em88110: Emulación de un procesador superescalar

Santiago Rodríguez de la Fuente
Rafael Méndez Cavanillas
M. Isabel García Clemente
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Universidad Politécnica de Madrid
Email: srodri@fi.upm.es, rmendez@fi.upm.es, mgarcia@fi.upm.es

Resumen

La enseñanza de la programación en ensamblador es un punto fundamental a la hora de conocer la arquitectura proporcionada por un computador. Por otra parte, las tendencias arquitectónicas de las máquinas actuales hacen que las técnicas de aumento de prestaciones basadas en la emisión de múltiples instrucciones por ciclo (máquinas superescalares) sean de uso generalizado hoy en día. La utilización de este tipo de plataformas para la realización de prácticas por el alumnado ofrece un entorno hostil y con dificultades añadidas para la depuración de los programas objeto de las prácticas. Estas razones nos llevaron a construir un emulador configurable de una máquina superescalar que permitiera elaborar un conjunto de prácticas, desde la más sencilla hasta prácticas más complejas, que utilicen todas las características de una máquina actual. Este emulador permite aportar una máquina virtual en la que el alumno programa sus prácticas independientemente de los problemas adicionales generados por una máquina real.

1 Descripción del emulador MC88110

El microprocesador MC88110 es un microprocesador RISC superescalar que se encuadra dentro de la familia 88000 de Motorola. Este procesador es capaz de emitir dos instrucciones por ciclo de reloj sin necesidad de que el programa de usuario tenga en cuenta las dependencias de datos generadas. El despacho de instrucciones se realiza hacia diez unidades funcionales distintas que trabajan en paralelo. Las instrucciones se emiten en el orden en que aparecen en el programa, pero debido al diferente tiempo de ejecución de cada una de las unidades funcionales las instrucciones pueden finalizar su ejecución fuera de orden. En la figura 1 se muestra la estructura general del procesador MC88110.

1.1 Descripción del *pipeline* de instrucciones

El componente principal, tanto de la máquina original como del emulador, es el secuenciador de instrucciones. El procesador original es capaz de emitir hasta dos instrucciones por ciclo de reloj. El secuenciador está organizado en un *pipeline* de instrucciones de cuatro etapas:

- **Fetch.** Se leen dos instrucciones de la cache de instrucciones. Su duración es de 1 ciclo.
- **Decodificación.** lectura de registros operandos de las instrucciones y cálculo de las direcciones de salto para hacer predicción estática. Su duración es de medio ciclo de reloj en el procesador original y de 1 ciclo en el emulador.

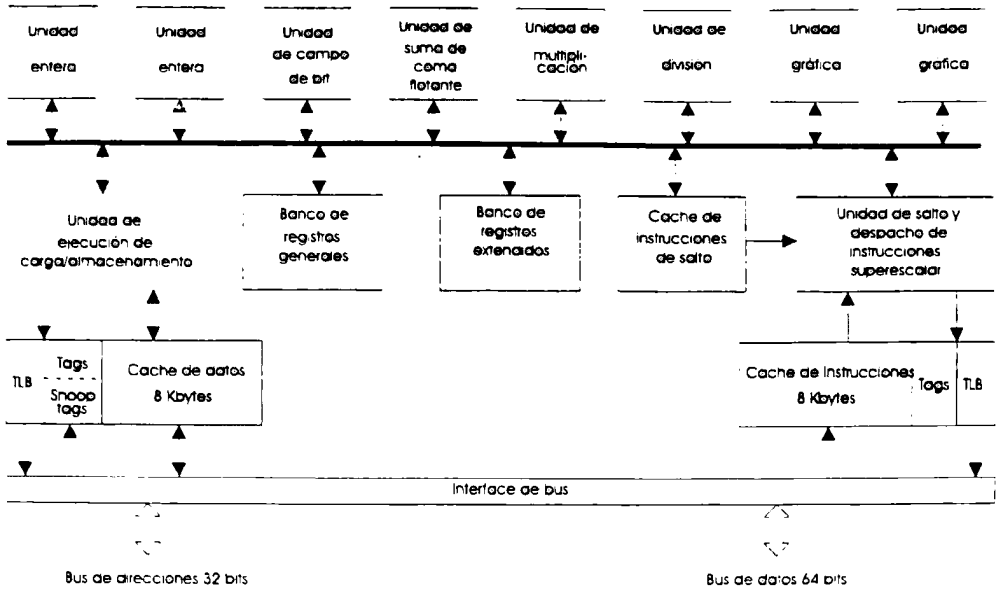

Figura 1: Estructura general del procesador MC88110

- **Ejecución**. evaluación de las condiciones de salto, cálculo de las direcciones de acceso a memoria para las instrucciones `ld` y `st` y acceso a memoria. Su duración depende de la unidad funcional que realice la ejecución.

