Jiuxe
www.wuolah.com/student/Jiuxe

3425

AC_ejercicios_tema4_resueltos.pdf

Ejercicios Resueltos

2° Arquitectura de Computadores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

2º curso / 2º cuatr.

Grado en Ing. Informática

Arquitectura de Computadores. Ejercicios y Cuestiones

Tema 4. Arquitecturas con Paralelismo a nivel de Instrucción (ILP)

Material elaborado por los profesores responsables de la asignatura:
Julio Ortega, Mancia Anguita

1 Ejercicios

Ejercicio 1. Para el fragmento de código siguiente:

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

y suponiendo que se pueden captar, decodificar, 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

b) Una ventana de instrucciones centralizada con emisión desordenada

c) Una estación de reserva de tres líneas para cada unidad funcional, con envío ordenado.

Nota: considere que hay una unidad funcional para la carga (2 ciclos), otra para el almacenamiento (1 ciclo), tres para la suma/resta (1 ciclo), y una para la multiplicación (4 ciclos). También puede considerar que, en la práctica, no hay límite para el número de instrucciones que pueden almacenarse en la ventana de instrucciones o en el buffer de instrucciones.

Solución

A continuación se muestran la evolución temporal de las instrucciones en sus distintas etapas para cada una de las alternativas que se piden. En esas figuras se supone que tras la decodificación de una instrucción, ésta puede pasar a ejecutarse sin consumir ciclos de emisión si los operandos y la unidad funcional que necesita están disponibles. Por esta razón, no se indicarán explícitamente los ciclos dedicados a la emisión en los casos (a) y (b), y al envío en el caso (c)

a) Emisión ordenada con ventana centralizada

Como se muestra la tabla correspondiente, las instrucciones se empiezan a ejecutar en orden respetando las dependencias de datos o estructurales que existan. Por ejemplo la segunda instrucción de acceso a memoria lw debe esperar que termine la instrucción lw anterior porque solo hay una unidad de carga de memoria. La

2x1

carné universitario

FOSTER'S HOLLYWOOD

*Consulta las condiciones de la promoción en fostershollywood2x1universitario.com

emisión de la instrucción `add r3,r3,r4` debe esperar a que termine la ejecución de la instrucción de multiplicación que la precede.

La instrucción `add r3,r0,r0` debe esperar a que se hayan emitido las instrucciones que la preceden a pesar de que no depende de ellas y podría haberse emitido en el ciclo 3, si la emisión fuese desordenada.

También debe comprobarse que no se emiten más de cuatro instrucciones por ciclo. En este caso, como mucho, se emiten tres instrucciones, en el ciclo 11.

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

b) Emisión desordenada con ventana centralizada

En este caso, hay que respetar las dependencias de datos y las estructurales. En cuanto las instrucciones tengan una unidad disponible y los datos que necesitan se pueden emitir, siempre que no sean menos de cuatro intrucciones por ciclo las que se emitan. En este caso, como mucho se emiten tres instrucciones en los ciclos 4 y 5

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

c) Estación de reserva con tres líneas para cada unidad funcional, y envío ordenado.

En esta alternativa, se ha supuesto que las instrucciones decodificadas se emiten a una estación de reserva desde la que se accede a la unidad funcional correspondiente enviando las instrucciones de forma ordenada (por eso las dos instrucciones sw se ejecutan ordenadamente). La asignación de instrucciones de suma o resta a las tres estaciones de reserva que existen para suma/resta se ha hecho de forma alternativa pero

tratando de minimizar tiempos. Es decir, se ha asignado a una estación de reserva que no tuviera ocupada su unidad funcional correspondiente para obtener los tiempos más favorables.

ESTACIÓN DE RESERVA	INSTRUCCIÓN	1	2	3	4	5	6	7	8	9	10	11	12	13
LW	*lw r1, 0x1ac*	IF	ID	EX										
LW	*lw r2, 0xc1f*	IF	ID			EX								
ADD(1)	*add r3, r0, r0*	IF	ID	EX										
MULT(1)	*mul r4, r2, r1*	IF	ID					EX						
ADD(2)	*add r3, r3, r4*		IF	ID								EX		
ADD(3)	*add r5, r0, 0x1ac*		IF	ID	EX									
ADD(1)	*add r6, r0, 0xc1f*		IF	ID	EX									
ADD(3)	*sub r5, r5, #4*		IF	ID		EX								
ADD(1)	*sub r6, r6, #4*			IF	ID	EX								
SW	*sw (r5), r3*			IF	ID								EX	
SW	*sw (r6), r4*			IF	ID									EX

Ejercicio 2. Considere que el fragmento de código siguiente:

```
1. lw      r3,0x10a      ; r3 ← M(0x10a)
2. addi    r2,r0,#128    ; r2 ← r0+128
3. add     r1,r0,0x0a    ; r1 ← r0+0x0a
4. lw      r4,0(r1)      ; r4 ← M(r1)
5. lw      r5,-8(r1)     ; r5 ← M(r1-8)
6. mult    r6,r5,r3      ; r6 ← r5*r3
7. add     r5,r6,r3      ; r5 ← r6+r3
8. add     r6,r4,r3      ; r6 ← r4+r3
9. sw      0(r1),r6      ; M(r1) ← r6
10.sw      -8(r1),r5     ; M(r1-8) ← r5
11.sub     r2,r2,#16     ; r2 ← r2-16
```

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 centralizada 2 instrucciones/ciclo; de escribir hasta 2 resultados/ciclo en los registros correspondientes (registros de reorden, o registros de la arquitectura según el caso), y completar (o retirar) hasta 3 instrucciones/ciclo.

