Enseñanza de Mecanismos Hardware de Ejecución Especulativa de Instrucciones

Sergio Sáez, Juan Luis Posadas, Pedro López
Departamento de Informática de Sistemas y Computadores (DISCA)
Facultad de Informática, Universidad Politécnica de Valencia
e-mail: { ssaez, jposadas, plopez } @disca.upv.es

Resumen

El presente trabajo describe la metodología empleada en las prácticas de cuarto curso de la Facultad de Informática para la enseñanza y aplicación de mecanismos hardware de especulación de instrucciones basados en el algoritmo de Sohi. Debido a la complejidad de dichos mecanismos y a la imposibilidad de trabajar directamente con ellos, se ha desarrollado un sistema que permite al alumno realizar su propia implementación del algoritmo de Sohi, simular su funcionamiento en la ejecución de programas y, además, detectar de forma automática los posibles errores de ejecución debidos a una implementación incorrecta. El objetivo del sistema es facilitar la comprensión del algoritmo de ejecución especulativa. Para ello, gracias a la detección automática de fallos, al alumno se le proporciona de forma precisa dónde y cuándo está fallando su desarrollo, lo cual le permite rectificarlo. Esta realimentación es la principal ventaja del sistema utilizado pues implica llegar a entender bien el algoritmo de Sohi para así conseguir su correcto funcionamiento.

1. Introducción

En los actuales planes de estudios para la obtención del título de Ingeniero de Informática (II) en la Universidad Politécnica de Valencia (UPV), el estudio de la arquitectura del computador es competencia de dos asignaturas de segundo ciclo de carácter obligatorio, Arquitectura de Computadores 1 (ARQ1) y Arquitectura de Computadores 2 (ARQ2) [6], ubicadas en el séptimo y octavo semestre de la titulación, respectivamente.

Previamente a estas asignaturas el alumno ha cursado en el primer ciclo las materias Estructura de Computadores 1 (EC1) y Estructura de Computadores 2 (EC2) ubicadas en el segundo y tercer semestres, respectivamente, donde se estudia la organización general del computador y ya se introduce el concepto de procesador segmentado [1,4].

En la asignatura ARQ1 se define el concepto moderno de arquitectura, y se aplican las técnicas de segmentación a la unidad de ejecución de instrucciones del procesador. Sólo se consideran las instrucciones enteras y los riesgos se solucionan mediante técnicas sencillas (inserción de ciclos de parada y cortocircuito).

La asignatura ARQ2 es una continuación de ARQ1 que comprende los siguientes objetivos:

- Aplicar las técnicas de segmentación a la unidad de instrucción del computador con operaciones multiciclo. Diferenciar las técnicas de gestión estática vs. dinámica de instrucciones. Conocer y comprender las técnicas de ejecución *especulativa* de instrucciones. Conocer y comprender el concepto de procesador superescalar.
- Distinguir los tipos de computadores orientados al procesamiento de vectores. Conocer y comprender los computadores vectoriales segmentados.
- Diferenciar los multiprocesadores de memoria compartida y los multicomputadores. Identificar los aspectos que más influencia tienen sobre las prestaciones de éstos.

La organización de la docencia de ambas asignaturas corresponde con 3 créditos de teoría y problemas y 1,5 créditos de trabajo en el laboratorio. Dicho trabajo consiste en la realización, de un modo coordinado con las clases de teoría y problemas, de un conjunto de prácticas

que refuerzan la comprensión y aplican los conceptos estudiados.

En este trabajo se presenta un sistema desarrollado específicamente para la realización de la tercera práctica de la asignatura ARQ2 que trata sobre la ejecución especulativa de instrucciones basada en el algoritmo de Sohi [2]. Previamente, el alumno ya ha trabajado en la primera práctica con la gestión estática de instrucciones y en la segunda práctica con la gestión dinámica basada en el algoritmo de Tomasulo [5]. Las principales características del sistema que se propone consisten en: permitir al alumno la implementación del algoritmo de Sohi, la simulación del mecanismo de ejecución especulativa de instrucciones, la generación de ficheros de resultados para reproducir la simulación, la visualización secuencial del estado del procesador en cada ciclo de ejecución y la detección automática de posibles fallos debidos a una implementación incorrecta. Este último punto permite al alumno detectar cuándo y dónde funciona mal su algoritmo. Precisamente, una de las ventajas del sistema es la realimentación que ofrece al alumno el mecanismo de detección de fallos, guiándole de esta forma en el estudio y comprensión del algoritmo de Sohi hasta su correcto funcionamiento. El tiempo disponible para la realización de la práctica es de dos sesiones de dos horas cada una.