- **Escritura en el banco de registros.** Se actualizan los registros especificados como destino de la instrucción. Su duración es de medio ciclo de reloj en el procesador original y de 1 ciclo en el emulador.

En cada ciclo de reloj el secuenciador lee dos instrucciones de la cache de instrucciones, las decodifica simultáneamente mientras lee los registros operandos del banco de registros. Si los operandos y las unidades funcionales están disponibles, el secuenciador despacha las dos instrucciones a sus correspondientes unidades funcionales. En el computador emulado se puede inhibir el *pipeline* de instrucciones. De esta forma el procesador se comporta como una máquina secuencial en la que no existe paralelismo interno ni, por supuesto, emisión múltiple de instrucciones. Esta característica permite plantear prácticas para alumnos noveles que realicen sus primeras tomas de contacto con el lenguaje ensamblador, obviando los problemas que presentan las máquinas superescalares (emisión de múltiples instrucciones por ciclo, finalización fuera de orden, etc.).

El secuenciador planifica las instrucciones según el orden en que aparecen en el programa, excepto para las instrucciones de salto y de almacenamiento en memoria que se pueden planificar sin que sus operandos estén disponibles. Para ello estas dos unidades funcionales disponen de sus respectivas estaciones de reserva (ver apartado 1.3). Con esta política se consigue que las instrucciones de salto y las instrucciones de almacenamiento no generen un parón en el *pipeline* por dependencias de datos. Una vez que la unidad funcional ha completado la instrucción pasa a escribir los resultados en el banco de registros. Debido a la diferente duración de las unidades funcionales (ver 1.3) las instrucciones pueden finalizar fuera de orden.

La emisión de una instrucción puede posponerse debido a un parón en el *pipeline*:

- La instrucción necesita un recurso que está ocupado (parón estructural)

- La instrucción necesita datos que son resultado de otra instrucción previamente emitida y no están disponibles (dependencia de datos).

- La instrucción que se va a emitir es un salto y provoca un cambio en el flujo de control del programa (dependencia de control).

Los parones estructurales se producen cuando la unidad funcional a la que se va a emitir la instrucción está ocupada. Para evitar que se produzca esta situación las unidades funcionales están segmentadas, de tal forma que pueden admitir una instrucción por ciclo de reloj. Por otra parte, puesto que el secuenciador es capaz de emitir dos instrucciones por ciclo, es posible que en el mismo ciclo de reloj se emitan dos instrucciones que deben ocupar una unidad funcional que no está segmentada. Por ejemplo en el mismo ciclo se desea emitir dos instrucciones de multiplicación o dos instrucciones de campo de bit. En este caso se produce un parón : la emisión de una de ellas debe aplazarse al ciclo siguiente. El computador emulado únicamente tiene dos puertas de escritura en el banco de registros (ver apartado 1.2). En el caso de que más de dos instrucciones finalicen su ejecución en una unidad funcional en el mismo ciclo de reloj, se producirá un parón debido a la no disponibilidad de puertas de escritura.

Los parones debidos a dependencias de datos [3] se dividen en:

- Dependencia *Read-after-Write*. Una instrucción toma como operando un registro que es el resultado de una instrucción emitida previamente.
- Dependencia *Write-after-Read*. Una instrucción toma como registro destino el mismo que una instrucción anterior ha tomado como fuente. Este tipo de dependencias tiene lugar cuando existen fases de lectura de registros posteriores a fases de escritura en el banco de registros o bien las instrucciones se emiten fuera de orden. Esto es lo que ocurre en este caso, pero las instrucciones que se emiten fuera de orden a las unidades funcionales no modifican el banco de registros y por tanto, esta dependencia no se genera.
- Dependencia *Write-after-Write*. Una instrucción toma como registro destino el mismo que una instrucción anterior ha tomado a su vez como destino de la instrucción. Esta dependencia se produce cuando hay varias fases del *pipeline* que realizan escrituras en el banco de registros, o la finalización de las instrucciones se puede realizar fuera de orden. Esta última situación es la que se produce en el caso que nos ocupa.

