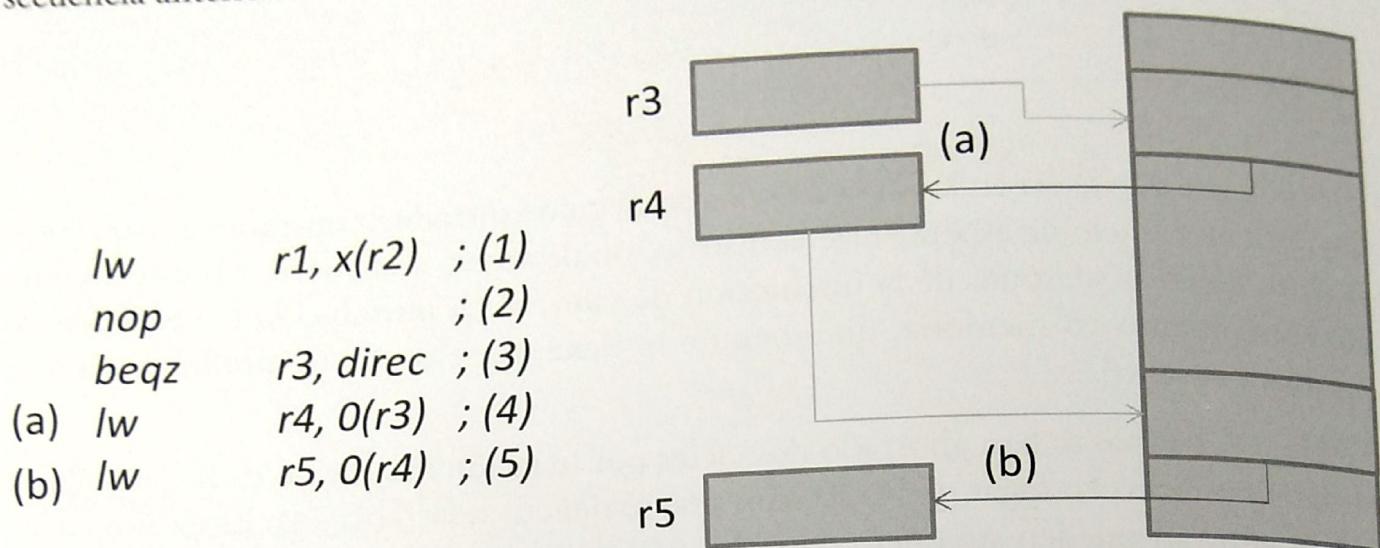


Figura 33. Uso de los registros $r3$, $r4$, y $r5$ en la secuencia inicial de instrucciones del *slot 1*

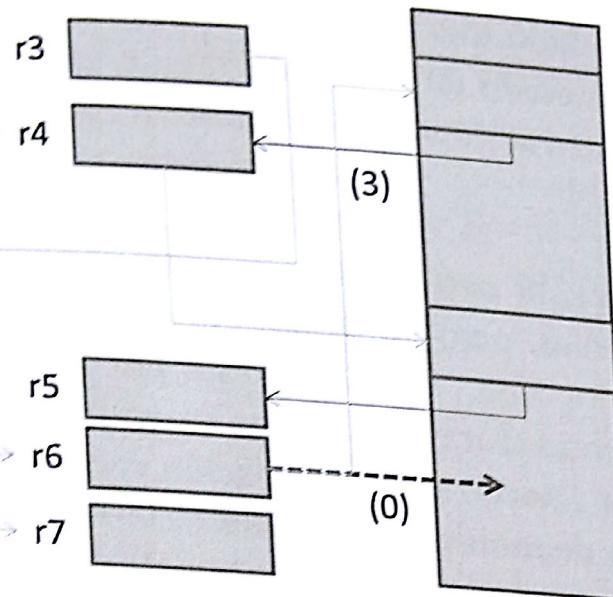
Así, si se adelantara la instrucción (4) para que estuviera delante de la instrucción de salto, la excepción que se produciría si $r3$ fuera igual a 0 haría que el programa terminase. Para que este cambio pueda realizarse es necesario que $r3$ sea distinto de cero siempre. En este caso, si existen registros disponibles, es posible utilizar instrucciones de movimiento condicional para evitar que se produzca la carga en caso de que se vaya a producir la excepción. El código sería:

```

addi    r6,r0,#1000 ; Se fija r6 a una dirección segura
lw      r1,x(r2)
mov    r7,r4          ; Se guarda r4 en r7
cmovnz r6,r3,r3      ; r3 a r6 si r3≠0
lw      r4,0(r6)       ; Carga especulativa
cmovz  r4,r7,r3      ; Si r3=0, r4 recupera su valor
beqz  r3,direc
lw      r5,0(r4)       ; Si r3≠0, hay que cargar r5
  