El presente trabajo está organizado como sigue: en la sección 2 se presenta brevemente el algoritmo de especulación, en la sección 3 se describe el trabajo de laboratorio, en la sección 4 se describe la simulación, generación y visualización de resultados, y la detección automática de errores. Finalmente, en la sección 5 se exponen algunas conclusiones de la experiencia.

2. Especulación de instrucciones mediante mecanismos hardware: algoritmo de sohi

El objetivo del sistema presentado en este trabajo es ofrecer al alumno la posibilidad de analizar de una forma didáctica y visual el correcto funcionamiento del algoritmo de Sohi para la ejecución especulativa y dinámica de instrucciones. La especulación de instrucciones es una técnica que pretende no sólo lanzar a ejecución sino ejecutar completamente (si es necesario) las instrucciones situadas tras los saltos condicionales antes de conocer el resultado del salto, teniendo en cuenta que la predicción puede ser incorrecta y que puede ser necesario volver atrás. Las instrucciones ejecutadas especulativamente no deben alterar el estado de la máquina (banco de registros y memoria) hasta que se valide la predicción efectuada en el salto.

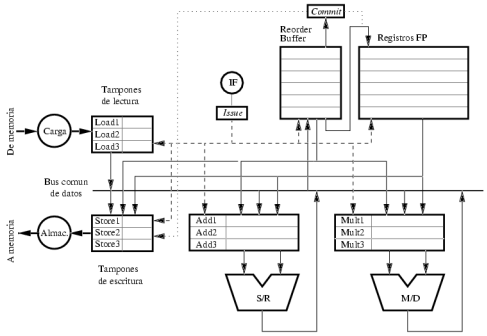


Figura 1. Ruta de datos para coma flotante

El mecanismo básico del algoritmo de Sohi consiste en: (1) lanzar las instrucciones en orden (fase *Issue*), (2) ejecutar las instrucciones fuera de orden (fase *EX*) y (3) finalizar las instrucciones en orden. Es necesario una fase adicional para finalizar las instrucciones: fase *Commit*. Las instrucciones llegan en orden a esta fase. En esta fase se actualizan los registros o la memoria y se reconocen las excepciones. Esta actualización sólo se lleva a cabo si la instrucción ya no es especulativa. Cuando una instrucción se lanza se reserva una entrada en una estructura denominada *reorder buffer*. Al terminar su ejecución no deposita el resultado en el registro destino, sino que lo hace en la entrada correspondiente del *reorder buffer*. De este modo, si esa instrucción tiene una dependencia de datos con otra posterior, la segunda deberá obtener sus operandos del *reorder buffer*, y no de los registros. Por otro lado, si una instrucción origina una excepción también se anota tal evento en su entrada del *reorder buffer*. Cuando la instrucción más antigua del *reorder buffer* finaliza (fase *Commit*): (1) se comprueba si ha producido alguna excepción y en tal caso se lanza la rutina de servicio, (2) se copia

el resultado desde su entrada del *reorder buffer* al registro destino o la posición de memoria correspondiente, (3) por último se libera el *reorder buffer*. Si un salto es incorrectamente predicho, cuando llega a la fase *Commit* limpia de instrucciones el *reorder buffer* y cancela todas las operaciones en curso. De esta forma las instrucciones lanzadas especulativamente (y ahora se sabe que incorrectamente) a ejecución después del salto no escriben sobre el registro destino (no se terminan) y no originan excepciones. La especulación y gestión dinámica de instrucciones se aplica a todas las instrucciones de la máquina, en la figura 1 se representa el esquema de la ruta de datos correspondiente a la parte de coma flotante.

