SIMULADOR DE UN COMPUTADOR SEGMENTADO CON ESPECULACIÓN *HARDWARE*.

Juan Carlos Martínez y Pedro López

Departamento de Informática de Sistemas y Computadores.
Escuela Universitaria y Facultad de Informática de Valencia.
Universidad Politécnica de Valencia.
e-mail: jc@gap.upv.es

RESUMEN: En el presente artículo se presenta un simulador de un computador DLX con especulación *hardware*. El simulador permite la ejecución ciclo a ciclo de pequeños programas, visualizando el estado de la unidad de instrucción segmentada y el avance de las instrucciones por la misma.

1.- INTRODUCCIÓN.

Las materias propias de "Arquitectura de Computadores" estudian, entre otras, las técnicas empleadas en los modernos procesadores para conseguir altas prestaciones. Así, es habitual encontrar el estudio de las técnicas de segmentación, planificación dinámica de instrucciones, predicción de saltos, especulación, procesadores superescalares, etc, en los programas de dichas asignaturas [2].

Para la comprensión de estas técnicas es conveniente la resolución de ejercicios en los que se realiza un seguimiento ciclo a ciclo (o paso a paso) del estado de los distintos componentes del procesador. La realización de estos ejercicios se ve facilitada mediante el empleo de simuladores que permitan ejecutar programas sencillos, aplicando las técnicas que se están estudiando. De esta manera, el estudiante puede validar los ejercicios efectuados, al tiempo que alcanzar una mejor comprensión de dichas técnicas. A diferencia de otros simuladores, cuyo objetivo es favorecer el aprendizaje del lenguaje máquina, en el caso que nos ocupa hay que hacer un especial hincapié en la utilización de un modelo detallado del sistema a simular, junto con una buena interfaz con el usuario. Así pues, el simulador debe ofrecer, además de los resultados de la ejecución del programa (tiempo transcurrido, estado de los registros y memoria), la posibilidad de simular con detalle y visualizar el avance de las instrucciones a lo largo de la unidad de instrucción segmentada, la reserva de estaciones con un algoritmo de planificación dinámica de instrucciones, etc,...

En este artículo se presenta un simulador de un computador segmentado con planificación dinámica de instrucciones y especulación (*xdlxvsim*), desarrollado en el lenguaje de programación C bajo el sistema operativo Linux y entorno gráfico X-Window. El resto del artículo está organizado como sigue. La sección 2 describe la arquitectura de la máquina simulada. La sección 3 describe el interfaz con el usuario. Finalmente, se presentan algunas conclusiones.

2. ARQUITECTURA DE LA MÁQUINA SIMULADA.

Xdlxvsim es un simulador de un computador segmentado con gestión dinámica de instrucciones y especulación. La arquitectura de la máquina simulada se basa en el computador DLX descrito por Hennessy-Patterson en [1]. El procesador simulado tiene un ciclo de instrucción segmentado, aplicando técnicas de planificación dinámica de instrucciones para permitir alterar el orden de lanzamiento a ejecución de las instrucciones, y predicción dinámica de instrucciones de salto basadas en un *branch target buffer* (*BTB*) de 1 bit y especulación hardware, siguiendo el algoritmo propuesto en [1]. Sólo los saltos que se predicen como tomados se almacenan en la tabla *BTB*.

La ruta de datos consta de dos buses de datos internos independientes (Figura 1), uno para instrucciones enteras y el otro para las instrucciones en coma flotante. Existe un único *reorder buffer* para ambos buses para conservar el orden de ejecución de las instrucciones. En la Figura 2 se puede observar la parte de la ruta de datos dedicada a las operaciones en coma flotante. Como podemos comprobar se dispone de una unidad de ejecución diferente según el tipo de operación. Cada operador tiene asociado un conjunto de estaciones reserva. Nótese que algunas instrucciones (cargas, almacenamientos o transferencia de datos) pueden necesitar utilizar el bus de enteros y el de coma flotante.

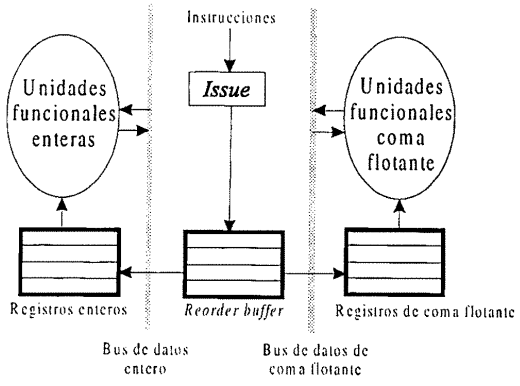

Figura 1. Ruta de datos del procesador simulado.