El procesador 88110, al igual que el procesador emulado, incorpora el mecanismo de marcado (*scoreboarding*) para almacenar las dependencias que se producen en la ejecución de los programas. Cada registro del banco incorpora un bit que almacena si el registro es destino de una instrucción en curso. En el momento de emitir una instrucción, se comprueba que los registros que actúan como operandos están libres para evitar dependencias *Read-after-Write*. Si están libres, la instrucción puede emitirse a su unidad funcional. Además se comprueba que el registro destino de la operación tampoco está ocupado. Con esta segunda comprobación se evitan las dependencias *Write-after-Write*.

Los parones de control (modificación del flujo de control del programa) provocan burbujas en el *pipeline*. La forma de reducir el impacto de estos parones utilizada en este procesador se basa en introducir instrucciones de bifurcación retardada de un *slot* y predicción de salto en la fase de decodificación en los saltos. Esta última técnica se mostrará en la sección 1.3.

Para reducir las consecuencias de los parones debidos a dependencias de datos el computador original puede adelantar (*forwarding*) [4] [2] los resultados de una instrucción simultáneamente a su escritura en el banco de registros. El adelantamiento de que dispone la máquina original es Ejecución-Ejecución. Esta característica no se ha incorporado en el procesador emulado.

1.2 Bancos de registros

El 88110 tiene dos bancos de registros: el banco general de registros y el banco extendido de registros. El banco general tiene 32 registros de 32 bits que almacenan direcciones y datos en coma fija o coma flotante. Todos los registros son accesibles por pares, de tal forma que pueden

almacenar operandos de 64 bits. El registro cero contiene la constante cero. Una escritura en este registro no tiene efecto.

El banco de registros extendido se compone de 32 registros de 80 bits. Se usan únicamente por las instrucciones de coma flotante y permiten almacenar datos en precisión simple (32 bits), doble (64 bits) o extendida (80 bits).

Cada uno de estos bancos de registros mantiene 4 puertos de lectura y 4 de escritura. De esta forma no es posible que se produzcan parones estructurales debidos a insuficiencia de puertos de escritura en los bancos. En el computador emulado se han simplificado los formatos del estándar de coma flotante, tal y como se explica en 1.3, y no se ha incorporado el banco de registros extendido. Por otra parte el banco de registros del emulador contiene cuatro puertos de lectura, pero únicamente dos de escritura y, por tanto, se pueden producir parones estructurales debidos a insuficiencia de puertos de escritura en el banco.

1.3 Descripción de las unidades funcionales

El 88110 dispone de diez unidades funcionales que trabajan simultáneamente. Todas ellas, a excepción de la unidad de división, están segmentadas para admitir una instrucción por ciclo de máquina: unas porque su tiempo de ejecución es de un ciclo y otras, la unidad de multiplicación, porque están segmentadas en etapas de un ciclo de latencia. El microprocesador original proporciona instrucciones de operación sobre datos de coma flotante. El formato de representación es el especificado en el estándar IEEE-754, y por tanto permite la manipulación de datos de precisión simple (32 bits), doble (64 bits) y extendida (80 bits). En el emulador para la realización de prácticas, se ha eliminado el formato de precisión extendida, puesto que no aporta nuevos conceptos que el alumno deba ejercitar que no queden representados en los formatos de precisión simple y doble. Las unidades funcionales se reparten de la siguiente forma:

- **Unidad entera.** Se han emulado dos unidades enteras independientes. Son simples unidades aritmético-lógicas que aceptan instrucciones lógicas y aritméticas de coma fija. El tiempo de ejecución para una instrucción es de un ciclo de reloj. Ésta es la única unidad funcional que está duplicada. La razón para ello es intentar evitar la mayor parte de los parones estructurales debidos a la no disponibilidad de unidades funcionales.

- **Unidad de campo de bit.** Ejecuta las instrucciones de manipulación de campo de bits en registros del computador. Su tiempo de ejecución es de un ciclo de reloj.

- **Unidad de multiplicación.** Ejecuta las instrucciones de multiplicación entera y coma flotante (IEEE-754). La ejecución de una instrucción emplea tres ciclos de reloj, pero esta unidad funcional está organizada en un *pipeline* de tres etapas que le permite aceptar una instrucción por ciclo.