Indique el número de ciclos que tardaría en ejecutarse el programa suponiendo finalización ordenada y:

a) Emisión ordenada

b) Emisión desordenada

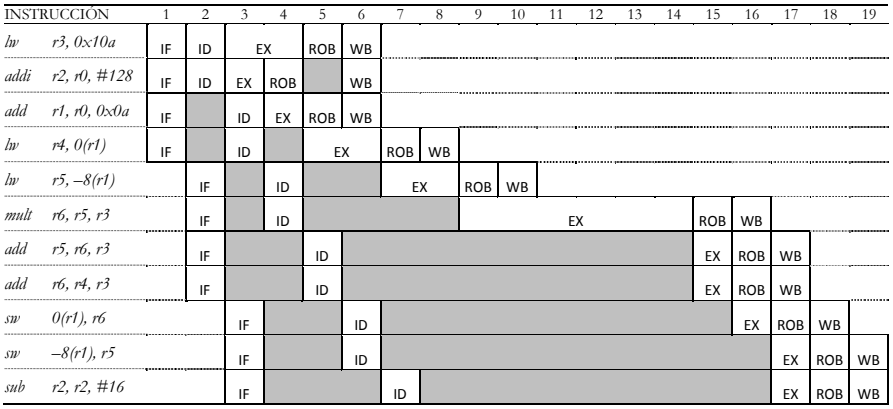
Nota: Considere que tiene una unidad funcional de carga (2 ciclos), una de almacenamiento (1 ciclo), tres unidades de suma/resta (1 ciclo), y una de multiplicación (6 ciclos), y que no hay limitaciones para el número de líneas de la cola de instrucciones, ventana de instrucciones, buffer de reorden, puertos de lectura/escritura etc.)

Solución

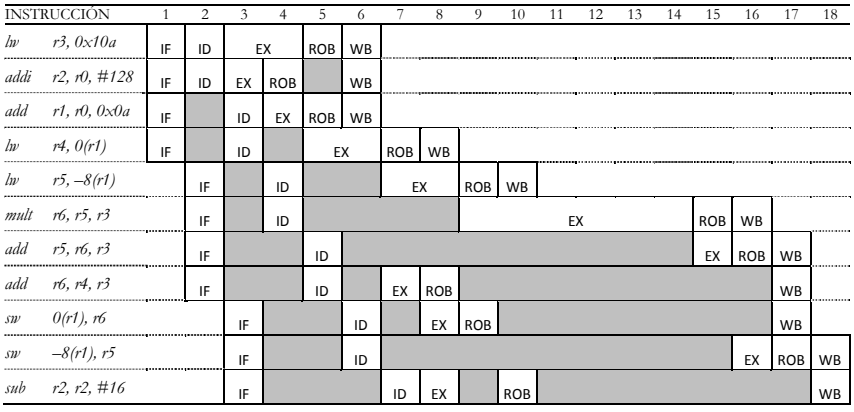
a) En la emisión ordenada los instantes en los que las instrucciones empiezan a ejecutarse (etapa EX) deben estar ordenados de menor a mayor, a diferencia de lo que ocurre en la emisión desordenada. Se supone que

el procesador utiliza un buffer de reordenamiento (ROB) para que la finalización del procesamiento de las instrucciones sea ordenada. Por ello, las etapas WB de las instrucciones (momento en que se retiran las instrucciones del ROB y se escriben en los registros de la arquitectura) deben estar ordenadas (tanto en el caso de emisión ordenada como desordenada). La etapa marcada como ROB es la que corresponde a la escritura de los resultados en el ROB (se ha supuesto que las instrucciones de escritura en memoria también consumen un ciclo de escritura en el ROB y un ciclo de escritura WB en el banco de registros, aunque estos ciclos se podrían evitar para estas instrucciones).

También tiene que comprobarse que no se decodifican, emiten, ni escriben en el ROB más de dos instrucciones por ciclo, ni se retiran más de tres instrucciones por ciclo.



b) En el caso de emisión desordenada, la traza de ejecución de las instrucciones es la siguiente:



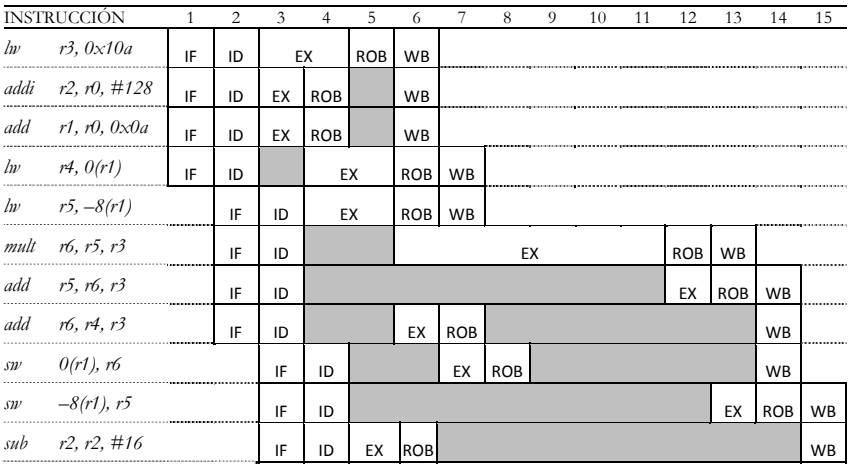
También en este caso hay que tener en cuenta que no se pueden decodificar, emitir, ni escribir en el ROB más de dos instrucciones por ciclo (obsérvese que la instrucción sw r2,r2,#16 debe esperar un ciclo para su etapa ROB por esta razón), ni se pueden retirar más de tres instrucciones por ciclo.