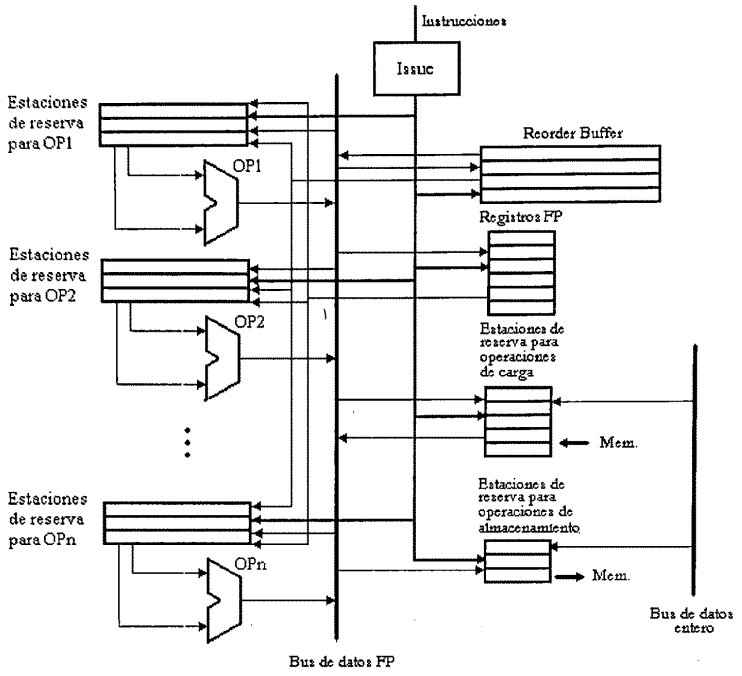

Figura 2 Unidad de coma flotante.

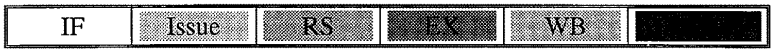

Figura 3: Segmentación del procesador dimulado.

El ciclo de instrucción está segmentado en seis fases (Figura 3). La fase *IF* realiza la búsqueda en memoria de la siguiente instrucción a ejecutar y calcula la dirección de la siguiente instrucción. En esta fase se realiza la predicción de saltos consultando el *BTB*.

La fase *Issue* decodifica la instrucción y la lanza según el algoritmo de planificación dinámica de instrucciones utilizado y resuelve las posibles dependencias de datos. Lo primero que se hace es decodificar la instrucción y extraer los operandos fuentes y destino, o la dirección destino de salto, en su caso. Después se busca una posición libre en el *reorder buffer* y una estación de reserva libre en el operador correspondiente, introduciendo en dichas tablas los datos necesarios. Si se dispone de todos los operandos y hay un operador libre la instrucción es pasada directamente a la fase *EX*. En otro caso la instrucción pasa a la fase *RS*.

La fase *RS* se limita a comprobar si la instrucción tiene los operandos fuentes disponibles y un operador libre para poder pasar la instrucción a la fase *EX*.

La fase *EX* realiza la ejecución de la instrucción y calcula el resultado de la operación. Esta fase es multiciclo, es decir, dependiendo de la operación a realizar esta puede durar varios ciclos. Por otro lado, varios operadores pueden trabajar en paralelo.

Una instrucción llega a la fase *WB* cuando ha terminado de calcular el resultado en la fase *EX* volcándolo sobre el bus de datos. Al tener buses separados para enteros y coma flotante, pueden entrar una instrucción entera y una en coma flotante a la vez a esta fase. Para diferenciarlas las denominaremos *WBI* (enteros) y *WBF* (coma flotante), respectivamente. En esta fase, la instrucción que toma el bus vuelca no solo el resultado de su operación en el bus sino también un identificador de la instrucción para que todas las estaciones de reserva lean el dato si es un operando fuente de alguna operación en espera. El *reorder buffer* almacenará el dato que se vuelca en el bus.

La última fase es la fase *Commit* que es la encargada de actualizar el estado de la máquina. Las instrucciones ejecutadas llegan en orden a esta fase. En esta fase, las operaciones que depositan su resultado en un registro actualizan el banco de registros y las instrucciones de almacenamiento lanzan la escritura en memoria. Las instrucciones de salto comprueban si se predijo correctamente el salto y en caso de error se anulan todas las instrucciones que se haya lanzado. En esta fase se comprueban también las excepciones que pudo provocar la instrucción en cualquier fase de la segmentación.

4.- INTERFAZ CON EL USUARIO.