- **Unidad de división.** Realiza todas las instrucciones de división entera y coma flotante. Su tiempo de ejecución es de trece ciclos de reloj y, a diferencia de las anteriores unidades funcionales, no está segmentada y no puede aceptar nuevas instrucciones mientras se encuentre ocupada.

- **Unidad de salto.** Ejecuta las instrucciones que desvían el flujo del programa. En la fase de decodificación de una instrucción condicional se calcula la dirección efectiva de salto y se realiza predicción estática de salto. En la fase de ejecución se evalúa la condición de salto. Si la predicción fue correcta, se elimina la marca de instrucción condicional de las instrucciones afectadas por la predicción. Si se ha fallado en la predicción, se abortan las instrucciones afectadas por dicha especulación. El procesador original mantenía una cache de direcciones de salto que le permitía acceder a la dirección destino del salto al

comienzo de la fase de decodificación. Esto se ha eliminado en el computador emulado y la dirección de salto siempre estará disponible al final de la fase de decodificación. Al igual que en el procesador original mantiene una estación de reserva, de tal forma que aunque los operandos no estén disponibles se puede emitir la instrucción a la unidad funcional y permanece en espera hasta que se permite el acceso a los operandos.

- **Unidad de carga-almacenamiento.** Es la unidad más sofisticada del computador. Dispone de dos estaciones de reserva donde se pueden almacenar cuatro instrucciones de carga y tres de almacenamiento, incluso si sus operandos no están disponibles. Esto permite la emisión de una instrucción de almacenamiento a la unidad funcional independientemente de la disponibilidad de los datos. Una vez que los datos están disponibles el almacén histórico de instrucciones permite la ejecución de la instrucción. Esta situación se produce cuando la instrucción de almacenamiento pasa a ser la primera del almacén histórico de instrucciones.

- **Unidad gráfica.** El computador original tiene dos unidades gráficas que permiten manipular estructuras utilizadas en algoritmos de representación de gráficos mediante instrucciones gráficas. Estas dos unidades funcionales no se han emulado puesto que no aportan nuevos conceptos relativos a arquitectura de computadores que el alumno deba conocer.

1.4 Almacén histórico de instrucciones

Puesto que la ejecución de instrucciones puede completarse fuera de secuencia, es necesario saber el orden de emisión de las instrucciones. Este es el caso de un error en la predicción de salto. Para poder volver hacia atrás se necesita saber cuales son las instrucciones que se han ejecutado y en qué orden. El almacén histórico de instrucciones es una cola FIFO que almacena las instrucciones que el secuenciador va emitiendo a las unidades funcionales el en orden lógico del programa. Cada entrada del almacén histórico contiene, además de la instrucción, el valor que almacenaba el registro destino de la operación. Cuando la instrucción que está como cabecera de la cola FIFO termina su ejecución, el almacén histórico de instrucciones elimina todas las instrucciones que hayan finalizado.

Si, por ejemplo, la instrucción que llega a la cabecera de la cola FIFO es un salto que se ha predicho mal, se eliminan del buffer histórico todas las instrucciones que se hayan ejecutado de forma condicional y se restaura el valor de los registros destino de estas instrucciones al valor que tuvieran antes de ejecutar la instrucción.

1.5 Sistema de memoria

El procesador original 88110 tiene arquitectura Harvard, es decir, caminos separados para instrucciones y para datos. Mantiene dos caches: una para instrucciones y otra para datos de 8Kbytes cada una de ellas. Las dos tienen política de ubicación asociativa por conjuntos de 128 conjuntos, dos bloques de cache por conjunto y 32 bytes por línea. La cache de instrucciones puede proporcionar dos palabras por ciclo de reloj. La cache de datos solamente puede proporcionar dos palabras por ciclo si están en la misma línea de cache. La política de coherencia de la cache de datos con la memoria principal es *copy-back*.

