UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE CARRERA

SIMULADOR DE UN SISTEMA JERÁRQUICO DE MEMORIA

Autor: Raúl Castillo Ruzafa
Director: José Luis Sánchez García
Francisco José Alfaro Cortés

Septiembre, 2006

Reunido en la fecha el Tribunal evaluador, que más abajo se cita, del Proyecto Fin de Carrera titulado:

Simulador de un sistema jerárquico de memoria

presentado por D/Dª

Raúl Castillo Ruzafa

y siendo su/s tutor/es

José Luis Sánchez García y Francisco José Alfaro Cortés

se otorga la calificación de _____

Y para que así conste, se firma la presente acta en

Albacete a ..... de ..... de 20.....

PRESIDENTE: _____

SECRETARIO: _____

VOCAL: _____

SECRETARIO PRESIDENTE VOCAL

RESUMEN:

En las enseñanzas de ingenierías en Informática, es fundamental la realización de prácticas para asimilar totalmente muchos de los conceptos que en ellas se imparten. Poner en práctica todos esos conceptos mediante la realización de ejercicios prácticos de forma manual, ya sea en el aula (pizarra) o cada alumno autónomamente, resulta claramente insuficiente.

Por esta razón resulta muy interesante poseer herramientas que favorezcan las tareas a los docentes y a los alumnos. De forma que los docentes proporcionarán a los alumnos los conocimientos básicos y los alumnos podrán, con el simple uso de las herramientas, profundizar completamente en los conocimientos.

El simulador que se presenta en este proyecto se construirá basándose en los conceptos teóricos que deben servir a la simulación y el estudio completo de las herramientas que actualmente existen, haciendo hincapié en los puntos fuertes y débiles que poseen las mismas.

Una vez expuesto con claridad el estado del arte se trata de exponer claramente todos los requisitos que se buscarán en el simulador y comenzar todo el proceso de análisis diseño y codificación.

Creada la aplicación se han realizado unas baterías de pruebas para comprobar el correcto funcionamiento, obteniendo una calidad software bastante aceptable.

ÍNDICE:

AGRADECIMIENTOS:

A mis padres Antonio y María, y hermanos José, Antonio y Alicia que me han apoyado, ayudado y aconsejado y sin los que no podría haber completado este proyecto.

A todos mis amigos de Caravaca de la Cruz de la peña calimocho. A mis compañeros de piso de estos años Jose Luis, Calos, Javi, Eduardo y Rogelio. A todos mis amigos de Albacete especialmente a Adolfo, Juanma y Juan Ángel.

CAPITULO 1 INTRODUCCIÓN Y MOTIVACIÓN

1.1 Introducción y motivación

En la enseñanza universitaria de cualquier titulación de carácter técnico, y en particular en las ingenierías en Informática, es fundamental la realización de prácticas para asimilar totalmente muchos de los conceptos que en ellas se imparten. Poner en práctica todos esos conceptos mediante la realización de ejercicios prácticos de forma manual, ya sea en el aula (pizarra) o cada alumno autónomamente, resulta claramente insuficiente. Esta tarea, debido a la complejidad de los ejercicios, resulta a menudo bastante tediosa y hace que a menudo sea abandonada. Por otro lado, y debido a la limitación de tiempo y espacio en el aula, los problemas que se resuelven son quizás insuficientes o excesivamente sencillos (cortos). Si además se quiere profundizar en la materia y hacer problemas que se asemejen a casos reales, la reproducción manual de dichos problemas es muy compleja, y como ya se ha dicho, acaba resultando un trabajo muy largo y pesado de realizar.

Por tanto, el uso de herramientas que de forma rápida y automática puedan reproducir situaciones más o menos reales, faciliten el seguimiento de su evolución en el tiempo, permitan corregir los problemas que generan o ayuden a la búsqueda de soluciones es sumamente útil, tanto a nivel de investigación [9] como a nivel docente [10]. En particular, y en cuanto a la práctica docente se refiere, se ha demostrado que la utilización de herramientas que ayuden y agilicen, a la vez que profundicen y mejoren, la enseñanza de conocimientos de partes complejas e importantes de algunas asignaturas hace que el aprovechamiento de los alumnos sea mayor, ya que además de contar el profesor con otra herramienta didáctica para conseguirlo, los alumnos pueden seguir aprendiendo y practicando sobre dichos conceptos de forma autónoma siguiendo su propio ritmo de aprendizaje.

En muchos casos, este tipo de aplicaciones permiten tanto indicar simplemente los resultados finales de la simulación de un proceso como mostrar cualquiera de los pasos intermedios por los que hay que pasar cuando se está observando de forma manual su evolución en el tiempo. De esta manera, el usuario/alumno que lo utilice puede corregir y verificar un problema hecho a mano en todos los pasos por los que ha pasado para resolverlo, lo cual es muy ventajoso para su aprendizaje.

El uso de simuladores en docencia universitaria, y en particular en las asignaturas de Arquitectura de Computadores, está hoy en día muy extendido. Los buenos simuladores suelen estar preparados tanto para atender las exigencias de usuarios que se pueden considerar expertos en la materia, y que lo que quieren es obtener resultados finales de la simulación para estudiar aspectos muy concretos, como para cubrir las necesidades de aquellos otros, más inexpertos, que quieren aprender paso a paso y con detalle el funcionamiento del proceso o elemento simulado.

En la docencia de la materia Arquitectura de Computadores, el uso de simuladores ([2], [3] y [5]) en lugar de procesadores reales resulta bastante común debido a una serie de importantes ventajas: ofrecen un entorno más amigable que una máquina real, permiten seguir la evolución de ciertos comportamientos de una manera sencilla y visual, se pueden detectar mejor los errores, y todo ello sin modificar elementos físicos del computador. Efectivamente, debido a que están realizados en software, los simuladores se pueden modificar fácilmente para añadir nuevas instrucciones, construir nuevos sistemas o simplemente ofrecer más información.

Al igual que el procesador, el sistema de memoria es una componente fundamental en cualquier computador, y por ello sus características y funcionamiento son ampliamente considerados en los contenidos de las ingenierías en Informática. Es importante, pues, que también para esta materia se disponga de herramientas de simulación adecuadas para facilitar el aprendizaje de los alumnos.

Hay un grupo numeroso de simuladores de memoria ([2], [3] y [5]), muchos de ellos de la memoria cache, y menos de la memoria virtual. Son bastantes los que disponen de una interfaz tal que los hace difíciles de usar para aquellos que no tienen conocimientos básicos de su manejo, o desconocen los principios básicos de funcionamiento de una jerarquía de memoria. También, casi todos los simuladores actuales de memoria no muestran los resultados de la forma más clara o intuitiva posible. La mayoría sólo muestra resultados, y son los usuarios los que sacan las conclusiones a partir de los datos obtenidos y su conocimiento del funcionamiento. Son pocos los simuladores de memoria que muestren el proceso de simulación paso a paso, de forma que un usuario pueda entender el funcionamiento de las simulaciones.

En la revisión que se ha hecho sobre este tipo de simuladores, no se ha encontrado ningún simulador orientado a la práctica docente que implemente de forma visual toda una jerarquía de memoria (tanto cache como su interacción con memoria virtual), de forma configurable y que permita mostrar su evolución en el tiempo de una forma simple y comprensible.

En resumen, parece claro que es importante disponer de adecuadas herramientas de simulación en el desarrollo de la práctica docente universitaria. Y, además, no se tiene conocimiento de la existencia de un simulador orientado a la docencia, completo y de fácil manejo de un sistema jerárquico de memoria completo. Por tanto, se considera más que justificada la decisión de realizar el simulador que se presenta en este proyecto.

Se pretende que su uso sea didáctico, de forma que alguien que desconozca totalmente en lo que consiste una jerarquía de memoria pueda obtener, a través del manejo y ayuda del mismo, unos conocimientos más o menos completos de la composición y funcionamiento de una jerarquía de memoria y las distintas alternativas que existen para implementarla. La idea es que el simulador sea una aplicación totalmente interactiva con una interfaz sencilla y un funcionamiento muy descriptivo y didáctico, siendo flexible en cuanto a la visualización de los detalles del proceso de simulación.

Como se detallará en capítulos posteriores, el sistema de memoria simulado está formado por los niveles de cache, memoria principal y memoria virtual. Hay que indicar que la parte correspondiente al nivel de cache se corresponde básicamente con el simulador DineroIV, en la medida en que es éste uno de los mejores en su categoría. La parte de memoria virtual ha sido desarrollada completamente partiendo desde cero y añadiendo las características que se han considerado más interesantes. Estas decisiones sobre el diseño y la arquitectura se explican más adelante una vez estudiadas las posibles alternativas a la hora de construir el simulador.

1.2 Distribución del documento

Además de éste, el documento se ha dividido en otros seis capítulos.

- En el **Capitulo 2** se indican claramente los objetivos del proyecto y la metodología seguida para el desarrollo del mismo.

- En el **Capitulo 3** se incluye una descripción de varios simuladores del sistema de memoria de un computador, con especial atención a DineroIV.

- En el **Capitulo 4** se describe el proceso de análisis y diseño e implementación que se ha seguido para la elaboración del simulador.

- En el **Capitulo 5** se muestra el entorno del simulador a nivel de manejo.

- Más adelante, en el **Capitulo 6** se consideran las pruebas que se han aplicado al proyecto para comprobar su correcto funcionamiento. Como cualquier producto software con un mínimo de calidad el funcionamiento del simulador ha sido detalladamente testeado.

- Finalmente, en el **Capítulo 7** se indican brevemente las conclusiones principales y se comentan los posibles trabajos futuros, mejoras y posibilidades del simulador.

CAPITULO 2 OBJETIVOS Y METODOLOGÍA

2.1 Objetivos

El objetivo principal de proyecto es obtener una herramienta de simulación docente completa y de fácil manejo que permita poner en práctica los conocimientos fundamentales de un sistema jerárquico de memoria.

Aunque no es el objetivo principal, también se trata de dar al usuario un soporte completo en todo el proceso de simulación, de forma que, aunque no tenga demasiados conocimientos sobre la materia, pueda obtenerlos y madurarlos con la herramienta.

El objetivo principal se alcanzará siempre y cuando se logre cubrir una serie de objetivos parciales. Esos objetivos son los siguientes:

- **Estudio de los actuales diseños del sistema de memoria de los computadores**: se debe alcanzar el conocimiento necesario sobre la organización de los sistemas de memoria de los computadores.
- **Establecimiento de los requisitos del sistema a desarrollar**: se deben fijar los requisitos que el sistema deberá cumplir.
- **Realización del análisis y diseño del sistema**: ajustado a los requisitos, se debe realizar un análisis del problema y un diseño.
- **Implementación de la herramienta**: utilizando herramientas de programación se debe implementar la herramienta a partir del diseño previo.
- **Comprobación del correcto funcionamiento del simulador**: se debe asegurar el adecuado funcionamiento del simulador desarrollado.

Algunos de estos objetivos serán fácil y rápidamente alcanzados en la medida en que lo que hace falta para ello son conocimientos que han sido adquiridos a lo largo de los estudios realizados y sólo se trata de refrescarlos y aplicarlos. Otros, sin embargo, deberán ser cubiertos realizando una serie de tareas de acuerdo con una determinada metodología. La siguiente sección está dedicada, precisamente, a indicar cuáles han sido las tareas y pasos programados para realizar este proyecto.

2.2 Metodología

En principio, para la realización de todas las tareas que llevarán a completar la implementación del simulador, se seguirá un proceso de desarrollo ordenado de análisis, diseño e implementación. Para el completo desarrollo del proyecto se seguirán los siguientes pasos:

- **Revisión de los simuladores de memoria existentes:** se hará una revisión de todos los simuladores de memoria que se consideran "relevantes". Este análisis consistirá en un estudio de la interfaz, resultados, puntos fuertes y débiles.

- **Revisión de los conceptos básicos sobre sistemas de memoria:** revisión de los conceptos que formarán parte del simulador.

- **Estudio profundo de DineroIV:** se estudiará el simulador DineroIV y la forma de incorporarlo como base a la implementación del nuevo simulador.

- **Diseño del entorno:** se estudiarán las posibles opciones que deberá contener el entorno.

- **Añadir funcionalidades:** por medio de programación se creará la funcionalidad necesaria.

- **Realización de una batería de pruebas:** se realizarán pruebas siguiendo algún tipo de planificación, de los elementos que se consideren necesarios.

Algunas de estas tareas se podrán solapar en el tiempo, como es el caso de las tres primeras, mientras que otras deben ser iniciadas una vez concluidas las anteriores en el orden indicado.

Cada una de estas tareas tendrá el correspondiente reflejo en uno o varios de los capítulos en los que se ha dividido este documento. Así, la revisión de los simuladores y el estudio de DineroIV se encuadra en el capítulo dedicado al estudio del estado de la cuestión, la revisión de conceptos básicos de memoria en el capítulo de análisis, diseño e implementación, también ahí se encuadran el diseño del entorno y las funcionalidades, mientras que la parte de pruebas tendrá su propio capítulo dividido en la planificación y la realización de las mismas.

2.3 Conocimientos necesarios

La realización de este proyecto ha requerido la aplicación de conocimientos, habilidades y destrezas adquiridas en distintas asignaturas de la carrera. Como se trata de un producto software sobre una materia de Arquitectura de Computadores, el desarrollo del mismo ha requerido un conjunto de conocimientos, principalmente sobre:

- **Análisis y diseño:** las capacidades de análisis adquiridas en asignaturas como Ingeniería del Software, así como la capacidad de modelar un sistema completo con el lenguaje de modelado UML [1], han sido un punto importante en todo el proceso de desarrollo.

- **Estructuras de datos:** las estructuras de datos marcan el funcionamiento de todo el sistema, como se muestra en el anexo II son el pilar fundamental sobre el que se apoya toda la implementación.

- **Lenguajes y entornos de programación:** el conocimiento de los lenguajes de programación, que se adquiere en las asignaturas de Fundamentos de la Programación y que luego se va afianzando en los siguientes cursos, ha sido una pieza fundamental sin la que no se podría haber convertido en realidad toda la especificación.

- **Arquitectura de computadores:** ya que se trata de un simulador de jerarquías de memoria la base de todo el proyecto viene del conocimiento de las jerarquías de memoria, las principales estructuras, los conceptos de rendimiento y la organización.

CAPITULO 3 SIMULADORES DE MEMORIA

En este capítulo se incluye un conjunto representativo de simuladores del sistema de memoria, a diversos niveles, de computadores mono y multiprocesador. Para cada uno de ellos se describen sus principales características. Se trata de mostrar cuál es el estado del arte sobre herramientas de simulación del sistema de memoria.

3.1 Dinero

Cuando se habla de simuladores de memoria cache casi siempre aparece uno de los más potentes que existen y que es además uno de los más utilizados, es Dinero. Dinero es un simulador de memoria cache que permite configurar varios niveles de cache: distintas políticas de escritura en caso de acierto y en caso de fallo, distintas políticas de reemplazo de bloques, varios tamaños de bloque, y distintos tipos de asociatividad, entre otros. Dinero fue escrito en lenguaje C utilizando una estructura dinámica de listas para simular tanto la composición como el funcionamiento de las caches.

El simulador Dinero es un simulador guiado por traza, es decir, el sistema recibe como entrada un fichero de trazas de memoria que contiene las referencias de memoria que realizaría el procesador en un sistema real, durante la ejecución de un programa. El simulador lee dicha secuencia de referencias, las procesa y posteriormente devuelve una serie de resultados.

Dinero recibe los parámetros desde la propia línea de comandos. Esta forma de configuración tiene como ventaja que se pueden lanzar muchas simulaciones con distintos parámetros de una sola vez.

A continuación pasan a describirse las principales características, haciendo mención a la última versión disponible: DineroIV.

3.1.1 Formatos de entrada de datos

El simulador DineroIV soporta distintos formatos de entrada de datos:

PIXIE32 y 64: formato de 32 o 64 bits. En su formato reducido los 24 últimos bits representan el campo de dirección. Los 4 bits superiores de la dirección son los tiempos en ciclos pasados hasta que se produce un evento de traza. Los 4 bits siguientes codifican el tipo de evento.