En este punto vamos a presentar el simulador utilizando un ejemplo. Para ello utilizaremos un programa que suma un escalar a un vector almacenado en la memoria, cuyo código es el siguiente:

```
tamanyo:    .word 256

start:      lw r1, tamanyo(r0)      ; r1=límite
            add r2,r0,r0            ; r2=0 sera el indice
            ld f0, a(r0)            ; f0=a

bucle:      ld f2, x(r2)            ; f2=X[i]
            addd f8, f2, f0         ; f8= a + X[i]
            sd z(r2), f8            ; Z[i]= a+ X[i]
            add r2, r2,8            ; i++
            slt r3, r2, r1          ; compara si es el último elemento
            bnez r3, bucle          ; bucle
            trap #0                 ; fin de programa.
```

La aplicación procura minimizar al máximo el espacio en pantalla. La ventana principal es una simple barra de herramientas (Figura 4).

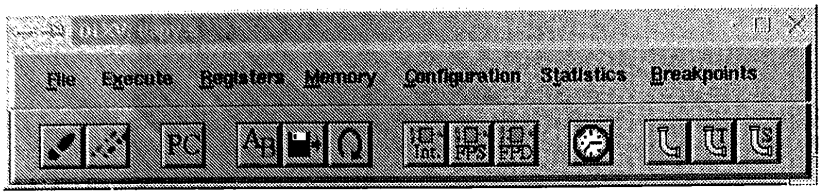

Figura 4: Ventana principal del programa.

Mediante los menús podemos acceder a todas las funciones del programa, y algunas de ellas han sido añadidas a la barra de herramientas en forma de botones con dibujos intuitivos. Las funciones del programa son las siguientes. Mediante el menú *File* se puede cargar un programa, abrir un editor para modificarlo, recargar el programa cargado si lo hemos modificado con el editor o salir del programa. Al cargar o recargar se ensambla automáticamente el programa. El menú *Execute* permite ejecutar el programa ciclo a ciclo, un cierto número de ciclos o hasta que termine el programa o al alcanzar un punto de ruptura. También en este menú se puede inicializar la máquina y las estadísticas. El menú *Registers* permite abrir acceder a los registros del procesador, enteros, de coma flotante de simple o doble precisión, así como a los registros internos del procesador. En el menú *Memory* podemos consultar o cambiar zonas de memoria. El menú *Configuration* sirve para configurar el simulador. Se puede configurar el tamaño de memoria, el número de estaciones de reserva, el número de operadores, sus latencias, el tamaño del *reorder buffer*, etc. También permite configurar el editor a utilizar para escribir el código fuente de los programas. Con el menú *Statistics* se accede a las estadísticas del programa ejecutado y con el menú *breakpoints* se pueden configurar puntos de ruptura para el programa.

Los tres botones de la derecha muestran el estado de las etapas de la segmentación. En el caso que nos ocupa, el botón es el ubicado al final cuya "S" hace referencia a la especulación (ver Figura 5). El botón con la forma de un reloj abre la ventana *Clock Cycles Diagram* (ver Figura 6) con un esquema de la ejecución de las instrucciones, indicando en qué fase de ejecución se encuentran en cada ciclo.

Vamos a pasar ahora a visualizar la simulación del programa. En la figura 6 se puede observar los primeros ciclos del programa. Obsérvese la ejecución de las instrucciones fuera de orden en los operadores y la terminación en orden (etapa *Commit*). Nótese que en el ciclo 9 se efectúa incorrectamente la predicción como "no tomado" (el salto no está todavía en la tabla BTB), cancelándose en el ciclo 17 las instrucciones lanzadas a ejecución incorrectamente. En la figura 5 se observa el estado de la unidad de instrucción segmentada en el ciclo 7.

El simulador también permite visualizar el estado de todas las tablas de la unidad de instrucción (estaciones de reserva, reorder buffer, BTB, etc). La Figura 7 muestra capturas de pantalla con esta información. Finalmente, la Figura 8 muestra un ejemplo de las estadísticas de ejecución que ofrece la aplicación (en este caso, las instrucciones ejecutadas clasificadas por códigos de operación).