Ejercicio 3. En el problema anterior, (a) indique qué mejoras realizaría en el procesador para reducir el tiempo de ejecución en la mejor de las opciones sin cambiar el diseño de las unidades funcionales (multiplicador, sumador, etc.) y sin cambiar el tipo de memorias ni la interfaz entre procesador y memoria

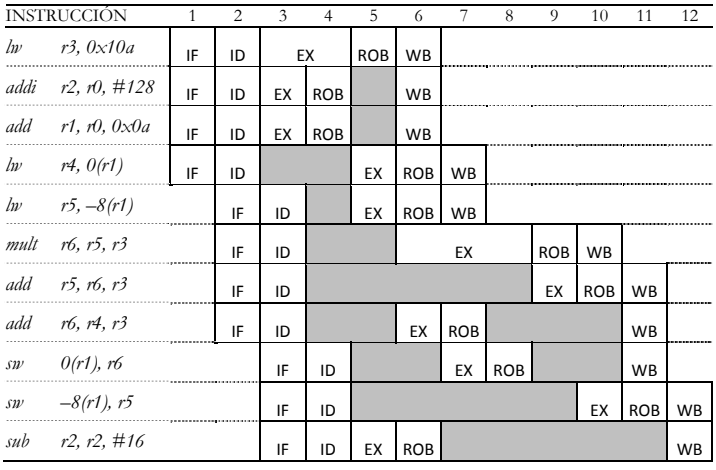
(no varía el número de instrucciones captadas por ciclo). (b) ¿Qué pasaría si además se reduce el tiempo de multiplicación a la mitad?.

Solución

(a) En primer lugar, se considera que se decodifican el mismo número de instrucciones que se captan, ya que no existen limitaciones impuestas por las instrucciones al ritmo de decodificación (éste viene determinado por las posibilidades de los circuitos de descodificación y la capacidad para almacenar las instrucciones decodificadas hasta que se emitan). También se considera que no existen limitaciones para el número de instrucciones por ciclo que se emiten, escriben el ROB, y se retiran. Por último, se consideran que están disponibles todas las unidades funcionales que se necesiten para que no haya colisiones (riesgos estructurales).



Teniendo en cuenta las características de la traza, si se redujese el tiempo de la multiplicación a la mitad (es decir tres ciclos) el tiempo de ejecución de dicha traza se reduciría también en esos mismos tres ciclos ya que todas las instrucciones que siguen a la multiplicación se encuentran esperando a que termine esta para poder proseguir. Es decir, la operación de multiplicación es el cuello de botella en este caso.



Por lo tanto se tendrían 12 ciclos. Si se tiene en cuenta que se tienen 11 instrucciones, que el tiempo mínimo que tarda la primera instrucción en salir son 6 ciclos (lo tomamos como tiempo de latencia de inicio del cauce), y que el tiempo total de ejecución en este caso es de 12 ciclos, se puede escribir:

$$T(n) = 12 = \text{TLI} + (n-1) \times \text{CPI} = 6 + (11-1) \times \text{CPI}$$

Y, si se despeja, se tiene que el procesador superescalar presenta una media de 0.6 ciclos por instrucción, o lo que es lo mismo, ejecuta 1.67 instrucciones por ciclo. Tiene un comportamiento superescalar, pero está muy lejos de las tres instrucciones por ciclo que pueden terminar como máximo.

Ejercicio 4. En el caso descrito en el problema 3, indique cómo evolucionaría el buffer de reorden, utilizado para implementar finalización ordenada, en la mejor de las opciones.

Solución

La Tabla que se proporciona a continuación muestra la evolución del buffer de reordenamiento (ROB) marcando en negrita los cambios que se producen en cada ciclo.

- En el ciclo 2 se decodifican las instrucciones (1) – (4) y se introducen en el ROB.
- En el ciclo 3 se decodifican las instrucciones (5) – (8) y se introducen en el ROB.
- En el ciclo 4 se decodifican las instrucciones (9) – (11) y se introducen en el ROB. Simultáneamente se almacena en el ROB los resultados de las instrucciones (2) y (3), cuya ejecución finalizó en el ciclo anterior.
- En el ciclo 5 se escribe en el ROB el resultado de la instrucción (1).
- En el ciclo 6 se retiran las tres primeras instrucciones y se escriben los resultados de las instrucciones (4), (5) y (11) en el ROB.
- En el ciclo 7 se retiran las instrucciones (4) y (5) y se escribe en el ROB el resultado de la instrucción (8).
- En el ciclo 8 se activa el bit valor válido de la instrucción (9) en el ROB, indicando que la instrucción de almacenamiento ya ha escrito en memoria.
- En el ciclo 9 se escribe el resultado de la instrucción (6) en el ROB.
- En el ciclo 10 se retira la instrucción (6) y se escribe el resultado de la instrucción (7) en el ROB.
- En el ciclo 11 se retiran las instrucciones (7), (8) y (9) y se indica que la instrucción (10) ya ha escrito en memoria.
- En el ciclo 12 se retiran las dos últimas instrucciones del ROB.

CICLO	#	CÓDIGO OPERACIÓN	REGISTRO DESTINO	VALOR	VALOR VÁLIDO
2	**0**	***Lw***	**r3**	**–**	**0**
	1	***Addi***	**r2**	**–**	**0**
	2	***Add***	**r1**	**–**	**0**
	3	***Lw***	**r4**	**–**	**0**

CICLO	#	CÓDIGO OPERACIÓN	REGISTRO DESTINO	VALOR	VALOR VÁLIDO
3	0	*Lw*	r3	–	0
	1	*Addi*	r2	–	0
	2	*Add*	r1	–	0
	3	*Lw*	r4	–	0
	4	***Lw***	**r5**	**–**	**0**
	5	***Mult***	**r6**	**–**	**0**
	6	***Add***	**r5**	**–**	**0**
	7	***Add***	**r6**	**–**	**0**
4	0	*Lw*	r3	–	0
	1	***Addi***	**r2**	**128**	**1**
	2	***Add***	**r1**	**0x0a**	**1**
	3	*Lw*	r4	–	0
	4	*Lw*	r5	–	0
	5	*Mult*	r6	–	0
	6	*Add*	r5	–	0
	7	*Add*	r6	–	0
	8	***Sw***	**–**	**–**	**0**
	9	***Sw***	**–**	**–**	**0**
	10	***Sub***	**r2**	**–**	**0**
5	**0**	***Lw***	**r3**	**[0x1a]**	**1**
	1	*Addi*	r2	128	1
	2	*Add*	r1	0x0a	1
	3	*Lw*	r4	–	0
	4	*Lw*	r5	–	0
	5	*Mult*	r6	–	0
	6	*Add*	r5	–	0
	7	*Add*	r6	–	0
	8	*Sw*	–	–	0
	9	*Sw*	–	–	0
	10	*Sub*	r2	–	0
6	**3**	***Lw***	**r4**	**[0x0a]**	**1**
	4	***Lw***	**r5**	**[0x0a – 8]**	**1**
	5	*Mult*	r6	–	0
	6	*Add*	r5	–	0
	7	*Add*	r6	–	0
	8	*Sw*	–	–	0
	9	*Sw*	–	–	0
	10	***Sub***	**r2**	**112**	**1**