BINARY: el formato binario, consiste en 32 bits de dirección, un byte para el tipo de acceso, y un byte de etiquetado.

DIN (versión extendida): El formato que más se utiliza es el formato DIN. Cada fichero tiene una línea por referencia a memoria, y está formado por dos campos separados por un espacio en blanco: el primero indica el tipo de acceso a memoria, y el segundo campo indica la dirección de memoria expresada en hexadecimal.

El tipo de acceso puede tener los siguientes valores:

1. Lectura de dato
2. Escritura de dato
3. Lectura de instrucción
4. Miscelánea
5. Post-escritura
6. Invalidación

El formato que más se utiliza es el formato DIN. Cada fichero tiene una línea por referencia a memoria, y está formado por dos campos separados por un espacio en blanco: el primero indica el tipo de acceso a memoria, y el segundo campo indica la dirección de memoria expresada en hexadecimal.

El tipo de acceso puede tener los siguientes valores:

Lectura de dato.
Escritura de dato.
Lectura de instrucción.
Miscelánea
Post-escritura
Invalidación

3.1.2 Formato de dirección

DineroIV utiliza direcciones de 32 bits, el formato de direcciones es de tamaño fijo, no así la forma de organizarlo en los campos etiqueta, ya que como se ha comentado anteriormente muchos de los parámetros que componen una memoria cache son configurables. Dependiendo de la configuración de la cache el formato será distinto (tablas 3.1 y 3.2).

Esquema	Número de conjuntos	Bloques por conjunto
Correspondencia directa	Número de bloques de la cache	1
Asociativa por conjuntos	Número de bloques de la cache/asociatividad	Asociatividad (normalmente de 2 a número limitado por el tamaño)
Totalmente asociativa	1	Número de bloques de la cache

Tabla 3.1 – Correspondencia y conjuntos.

Asociatividad	Método de selección	Comparaciones necesarias
Correspondencia directa	Índice	1
Asociativa por conjuntos	Índice del conjunto, búsqueda entre los elementos del conjunto	Grado de asociatividad
Totalmente asociativa	Buscar en todas las entradas de la cache	Tamaño de la cache
	Tabla de búsqueda separada	0

Tabla 3.2 – Correspondencia y método de selección.

3.1.3 Algoritmos que implementa

DineroIV implementa los algoritmos de reemplazo típicos, políticas de escritura en caso de acierto y políticas de escritura en caso fallo.

Cuando ocurre un fallo en una cache asociativa, se tiene que decidir qué bloque se reemplaza. En una cache totalmente asociativa, todos los bloques son candidatos. Si la cache es asociativa por conjuntos, se tiene que escoger entre los bloques del conjunto. Es obvio que el reemplazo es fácil en una cache de correspondencia directa porque sólo hay un candidato [6].

Existen varias estrategias a la hora de reemplazar bloques en cache. DineroIV implementa alguna de ellas:

-Aleatorio: los bloques candidatos son escogidos aleatoriamente.

-Usado menos recientemente (LRU): El bloque reemplazado es el que ha estado más tiempo sin utilizar.

-FIFO: El primero en entrar en cache es el primero en ser sustituido.

Otro punto importante a la hora de considerar los algoritmos que implementa DineroIV son las políticas de escritura. En este caso se deberá distinguir la política de escritura tanto en caso de acierto como en caso de fallo. En caso de acierto habrá dos posibilidades:

-Escritura directa: la información se escribe tanto en el bloque de la cache como en el bloque del siguiente nivel de la jerarquía de memoria.

-Postescritura (también llamada copia diferida o escritura cuando reemplazo): la información se escribe sólo en el bloque de la cache. Cuando el bloque es reemplazado es cuando se actualiza el nivel inferior de la jerarquía de memoria.

Por otra parte, en caso de fallo en cache del bloque a escribir se tienen las siguientes posibilidades:

-Carga en escritura: la información se escribe en el bloque en caso de fallo y éste es traído a cache en ese momento.

-No carga en escritura: la información se escribe sólo en el bloque de la memoria. El bloque no se trae a cache.

3.1.4 Otras características

DineroIV tiene algunas características más que lo hacen muy interesante.

-Número máximo de niveles: el simulador admite hasta 5 niveles de caches configurables.
-Tipo de caches: permite caches separadas, de datos o instrucciones, o unificada.
-Asociatividad: permite configurar caches asociativas, con distintos tamaños de conjunto.

3.2 Xcache

Xcache es un simulador diseñado para ser utilizado en el aprendizaje en arquitectura de computadores. Xcache permite simulación, visualización, ejecución paso a paso, de una traza formato “din” de entrada.

Xcache utiliza DineroIII como base para realizar sus tareas de ejecución paso a paso, soportando todas las opciones que vienen con este simulador.

Xcache fue creado por Herbert Grünbacher en la **Vienna University of Technology**.

3.3.1. Funcionamiento

Xcache permite mostrar los pasos que se siguen desde que el procesador hace una petición a memoria cache hasta que se reciben los datos de cache. Además, permite mostrar algunos datos como tiempos de acceso, que no se dan en DineroIII. En la figura 3.1 se muestra la ventana de configuración.

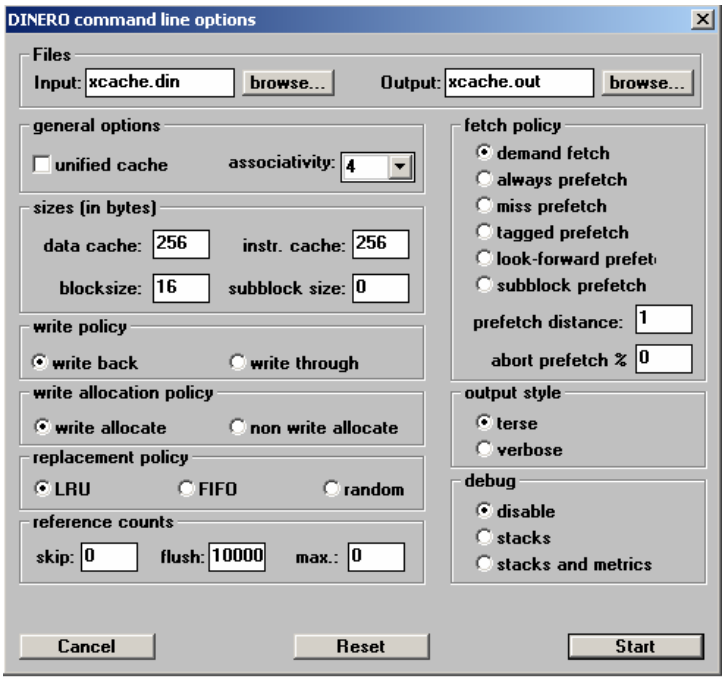

Figura 3.1 – Configuración de Xcache.

Xcache permite en todo momento ver el estado de las caches y de la memoria principal. Utiliza un código de colores para mostrar las distintas operaciones que se van produciendo en los bloques de cache. La idea principal de este simulador es poder navegar por distintas pantallas que dan una información más detallada de lo que está pasando en cada momento. A continuación se pasa a describir las más representativas (véanse figuras 3.2 y 3.3).

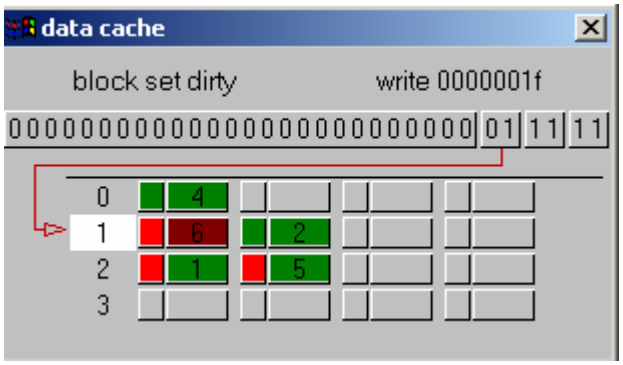

Figura 3.2 – Cache de datos en Xcache.

La ventana de visualización de cache contiene una lista, mostrando el contenido de la cache, además de campos de texto para ver las direcciones de memoria, que han sido incluidas en el fichero de entrada.

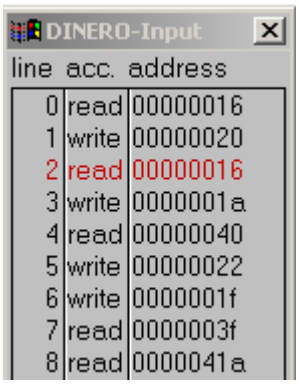

Figura 3.3 – Entrada de datos.

La ventana de visualización *fichero de entrada* (figura 3.3) contiene la forma de acceso, la línea y la dirección a la que se quiere acceder. En todo momento esta ventana muestra el estado de la ejecución del fichero de trazas, mostrando en color rojo la línea actual.

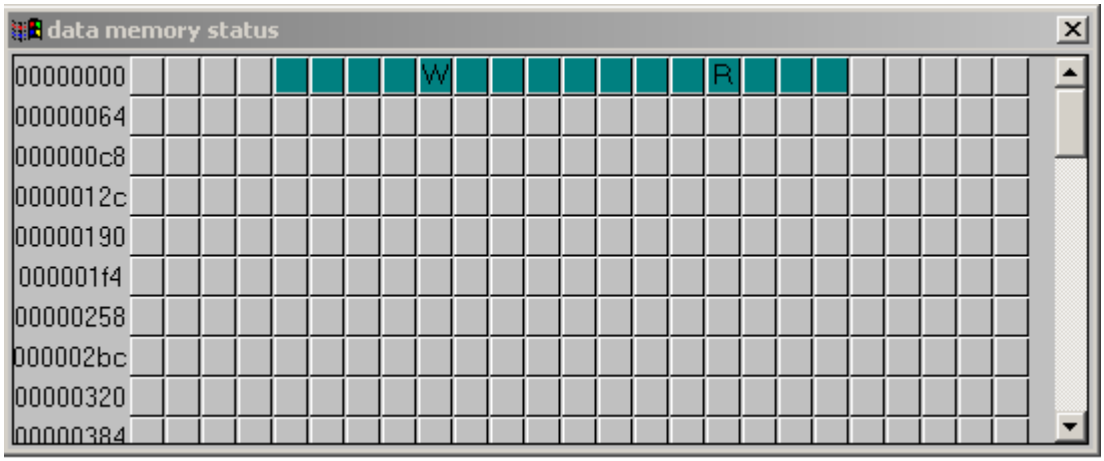

Figura 3.4 – Representación de la memoria principal.

La ventana de memoria muestra el estado de la memoria principal. Cada rectángulo corresponde a una palabra de memoria. Al principio de cada línea se puede ver la dirección del primer bloque. Los bloques que están en cache se muestran en

verde, los bloques que son cargados y desalojados aparecen en rojo. Un carácter dentro del rectángulo indica el tipo de referencia (W=escritura, R=Lectura) (figura 3.4).

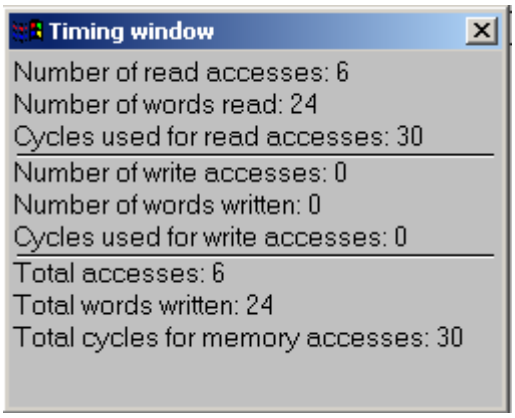

Figura 3.5 – Estadísticas de tiempos.

La ventana de tiempos (figura 3.5) muestra en cada momento datos estadísticos del acceso a memoria.

3.3.2. Puntos fuertes y débiles de Xcache

Xcache tiene muchos puntos fuertes. Quizás su mejor cualidad es la claridad de su interfaz para mostrar el funcionamiento de los distintos algoritmos de sustitución. Otro de sus puntos fuertes es su facilidad de configuración. Mientras que en DineroIII se tenía que trabajar con línea de comandos, aquí con una sola pantalla de configuración se puede definir cualquier tipo de configuración.

Finalmente, Xcache hace su propia aportación añadiendo tiempos de acceso a memoria, característica que no está incluida ni en DineroIII, ni tampoco en DineroIV.

En todo momento, Xcache utiliza como base el simulador DineroIII lo que ha llevado a una limitación en cuanto al número de niveles de cache. DineroIII sólo implementa un nivel de cache mientras que DineroIV ya permite hasta cinco niveles de cache.

Otro problema que plantea este simulador es que no permite ejecución en modo comando, de forma que se puedan lanzar varias simulaciones y luego obtener ficheros de resultados.

Por último, otro pequeño inconveniente es que aunque permite configurar de forma gráfica, no permite guardar y cargar esa configuración.

3.4 SMPCACHE

El simulador permite ver con detalle las transacciones que tienen lugar en un sistema multiprocesador con coherencia de cache. El simulador sólo sirve para sistemas con memoria de acceso uniforme (UMA). Implica que los sistemas a simular están basados en un único bus compartido por todos los procesadores. El tipo de protocolos de coherencia de caches en estos sistemas es el de sondeo, habiéndose implementado tres de estos protocolos: MSI, MESI y DRAGON. El MSI y el MESI son de invalidación, mientras que el DRAGON es de actualización.

SMPCache funciona bajo Windows y es totalmente visual y amistoso. El simulador funciona a partir de ficheros de trazas donde se especifica el tipo de acceso a memoria (instrucción, lectura o escritura) y la dirección a la que se accede. Estos ficheros de trazas están formados por dos columnas, en la primera se pone la operación y en la segunda la dirección hexadecimal.

3.4.1. Funciones básicas

El simulador permite establecer la configuración de los parámetros que definen la arquitectura del sistema, ofreciendo la respuesta del mismo al someterlo a los accesos de memoria generados por los programas (trazas de memoria que se hayan cargado en los distintos procesadores). Por tanto, se trata de una aplicación que podría encargarse de la evaluación de sistemas de memoria en multiprocesadores con fines de investigación (figura 3.6).

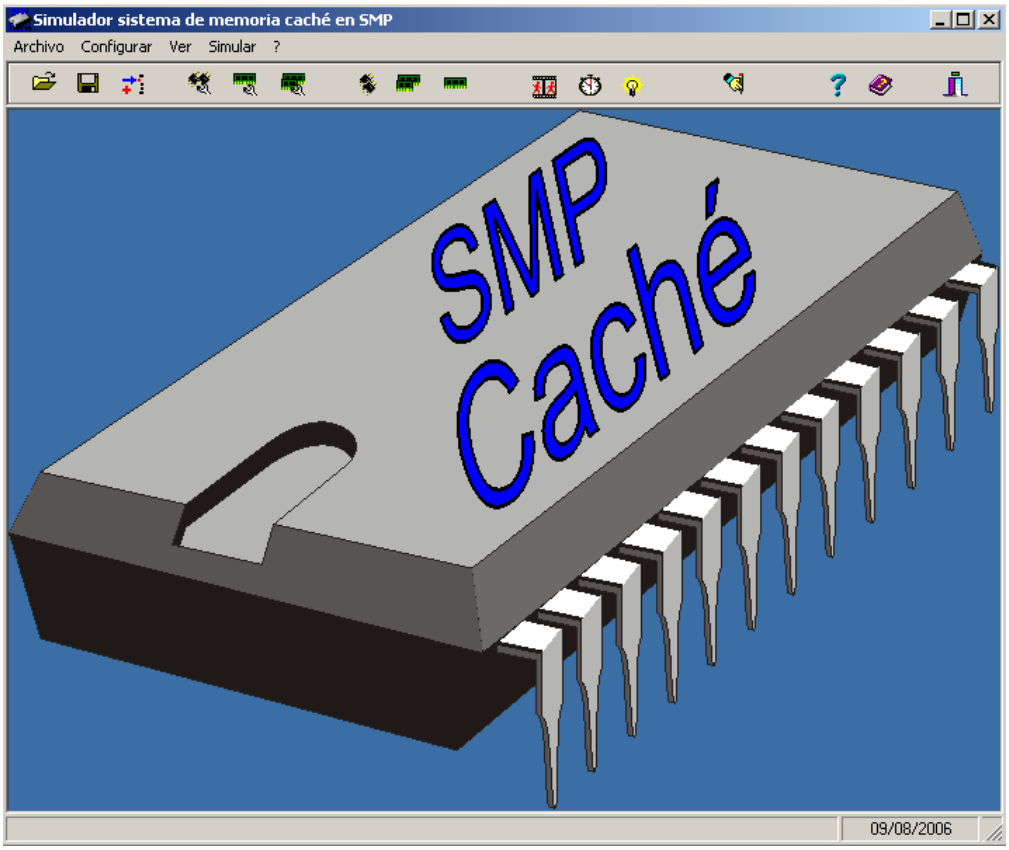

Figura 3.6 – Entorno SMPCache.