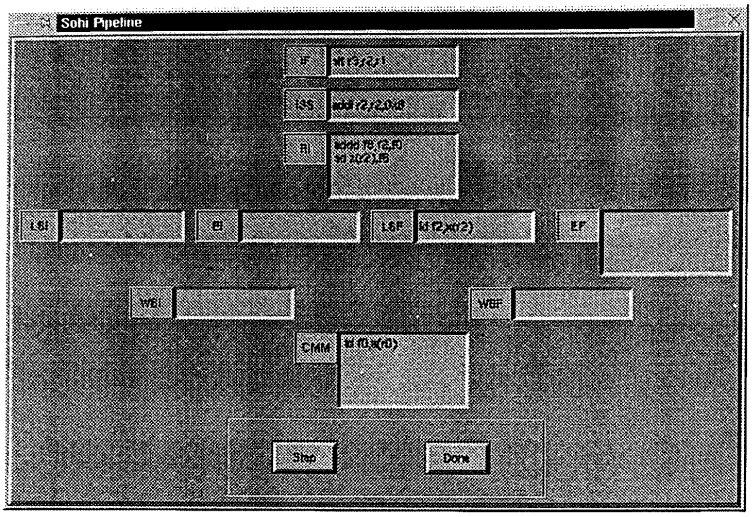

Figura 5: Avance de las instrucciones en la unidad de instrucción.

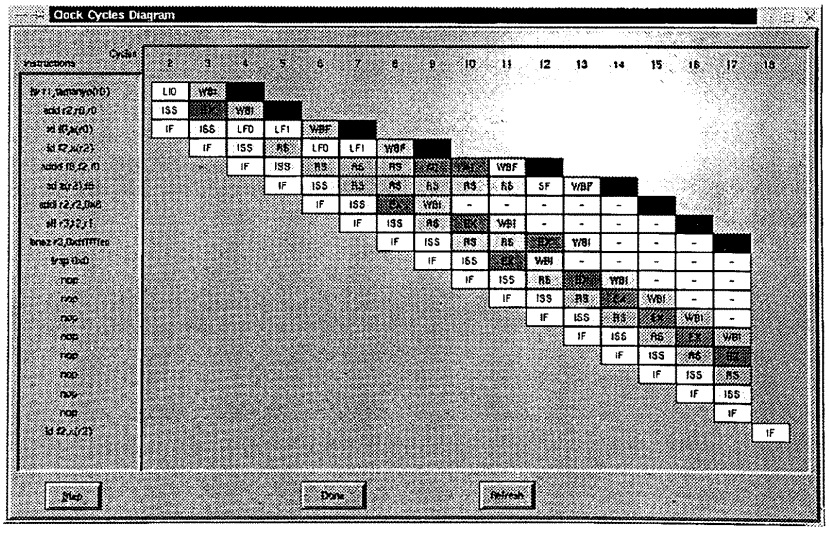

Figura 6: Esquema de ejecución de una iteración del programa.

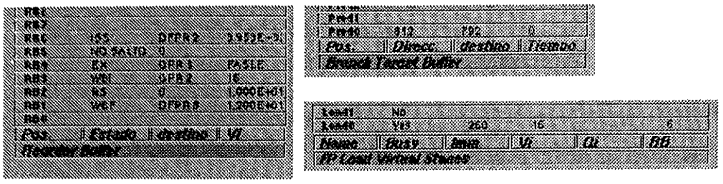

Figura 7: Estructuras de datos de la unidad de instrucción.

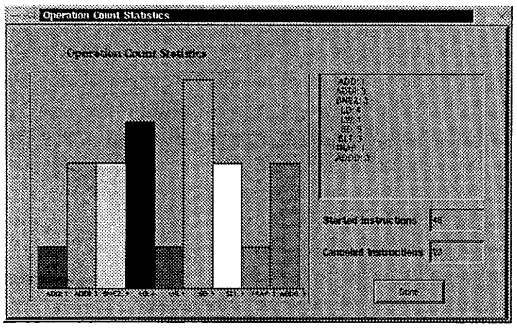

Figura 8: Ejemplo de estadísticas de ejecución.

5.- CONCLUSIONES

En este artículo se ha presentado un simulador de un computador con planificación dinámica de instrucciones, predicción de saltos y ejecución especulativa. El simulador es capaz de ejecutar programas escritos en lenguaje ensamblador del computador *DLX* y ofrece la posibilidad de visualizar el avance de las instrucciones por las etapas de la unidad de instrucción, el diagrama de ejecución ciclo a ciclo y el contenido de todas las estructuras de datos. El simulador se ha desarrollado como una herramienta de apoyo a la docencia en la materia de "Arquitectura de Computadores", permitiendo a los estudiantes la realización y ejecución de pequeños programas, alcanzando una mejor compresión del funcionamiento de las técnicas de especulación hardware.

BIBLIOGRAFÍA

[1] J. Hennessy, D.A. Paterson, *Computer Architecture: A quantitative approach.* Morgan Kauffman, 1996.

[2] http://dlxv.disca.upv.es, Web de las asignaturas *Arquitectura de Computadores I y II* Facultad de Informática. Univ. Politécnica de Valencia.