```

donde $r6$ y $r7$ son registros auxiliares. En $r6$ se carga primero una dirección segura (se sabe que no genera excepción porque contiene una dirección de memoria del espacio de usuario), y en $r7$ se introduce el valor previo de $r4$ para poder recuperarlo si la carga especulativa no debía realizarse. Como se puede comprobar, la especulación tiene un coste en instrucciones cuyo efecto final en el tiempo de ejecución depende de la probabilidad de que la especulación sea correcta o no. En la Figura 34 se proporciona una descripción gráfica de los efectos de las primeras transformaciones realizadas en la secuencia de operaciones del *slot 1*.

(0) addi	r6, r0, #1000	; Fijamos r6 a una dirección segura
lw	r1, x(r2)	
(1) mov	r7, r4	; Guardamos el contenido original de r4 en r7
(2) cmovnz	r6, r3, r3	; Movemos r3 a r6 si r3 es distinto de cero
(3) lw	r4, 0(r6)	; Carga especulativa
(4) cmovz	r4, r7, r3	; Si r3 es 0, hay que hacer que r4 recupere su valor
beqz	r3, direc	
lw	r5, 0(r4)	; Si r3 no es cero, hay que cargar r5



Si r6 no se carga con r3 en `cmovnz` como luego se hace `lw` hay que asegurarse que esa dirección no genera excepción

Figura 34. Primera transformación de las instrucciones del slot 1

También es posible evitar la instrucción de salto si se utilizan cargas especulativas con la misma estrategia de la Figura 34) para las dos instrucciones de carga protegidas por el salto en el código inicial. En este caso el código sería el siguiente:

addi	r6, r0, #1000	; Fijamos r6 a una dirección segura
lw	r1, x(r2)	
mov	r7, r4	; Se guarda el r4 inicial en r7
mov	r8, r5	; Se guarda r5 en r8
cmovnz	r6, r3, r3	; r3 a r6 si r3 es distinto de cero
lw	r4, 0(r6)	; Carga especulativa
lw	r5, 0(r4)	; Carga también especulativa
cmovz	r4, r7, r3	; Si r3=0, r4 recupera su valor
cmovz	r5, r8, r3	; Si r3=0, r5 recupera su valor

Hay que tener en cuenta que la dirección `direc` a la que se produce el salto viene a continuación de este trozo de código. Si no fuese así, no se podría aprovechar tan eficientemente el procesamiento especulativo. Así, en general, cuando existen saltos a distintas direcciones y no existe un punto de confluencia de esos caminos no suele ser posible obtener mejores prestaciones mediante cambios especulativos que eliminan la instrucción de salto.

Tabla 25 Código VLIW final sin instrucciones de salto

Inst.		slot 1	slot 2
1	addi	r6, r0, #1000	add r10, r11, r12
2	aw	r1, x(r2)	add r13, r10, r14
3	cmovnz	r6, r3, r3	mov r7, r4
4	lw	r4, 0(r3)	mov r8, r5
5	lw	r5, 0(r4)	nop
6	cmovz	r4, r7, r3	cmovz r5, r8, r3

Finalmente, si suponemos que las operaciones pueden ubicarse en cualquiera de los *slots* de las instrucciones VLIW dado que no se nos indica nada en el enunciado, podemos generar el código que se muestra en la Tabla 25, en el que se han respetado las dependencias entre las distintas instrucciones y no se utilizan instrucciones de salto, las dependencias entre las distintas instrucciones y no se utilizan instrucciones de salto,