Dada su facilidad de manejo, el simulador también puede ser utilizado igualmente con fines didácticos, ya que permite observar de una forma clara y gráfica cómo va actualizándose el sistema completo a medida que se avanza en la ejecución de los programas (las trazas son leídas). Permite una visión completa de la evolución del multiprocesador, de una cache en concreto o de un determinado bloque de memoria. Muestra, en cada instante, los accesos a memoria realizados por cada procesador, el estado del bus, de cada cache, de cada bloque dentro de cada cache, y otros.

El simulador también permite estudiar el sistema de memoria que mejor se ajusta a unas determinadas necesidades antes de su implementación física, o simplemente ayuda a simular sistemas actuales para ver su eficacia y poder comparar los resultados de forma fácil. Algunos de los factores que se pueden estudiar con el simulador son: localidad de los distintos programas, influencia del número de procesadores, del protocolo de coherencia cache, del algoritmo de arbitraje del bus, del tipo de correspondencia, de la política de reemplazo, del tamaño de la cache (bloques en cache), del número de conjuntos de cache (en correspondencia asociativa por conjuntos), del número de palabras por bloque (tamaño del bloque), del ancho de la palabra, y otros.

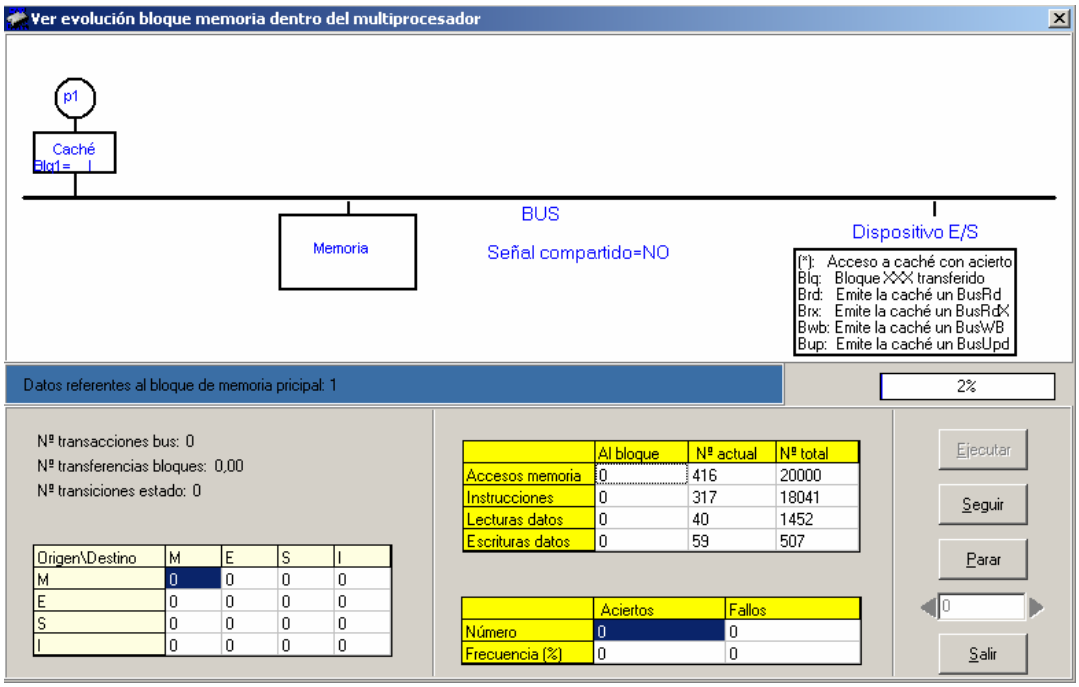

Figura 3.7 – Evolución memoria en SMPCache.

Además, SMPCache presenta mediante datos estadísticos (figura 3.7) y todo tipo de gráficas, medidas de interés como:

-Número de transacciones en el bus.
-Número de transferencias de bloques a través del bus.
-Tráfico en el bus teniendo en cuenta las dos medidas anteriores.
-Número de transiciones de estado (cada bloque en caché tiene un estado asociado).
-Número de transiciones de estado según el estado origen y el destino.
-Accesos a memoria realizados, desglosándolos por tipos: captura de instrucción, lectura de dato o escritura de dato.
-Número de aciertos y fallos de cache, así como frecuencia de aciertos y fallos.

Todos estos datos se presentan a distintos niveles (figura 3.7), aunque siempre teniendo en cuenta la interrelación de todos los elementos del sistema. Se puede, por tanto, realizar una simulación observando el multiprocesador completo, y todos los bloque de memoria o un único bloque. También se puede observar una cache específica, y todos los bloques de memoria o un bloque concreto.

La simulación realiza una aproximación programada de las operaciones que realizarían los componentes del sistema de memoria cache en un multiprocesador real. Para ello, se realizan los cálculos adecuados y, al mismo tiempo se van ofreciendo los resultados, tanto actuales como acumulados. Existen tres tipos de simulación (paso a paso, con punto de interrupción y ejecución completa) admitiéndose el cambio de un tipo de simulación a otro a mitad de la misma. También existe la posibilidad de abortar la simulación en cualquier instante, con el objetivo de corregir algún detalle de la organización.

3.4.2. Características técnicas

- Procesadores en el SMP: varía de 1 a 8
- Protocolos de coherencia cache: admite MSI, MESI (Illinois) y Dragon
- Algoritmos de arbitraje del bus: aleatorio, LRU o LFU
- Anchos de palabra (en bits): pueden ser 8, 16, 32 o 64
- Palabras en un bloque: varía desde 1 hasta 1024, en potencias de 2
- Bloques en memoria principal: varía desde 1 hasta 4194304, en potencias de 2
- Bloques en cache: varía desde 1 hasta 2048, en potencias de 2
- Funciones de correspondencia: pueden ser directa, asociativa por conjuntos o totalmente asociativa
- Conjuntos de cache (para asociativa por conjuntos): varía desde 1 hasta 2048, en potencias de 2
- Algoritmos de reemplazo: Aleatorio, LRU, FIFO o LFU
- Políticas de escritura: sólo admite post-escritura, debido a los protocolos de coherencia utilizados
- Niveles de cache: admite un único nivel
- Referencias: a nivel de palabra de memoria
- Tamaño de bloque máximo: admite bloques de hasta 8 Kbytes
- Tamaño de memoria principal máximo: es de 32 Gbytes
- Tamaño de cache máximo: es de 16 Mbytes.

3.4.3. Limitaciones y puntos fuertes de SMPCACHE

Entre los puntos fuertes de SMPCache se han enumerado:

-Ejecución en modo gráfico.
-Gran cantidad de datos estadísticos.
-Estadísticas muy detalladas incluso en modo gráfico.

En cuanto a las limitaciones que se pueden encontrar en este simulador, la principal es que no implementa la memoria virtual propiamente dicha, aún así no se puede decir que el simulador sea limitado, ya que se trata de uno de los más completos de los que se han estudiado.

3.5 PAGE

Es un simulador implementado para usos docentes. El programa muestra distintos algoritmos de reemplazos de páginas de memoria. El programa incluye cinco algoritmos de reemplazo de páginas: FIFO, segunda oportunidad, reloj, LRU con distintas variantes.

3.5.1. Funcionamiento

El simulador permite escoger el tipo de algoritmo a utilizar para el reemplazo de las páginas, véase figura 3.8. Para cualquiera se mostrará una pantalla en la que se podrá ir controlando la simulación y se mostrará de forma gráfica la estructura de datos que soporta el algoritmo, véase figura 3.9.

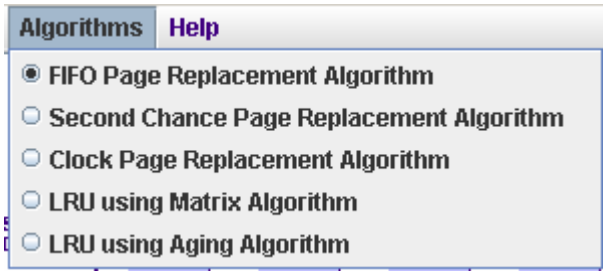

Figura 3.8 – Algoritmos para el reemplazo de página.

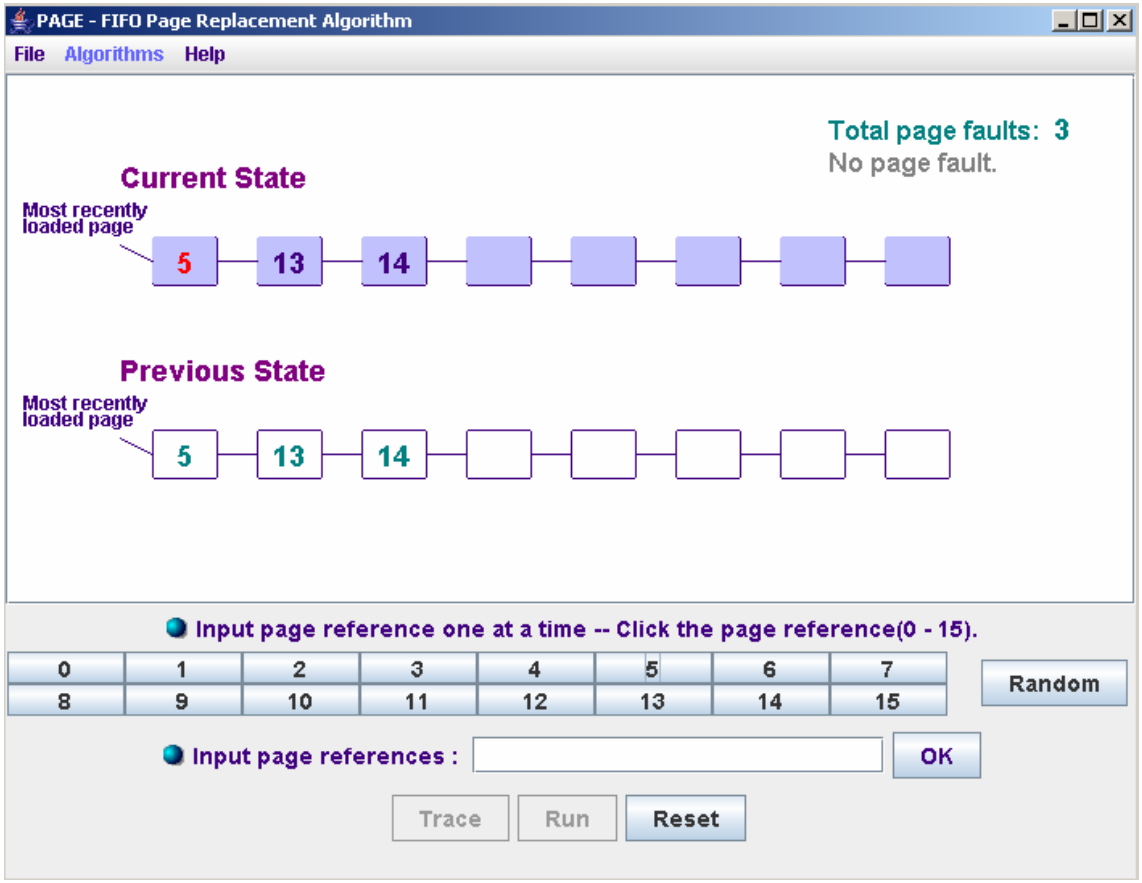

Figura 3.9 – Algoritmo FIFO en PAGE.

3.5.2. Limitaciones y puntos fuertes de PAGE

Entre los puntos fuertes de PAGE se pueden enumerar:

-Ejecución en modo gráfico.
-Entorno muy atractivo.
-Facilidad de uso.

En cuanto a las limitaciones que se pueden encontrar en este simulador, la principal es que es muy restringido en cuanto a las simulaciones que se pueden realizar, y el usuario tiene que controlar en todo momento la simulación.

3.6 MLFQ

Se trata de un simulador implementado para usos docentes. Implementa un sistema de CPU y entrada/salida indicando en cada momento, según una traza de entrada, el estado de ocupación de cada uno de los elementos.

3.6.1. Funcionamiento

En el simulador se podrá escoger entre los distintos tipos de trabajo que se quieren realizar (batch o regular). Permite escoger el tipo de algoritmo a utilizar para el desalojo de trabajos en el procesador o en el bus (figura 3.10). En la ventana principal se muestran los datos de la ejecución, el quantum asignado a cada trabajo y algunos datos más como el tiempo de llegada (figura 3.11).

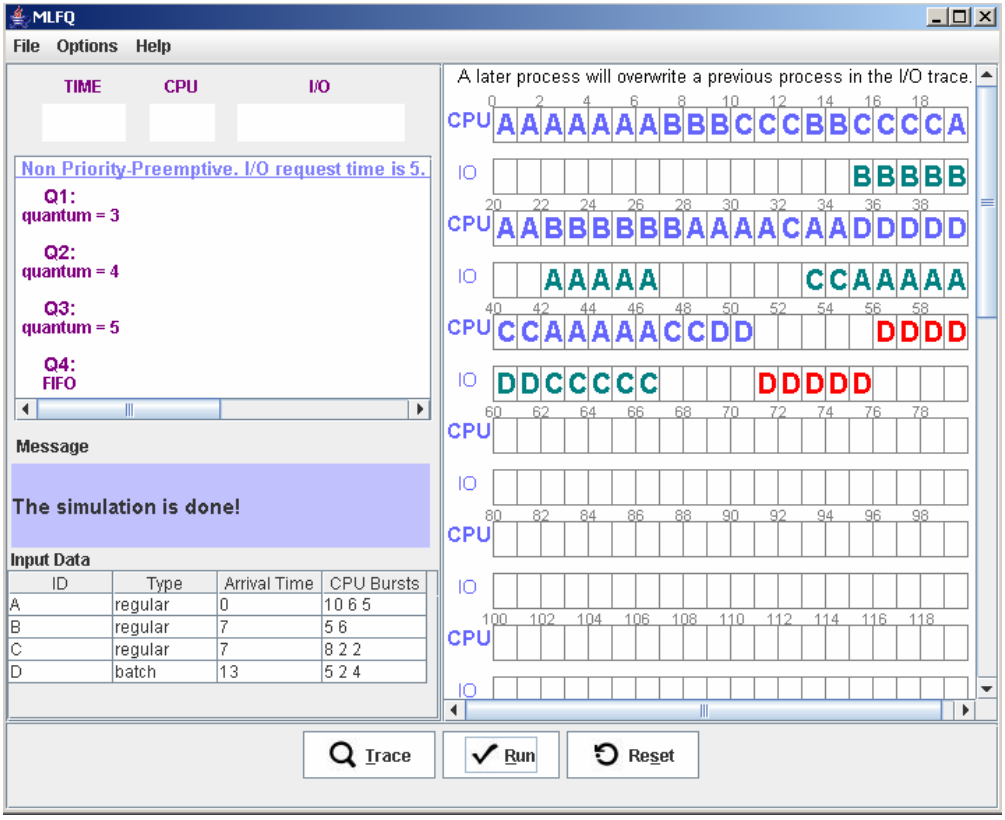

Figura 3.10 – Ejecución en MLFQ.