3. Desarrollo de la práctica

Al alumno se le proporciona un simulador en el que el código correspondiente a las fases de escritura de resultados y *Commit* se ha eliminado, dejando únicamente un esqueleto de dichos procedimientos. También se le proporciona un algoritmo en pseudocódigo en el que se definen todas las acciones que tienen lugar en cada una de las fases del algoritmo. El trabajo del alumno consiste en escribir dichos procedimientos y conseguir un simulador que funcione. El simulador ha sido implementado en lenguaje C bajo el sistema operativo GNU/Linux.

La práctica está dividida principalmente en dos apartados: una primera parte donde el alumno debe familiarizarse con la estructura de datos utilizada en la implementación del simulador y acabar de desarrollar el código correspondiente al algoritmo de Sohi, y una segunda parte en la que el alumno tiene que comprobar el funcionamiento correcto del código implementado simulando la ejecución de varios programas en ensamblador. La primera parte de implementación obliga al alumno a comprender cómo funciona el algoritmo así como analizar qué recursos se necesitan para diseñar un procesador que lo aplique. En la segunda parte, con la simulación de la ejecución de programas y posterior visualización de los resultados obtenidos, el alumno puede comprobar si la versión del algoritmo que ha realizado es correcta. En caso contrario, el simulador indica el error cometido.

Los resultados que se generan en una simulación son ficheros HTML que permiten visualizar el cronograma de ejecución de las instrucciones junto con el estado del procesador ciclo a ciclo. En el caso de obtenerse un funcionamiento erróneo, el simulador indica en qué ciclo se ha producido el primer error y qué parte del estado del procesador es incorrecta. El alumno puede observar paso a paso la evolución de la ejecución de las instrucciones y ver dónde falla el algoritmo, para así proceder a su rectificación. Este método permite profundizar de forma sencilla en el estudio y comprensión del algoritmo al poderse comprobar de forma práctica y visual cualquier situación teórica.

La evaluación de la práctica y comprensión que el alumno tiene que haber adquirido se realiza a través de un cuestionario con preguntas relativas a la ejecución de varios programas ejemplo y así forzar a su seguimiento visual ciclo a ciclo.

La experiencia llevada a cabo ha sido satisfactoria y los alumnos agradecen el material disponible ya que les facilita la comprensión de los conceptos estudiados en clase, además de permitirles preparar el examen con mayor facilidad al disponer de múltiples ejemplos de ejecución que ellos mismos pueden generar y visualizar tantas veces como necesiten.

4. Simulación, visualización de resultados y detección automática de errores

El simulador, denominado DLX/ESP, está basado en el procesador DLX [2,3] con instrucciones de coma flotante. Permite interpretar código ensamblador desarrollado con un conjunto de instrucciones enteras reducido, e instrucciones de coma flotante aritméticas y de carga/almacenamiento de simple precisión.

Una vez implementado (la parte correspondiente a la especificación del algoritmo de Sohi) y compilado el simulador, se comprueba su funcionamiento mediante la simulación de la ejecución de varios programas.