Problema 15. En un procesador VLIW con dos *slots*, las operaciones de comparación solo pueden utilizar el primero de ellos, pero las demás operaciones pueden ubicarse indistintamente en cualquiera de los *slots*. Además, todas las operaciones pueden prediseñarse, igual que en el caso del procesador del problema anterior. Las latencias de las cargas de memoria son muy elevadas (cinco ciclos) en relación con las de las unidades de suma/resta (un ciclo) e incluso las de multiplicación (dos ciclos). Precisamente, para paliar el efecto de las latencias elevadas en el acceso a memoria, se incluye una instrucción de carga de memoria especulativa *lw.srd*, *desplaz(rs)*, compensa que permite indicar la operación de carga del dato de forma anticipada, incluso adelantando a instrucciones de salto condicional. Además, todos los registros del procesador disponen de un bit de marca para gestionar las excepciones como se indica a continuación:

- (1) Si la carga adelantada da lugar a una excepción, se marca el registro de destino (como registro envenenado) pero no se atiende la excepción.
- (2) Si una operación utiliza un operando marcado provocará que también se marque el registro resultado de esa operación como registro envenenado.
- (3) La excepción se atenderá si se intenta almacenar un resultado envenenado. En dicho caso, tras atender la excepción se ejecutará el código de reparación almacenado a partir de la dirección compensa.

Optimice el código

add	r3, r2, r1
add	r4, r5, r3
beqz	r4, cero
lw	r7, dato
add	r8, r4, r7
mult	r2, r1, r8
cero: sub	r9, r1, r4
mult	r1, r9, r2
sw	resul, r1

de manera que no aparezca ninguna instrucción de salto, se tengan en cuenta las latencias de las operaciones, las ubicaciones permitidas de las operaciones en los slots, y se adelante la instrucción de carga especulativamente todo lo que sea posible para tratar de ocultar su latencia.

¿A partir de qué probabilidad de que se produzca una excepción es beneficioso utilizar esta estrategia especulativa?

Solución

En primer lugar, podemos evitar la instrucción de salto condicional utilizando un predicado:

		add	r3, r2, r1
		add	r4, r5, r3
	p1	cmp.ne	r4, r0
(p1)		lw	r7, dato
(p1)		add	r8, r4, r7
(p1)		mult	r2, r1, r8
		sub	r9, r1, r4
		mult	r1, r9, r2
		sw	resul, r1

y ubicamos las operaciones en los dos slots de que disponen las instrucciones VLIW, tal y como se muestra en la Tabla 26. Para situar las operaciones se han tenido en cuenta sus dependencias y el hecho de que las operaciones de comparación tienen que ubicarse en el primer slot. Por otra parte, hay que introducir siete slots que quedan sin operación (los nop que hay que introducir).

Tabla 26. Código VLIW para la secuencia de instrucciones escalares del problema

Inst.		slot 1	slot 2
1		add r3, r2, r1	nop
2		add r4, r5, r3	nop
3	p1	cmp.ne r4, r0	sub r9, r1, r4
4	(p1)	lw r7, dato	nop
5	(p1)	add r8, r4, r7	nop
6	(p1)	mult r2, r1, r8	nop
7		nop	mult r1, r9, r2
8		sw resul, r1	nop

El número de ciclos que tardan en emitirse las instrucciones de la Tabla 26 se obtienen a partir de la Tabla 27. Hacen falta 14 ciclos.

Como se puede ver en la Tabla 27, la carga introduce varios ciclos de espera que ralentizan la emisión de las instrucciones. Por eso sería interesante adelantar esta instrucción lo más posible. Dado que existen dependencias RAW de la instrucción 1 con la 2, de la 2 con la 3, y de la 3 con la propia instrucción de carga (está vigilada por p1), sólo se podría adelantar la carga especulativamente para evitar el efecto de las

Fundamentos de...

excepciones que puedan ocurrir en el acceso a memoria (por fallo de página, etc.), ya que si r_4 toma el valor 0 al calcularse en la instrucción 2, la carga no debería haberse realizado y por tanto, no debería haber ocurrido ninguna excepción. La Tabla 28 muestra el nuevo código VLIW.