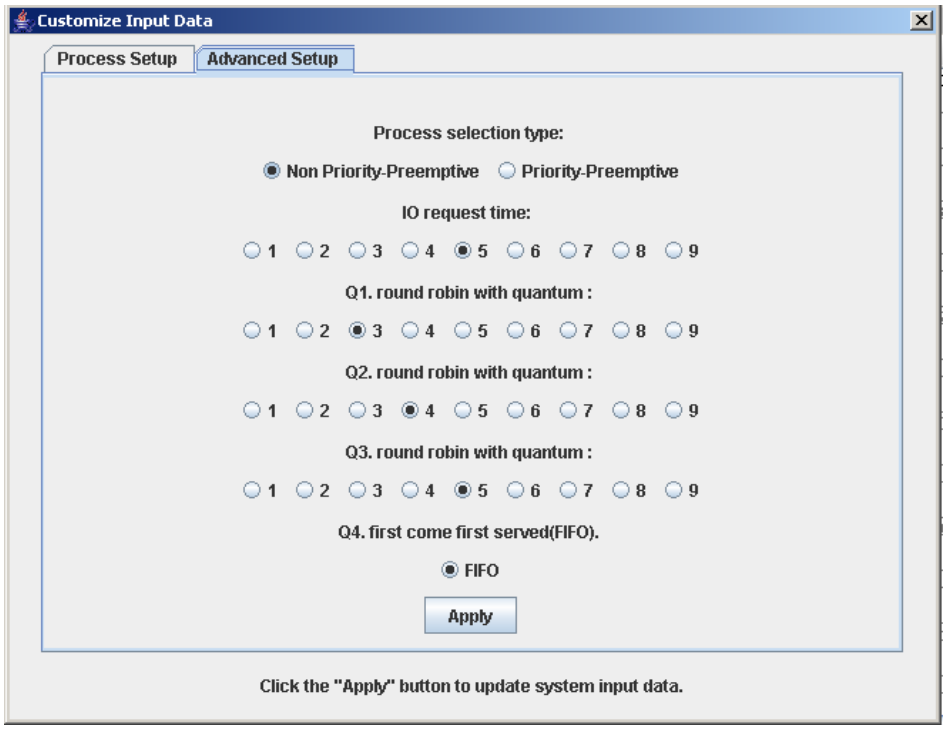

Figura 3.11 – Configuración en MLFQ.

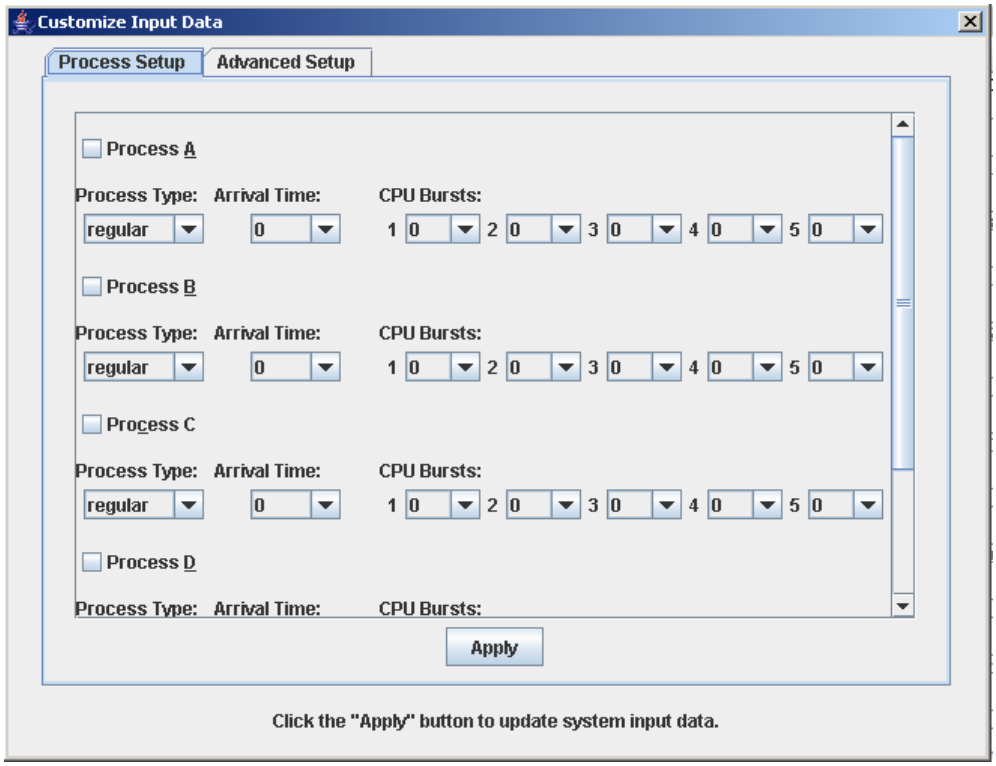

Figura 3.12 – Configuración avanzada de trabajos.

3.6.2. Limitaciones y puntos fuertes de MLFQ.

Entre los puntos fuertes de MLFQ se pueden enumerar:

- Ejecución en modo gráfico.
- Entorno muy atractivo.
- Facilidad de uso.

La limitación principal de este entorno es que no implementa ningún control específico de la memoria, aunque se producen escrituras y lecturas en disco. Además, después de la ejecución casi no se obtienen datos estadísticos.

No se puede considerar propiamente como un simulador de memoria.

3.7 Cache Demostrator

Es un simulador de caches que muestra paso a paso los distintos conceptos de caches de una forma gráfica, vía Web. El simulador ha sido creado con fines docentes.

3.7.1. Funcionamiento

Una vez se entra en la página del simulador se muestran distintas opciones. Pulsando sobre cualquiera de ellas se accede a distintas pantallas que permitirán configurar: políticas de reemplazo, tamaño de caches, tiempos, entre otros (figura 3.13).

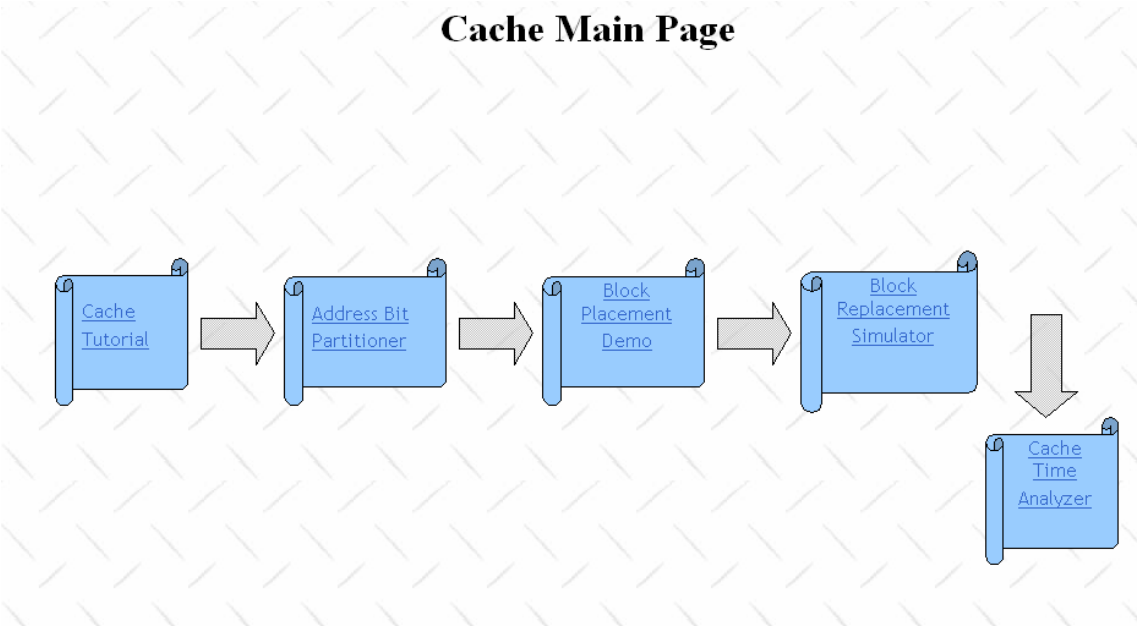

Figura 3.13 – Página principal de Cache Demostrator**.**

Para cada una de las opciones el simulador muestra, además de resultados, los conceptos que explican esos resultados (figuras 3.13 y 3.14).

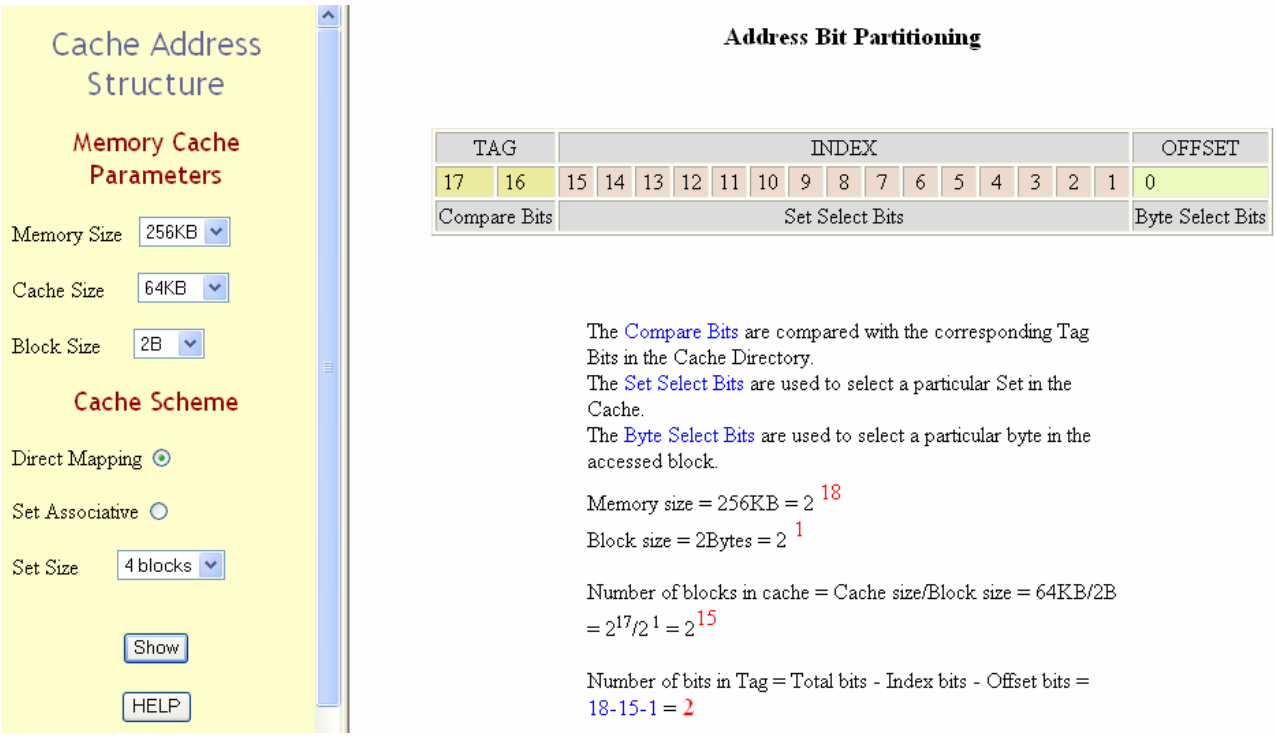

Figura 3.14 – Formato de direcciones.

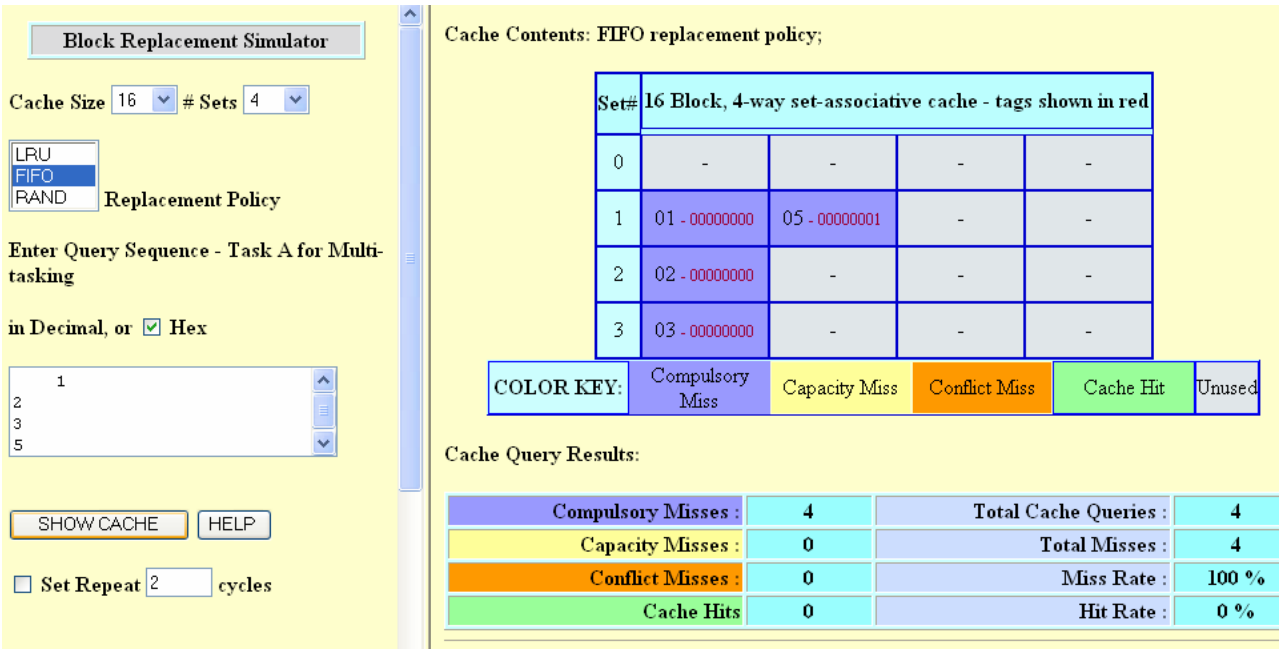

Figura 3.15 – Explicando las políticas de reemplazo

3.7.2. Limitaciones y puntos fuertes de Cache Demostrator

Entre los puntos fuertes se pueden enumerar:

- Ejecución en modo gráfico.
- Entorno muy atractivo.
- Facilidad de uso.
- Soporte en todo momento, muy completo y atractivo.

La principal limitación consiste en que el entorno está muy limitado a la hora de introducir datos y modificar configuraciones. Además, en todo momento el proceso tiene que ser guiado por el usuario. Quizás el punto débil más grande que tiene este simulador es que no tiene una opción para probar todas las opciones juntas.

3.8 Multitask Cache Demostrator

Ha sido creado con fines docentes Es un simulador de caches que crea una simulación de memoria cache con distintos elementos, vía Web [5].

3.8.1. Funcionamiento

El simulador se muestra como una página web donde se permite configurar distintos tamaños de cache, políticas de reemplazo y tiempos, entre otros (figura 3.16).