El espacio de direcciones en el computador original es de 4Gbytes. En el computador emulado hemos estimado que para la realización de prácticas no es necesario más de 256Kbytes que es el tamaño emulado de la memoria. La memoria principal del sistema es configurable. Los parámetros que se pueden especificar son: tiempo de acceso y el número de bloques en un entrelazado simple de orden inferior. El modelo original de las caches del 88110 se ha mejorado en el computador emulado. Las principales modificaciones se basan en poder configurar los parámetros de cada una de las caches por separado:

```
        or r4, r0, low(N)     ;Se carga en r4, r11, r1 y r2
        or r11, r0, low(RESULTADO) ;las direcciones de las variables
        or r1, r0, low(V1)    ;N, RESULTADO, V1 y V2
        or r2, r0, low(V2)
        or.u r4, r4, high(N)
        or.u r11, r11, high(RESULTADO)
        or.u r1, r1, high(V1)
        or.u r2, r2, high(V2)
        ld r4, r4, r0         ;Se carga en r4 el numero de
        and r7, r0, r0        ;elementos de los vectores
        cmp r3, r4, r0
        bb0.n 3, r3, error    ;Si es cero se salta a error
        and r8, r0, r0
bucle:  ld r5, r1, r0         ;Se carga en r5 y r6 los elementos
        ld r6, r2, r0         ;de los respectivos vectores
        sub r4,r4,1           ;Queda un elemento menos
        add r1, r1, 4         ;Se incrementa el puntero de V1
        mulu.d r9, r5, r6     ;El resultado de la multiplicacion se deja en r9 y r10
        add r2, r2, 4         ;Se incrementa el puntero de V2
        cmp r3, r4, r0
        add.co r7, r7, r10    ;Se acumula el resultado en doble precision
        bb1.n 3, r3, bucle    ;Si el contador no ha llegado a cero
        add.ci r8, r8, r9     ;se salta a bucle
        st r7, r11, r0
        st r8, r11, 4
error:  stop                  ;Fin de la emulacion
```

Figura 2: Programa ensamblador que realiza un producto escalar de dos vectores

- Las caches pueden desactivarse a voluntad para la realización de prácticas para los alumnos que no hayan cursado los principios de jerarquía de memoria.
- El tiempo de acceso a la cache. el tamaño y el tamaño del bloque son configurables.
- Las políticas de ubicación que se pueden elegir son: asociativa, asociativa por conjuntos o directa.
- Las políticas de escritura en memoria de la cache de datos es configurable. Sus posibles valores son: *Write-through* y *Copy-back*.

2 Visualización de la ejecución de un programa

Para facilitar la realización del código que compone la práctica se dispone de un programa ensamblador que permite generar el fichero binario ejecutable por el emulador [1]. Este ensamblador admite las instrucciones del MC88110 y algunas pseudoinstrucciones especificadas en el estándar IEEE 694 (`org`. `res` y `data`). Las referencias a variables se pueden realizar a través de sus nombres (etiquetas). La dirección del destino del salto en las bifurcaciones con direccionamiento relativo al contador de programa se pueden especificar mediante la etiqueta asociada a la instrucción a la que se desea saltar. El ensamblador incorpora la posibilidad de especificar macros para aquellos fragmentos de código que el usuario debe repetir con frecuencia.

```
PC=56          ld       r06,r02,r00     Tot. Inst: 10   << Ciclo : 28
FL=1 FE=1 FC=0 FV=0 FR=0
R01 = 00000074 h R02 = 0000009C h R03 = 00000000 h R04 = 0000000A h
R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
R09 = 00000000 h R10 = 00000000 h R11 = 0000006C h R12 = 00000000 h
R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000000 h
R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
R29 = 00000000 h R30 = 00000000 h R31 = 00000000 h
  Cache de instrucciones : 7 acceso(s), 2 fallo(s), Hit ratio 71.4
  Cache de datos : 1 acceso(s), 1 fallo(s), Hit ratio 0.0
FETCH: C          60       sub     r04,r04,1
C                 56       ld      r06,r02,r00
DEC: C            52       ld      r05,r01,r00
C                 48       and     r08,r00,r00
EXEC:             44       bb0.n   03,r03,14
                  40       cmp     r03,r04,r00
WBCK:
```

Figura 3: Estado del emulador tras ejecutar 27 ciclos de máquina

```
FETCH:            68       mulu.d  r09,r05,r06
                  64       add     r01,r01,4
DEC:
EXEC:             52       ld      r05,r01,r00
                  60       sub     r04,r04,1
                  56       ld      r06,r02,r00
WBCK:             48       and     r08,r00,r00
                  44       bb0.n   03,r03,14
```

Figura 4: Estado del *pipeline* tras ejecutar 29 ciclos de máquina

En la figura 2 se muestra un fragmento de programa ensamblador que realiza el producto escalar de dos vectores. Las 8 primeras líneas de código inicializan las variables que se van a utilizar a lo largo del programa. La instrucción `bb0.n 3,r3,error` bifurca si el bit 3 de `r3` es cero. Esta instrucción hace predicción estática de salto no efectivo. Nótese que `r3` es el resultado de una instrucción `cmp`. Esta instrucción compara los registros `r4` y `r0` y almacena el resultado de la comparación en `r3`. Este registro es una cadena de bits en el que cada bit representa una condición. El bit 3 de `r3` se activa si los registros `r3` y `r0` no son iguales. El sufijo `.n` indica que la instrucción siguiente sea ejecutada antes de efectuar el salto (bifurcación retardada).