CICLO	#	CÓDIGO OPERACIÓN	REGISTRO DESTINO	VALOR	VALOR VÁLIDO
7	5	*mult*	r6	–	0
	6	*add*	r5	–	0
	7	***add***	**r6**	**r4 + r3**	**1**
	8	*sw*	–	–	0
	9	*sw*	–	–	0
	10	*sub*	r2	112	1
8	5	*mult*	r6	–	0
	6	*add*	r5	–	0
	7	*add*	r6	r4 + r3	1
	8	***sw***	**–**	**–**	**1**
	9	*sw*	–	–	0
	10	*sub*	r2	112	1
9	**5**	***mult***	**r6**	**r5 × r3**	**1**
	6	*add*	r5	–	0
	7	*add*	r6	r4 + r3	1
	8	*sw*	–	–	1
	9	*sw*	–	–	0
	10	*sub*	r2	112	1
10	**6**	***add***	**r5**	**r6 + r3**	**1**
	7	*add*	r6	r4 + r3	1
	8	*sw*	–	–	1
	9	*sw*	–	–	0
	10	*sub*	r2	112	1
11	**9**	***sw***	**–**	**–**	**1**
	10	*sub*	r2	112	1

Ejercicio 5. En un procesador superescalar con renombramiento de registros se utiliza un buffer de renombramiento para implementar el mismo. Indique como evolucionarían los registros de renombramiento al realizar el renombramiento para las instrucciones:

```
1. mul r2,r0,r1     ; r2 ← r0*r1
2. add r3,r1,r2     ; r3 ← r1+r2
3. sub r2,r0,r1     ; r2 ← r0-r1
4. add r3,r3,r2     ; r3 ← r3+r2
```

Solución

Las instrucciones se renombra de forma ordenada, aunque podrían emitirse desordenadamente a medida que estén disponibles los operandos y las unidades funcionales. La estructura típica de un buffer de renombramiento con acceso asociativo podría ser la que se muestra a continuación, en la que se ha supuesto que se han hecho las asignaciones (renombrados) de r0 y r1 (las líneas 0 y 1 tienen valores de "Entrada válida" a 1 correspondiente a los registros r0 y r1, con sus valores válidos correspondientes y los bits de último a 1 indicando que han sido los últimos renombramientos de esos registros)

:

#	Entrada válida	Registo destino	Valor	Valor Válido	Bit de Último
0	1	r0	[r0]	1	1
1	1	r1	[r1]	1	1
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				
8	0				

Al decodificarse la instrucción `mul r2,r0,r1` los valores de r0 y r1 se tomarán de las líneas 0 y 1 del buffer de renombrado y se renombrará r2. La instrucción se puede emitir para su ejecución y el resultado de la misma se almacenará en la línea 2 del buffer de renombrado. El contenido del campo "Valor válido" de esta línea será igual a 0 hasta que el resultado no se haya obtenido

#	Entrada válida	Registo destino	Valor	Valor Válido	Bit de Último
0	1	r0	[r0]	1	1
1	1	r1	[r1]	1	1
2	1	r2		0	1
3	0				
4	0				
5	0				
6	0				
7	0				
8	0				

Después se considera la instrucción `add r3,r1,r2`, como r2 no está disponible no se podrá empezar a ejecutar, pero se utilizará una marca en la estación de reserva indicando que se espera el valor que se escribirá en la línea 2. También se hace el renombrado de r3. El valor de r1 sí se puede tomar de la línea 1.

#	Entrada válida	Registo destino	Valor	Valor Válido	Bit de Último
0	1	r0	[r0]	1	1
1	1	r1	[r1]	1	1
2	1	r2		0	1
3	1	r3		0	1
4	0				
5	0				
6	0				
7	0				
8	0				

Cuando se decodifica la instrucción `sub r2,r0,r1`, se realiza otro renombramiento de r2. Como r0 y r1 tienen valores válidos, esta instrucción podría empezar a ejecutarse si existe una unidad funcional disponible. El resultado de esta operación se almacenará en la línea 4 (último renombrado de r2) y de ahí se tomarán los operandos r2 en instrucciones posteriores (para eso se utiliza el "Bit de Último"):

#	Entrada válida	Registo destino	Valor	Valor Válido	Bit de Último
0	1	r0	[r0]	1	1
1	1	r1	[r1]	1	1
2	1	r2		0	0
3	1	r3		0	1
4	1	r2		0	1
5	0				
6	0				
7	0				
8	0				

La última instrucción, `add r3,r3,r2`, origina otro renombrado de r3. Sus operandos r3 y r2 se tomarían de las líneas 3 y 4 respectivamente porque son los últimos renombrados que se han hecho para r3 y r2. Como no tienen valores válidos, se utilizarán las marcas correspondientes a dichas líneas 3 y 4 del buffer para que se escriban los resultados correspondientes en la línea de la ventana de instrucciones donde se almacena esta instrucción hasta que pueda emitirse.

#	Entrada válida	Registo destino	Valor	Valor Válido	Bit de Último
0	1	r0	[r0]	1	1
1	1	r1	[r1]	1	1
2	1	r2		0	0
3	1	r3		0	0
4	1	r2		0	1
5	1	r3		0	1
6	0				
7	0				
8	0				

Cuando termine `mul r2,r0,r1,` se escribirá el resultado em la línea 1 y podrá continuar `add r3,r1,r2`, y cuando termine `sub r2,r0,r1`, se escribirá el resultado en la línea 4. Cuando termine `add r3,r1,r2`, se escribirá el resultado en la línea 3 y podrá continuar `add r3,r3,r2`.