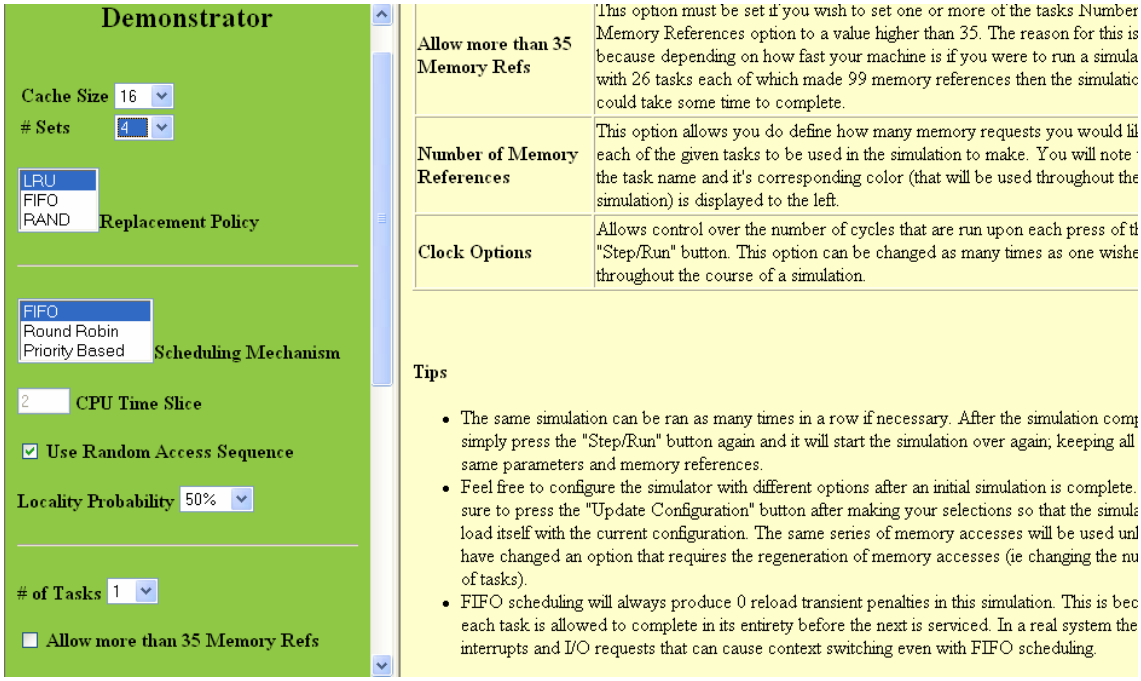

Figura 3.16 – Soporte ofrecido por el simulador.

Una vez configurado se muestra una tabla donde está representada la cache, y debajo de ésta irán apareciendo los datos estadísticos y referencias a la misma (figura 3.17).

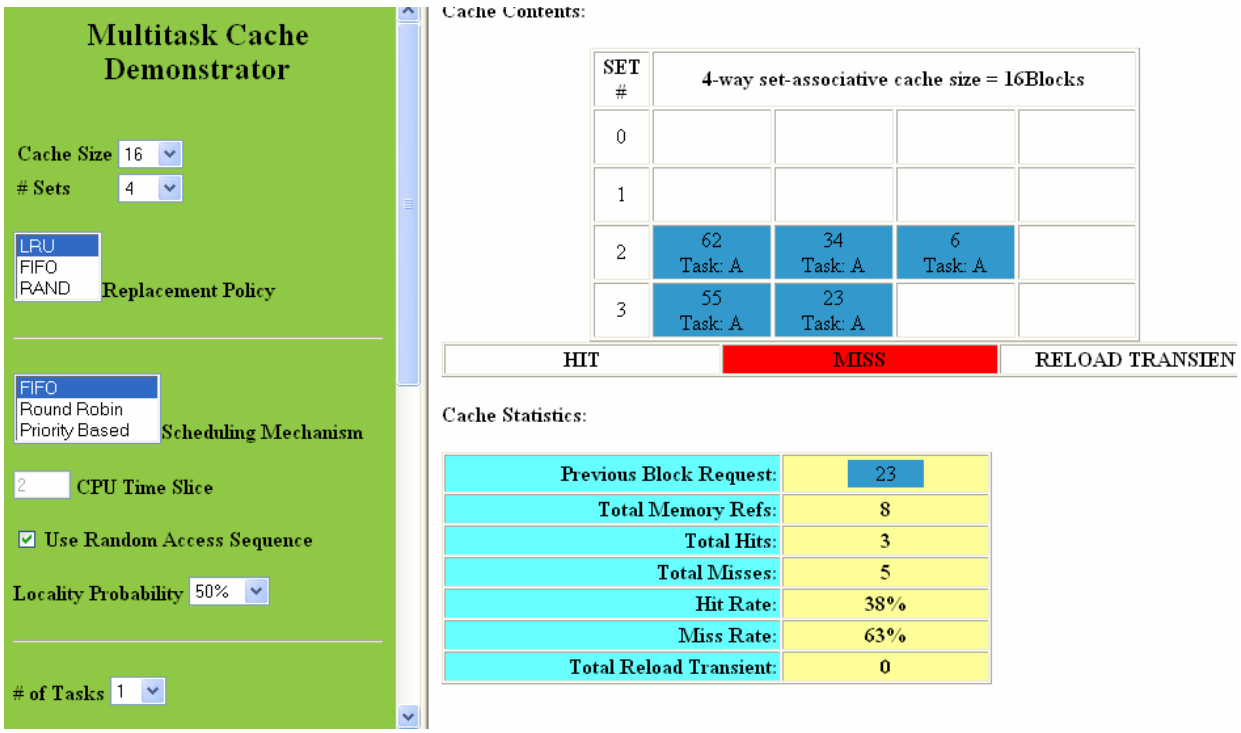

Figura 3.17 – Página principal Cache Demostrator.

3.8.2. Limitaciones y puntos fuertes de Multitask Cache Demostrator

Entre los puntos fuertes se pueden enumerar:

- Ejecución en modo gráfico.
- Entorno muy atractivo.
- Facilidad de uso.
- Soporte bastante completo.
- Implementa bastantes algoritmos de reemplazo.

Al igual que ocurre con el anterior simulador, la principal limitación consiste en que el entorno está muy limitado a la hora de introducir datos y modificar configuraciones. Aunque aquí el proceso es guiado por traza no hay forma de modificar la misma, por lo que el usuario no tiene demasiado control sobre la ejecución. No implementa ningún algoritmo de escritura.

3.9. Page Replacement Policies

Se trata de un simulador que funciona vía Web. El simulador muestra el comportamiento de los algoritmos de sustitución de páginas paso a paso y con soporte de ayuda en todo momento.

3.9.1. Funcionamiento.

El simulador se muestra como una página web donde se permite introducir distintas páginas y escoger el algoritmo de reemplazo (figura 3.18). El entorno no requiere ningún tipo de configuración, sólo el número de páginas en memoria (marcos de página). Una vez que el usuario ha escogido la política a aplicar y la página o páginas a referenciar puede ver a su derecha los resultados de la simulación (figura 3.19).

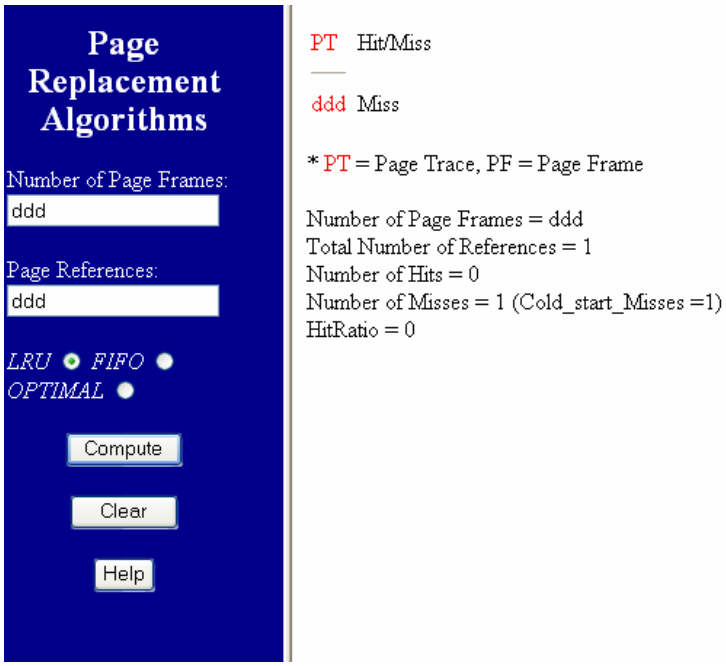

Figura 3.18 – Ventana de simulación.

3.9.2. Limitaciones y puntos fuertes de Page Replacement Policies

Entre los puntos fuertes se pueden enumerar:

- Ejecución en modo gráfico.
- Facilidad de uso.
- Implementa algunos algoritmos de reemplazo.

Aunque tiene sus puntos fuertes, el entorno es demasiado simple como simulador de memoria virtual, ya que no permite casi configuraciones y todo el proceso de simulación está guiado por el usuario.

3.10. Conclusiones

Del estudio de los simuladores que se han analizado a lo largo del capítulo, se desprende, que DineroIV es el simulador más completo de memoria cache, su único fallo consiste en no tener un entorno gráfico alternativo. Aunque Xcache podría ser ese entorno alternativo, sólo está preparado para la versión III de Dinero, que es algo más limitada. Así, una vez completado el estudio, se ha considerado DineroIV como la base de desarrollo del simulador, de forma que se integre totalmente en la parte de simulación de memoria cache.

En cuanto a los simuladores de memoria virtual, no se han encontrado muchos. Los que se han encontrado implementan varios algoritmos, pero no mezclan memoria virtual y cache, y ninguno permite demasiadas opciones de configuración. Por esta razón la parte de memoria virtual se implementará completamente nueva, basándose en alguna de las interfaces que se han visto anteriormente.

CAPITULO 4 ANÁLISIS, DISEÑO E IMPLEMENTACIÓN

4.1. Introducción

En este capítulo se ofrece una visión completa de los contenidos concretos de la herramienta de simulación a desarrollar, El capítulo se centra también en los aspectos que tienen que ver con el análisis y diseño del entorno de simulación.

La finalidad de esta sección será describir la especificación de los requisitos necesarios para el desarrollo del simulador, que permitirá al usuario realizar simulaciones de la jerarquía de memoria. Previamente han sido diseñados mediante modelado UML [1], mediante la incorporación de estructuras propias del diseño que se irá mostrando a lo largo del capítulo.

4.2. Análisis de requisitos

A continuación pasarán a describirse los conceptos básicos de los sistemas de jerarquías de memoria sobre los que se basará el desarrollo. Más adelante, se pasará a mostrar las principales características y los requisitos concretos del sistema.

4.2.1 Conceptos teóricos sobre el sistema de memoria

En esta parte se describe la especificación de los requisitos necesarios para el desarrollo del simulador. Dadas las características del modelo a simular, es imprescindible ajustarse a una serie de propiedades típicas de los sistemas de memoria. Por ello, se incluyen a continuación una serie de conceptos básicos sobre ellos.

A continuación se hará un breve repaso a los conceptos básicos de jerarquías de memoria que servirán de marco de trabajo para todo el proceso de análisis del problema.

Cache fue el término escogido para representar el nivel de la jerarquía de memorias entre la CPU y la memoria principal en el primer computador que tuvo ese nivel extra [6].

Cuando un procesador necesita obtener un dato que no se encuentra en sus registros, tratará de acceder a la memoria cache, y en el caso de que no se encuentre en la misma se accederá a los niveles superiores de la jerarquía (figura 4.1).

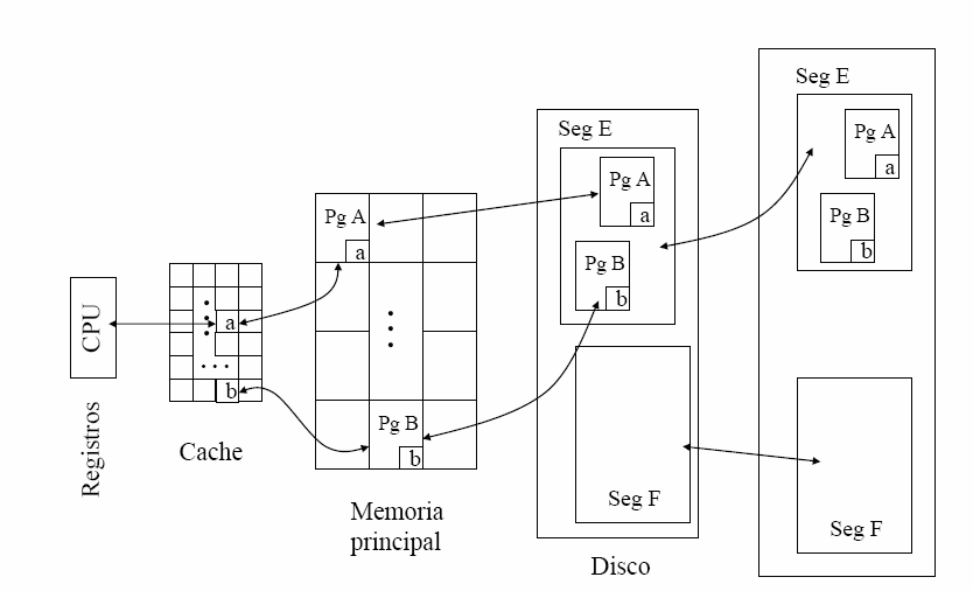

Figura 4.1 – Organización de un sistema jerárquico de memoria.

El primer problema que surge es saber cómo se determina si un dato está o no en cache. Para determinar si una palabra de la cache se corresponde con la palabra pedida se utilizan las etiquetas. Las etiquetas contienen la información de la dirección requerida para identificar si una palabra de la cache corresponde a la palabra pedida. La etiqueta puede estar formada sólo por la parte alta de la dirección, correspondiente a los bits que no se usan para indexar la cache [6].

Políticas de ubicación

Hay varios enfoques a la hora de determinar la localización de un bloque en cache:

Correspondencia directa: un bloque de memoria principal sólo puede ir a una posición de memoria cache. La forma de obtener esa posición suele ser **(Dirección del bloque) módulo (Número de bloques de la cache).** Se realiza una comparación de etiquetas para ver si el bloque está o no en esa posición de cache.

Correspondencia asociativa por conjuntos: la memoria cache se divide en conjuntos de forma que un conjunto está formado por dos o más bloques, y un bloque

de memoria principal podrá asignarse a cualquier bloque de cache de un conjunto dado. El número de bloques por conjunto se llama número de vías. Para saber cúal es el conjunto al que pertenece un bloque, se aplica la expresión: **bloque i módulo (nº de conjuntos).**

Correspondencia totalmente asociativa: un bloque cualquiera de memoria puede situarse en cualquier bloque de memoria cache. Para identificar el bloque se utiliza la etiqueta.

Políticas de reemplazo

Se ha descrito cómo se ubica un bloque en cache y cómo se determina si un bloque está o no en cache. Cuando un bloque debe ser sustituido por otro, se usan diversas políticas de reemplazo.

- **Emplazamiento directo:** un bloque sólo podrá ir a una posición, así que no habrá problema.

- **Emplazamiento asociativo y asociativo por conjuntos:** aquí la elección ya no resulta trivial, ahora las posibilidades pueden ser muchas. Para esto se utilizan distintos algoritmos. Se expondrán los que son de utilidad en este proyecto:

-Aleatorio: el bloque a sustituir se selecciona aleatoriamente.
-FIFO (First In First Out): es desalojado siempre el bloque que más tiempo lleva en cache.
-LRU (Least Recently Used): se reemplaza el bloque menos recientemente utilizado.

Cabe mencionar una implementación concreta del algoritmo LRU que es la basada en un **contador** en la que un contador es asociado a cada bloque de forma que el contador se va incrementando conforme van pasando ciclos sin acceder a ese bloque, el contador se pone a cero una vez el bloque se ha referenciado (el bloque con el contador más alto será el que se sustituya). También es interesante comentar la implementación LRU del **reloj** que utiliza una lista circular para implementar LRU de forma que un puntero se va moviendo en el sentido de las agujas del reloj actualizando unos contadores y en el caso de que se actualice un bloque el puntero se sitúa en él modificando su contador, de forma que ese siempre será el menos recientemente utilizado y el resto irán incrementando su contador.

Políticas de escritura

Se ha descrito lo que ocurre cuando se está leyendo un dato y se produce un fallo, pero normalmente no sólo hay lecturas de datos o instrucciones. Es también normal que haya escrituras, y éstas suelen ser mucho más costosas que las lecturas.