Por otra parte, el emulador dispone de un depurador que permite al usuario controlar la ejecución de un programa. El usuario puede especificar puntos de ruptura, ejecutar ciclo a ciclo, visualizar valores de registros y posiciones de memoria y modificar el contenido de registros y posiciones de memoria. En cada instante en que el programa da control al usuario, se muestra el estado interno del procesador: contenido de los registros, registro de estado y estado del *pipeline*.

Después de ejecutar 27 ciclos de máquina el emulador muestra los datos que se muestran en la figura 3. En esta figura se muestra el estado del procesador antes de ejecutar el ciclo 28. Se muestra la instrucción que se está ejecutando, el valor del PC, el contenido de los registros de la máquina, el valor del registro de estado del procesador, las estadísticas de acceso a las caches del sistema y el estado del *pipeline*. En este caso una unidad entera está ocupada ejecutando

```
FETCH: C        56      ld      r06,r02,r00
C               52      ld      r05,r01,r00
DEC: C          92      st      r07,r11,r00
C               88      add.ci  r08,r08,r09
EXEC:           84      bb1.n   03,r03,-8
                80      add.co  r07,r07,r10
WBCK:
```

Figura 5: Estado del *pipeline* tras ejecutar 52 ciclos de máquina

la instrucción `cmp`. Ésta genera una dependencia con la instrucción de bifurcación que ocupa la dirección de memoria 44 (`bb0`) y, por tanto, se ha despachado a la estación de reserva de la unidad de salto. Las instrucciones que siguen a esta instrucción en memoria son resultado de haber realizado una predicción estática de salto y por tanto están marcadas como condicionales (`C`). La figura 4 muestra el estado del *pipeline* cuando la instrucción `bb0` ha pasado a la etapa `WBCK`. En este caso se han cambiado el estado de las instrucciones que inicialmente estaban marcadas como condicionales a ejecución normal puesto que la predicción ha sido correcta.

En el ejemplo presentado en la figura 5 se muestra la situación al finalizar la primera iteración del bucle. La instrucción `bb1` se ha emitido a la unidad de salto y previamente se ha realizado una predicción de salto efectivo. Esto se aprecia puesto que las instrucciones 52 y 56 están etiquetadas como condicionales. Obsérvese que al finalizar este ciclo se decidirá que el salto se ha predicho correctamente y se eliminarán estas marcas, así como la marca condicional de la instrucción 88 (bifurcación retardada). Por otra parte se abortará la instrucción que ocupa la dirección de memoria 92.

3 Conclusiones y Trabajos futuros

Hasta el momento se han realizado prácticas con este entorno, configurando el emulador en modo serie, y se están preparando nuevas prácticas relativas a computadores segmentados. Por otra parte, las prestaciones del emulador se mejorarán incorporando los adelantamientos de que dispone la máquina original y permitiendo un nuevo parámetro de configuración que deje la posibilidad de especificar si se desean emitir una o dos instrucciones por ciclo. Creemos que con esta nueva posibilidad se puede plantear un conjunto de prácticas más gradual, empezando por una máquina completamente secuencial, pasando por una máquina segmentada básica que emite una instrucción por ciclo y concluyendo el ciclo de prácticas con una máquina superescalar (emisión de dos instrucciones por ciclo).

4 Referencias

[1] Departamento de Arquitectura y Tecnología de Sistemas Informáticos. *Prácticas de Fundamentos de los Computadores: Programación en Ensamblador*. Marzo 1997.

[2] Keith Diefendorff, Michael Allen. Organization of the Motorola 88110 superscalar RISC microprocessor. *IEEE Micro*, 12(2):40-63, Abril 1992.

[3] John L. Hennessy, David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2ª edición, 1995.

[4] Motorola. *MC88110: Second Generation RISC Microprocessor. User's Manual*. Motorola Inc., 1991.