Ejercicio 6. Considere el bucle:

```
i=1
do
{
  b[i]=a[i]*c;
  c=c+1;
  if (c>10) then goto etiqueta;
  i=i+1;
} while (i<=10);
etiqueta:......
```

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 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 no)

c) Predicción dinámica con un bit (1= Saltar; 0 = No Saltar; Inicialmente está a 1)

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

Solución

Al traducir el fragmento de código a ensamblador se codificarán dos saltos: S_1, que estará dentro del bucle y saltará hacia *etiqueta*, y S_2, que será el salto al final del bucle para iniciar otra iteración. Dependiendo del valor inicial de *c* se pueden distinguir tres comportamientos diferentes de los saltos:

1. $c \leq 0$: El salto S_1 no salta nunca y el salto S_2 salta 9 veces (el bucle realiza 10 iteraciones, el máximo de iteraciones del bucle). Así pues, si N significa que no salta y S que sí se salta, y los subíndices 1 y 2 hacen referencia a las instrucciones de salto primera (instrucción dentro del bucle) y segunda (instrucción de salto al final del bucle), respectivamente:

$$(N_1 S_2)^9 N_1 N_2$$

2. $1 \leq c < 10$:

   - Si $c = 9$: $N_1 S_2 S_1$
   - Si $c = 8$: $(N_1 S_2)^2 S_1$
   - ...
   - Si $c = 1$: $(N_1 S_2)^9 S_1$

   Es decir: $(N_1 S_2)^{10-c} S_1$

3. $c \geq 10$: S_1

   ya que no se realizaría ninguna iteración completa, al verificarse la condición de salto de la instrucción condicional que hay dentro del bucle.

Una vez obtenidos los comportamientos de los saltos correspondientes a los distintos valores posibles de *c*, se pasa a considerar cada uno de los esquemas de predicción indicados. Para el caso de un predictor fijo que esté diseñado para predecir que siempre se va a tomar el salto tendríamos las siguientes penalizaciones en función del valor de *c*:

1. $c \leq 0$: Como el comportamiento de la secuencia de saltos es $(N_1 S_2)^9 N_1 N_2$ y la predicción es $(S_1 S_2)^{10}$, tendríamos un fallo por cada salto que no se tome, es decir:

$$P_{\text{fijo}_1} = F_{\text{fijo}_1} \times P = (9+1+1) \times 4 = 44 \text{ ciclos}$$

2. $1 \leq c < 10$: En este caso, el comportamiento es $(N_1 S_2)^{10-c} S_1$, la predicción sigue -siendo saltar siempre, por lo que tendríamos una penalización de:

$$P_{\text{fijo}_2} = F_{\text{fijo}_2} \times P = (10-c) \times 4 \text{ ciclos}$$

3. $c \geq 10$: En este caso, como el comportamiento de la secuencia es S_1 y la predicción será siempre saltar, no se producirá ningún fallo, por lo que:

$$P_{\text{fijo}_3} = F_{\text{fijo}_3} \times P = 0 \times 4 = 0 \text{ ciclos}$$

En el segundo apartado del problema se nos pide que estimemos la penalización en el caso de usar un predictor estático que suponga que los saltos hacia atrás se tomarán siempre y los saltos hacia adelante no se tomarán nunca. Dependiendo del valor de *c* tendremos:

1. *c* ≤ 0: El comportamiento de los saltos sigue siendo $(N_1S_2)^9N_1N_2$, pero en este caso la predicción es $(N_1S_2)^{10}$. Por tanto, tendemos un fallo en la última ejecución del segundo salto. La penalización será de:

$$P_{\text{estático}_1} = F_{\text{estático}_1} \times P = 1 \times 4 = 4 \text{ ciclos}$$

2. 1 ≤ *c* < 10: En este caso, el comportamiento es $(N_1S_2)^{10-c}S_1$, por lo que se cometería un fallo cuando el salto S_1 salte hacia adelante:

$$P_{\text{estático}_2} = F_{\text{estático}_2} \times P = 1 \times 4 = 4 \text{ ciclos}$$

3. *c* ≥ 10: En este caso, como el comportamiento de la secuencia es S_1 y el predictor asume que los saltos hacia adelante no se toman, se cometerá un único fallo, por lo que la penalización será:

$$P_{\text{estático}_3} = F_{\text{estático}_3} \times P = 1 \times 4 = 4 \text{ ciclos}$$

Por consiguiente, para cualquier valor de *c*, este esquema de predicción siempre da lugar a 4 ciclos de penalización.

El último esquema de predicción que se propone en este ejercicio es un predictor dinámico con un bit de historia. En este caso, al tratarse de un esquema dinámico, habrá un predictor para cada salto, cuyo bit de historia estará inicializado a 1, e irá cambiando a 1 ó 0 en función de la última ejecución del salto (1 si saltó y 0 si no lo hizo). En función del valor de *c* tendremos:

1. *c* ≤ 0: El comportamiento $(N_1S_2)^9N_1N_2$, se debe separar para cada salto, ya que cada salto tendrá un bit de historia independiente.

   - Para el primer salto tendremos el comportamiento $(N_1)^{10}$ y la predicción $S_1(N_1)^9$, ya que como los bits de historia están inicializados a 1, la primera vez se predecirá saltar y se fallará la predicción. El resto de las predicciones se realizarán en función de la última ejecución del salto, y como nunca salta, se predecirán todas correctamente.

   - Para el segundo salto tendremos el comportamiento $(S_2)^9N_2$ y la predicción $(S_2)^{10}$, ya que el bit de historia comenzará inicializado a cero y el salto se tomará todas las veces menos la última.

   Por tanto, la penalización total será de:

$$P_{\text{dinámico}_1} = (F_1 + F_2) \times P = (1+1) \times 4 = 8 \text{ ciclos}$$

2. 1 ≤ *c* < 10:

Para el primer salto tendremos el comportamiento $(N_1)^{10-c}S_1$ y la predicción $S_1(N_1)^{10-c}$. Por tanto, se cometerán dos fallos, en el primer y en el último salto de la secuencia.