Según si se produce un fallo o un acierto se puede utilizar una política distinta de escritura.

-En caso de acierto:

- **Escritura inmediata (*Write Through*):** se actualiza la cache a la vez que la memoria principal.
- **Post-escritura (*Write Back*):** si hay que escribir y se produce acierto se escribe el dato sólo en memoria cache y se marca el bloque como que ha sido modificado. Cuando ese bloque sea desalojado de memoria cache se escribe en memoria principal.

-En caso de fallo:

- **Carga en escritura (*Write Allocate*):** se carga el bloque de memoria principal y se escriben los datos.
- **No carga en escritura (*No Write Allocate*):** se modifica el dato en memoria principal solamente, el bloque no se carga en cache.

Medidas de rendimiento

Hay algunas medidas del rendimiento de caches que son bastante interesantes y que se aplican directamente en la realización del proyecto. A continuación pasan a describirse.

-Tasa de fallos (m): m = referencias falladas / referencias totales

-Tasa de aciertos (h): h = 1 – m

-Tiempo de servicio en caso de acierto (tsa): tiempo medio que tardará la cache en obtener un dato en caso de acierto.

-Tiempo de servicio en caso de fallo (*tsf*):

tsf = *tsa* + *tpf*

tpf = tiempo de penalización en caso de fallo

-El tiempo de penalización por fallo es el tiempo de servicio en caso de acierto de los niveles superiores.

-Tiempo medio de acceso a memoria (*tma*) (modelo simple)

tma = tsa + m · tpf

Memoria virtual

Se ha indicado cómo las caches sirven para lograr accesos rápidos para los datos y el código usados más recientemente. De la mima forma, la memoria principal puede actuar como una "cache" para el almacenamiento secundario, normalmente en discos magnéticos. Se ha llegado al nivel de memoria virtual. Hay dos razones principales por las que se utiliza memoria virtual: permitir la compartición eficiente y sin peligros de memoria entre múltiples programas, y eliminar el inconveniente de tener un espacio de memoria principal pequeño y limitado [6].

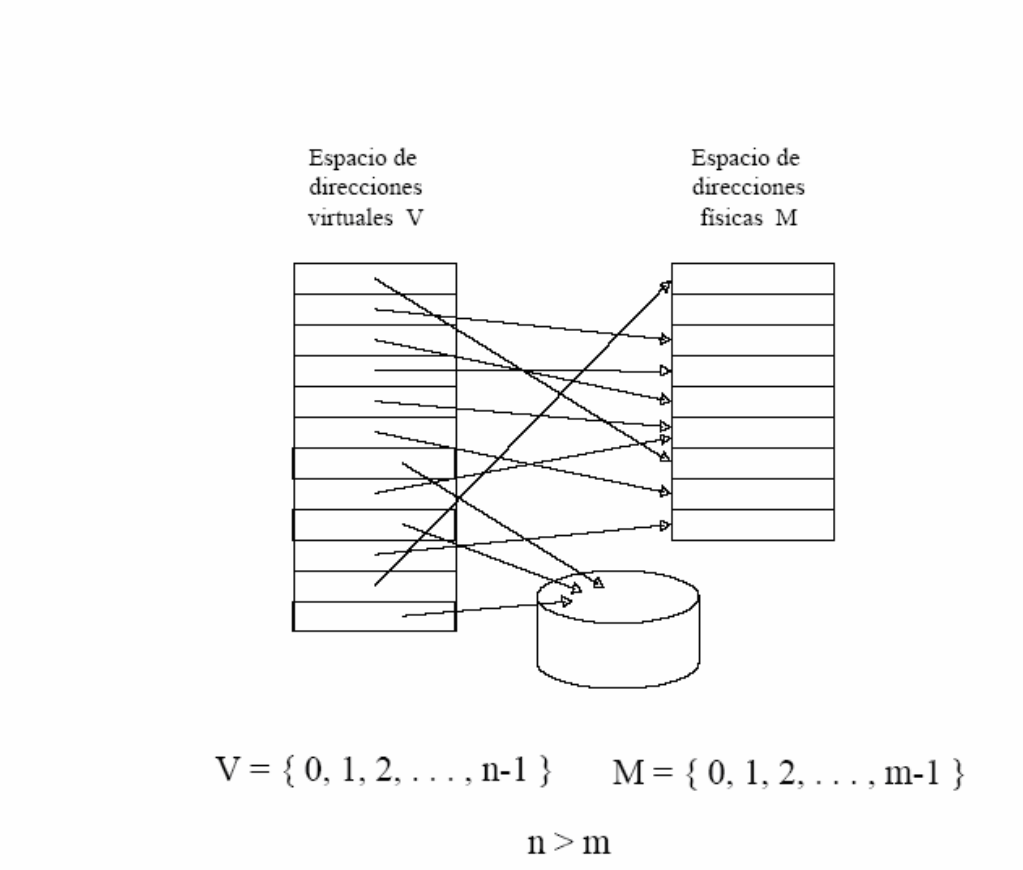

Figura 4.2 – Traducción de direcciones.

Hay distintas concepciones de la distribución del espacio aunque en este proyecto sólo se ha utilizado la paginación. Los conceptos de funcionamiento de la memoria virtual y de la cache son iguales, las diferentes raíces históricas han llevado al uso de diferentes terminologías. Un bloque de memoria virtual se llama página, y un fallo de memoria virtual se llama fallo de página. Con la memoria virtual, la CPU produce direcciones virtuales, que se traducen con una combinación de circuitería y software a una dirección física, que se usa para acceder a memoria principal [6] (figura 4.2)

La paginación divide el espacio de direcciones virtuales en bloques del mismo tamaño, el espacio de direcciones físicas en bloques iguales y con el mismo tamaño que las páginas de memoria virtual (es lo que se llamarán marcos de página o páginas físicas o bloques).

Cada dirección virtual se puede considerar como un par (p, n), donde p indica el número de página, y n el desplazamiento de la página.

Las páginas pasan de memoria principal a memoria secundaria, y viceversa, en un proceso bastante costoso, ya que el tiempo de acceso suele ser mucho mayor que el de acceso a memoria principal.

Un fallo de página requiere que el sistema operativo recupere el control de la ejecución, guarde la dirección virtual del fallo, posteriormente llevar a memoria principal la página que se ha solicitado, en caso necesario sustituir, y posteriormente el sistema operativo tendrá que recuperar la ejecución. Ya que el proceso es bastante costoso se suelen utilizar algoritmos de reemplazo y de escritura que realicen la menor cantidad de trasiegos entre memoria principal y memoria secundaria. Así, se suele utilizar LRU como algoritmo de sustitución de páginas y postescritura como política de escritura.

Traducción rápida de direcciones

A veces el acceso a la tabla de páginas es demasiado lento, o se accede de forma reiterada a la misma página o páginas. Este retraso “innecesario” se puede evitar guardando de alguna forma ciertas entradas de la tabla de páginas que están presentes en memoria principal en un buffer intermedio de forma que cualquier página que esté en este buffer obligatoriamente estará en la tabla de páginas y además también habrá sido accedida hace muy poco tiempo. Esto es lo que se llama Buffer de Traducción anticipada de direcciones (TLB) (figura 4.3).

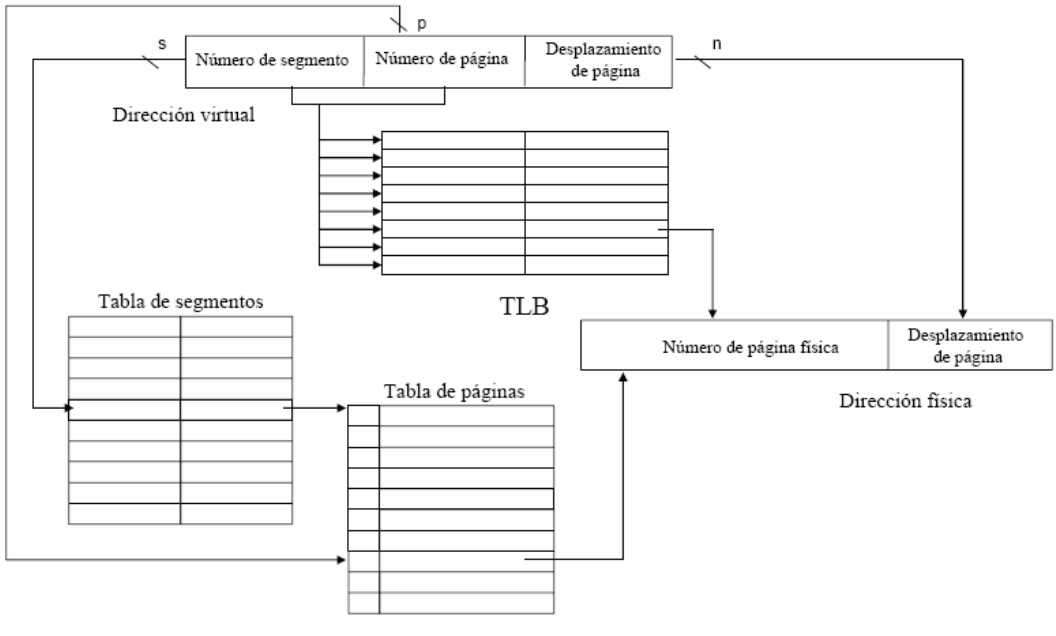

Figura 4.3 – Funcionamiento de un TLB.

.

Combinaciones de caches y memoria virtual

Se pueden utilizar distintas combinaciones de traducción entre caches y memoria virtual. Normalmente una cache trabaja con direcciones físicas y en el caso de que se produzca algún fallo deben haber sido traducidas previamente las direcciones virtuales en la tabla de páginas para recuperar de disco esos bloques. Aunque ese suele ser el funcionamiento normal también se pueden encontrar otras combinaciones de direccionamiento.

Existen tres tipos de memoria cache en cuanto a su forma de direccionamiento:

Cache física: tanto el índice de la cache como las etiquetas son de direcciones físicas, y no virtuales. El procesador genera direcciones virtuales, así que para acceder a cualquier nivel de cache tendrá que realizar todo el proceso de traducción completo, acceso a la tabla de páginas para comprobar si el bloque está en memoria y recibir la dirección física (figura 4.4).

Cache virtual: la cache usa tetiquetas de direcciones virtuales, por lo tanto, se puede acceder en paralelo al TLB y a la cache de forma que si no se produce un fallo de página no es necesario realizar la traducción (figura 4.5).

Cacher virtualmente indexada pero físicamente etiquetada: estas caches se indexan utilizando direcciones virtuales, pero usan etiquetas de direcciones físicas. El acceso a cache y al TLB se hace en praralelo, y las etiquetas de direcciones físicas de la cache se comparan con la dirección física procedente del TLB (figura 4.6).

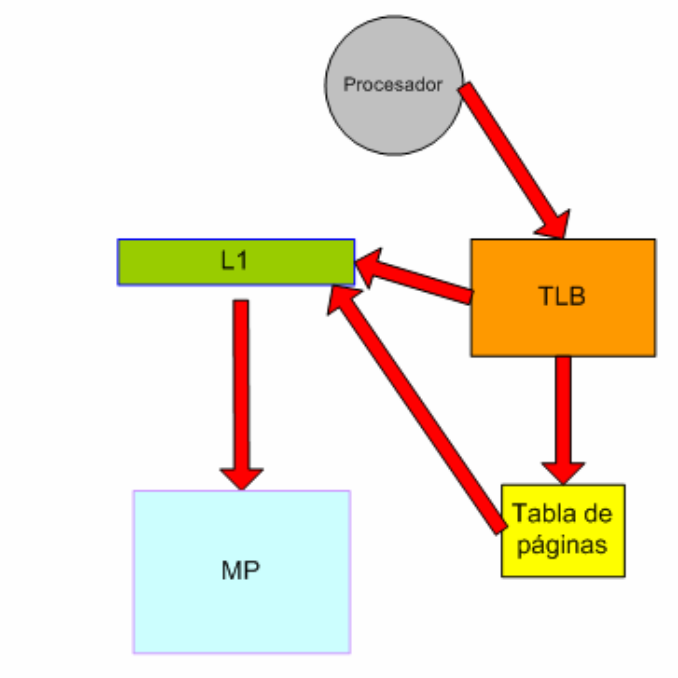

Figura 4.4 – Cache L1 física.

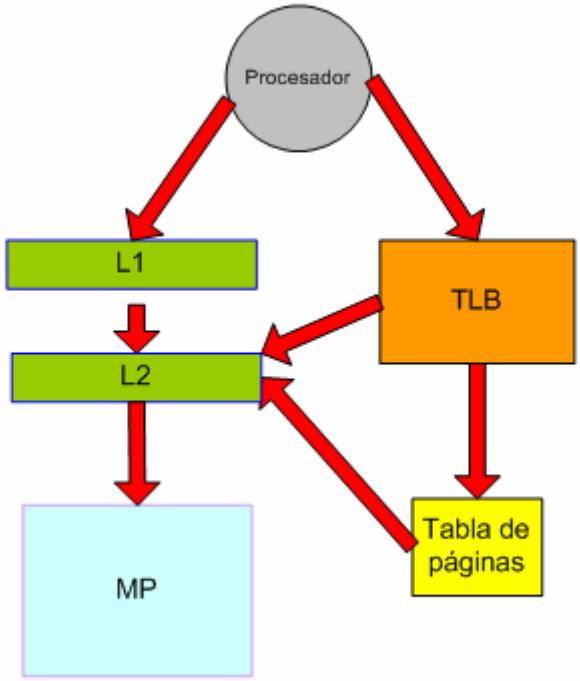

Figura 4.5 – Cache L1 virtual.

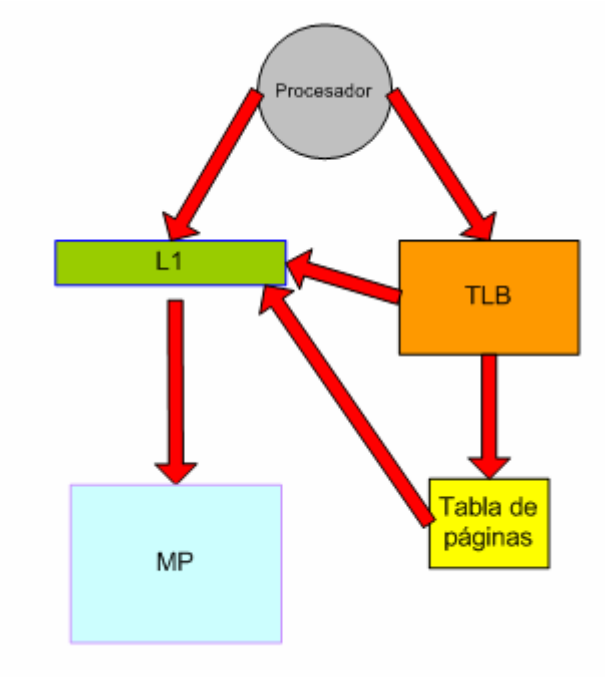

Figura 4.6 – Cache L1 virtualmente indexada pero físicamente etiquetada.

Una vez descritos los conceptos teóricos básicos que serviran al desarrollo se pasará a exponer los requisitos claramente, que será el objetivo del siguiente apartado.

4.2.2 Descripción de requisitos

Objetivo principal: el objetivo principal de este trabajo es crear un simulador que de forma visual permita simular los accesos a jerarquía de memoria que se producen en una ejecución completa de un programa tal y como lo hace una jerarquía de memoria virtual. En el anterior capítulo se han visto varios entornos de simulación, con sus limitaciones y sus puntos fuertes. A partir de ellos ([2], [3] y [5]), y pensando en utilizar algunas de las ideas que ya se estaban mostrando en los anteriores simuladores, se ha creado una especificación que en principio trata de ser lo más completa posible dentro de los límites que el tiempo y los propios objetivos del proyecto fin de carrera imponen.