Tabla 27. Ciclos en los que se emiten las instrucciones del código VLIW de la Tabla 26

Ciclo		slot 1	slot 2
1		add r_3, r_2, r_1	nop
2		add r_4, r_5, r_3	nop
3	p1	cmp.ne r_4, r_0	sub r_9, r_1, r_4
4	(p1)	lw r_7, dato	nop
5			
6			
7			
8			
9	(p1)	add r_8, r_4, r_7	nop
10	(p1)	mult r_2, r_1, r_8	nop
11			mult r_1, r_9, r_2
12		nop	
13			
14		sw resul, r_1	nop

Tabla 28. Código VLIW con carga especulativa

Inst.		slot 1	slot 2
1		add r_3, r_2, r_1	lw.s $r_7, \text{dato, compensa}$
2		add r_4, r_5, r_3	nop
3	p1	cmp.ne r_4, r_0	sub r_9, r_1, r_4
5	(p1)	add r_8, r_4, r_7	nop
6	(p1)	mult r_2, r_1, r_8	nop
7		nop	mult r_1, r_9, r_2
8		sw resul, r_1	nop

El número de ciclos que se tardaría en emitir las instrucciones VLIW se muestra en la Tabla 29, donde se han tenido en cuenta los ciclos de espera de la carga y de la multiplicación entre instrucciones dependientes. Como se puede ver, se consigue la emisión de todas las instrucciones en once ciclos, tres ciclos menos que en el caso que no utilizaba la carga especulativa.

No obstante, hay que tener en cuenta que si se produce una excepción y la carga especulativa falla pero $r_4 \neq 0$ y, por lo tanto $p1=1$, el resultado de r_7 no sería correcto. Con el procedimiento descrito en el problema, la excepción que se produciría al ejecutarse `lw.s` no se atendería y se marca el registro r_7 . Esto hace que también se marquen los registros r_8 , r_2 , y r_1 , y cuando se ejecute el acceso a memoria en la instrucción 8 de la Tabla 28, al intentar almacenar en memoria el registro marcado r_1 , se atenderá la excepción y se ejecutarán el código de compensación que permite obtener el resultado correcto en los registros. Es decir:

compensa:	lw	r7, dato
	add	r8, r4, r7
	mult	r2, r1, r8
	mult	r1, r9, r2
	sw	resul, r1

Tabla 29. Ciclos en emitirse el código VLIW de la Tabla 28

Ciclo		slot 1	slot 2
1		add r3, r2, r1	lw.s r7, dato, compensa
2		add r4, r5, r3	nop
3	p1	cmp.ne r4, r0	sub r9, r1, r4
4			
5			
6	(p1)	add r8, r4, r7	nop
7	(p1)	mult r2, r1, r8	nop
8			
9		nop	
10			mult r1, r9, r2
11		sw resul, r1	nop

Los ciclos necesarios para emitir estas instrucciones (teniendo en cuenta la dependencia entre operaciones) se proporcionan en la Tabla 30.

Por tanto, el tiempo necesario para completar la emisión del código cuando hay excepción en el acceso a memoria especulativo sería igual a la suma del tiempo para el código optimizado (11 ciclos) más la penalización correspondiente al tiempo del código de compensación (11 ciclos). Es decir 22 ciclos.

Tabla 30. Ciclos en emitirse el código VLIW de compensación

Ciclo		slot 1	slot 2
1	lw	r7, dato	nop
2			
3			
4			
5			
6	add	r8, r4, r7	nop
7	mult	r2, r1, r8	nop
8			
9	mult	r1, r9, r2	nop
10			
11	sw	resul, r1	nop

Por tanto, el tiempo necesario para completar la emisión del código cuando hay excepción en el acceso a memoria especulativo sería igual a la suma del tiempo para el código optimizado (11 ciclos) más la penalización correspondiente al tiempo del código de compensación (11 ciclos). Es decir 22 ciclos.

Para que sea mejor utilizar la alternativa especulativa que la inicial (14 ciclos) debería ocurrir que:

$$14 \geq p \times 22 + (1 - p) \times 11$$

donde p es la probabilidad de que se produzca una excepción. Así, despejando

$$p \leq \frac{3}{11} = 0.2727$$

Por tanto, para que resulte beneficiosa la alternativa, debe ocurrir que en más del 72.7% de los casos ($1-p=0.727$) no se produzcan excepciones.



5. Bibliografía

- Julio Ortega Lopera, Mancia Anguita López, Alberto Prieto Espinosa, 2005, "Arquitectura de Computadores". Thomson.
- Julio Ortega Lopera, Jesús López Peñalver, 2008, "Problemas de Ingeniería de Computadores. Cien problemas resueltos de Procesadores Paralelos". Editorial Copicentro.