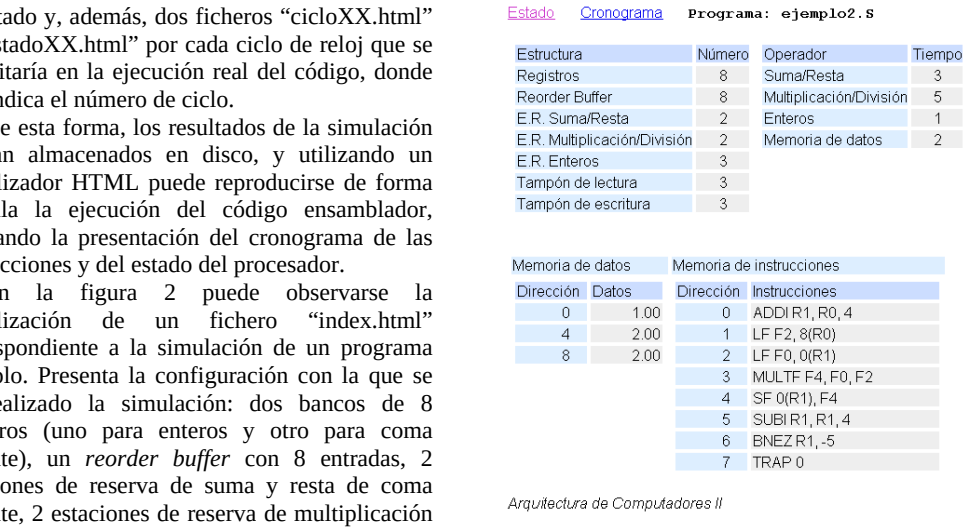
El programa con el código en ensamblador a ejecutar se le proporciona al simulador mediante un fichero de texto que se pasa como parámetro (p. ej.: *dlxesp –f ejemplo2.s*). Entonces, el simulador genera un fichero “index.html” con las características del simulador y el código que se ha

ejecutado y, además, dos ficheros “cicloXX.html” y “estadoXX.html” por cada ciclo de reloj que se necesitaría en la ejecución real del código, donde XX indica el número de ciclo.

De esta forma, los resultados de la simulación quedan almacenados en disco, y utilizando un visualizador HTML puede reproducirse de forma sencilla la ejecución del código ensamblador, utilizando la presentación del cronograma de las instrucciones y del estado del procesador.

En la figura 2 puede observarse la visualización de un fichero “index.html” correspondiente a la simulación de un programa ejemplo. Presenta la configuración con la que se ha realizado la simulación: dos bancos de 8 registros (uno para enteros y otro para coma flotante), un *reorder buffer* con 8 entradas, 2 estaciones de reserva de suma y resta de coma flotante, 2 estaciones de reserva de multiplicación y división de coma flotante, 3 estaciones de reserva para operaciones con enteros, 3 tampones de lectura, 3 tampones de escritura, un operador de suma y resta de coma flotante con un tiempo de 3 ciclos, un operador de multiplicación y división de coma flotante con un tiempo de 5 ciclos, un operador para operaciones con enteros con un tiempo de 1 ciclo y, por último, una memoria de datos con un tiempo de acceso de 2 ciclos, de la que se visualiza su contenido una vez inicializada.

Además, también se presenta la memoria de



Estado Cronograma **Programa: ejemplo2.S**

Estructura	Número	Operador	Tiempo
Registros	8	Suma/Resta	3
Reorder Buffer	8	Multiplicación/División	5
E.R. Suma/Resta	2	Enteros	1
E.R. Multiplicación/División	2	Memoria de datos	2
E.R. Enteros	3		
Tampón de lectura	3		
Tampón de escritura	3		

Memoria de datos

Dirección	Datos
0	1.00
4	2.00
8	2.00

Memoria de instrucciones

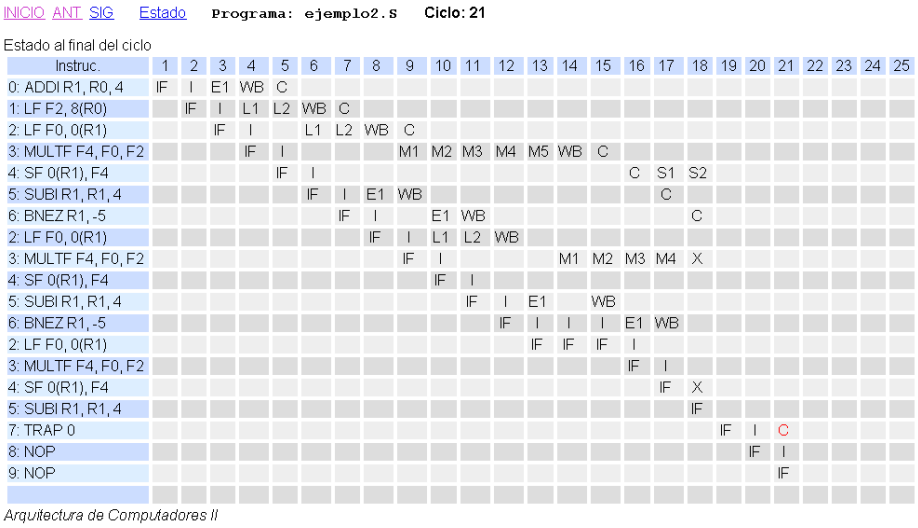
Dirección	Instrucciones
0	ADDI R1, R0, 4
1	LF F2, 8(R0)
2	LF F0, 0(R1)
3	MULTF F4, F0, F2
4	SF 0(R1), F4
5	SUBI R1, R1, 4
6	BNEZ R1, -5
7	TRAP 0