Configuración: se persigue implementar un sistema de jerarquías de memoria totalmente configurable. Este sistema de jerarquía incluirá:

- *Múltiples niveles configurables de cache:* parámetros a configurar son el número de niveles de cache, la asociatividad, las políticas de búsqueda,

las políticas de reemplazo, las políticas de escritura y los tiempos de acceso por cada nivel.

- *TLB configurable:* se ha de permitir configurar un TLB, de forma que se pueda configurar unificado o separado (para datos e instrucciones), tiempos de acceso y el número de entradas que podrá almacenar el mismo.

- *Tabla de páginas configurable:* para configurar la tabla de páginas se introducirán el número de entradas de la misma, así como el tiempo de acceso.

- *Tabla de marcos de página configurable:* aquí se especificará el número de entradas máximo de la tabla de marcos de página, así como el tiempo de acceso y el tiempo de acceso a disco.

- *Memoria principal:* en la memoria principal además de los datos anteriores, se considerará como parámetro el tiempo de acceso.

Entrada de datos: la entrada de datos, así como el manejo de la simulación deben hacerse por medio de una traza de entrada en formato DIN.

Interfaz de usuario: la interfaz de usuario es una de las partes más importantes de este simulador. Debe ser fácil de utilizar y en todo momento debe mostrar el máximo de información posible. En principio, se optará en la medida que sea posible por utilizar un formato de representación parecido al que utiliza el simulador Xcache para representar una simulación de caches. El resto de estructuras se visualizarán en todo momento en forma de tablas. Además, se quiere que en una ventana se puedan ver todos los datos de la simulación (formato de direcciones, transformaciones, fichero de entrada). Esta ventana será la que controlará toda la simulación y según el elemento que se seleccione se podrá obtener cierta información adicional.

Ayuda: otra parte muy importante es la referente a la ayuda proporcinada a los usuarios. Un hipotético usuario tendrá que ser capaz, sin tener demasiados conocimientos de las jerarquías de memoria, de poder resolver todas sus posibles dudas sobre cualquier tema relacionado con el simulador o la teoría que le sirve de base. Además, se diferenciará entre una ayuda rápida que se dará al usuario en las distintas pantallas y una ayuda más completa por la que el usuario podrá navegar libremente y que contendrá conceptos mucho más ampliados con ejemplos de utilización del propio simulador.

4.2.3 Elementos de menú

Hay ciertas funcionalidades que han sido fijadas a priori como indispensables. Dichas funcionalidades son las típicas que pueden pedirse a un entorno de estas características. Éstas son:

- *Cargar traza:* este elemento de menú debe permitir cargar una traza en el simulador.

- *Cargar configuración:* con esta opción se debe poder cargar una configuración que previamente haya sido guardada.

- *Guardar configuración:* debe permitir guardar la configuración de los distintos elementos de memoria del simulador.

- *Crear o modificar configuración:* de alguna forma más o menos intuitiva debe poderse modificar una configuración.

- *Simular y ver el estado de la simulación:* una vez configurada la jerarquía debe poderse realizar una simulación a partir de una configuración y de una traza de accesos a memoria.

- *Ver estadísticas:* se deben mostrar de alguna forma los posibles resultados estadísticos de la simulación.

- *Modificar la distribución de ventanas:* una vez que se esté realizando la simulación las ventanas deben poder distribuirse con un menú de acceso sencillo de forma que se pueda modificar su distribución para ver mejor la evolución de la ejecución.

- *Acceso a la ayuda:* se debe poder acceder a la ayuda de forma rápida y corta y en un formato mucho más completo.

- *Salir:* Salir de la aplicación

4.3 Diseño

En el apartado que comienza se describirán los distintos modelos creados para realizar el análisis de requisitos del sistema, utilizando el lenguaje de modelado UML [6]. En la figura 4.7 se muestra el modelo de casos de uso extraído de los requisitos que se han comentado en la sección anterior.

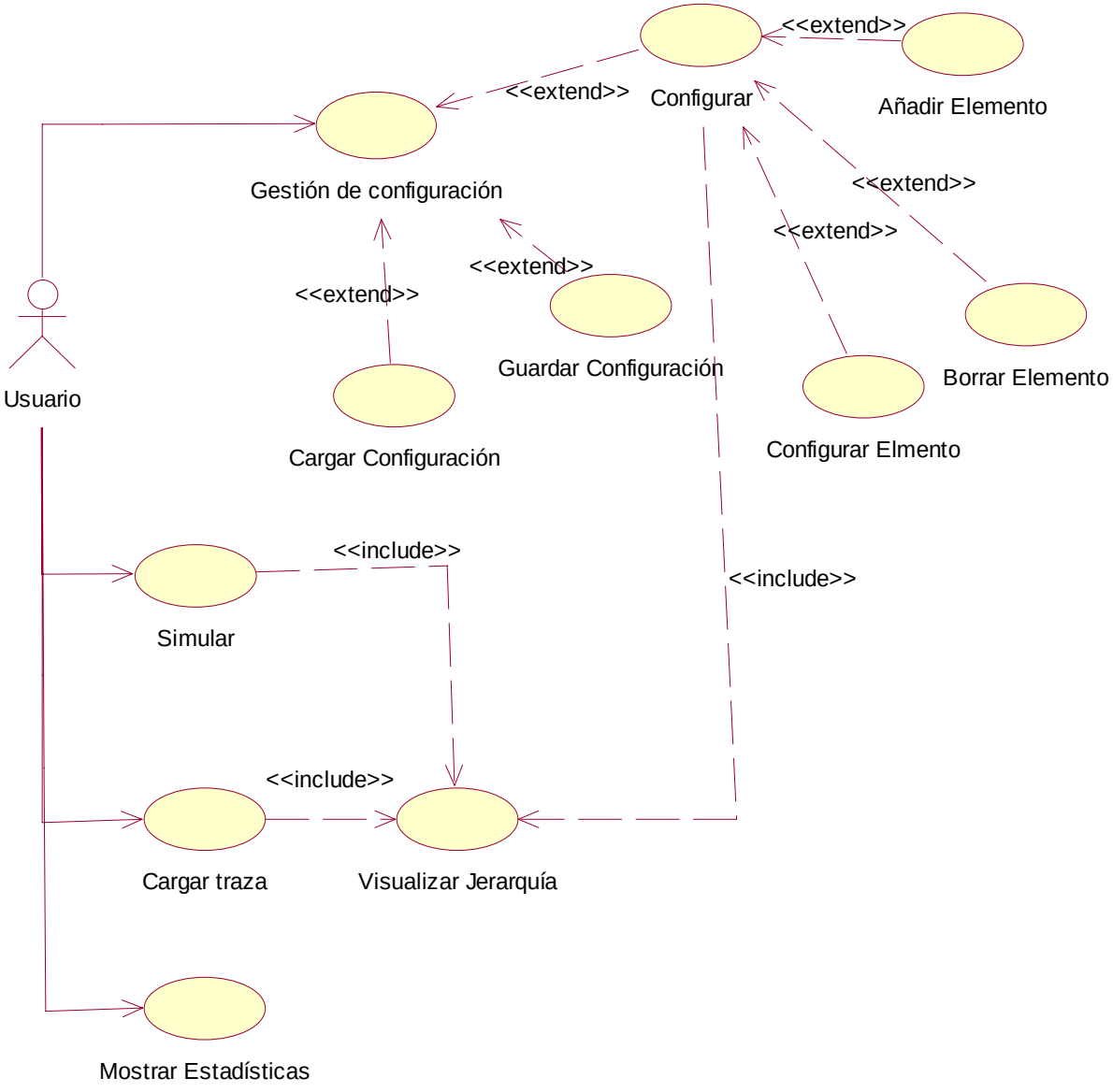

Figura 4.7 – Diagrama de Casos de Uso.

4.3.1 Descripción de los casos de uso

<table>
<tr><th colspan="2">Caso de uso Gestión de configuración</th></tr>
<tr><td colspan="2">Breve descripción

Este caso de uso contempla la gestión de la configuración del proyecto, es decir, la creación, modificación y eliminación de elementos de la jerarquía.</td></tr>
<tr><td>Flujo de eventos</td><td>Flujo Básico:

El usuario selecciona una opción del menú de configuración. A continuación se ejecuta la opción escogida.
Flujo Alternativo:

El usuario selecciona cualquier otra opción, como cargar traza o cierra el entorno.</td></tr>
<tr><td colspan="2">Poscondiciones

El programa queda preparado para empezar la ejecución en el caso de que una traza haya sido cargada. La configuración ha sido realizada correctamente.</td></tr>
<tr><td colspan="2">Excepciones
Hasta que la configuración que se ha introducido no sea correcta no se puede comenzar la ejecución.</td></tr>
<tr><td>Relaciones</td><td>Actores
Usuario
Extensiones
Configurar
Guardar configuración
Cargar configuración</td></tr>
</table>

Tabla 4.1 – Caso de Uso ’Gestión de configuración’

Caso de uso *Guardar configuración*	
Breve descripción	
Este caso de uso contempla cómo guardar una configuración.	
Flujo de eventos	**Flujo Básico:** El usuario selecciona la opción del menú de configuración. A continuación se ejecuta la opción y se guarda la configuración. Una vez cargada se muestra en una ventana. **Flujo Alternativo:** El usuario cierra sin salvar.
Poscondiciones	
Una vez que la traza ha sido guardada se podrá recuperar la configuración. La configuración ha sido realizada correctamente.	
Excepciones	
Si la configuración es errónea el programa mostrará un mensaje de error.	
Relaciones	**Extensiones** Extiende a Gestión de configuración.

Tabla 4.2 – Caso de Uso ’Guardar configuración’

Caso de uso *Cargar configuración*	
Breve descripción	
Este caso de uso contempla como cargar una configuración.	
Flujo de eventos	**Flujo Básico:** El usuario selecciona la opción del menú de configuración. A continuación se ejecuta la opción y se carga la configuración. Una vez cargada se muestra en una ventana. **Flujo Alternativo:** Si hay una configuración previa ésta será borrada y se aplicará la nueva.
Poscondiciones	
Una vez cargada, el programa queda preparado para la ejecución si la traza ha sido cargada. La configuración ha sido realizada correctamente.	
Excepciones	
Si la configuración es errónea el programa mostrará un mensaje de error.	
Relaciones	**Extensiones** Extiende a Gestión de configuración.

Tabla 4.3 – Caso de Uso ’Cargar configuración’

Caso de uso *Configurar*	
Breve descripción	
En este caso de uso se contempla la configuración de la jerarquía de memoria.	
Flujo de eventos	**Flujo Básico:** El usuario selecciona la opción del menú de configuración. A continuación se pueden añadir, eliminar o configurar elementos.
Poscondiciones	
Para que la jerarquía quede configurada al menos se debe tener un nivel de cache. Dependiendo del tipo de configuración, requerirá algunas condiciones especiales. La configuración queda guardada.	
Excepciones	
Si la configuración es errónea el programa mostrará un mensaje de error.	
Relaciones	**Extensiones** Extiende a Gestión de configuración.

Tabla 4.4 – Caso de Uso ’Configurar’

Caso de uso *Añadir elemento*	
Breve descripción	
Se añade un elemento a la jerarquía de memoria.	
Flujo de eventos	**Flujo Básico:**
	El usuario selecciona el tipo de elemento a añadir. La ventana se refresca mostrando el nuevo elemento.
Poscondiciones	
Todos los elementos que se pueden añadir tienen una cantidad limitada.	
Excepciones	
Si se sobrepasa el número de elementos quedarán deshabilitadas las opciones para añadirlos hasta que se elimine alguno de ellos.	
Relaciones	**Extensiones**
	Extiende a Configuración.

Tabla 4.5 – Caso de Uso ’Añadir elemento’

Caso de uso *Configurar elemento*	
Breve descripción Se configura un elemento de la jerarquía de memoria que previamente ha sido añadido.	
Flujo de eventos	**Flujo Básico:** El usuario selecciona el elemento de la jerarquía a modificar. Se muestra una ventana de configuración, que dependiendo del tipo de elemento permitirá un conjunto de opciones distintas. Se introducen los valores correctos. **Flujo Alternativo:** El usuario selecciona el elemento de la jerarquía a modificar. Se muestra una ventana de configuración, que dependiendo del tipo de elemento permitirá una cantidad de opciones distinta. Se introduce una configuración incorrecta. Se muestra un mensaje de error
Poscondiciones El elemento debe ser configurado correctamente. **Excepciones** Si se introduce una configuración no válida se producirá un mensaje de error.	
Relaciones	**Extensiones** Extiende a Configuración.

Tabla 4.6 – Caso de Uso ’Configurar elemento’

Caso de uso *Borrar elemento*	
Breve descripción	
Un elemento de la jerarquía que previamente se haya añadido es eliminado.	
Flujo de eventos	**Flujo Básico:** El usuario selecciona el tipo de elemento que desea borrar y la representación se refresca mostrando la jerarquía sin ese elemento.
Poscondiciones	
El elemento tiene que haber sido añadido o cargado previamente para poder eliminarlo de la jerarquía.	
Excepciones	
Dependiendo del tipo de configuración no se pueden eliminar ciertos elementos.	
Relaciones	**Extensiones** Extiende a Configuración.

Tabla 4.7 – Caso de Uso ’Borrar elemento’

Caso de uso *Visualizar jerarquía*	
Breve descripción	
Se realiza una representación de la jerarquía de forma que se pueda ver una configuración previamente realizada.	
Flujo de eventos	**Flujo Básico:** Se muestra al usuario la jerarquía configurada y lista para ejecución, además del fichero de entrada que dirigirá la simulación.
Poscondiciones	
La jerarquía tiene que tener al menos un elemento cache.	
Excepciones	
Dependiendo del tipo de configuración se mostrarán los elementos de una forma distinta.	
Relaciones	**Extensiones** Extiende a Configuración. Extiende a Simular. Extiende a Cargar traza.

Tabla 4.8– Caso de Uso ’Visualizar jerarquía’

Caso de uso *Simular*	
Breve descripción	
En este caso de uso se describe la ejecución de una simulación.	
Flujo de eventos	**Flujo Básico:** Se muestra en todo momento el estado de todos los elementos de la jerarquía. También se va produciendo una actualización continua de los mismos.
Poscondiciones	
Los datos de la ejecución deben haber sido almacenados.	
Excepciones	
Dependiendo del tipo de configuración se mostrarán los elementos de una forma distinta.	
Relaciones	**Inclusiones** Usa Visualizar

Tabla 4.9– Caso de Uso ’Simular’

Caso de uso *Cargar traza*	
Breve descripción	
En este caso de uso se describe la carga de una traza de accesos a memoria.	
Flujo de eventos	**Flujo Básico:** Se selecciona la correspondiente opción del menú y se escoge el fichero de trazas correspondiente. Se refresca la ventana de visualización de la simulación. **Flujo Excepcional:** Se produce un error al cargar la traza. La traza es demasiado grande. El formato no es el esperado.
Poscondiciones	
La traza queda guardada para ser accedida más adelante.	
Excepciones	
Dependiendo del tipo de entradas en la traza la forma de mostrarla será distinta.	
Relaciones	**Inclusiones** Usa a Visualizar

Tabla 4.10– Caso de Uso ’Cargar traza’

Caso de uso *Mostrar estadísticas*	
Breve descripción	
El usuario recibe información de la simulación.	
Flujo de eventos	**Flujo Básico:** Se selecciona la correspondiente opción del menú. Se abre una ventana que permite ver al usuario las estadísticas de ejecución.
Poscondiciones	
Si no hay elementos configurados no es posible el acceso a esta opción.	
Excepciones	
Dependiendo del tipo de entradas en la traza la forma de mostrarla será distinta.	
Relaciones	**Inclusiones** Usa a Visualizar

Tabla 4.11– Caso de Uso ’Mostrar estadísticas’

4.4 Arquitectura

A continuación se dará una visión parcial de la arquitectura del sistema, mostrando distintos aspectos del mismo. En principio, la arquitectura es muy sencilla, por lo que no se entrará en ningún tipo de detalle demasiado fino.