Para el segundo salto tendremos el comportamiento $(S_2)^{10-c}$, que coincide con la predicción, por lo que no se cometerá ningún fallo para este salto.

Por tanto, la penalización total será de:

$$P_{\text{dinámico}_2} = (F_1 + F_2) \times P = (2+0) \times 4 = 8 \text{ ciclos}$$

3. $c \geq 10$: Como el comportamiento de la secuencia es S_1, el predictor del segundo salto no se llega a usar, y como el predictor del primero está inicializado a 1, acertará la predicción, por lo que no se producirá ningún fallo. Por tanto:

$$P_{\text{dinámico}_3} = (0+0) \times P = 0 \text{ ciclos}$$

Ejercicio 7. En la situación descrita en el problema anterior. ¿Cuál de los tres esquemas es más eficaz por término medio si hay un 25% de probabilidades de que c sea menor o igual a 0, un 30% de que sea mayor o igual a 10; y un 45% de que sea cualquier número entre 1 y 9, siendo todos equiprobables?

Solución

En el ejercicio 6 se ha determinado la penalización de la secuencia para cada tipo de predictor, en este ejercicio de trata de utilizar la distribución de probabilidad de la variable *c* que se da en el enunciado para evaluar la penalización media de cada predictor, y por lo tanto cuál es el mejor para este código y estas circunstancias. Para el caso del predictor fijo, y teniendo en cuenta que todos los valores de *c* entre 1 y 9 son equiprobables, tenemos:

$$P_{\text{fijo}} = 0.25 \times P_{\text{fijo}_1} + 0.45 \times P_{\text{fijo}_2} + 0.30 \times P_{\text{fijo}_3} = 0.25 \times 44 + 0.45 \times \left(\frac{\sum_{c=1}^{9} (10-c) \times 4}{9} \right) + 0.30 \times 0 = 20 \text{ ciclos}$$

Para el predictor estático tenemos:

$$P_{\text{estático}} = 0.25 \times P_{\text{estático}_1} + 0.45 \times P_{\text{estático}_2} + 0.30 \times P_{\text{estático}_3} = 0.25 \times 4 + 0.45 \times 4 + 0.30 \times 4 = 4 \text{ ciclos}$$

Y para el predictor dinámico:

$$P_{\text{dinámico}} = 0.25 \times P_{\text{dinámico}_1} + 0.45 \times P_{\text{dinámico}_2} + 0.30 \times P_{\text{dinámico}_3} = 0.25 \times 8 + 0.45 \times 8 + 0.30 \times 0 = 5.6 \text{ ciclos}$$

Como se puede ver, para este bucle, y para la distribución de probabilidad de los valores de *c*, el esquema de predicción más eficiente corresponde a la predicción estática.

En cualquier caso, esta situación depende de las probabilidades de los distintos valores de *c*. Si, por ejemplo, la probabilidad de que *c* esté entre 1 y 9 fuera de 0.15, la penalización para la predicción dinámica con 1 bit

de historia sería de 3.2 ciclos, mientras que la predicción estática seguiría presentando una penalización de 4 ciclos.

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

SSNNNSSNSNSNSSSSSN

donde S indica que se produce el salto y N que no. Indique la penalización efectiva que se introduce si se utiliza:

a) Predicción fija (siempre se considera que se no se va a producir el salto)

b) Predicción estática (si el desplazamiento es negativo se toma y si es positivo no)

c) Predicción dinámica con dos bits, inicialmente en el estado (11).

d) Predicción dinámica con tres bits, inicialmente en el estado (111).

Nota: La penalización por saltos incorrectamente predichos es de 5 ciclos y para los saltos correctamente predichos es 0 ciclos.

Solución

En el caso de usar predicción fija, se produciría un fallo del predictor cada vez que se tome el salto, tal y como muestra la Tabla siguiente. Por tanto, la penalización total sería de:

$$P_{\text{fijo}} = F_{\text{fijo}} \times P = 11 \times 5 = 55 \text{ ciclos}$$

PREDICCIÓN FIJA	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
EJECUCIÓN	S	S	N	N	N	S	S	N	S	N	S	N	S	S	S	S	S	N
PENALIZACIÓN	P	P				P	P		P		P		P	P	P	P	P	

Si se usa el predictor estático, como la dirección de destino del salto es anterior, se producirá un fallo en la predicción cuando no se produzca el salto, tal y como muestra la tabla correspondiente. Por tanto, la penalización total sería de:

$$P_{\text{estático}} = N_{\text{estático}} \times P = 7 \times 5 = 35 \text{ ciclos}$$

PREDICCIÓN ESTÁTICA	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
EJECUCIÓN	S	S	N	N	N	S	S	N	S	N	S	N	S	S	S	S	S	N
PENALIZACIÓN			P	P	P			P		P		P						P

En cuanto al predictor dinámico con dos bits de historia, la tabla siguiente muestra el estado del predictor antes de ejecutar el salto, la predicción que realiza el predictor, el comportamiento del salto y si se produce o no penalización. Teniendo en cuanta los fallos que se producen, la penalización total sería de:

$$P_{2\,\text{bits}} = F_{2\,\text{bits}} \times P = 11 \times 5 = 55\ \text{ciclos}$$

ESTADO	11	11	11	10	01	00	01	10	01	10	01	10	01	10	11	11	11	11
PREDICCIÓN DINÁMICA (2 BITS)	S	S	S	S	N	N	N	S	N	S	N	S	N	S	S	S	S	S
EJECUCIÓN	S	S	N	N	N	S	S	N	S	N	S	N	S	S	S	S	S	N
PENALIZACIÓN			P	P		P	P	P	P	P	P	P	P					P

Por último, para el predictor dinámico de tres bits, en la tabla siguiente se indican los bits de historia que existen antes de que se ejecute la instrucción, la predicción que determinan esos bits, y lo que finalmente ocurre (según se indica en la secuencia objeto del problema). La última fila indica si ha habido penalización (no coinciden la predicción y lo que ocurre al final). Teniendo en canta esta información, tenemos que la penalización total es de:

$$P_{3\,\text{bits}} = F_{3\,\text{bits}} \times P = 10 \times 5 = 50\ \text{ciclos}$$