Arquitectura de Computadores II

Figura 2. Configuración del procesador

instrucciones con el código ensamblador simulado. En este caso puede observarse un código sencillo correspondiente a un bucle donde se realizará el escalado de un vector almacenado en memoria.

Es importante destacar que el simulador permite cambiar la configuración para así poder comparar diferentes resultados dependientes de



INICIO ANT SIG Estado **Programa: ejemplo2.S** **Ciclo: 21**

Estado al final del ciclo

Instruc.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0: ADDI R1, R0, 4	IF	I	E1	WB	C																				
1: LF F2, 8(R0)		IF	I	L1	L2	WB	C																		
2: LF F0, 0(R1)			IF	I		L1	L2	WB	C																
3: MULTF F4, F0, F2				IF	I				M1	M2	M3	M4	M5	WB	C										
4: SF 0(R1), F4					IF	I										C	S1	S2							
5: SUBI R1, R1, 4						IF	I	E1	WB								C								
6: BNEZ R1, -5							IF	I		E1	WB							C							
2: LF F0, 0(R1)								IF	I	L1	L2	WB													
3: MULTF F4, F0, F2									IF	I				M1	M2	M3	M4	X							
4: SF 0(R1), F4										IF	I														
5: SUBI R1, R1, 4											IF	I	E1		WB										
6: BNEZ R1, -5												IF	I	I	I	E1	WB								
2: LF F0, 0(R1)													IF	IF	IF	I									
3: MULTF F4, F0, F2																IF	I								
4: SF 0(R1), F4																	IF	X							
5: SUBI R1, R1, 4																		IF							
7: TRAP 0																			IF	I	C				
8: NOP																				IF	I				
9: NOP																					IF				

Arquitectura de Computadores II

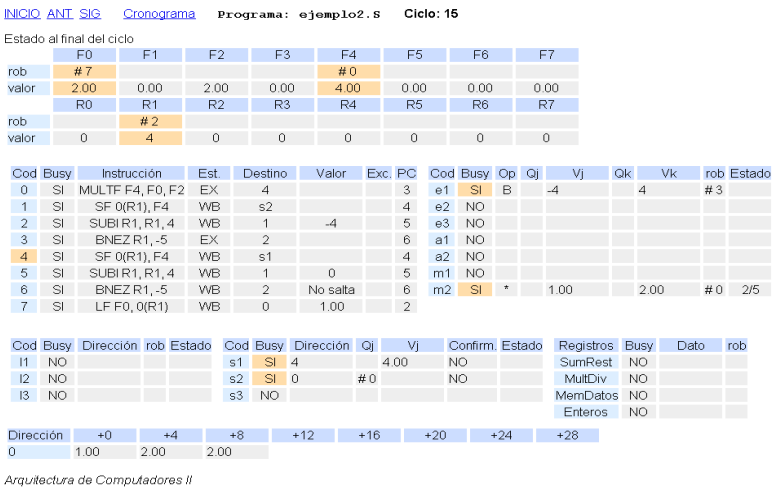
Figura 3. Cronograma de ejecución

ésta (número de entradas en el *reorder buffer*, número de estaciones de reserva, tiempo de los operadores, etc.).

En la figura 3 puede observarse el cronograma de ejecución de las instrucciones correspondientes al ejemplo. El simulador ha generado un total de 21 ciclos de reloj que son, empleando la configuración indicada, el tiempo global de ejecución del código ejemplo. En la figura se muestra el estado del cronograma en el ciclo 21 (último). Puede observarse que cada ventana dispone en la parte superior de un enlace al ciclo anterior y otro al posterior, permitiendo fácilmente avanzar o retroceder ciclo a ciclo en la ejecución de las instrucciones. Viendo los diferentes cronogramas generados (uno por cada ciclo) se comprueba cómo el *hardware* lanza a ejecución instrucciones posteriores a otras que se han parado y cómo se altera dinámicamente el orden de la ejecución, evitando que el hecho de parar una instrucción afecte a las que le siguen. En los cronogramas también puede apreciarse la existencia de otro enlace con nombre *estado* que sirve para abrir una nueva ventana donde se visualiza el estado del procesador en ese ciclo. En la figura 4 puede apreciarse el estado del procesador al finalizar el ciclo 15.

En las ventanas del estado del procesador se visualizan principalmente: los bancos de registros de enteros y de coma flotante con las marcas (*rob*) asociadas a cada uno de ellos en ese instante (cada marca es la entrada del *reorder buffer* correspondiente a la última instrucción lanzada que tenga que escribir en el registro); el contenido del *reor-der buffer*; las estaciones de reserva, indicándose las operaciones que las ocupan, la disponibilidad de los operandos (valores, o, en su defecto, las entradas del *reorder buffer* asociadas), la entrada del *reorder buffer* correspondiente a la instrucción que está utilizando el operador y el número de ciclos ejecutados; los tampones de lectura y escritura con las direcciones de acceso y el contenido de la memoria de datos para permitir comprobar si al final de la ejecución del código se han obtenido los resultados correctos.

Imaginemos que el alumno ha cometido un error en la implementación del algoritmo de Sohi. En este caso, detectar exactamente dónde falla su algoritmo no es una tarea sencilla debido a la cantidad de información que hay que controlar en la ejecución de cualquier programa. Para ello, el sistema desarrollado proporciona un mecanismo de generación y validación de *firmas* que permite al alumno comprobar si su algoritmo ejecuta correctamente un programa dado. El simulador, además, le indicará en qué ciclo falla su algoritmo y cuál es el error detectado. Dicho error se visualiza en el estado del procesador correspondiente al ciclo.



INICIO ANT SIG Cronograma Programa: ejemplo2.s **Ciclo: 15**

Estado al final del ciclo

	F0	F1	F2	F3	F4	F5	F6	F7
rob	#7				#0			
valor	2.00	0.00	2.00	0.00	4.00	0.00	0.00	0.00
	R0	R1	R2	R3	R4	R5	R6	R7
rob		#2						
valor	0	4	0	0	0	0	0	0

Cod	Busy	Instrucción	Est.	Destino	Valor	Exc.	PC
0	SI	MULTF F4, F0, F2	EX	4			3
1	SI	SF 0(R1), F4	WB	s2			4
2	SI	SUBI R1, R1, 4	WB	1	-4		5
3	SI	BNEZ R1, -5	EX	2			6
4	SI	SF 0(R1), F4	WB	s1			4
5	SI	SUBI R1, R1, 4	WB	1	0		5
6	SI	BNEZ R1, -5	WB	2	No salta		6
7	SI	LF F0, 0(R1)	WB	0	1.00		2

Cod	Busy	Op	Qj	Vj	Qk	Vk	rob	Estado
e1	SI	B		-4		4	#3	
e2	NO							
e3	NO							
a1	NO							
a2	NO							
m1	NO							
m2	SI	*		1.00		2.00	#0	2/5