ESTADO	111	111	111	011	001	000	100	110	011	101	010	101	010	101	110	111	111	111
PREDICCIÓN DINÁMICA (3 BITS)	S	S	S	S	N	N	N	S	S	S	N	S	N	S	S	S	S	S
EJECUCIÓN	S	S	N	N	N	S	S	N	S	N	S	N	S	S	S	S	S	N
PENALIZACIÓN			P	P		P	P	P		P	P	P	P					P

Como se puede ver, en la secuencia de ejecuciones de la instrucción de salto considerada en este problema, el mejor esquema de predicción es la predicción estática que aquí se utiliza. El esquema de predicción de salto dinámico con dos bits es igual de bueno (o malo) que la predicción fija.

Así, se puede indicar que la eficacia de un esquema de salto depende del perfil de saltos a que de lugar la correspondiente instrucción de salto condicional. En la práctica, los esquemas de predicción dinámica suelen funcionar mejor que los de predicción estática porque las instrucciones de salto suelen repetir el comportamiento de su ejecución previa. En ese sentido, la secuencia utilizada en este problema es bastante atípica.

Ejercicio 9. Indique cómo transformaría la secuencia de instrucciones que se muestran en la Tabla un core de procesamiento de forma que sólo se utilicen instrucciones de movimiento de datos condicionales (no debe haber ninguna instrucción de salto condicional).

#	OP1	OP2
1	*lw r1, x(r2)*	*add r10, r11, r12*
2		*add r13, r10, r14*

3	beqzr3, direc	
4	lw r4, 0(r3)	
5	lw r5, 0(r4)	

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:

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

En esta secuencia de instrucciones, 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. 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      ; Fijamos r6 a una dirección segura
lw      r1, x(r2)
mov     r7, r4             ; Guardamos el contenido original de r4 en r7
cmovnz  r6, r3, r3         ; Movemos r3 a r6 si r3 es distinto de cero
lw      r4, 0(r6)          ; Carga especulativa
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
```

donde *r6* y *r7* son registros auxiliares. En *r6* se carga primero una dirección segura, 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.

Es posible evitar la instrucción de salto si se utilizan cargas especulativas para las dos instrucciones de carga protegidas por el salto en el código inicial. En este caso el código sería el siguiente (r0 se supone a 0):

```
addi    r6, r0, #1000      ; Fijamos r6 a una dirección segura
lw      r1, x(r2)
mov     r7, r4             ; Guardamos el contenido original de r4 en r7
mov     r8, r5             ; Guardamos r5 en otro registro temporal r8
cmovnz  r6, r3, r3         ; Movemos r3 a r6 si r3 es distinto de cero
lw      r4, 0(r6)          ; Carga especulativa
lw      r5, 0(r4)          ; Esta carga también es especulativa
cmovz   r4, r7, r3         ; Si r3 es 0, hay que hacer que r4 recupere su valor
cmovz   r5, r8, r3         ; Si r3 es 0 hay que hacer que r5 recupere su valor
```

Se supone 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 eliminen la instrucción de salto.

Ejercicio 10. En un procesador todas las instrucciones pueden predicarse. Para establecer los valores de los predicados se utilizan intrucciones 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, gt, eq, ne). Si la condición es verdadera p1=1 y p2=0, y si es falsa, p1=0 y p2=1. La instrucción sólo se ejecuta si el predicado p=1 (habrá sido establecido por otra instrucción de comparación).

En estas condiciones, utilice instrucciones con predicado para escribir sin ninguna instrucción de salto el siguiente código:

```
if (A>B) { X=1;}
else {if (C<D) X=2; else X=3;}
```

Solución

A continuación se muestra el código que implementa el programa anterior usando predicados (r0 se supone a 0):