Cod	Busy	Dirección	rob	Estado
I1	NO			
I2	NO			
I3	NO			

Cod	Busy	Dirección	Qj	Vj	Confirm	Estado
s1	SI	4		4.00	NO	
s2	SI	0	#0		NO	
s3	NO					

Registros	Busy	Dato	rob
SumRest	NO		
MultDiv	NO		
MemDatos	NO		
Enteros	NO		

Dirección	+0	+4	+8	+12	+16	+20	+24	+28
0	1.00	2.00	2.00					

Arquitectura de Computadores II

Figura 4. Estado del procesador al final del ciclo 15

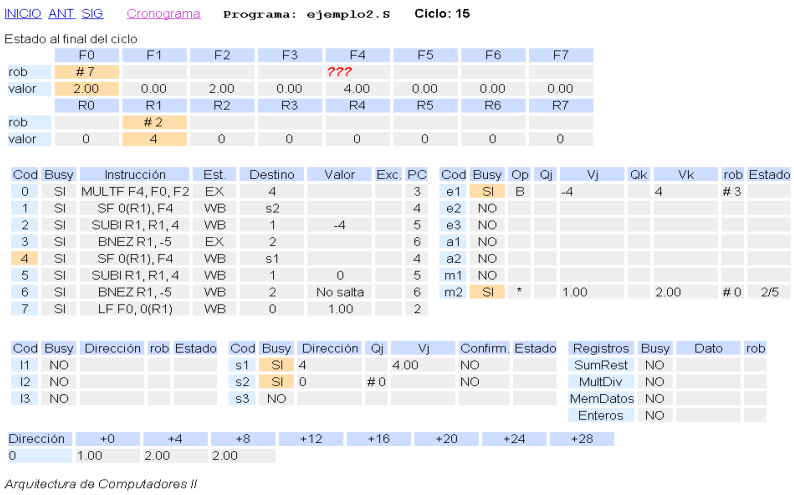


Figura 5. Detección de error en el ciclo 15

La validación mediante *firmas* funciona de la siguiente forma: cuando se simula la ejecución de un programa el simulador genera un fichero (*firma*) con el estado del procesador en cada ciclo, el profesor entrega al alumno las firmas de los programas que luego éste tendrá que ejecutar y el alumno indica al simulador que compare su ejecución con el fichero de *firmas* correcto. De esta forma, mediante la comparación, el simulador puede indicar los errores que se produzcan (si el alumno implementó el algoritmo correctamente sus ficheros de firmas coincidirán exactamente con los originales).

Por ejemplo, supóngase que el alumno comete el error de liberar la marca de un registro al finalizar una instrucción que no es la última lanzada a ejecución que escribe en dicho registro (por lo que la marca del registro no coincide con su entrada del *reorder buffer*). En este caso, cuando se simulara la ejecución del ejemplo, el simulador le indicaría en que ciclo se ha producido el error y marcaría el estado del registro como invalido. La figura 5 muestra un ejemplo de este error en el registro F4, mientras que en la figura 4 se muestra que el registro F4 todavía debería estar reservado por la instrucción que se encuentra en la entrada 0 del *reorder buffer*.

5. Conclusiones

En este trabajo se ha presentado un simulador para la ejecución especulativa de instrucciones y sobretodo, un mecanismo de detección de fallos en la implementación, que le ofrece al alumno una retroalimentación adicional a la ofrecida por el profesor y la simulación en sí. El mecanismo aquí presentado se está incorporando a la totalidad de las prácticas en las que el objetivo principal es la implementación, en un simulador, de algún mecanismo hardware de gestión de instrucciones.

Referencias

[1] D.A. Patterson, J.L. Hennessy. *Organización y Diseño de Comput*. Mc. Graw Hill, 1995.

[2] J.L. Hennessy, D.A. Patterson. *Arquitectura de Computadores, un enfoque cuantitativo*. New York. Mc. Graw Hill, 1993.

[3] J.L. Hennessy, D.A. Patterson. *Herramientas y Simuladores: http://www.mkp.com/books_catalog/ca/hp2e_res.asp#other*

[4] J.L. Posadas, A. Robles. *Innovación Educativa en la Enseñanza del Procesador Segmentado*. JENUI, 1999.

[5] Sergio Sáez, Juan Luis Posadas, Pedro López. *Enseñanza de la planificación dinámica de instrucciones*. JENUI, 2000

[6] *WEB de las asignaturas ARQ1 y ARQ2: http://dlxv.disca.upv.es*