```
        lw      r1, a       ; r1 = A
        lw      r2, b       ; r2 = B
p1, p2  cmp.gt  r1,r2       ; Si a > b p1 = 1 y p2 = 0 (si no, p1 = 0 y p2 = 1)
(p1)    addi    r5, r0, #1
(p1) p3 cmp.ne  r0, r0      ; Inicializamos p3 a 0 (sin utilizar p1 también funciona bien)
(p1) p4 cmp.ne  r0, r0      ; Inicializamos p4 a 0 (sin utilizar p1 también funciona bien)
(p2)    lw      r3, c       ; r3 = c (sin utilizar p2 también funciona bien)
(p2)    lw      r4, d       ; r4 = d(sin utilizar p2 también funciona bien)
(p2) p3, p4 cmp.lt  r3, r4  ; Sólo si p2 = 1 p3 o p4 pueden ser 1
(p3)    addi    r5, r0, #2  ; Se ejecuta si p3 = 1 (y p2 = 1)
(p4)    addi    r5, r0, #3  ; Se ejecuta si p4 = 1 (y p2 = 1)
        sw      r5, x       ; Almacenamos el resultado
```

2 Cuestiones

Cuestión 1. ¿Qué tienen en común un procesador superescalar y uno VLIW? ¿En qué se diferencian?

Solución

Tanto un procesador superescalar como uno VLIW son procesadores segmentados que aprovechan el paralelismo entre instrucciones (ILP) procesando varias instrucciones u operaciones escalares en cada una de ellas. La diferencia principal entre ellos es que mientras que el procesador superescalar incluye elementos hardware para realizar la planificación de instrucciones (enviar a ejecutar las instrucciones decodificadas

independientes para las que existen recursos de ejecución disponibles) dinámicamente, los procesadores superescalares se basan en el compilador para realizar dicha planificación (planificación estática).

◀

Cuestión 2. ¿Qué es un buffer de renombramiento? ¿Qué es un buffer de reordenamiento? ¿Existe alguna relación entre ambos?

Solución

Un buffer de renombramiento es un recurso presente en los procesadores superescalares que permite asignar almacenamiento temporal a los datos asignados a registros del banco de registros de la arquitectura. La asignación de registros de la arquitectura la realiza el compilador o el programador en ensamblador. Mediante la asignación de registros temporales a los registros de la arquitectura se implementa el renombramiento de estos últimos con el fin de eliminar riesgos (dependencias) WAW y WAR.

Un buffer de reordenamiento permite implementar la finalización ordenada de las instrucciones después su ejecución.

Es posible implementar el renombramiento de registros aprovechando la estructura del buffer de reordenamiento.

◀

Cuestión 3. ¿Qué es una ventana de instrucciones? ¿Y una estación de reserva? ¿Existe alguna relación entre ellas?

Solución

Una ventana de instrucciones es la estructura a la que pasan las instrucciones tras ser decodificadas para ser emitidas desde ahí a la unidad de datos donde se ejecutarán cuando dicha unidad esté libre y los operandos que se necesitan para realizar la correspondiente operación estén disponibles. La emisión desde esa ventana de instrucciones puede ser ordenada o desordenada.

Una estación de reserva es una estructura presente en los procesadores superescalares en los que se distribuye la ventana de instrucciones de forma que en las instrucciones decodificadas se envían a una u otra estación de reserva según la unidad funcional (o el tipo de unidad funcional) donde se va a ejecutar la instrucción.

Una ventana de instrucciones puede considerarse un caso particular de estación de reserva desde la que se puede acceder a todas las unidades funcionales del procesador. La transferencia de una instrucción desde la unidad de decodificación a la estación de reserva se denomina emisión, y desde la estación de reserva correspondiente a la unidad funcional, envío.

◀

Cuestión 4. ¿Qué utilidad tiene la predicación de instrucciones? ¿Es exclusiva de los procesadores VLIW?

Solución

La predicación de instrucciones permite eliminar ciertas instrucciones de salto condicional de los códigos. Por lo tanto, contribuye a reducir el número de riesgos de control (saltos) en la ejecución de programas en procesadores segmentados.

Aunque es una técnica muy importante en el contexto de los procesadores VLIW, porque al reducir instrucciones de salto condicional se contribuye a aumentar el tamaño de los bloques básicos (conjunto de

instrucciones con un punto de entrada y un punto de salida), existe la posibilidad de utilizar predicados tanto en procesadores superescalares como VLIW. Por ejemplo, en el conjunto de instrucciones de ARM se pueden predicar todas las instrucciones.

Cuestión 5. ¿En qué momento se lleva a cabo la predicción estática de saltos condiciones? ¿Se puede aprovechar la predicción estática en un procesador con predicción dinámica de saltos?

Solución

La predicción estática se produce en el momento de la compilación teniendo en cuenta el sentido más probable del salto (sobre todo es efectiva en instrucciones de control de bucles).

En el caso de los procesadores con predicción dinámica, se suele tener en cuenta algún tipo de predicción estática para cuando no se dispone de información de historia de la instrucción de salto (en la primera aparición de la misma).

Cuestión 6. ¿Qué procesadores dependen más de la capacidad del compilador, un procesador superescalar o uno VLIW?

Solución

Los procesadores superescalares pueden mejorar su rendimiento gracias a algunas técnicas aplicadas por el compilador, sobre todo en los procesadores de Intel a partir de la microarquitectura P6 que llevaba a cabo una traducción de las instrucciones del repertorio x86 a un conjunto de microoperaciones internas con estructuras fijas como las instrucciones de los repertorios RISC. Sin embargo, el papel del compilador en un procesador VLIW es fundamental, ya que es él el que se encarga de realizar la planificación de las instrucciones para aprovechar el paralelismo entre instrucciones. En los procesadores superescalares existen estructuras como las estaciones de reserva, los buffer de renombramiento, los buffer de reordenamiento, etc. que extraen el paralelismo ILP dinámicamente.

Cuestión 7. ¿Qué procesadores tienen microarquitecturas con mayor complejidad hardware, los superescalares o los VLIW? Indique algún recurso que esté presente en un procesador superescalar y no sea necesario en uno VLIW.

Solución

Los procesadores superescalares son los que tienen una microarquitectura más compleja. Precisamente, la idea de dejar que sea el compilador el responsable de extraer el paralelismo entre instrucciones perseguía conseguir núcleos de procesamiento más sencillos desde el punto de vista hardware para reducir el consumo energético y/o sustituir ciertos elementos hardware por un mayor número de unidades funcionales que permitiesen aumentar el número de operaciones finalizadas por ciclo.

Estructuras como las estaciones de reserva, los buffers de renombramiento, y los buffers de reordenamiento son típicas de los procesadores superescalares.

Cuestión 8. Haga una lista con 5 microprocesadores superescalares o VLIW que se hayan comercializado en los últimos 5 años indicando alguna de sus características (frecuencias, núcleos, tamaño de cache, etc.)

Solución

Procesador	Año	Frecuencia (GHz)	Fabricante	Observaciones
Core i7	2008	2.66 – 3.33	Intel	Intel x86-64. HyperThreading Tecnología de 45 nm o 32 nm QPI Transistores: 1.170 millones en el Core i7 980x, con 6 núcleos y 12 MB de memoria caché
Phenom II	2008	2.5 -3.8	AMD	x86-64 Proceso: 45 nm Serie 1000: 6 núcleos con 3MB de cache L2 (512KB por núcleo) y 6MB de cache L3 compartidos
Itanium 9300 (Tukwila)	2010	1.33 – 1.73	Intel	2 – 4 Cores Cache L2: 256 kB (D)+ 512 kB (I) Cache L3: 10–24 MB QPI Proceso: 65 nm
POWER7	2010	2.4 – 4.25	IBM	Repertorio: Power Arch. 4, 6, 8 Cores Proceso 45 nm CacheL1 32+32KB/core CacheL2 256 KB/core Cache L3 32 MB
Xeon 5600	2010	2.4 – 3.46	Intel	6 Cores (HT: 12 threads) QPI Cache 12 MB