Como se ha comentado desde el principio del proyecto uno de sus principales objetivos es el incorporar toda la funcionalidad de DineroIV a la interfaz de simulador que se ha creado. De esta forma gran parte del código fuente que implementa la simulación de jerarquías de cache se ha mantenido o modificado respecto a las librerías que incluye dinero IV.

Así pues, se pasa a describir en forma de diagrama de clases los elementos que se han considerado imprescindibles. Por sencillez, se han omitido métodos y atributos de las clases, ya que no aportan información relevante.

El diseño se divide en dos paquetes, por un lado tenemos un paquete que contiene toda la lógica de la simulación, mientras que por otro lado tenemos un paquete que se encarga de la representación. A continuación se muestra el diagrama de paquetes:

Figura 4.8 – Diagrama de paquetes.

La vista de los casos de uso nos ha permitido identificar toda la funcionalidad que la aplicación deberá soportar. A continuación la descripción de las clases dará una vista más cercana a la implementación. Se han omitido métodos y atributos, ya que la información que ofrecen no es interesante a la hora de analizar la aplicación a grandes rasgos.

Las clases que implementan la jerarquía se encuentran en el paquete de lógica, la gran parte del trabajo de implementación se ha centrado en la parte de diseño de memoria virtual.

Los números que aparecen en el diagrama de clases indican la cardinalidad máxima, por ejemplo el número de caches (0..4) significa que puede haber de 0 a 4 caches.

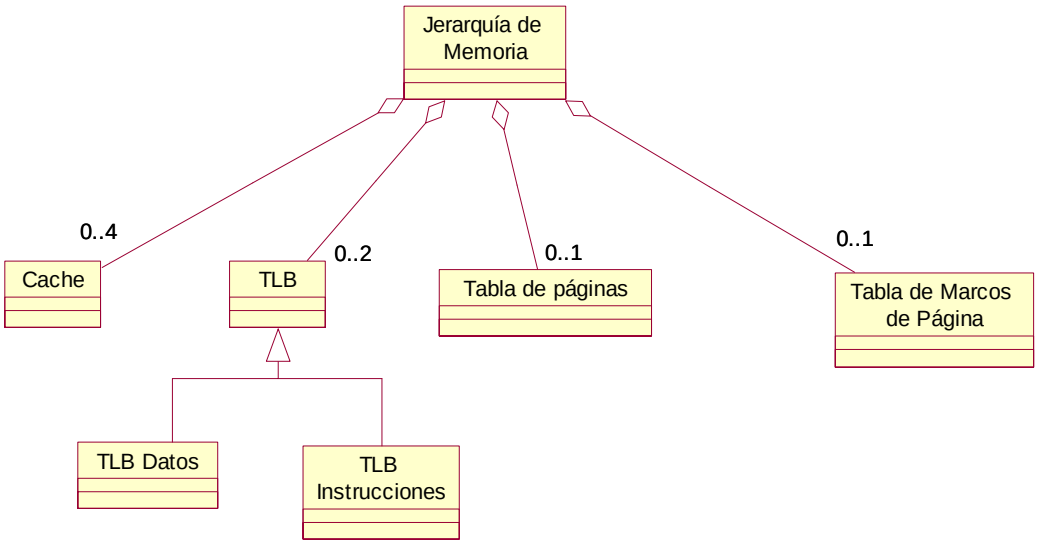

Figura 4.9 – Diagrama de clases paquete Lógica.

Jerarquía de Memoria: es la clase que representa una jerarquía de memoria y que va a contener todo el soporte para el manejo de una jerarquía de memoria.

Tabla de Marcos de Página: es la clase que representa una tabla de marcos de página. Tendrá todas las operaciones necesarias para sustituir un marco de página, hacer una referencia, así como para guarda las estadísticas que sean necesarias.

Tabla de páginas: es la clase que representa una tabla de páginas. Al igual que las demás contendrá todas las operaciones necesarias para manejar la tabla de páginas.

TLB: es la clase que representa un TLB. Un TLB puede ser de datos o de instrucciones.

Cache: esta clase contendrá toda la lógica de DineroIV. Es una de las partes principales del simulador ya que sobre ésta se realizan todos los accesos posteriores.

Las clases que implementan la representación de una simulación, supone la parte principal de la aplicación ya que sin una representación adecuada no tiene sentido la implementación del simulador.

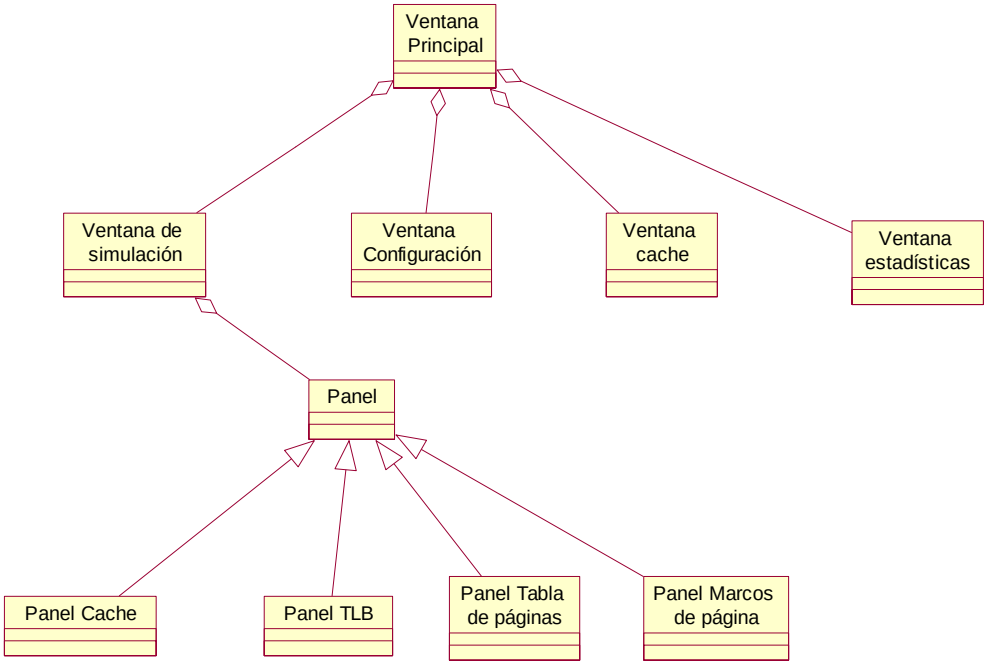

Figura 4.10 – Diagrama de clases paquete Representación.

Ventana Principal: Es la clase que va a contener la interfaz de usuario, así como el soporte para la ayuda. Además, servirá de marco para el resto de componentes. Esta ventana está compuesta por paneles cada uno de los cuales representará a un componente distinto de memoria principal.

Ventana Simulación: esta clase representará la simulación. Aunque no se trata de la ventana principal es la ventana más importante ya que es el núcleo de la simulación.

Ventana Cache: esta clase representará la simulación de una cache, la cache es uno de los componentes de la jerarquía que más se deben cuidar en la representación de la simulación. Este componente se actualiza según la simulación avanza.

Ventana estadísticas: esta clase representará a la ventana que dará las estadísticas de la simulación.

El resto de elementos del diagrama no tienen gran importancia por lo que no se han descrito.

4.5 Implementación

A continuación se comentarán aspectos de la implementación que se ha realizado en este proyecto. Sólo se hará un comentario a grandes rasgos, desde el lenguaje escogido y el entorno de programación, así como las partes que se han podido reutilizar de otros simuladores.

El lenguaje que se ha utilizado, como se comentaba al principio, ha sido el lenguaje C++ ¿Por qué utilizar un lenguaje que está dejando de utilizarse y para el que no hay grandes comunidades de desarrolladores trabajando? La respuesta es sencilla, por comodidad. Desde el principio, uno de los objetivos del proyecto fue poder incorporar todos los algoritmos que implementaba el simulador DineroIV. Este simulador está completamente escrito en lenguaje C, por lo que la conversión a C++ es casi inmediata. Además, hay entornos de programación bastante potentes de C++ que permiten programar de forma bastante organizada y documentada.

Entre los problemas que se plantearon en la implementación surgieron algunos problemas con la compatibilidad del código en lenguaje C de DineroIV, entre ellos: gestión de memoria, incompatibilidad de funciones y problemas al ensamblar los distintos módulos. Esta es la razón por la que se escogió C++ Builder de Borland, a parte de ser un entorno bastante intuitivo y fácil de usar. Hay otros productos comerciales con una tecnología parecida y de la misma generación como es Visual C de Microsoft, en principio el fundamento es el mismo aunque con bastantes variantes.

El código se ha estructurado usando clases de forma que cada una implemente una tarea lo más específica posible. En el caso de la clase cache en todo momento se ha utilizado el antiguo código de DineroIV, modificando lo poco que es necesario para tener una clase que pudiera implementar cualquier jerarquía de caches. En principio, el funcionamiento de esta parte en ningún momento ha debido cambiar, por lo que se trabaja con el código como si de una caja negra se tratase. De esta forma, esta parte ha sido como el resto testada a parte para comprobar que se mantiene el funcionamiento que tenía el simulador DineroIV. Esta parte se detalla completamente en el capítulo 5 y en el anexo 2.

4.6. Limitaciones

Aunque el diseño ha tratado de ser lo más completo y sencillo posible, esto supone que la representación, sobre todo en lo que se refiere a la interfaz, puede ser prácticamente de cualquier forma. De este modo la interfaz se podría diseñar prácticamente como se quiera. Pero anteriormente ya se han analizado bastantes entornos y ya se tiene más o menos una idea clara de lo que se quiere como interfaz. Aún así, no se han podido incluir todas las características –ni a la interfaz ni a la implementación de la memoria virtual- como habrían podido desearse.

Entre las limitaciones que se pueden enumerar, las más obvias son:

- Pantalla de **estadísticas más completa** de forma que se puedan generar en modo gráfico historiales de ejecución o comparar varias ejecuciones.

- Permitir una **configuración mucho más versátil** de las estructuras de memoria que se pueden utilizar, así como permitir crear cualquier tipo de componente que se quiera. Habiendo definido previamente su comportamiento y propiedades.

CAPITULO 5 ENTORNO

5.1. Introducción

A continuación se describe el entorno de simulación que se ha desarrollado. La descripción se centrará en las opciones básicas, sin entrar en demasiado detalle pues en este punto no se trata de proporcionar una guía de usuario, sino que simplemente se quieren mostrar los rasgos más representativos y dar una visión general.

5.2. Características técnicas

Se pasa a describir las principales características del simulador (tabla 5.1).

Características	*Máximo*
Niveles cache	1 como mínimo, 4 máximo.
Bloques por conjunto para cache	1 como mínimo, máximo ilimitado, pero los tamaños parciales limitados según el tamaño de cache y de bloque.
Nº de conjuntos para cache	1 como mínimo (memoria directa), máximo ilimitado, pero los tamaños parciales limitados según el tamaño de cache y de bloque.
Algoritmos de reemplazo para cache	LRU, FIFO y aleatorio.
Políticas de búsqueda para cache	Por demanda, siempre prebúsqueda, prebúsqueda con fallo, prebúsqueda con marcado, carga adelante y subbloque.
Políticas de escritura en caso de acierto en cache	Postescritura y escritura inmediata.
Políticas de escritura en caso de fallo en cache	Carga en escritura y no carga en escritura.
Tipo de acceso a cache	Física, virtual y virtualmente indexada pero físicamente etiquetada.
Tipo de cache	Unificada o separada.
TLB	Unificada o separada.
Número entradas TLB	1 como mínimo, 50000 máximo.
Número entradas TP	1 como mínimo, 50000000 máximo.
Algoritmo de reemplazo de páginas (Memoria virtual)	LRU

Políticas de escritura en caso de acierto(Memoria virtual)	Postescritura
Políticas de escritura en caso de acierto (Memoria virtual)	Carga en escritura

Tabla 5.1 – Características técnicas.

5.3. Perfiles y modelos de Uso

Se comenzará describiendo paso a paso cada una de las ventanas más importantes de la aplicación, y cómo serán mostradas a un usuario.

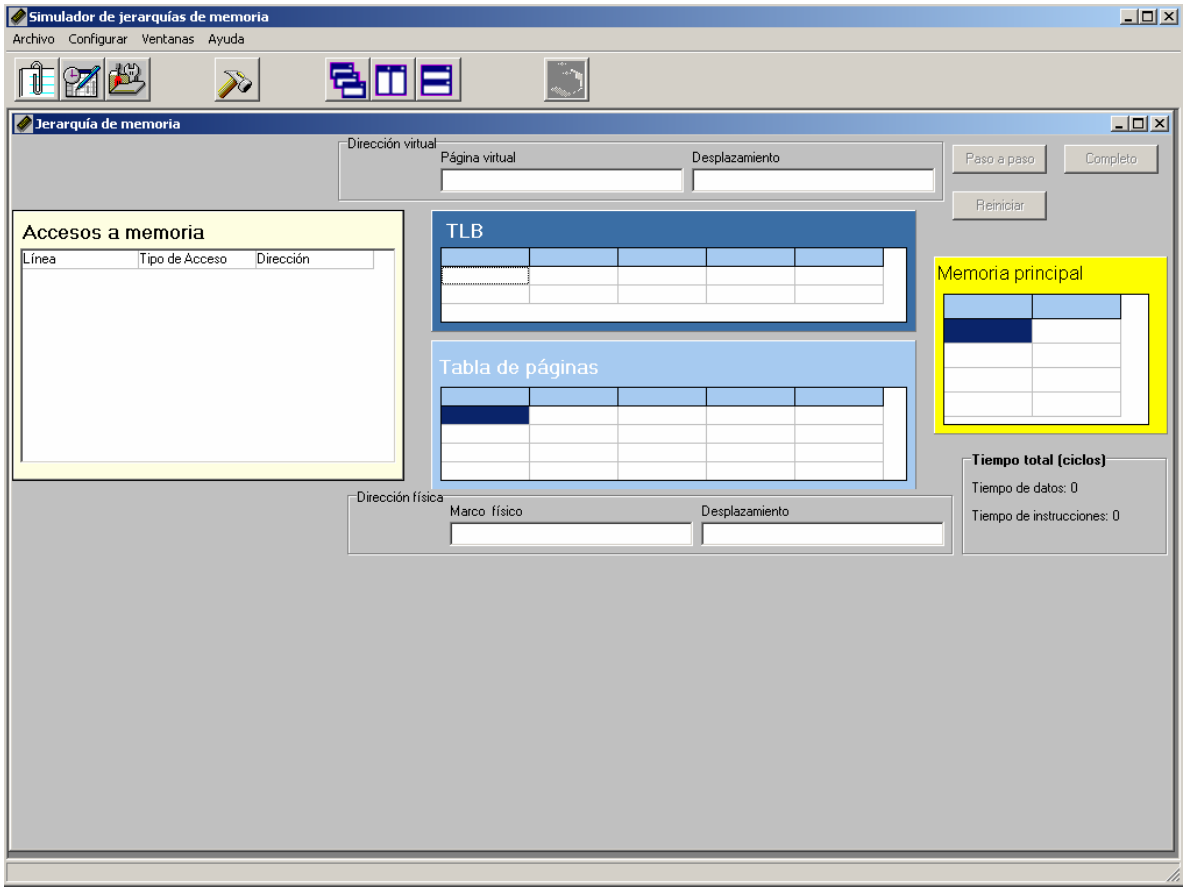

Figura 5.1 – Pantalla principal.

Cuando se ejecuta el simulador lo primero que se ve es una pantalla (figura 5.1) donde aparecen distintos elementos diferenciados En el momento de ejecutar el

simulador, esta ventana aparece deshabilitada de forma que no se puede tener acceso a ninguno de sus elementos. Esta será la ventana que controlará la simulación, así como en la que se irá mostrando el progreso de la misma.

Los elementos que aparecen en esta ventana son:

- *Accesos a memoria*: muestra los accesos que se realizarán a la jerarquía de memoria.
- *TLB:* muestra las entradas del TLB así como su estado en todo momento.
- Tabla de páginas: muestra el contenido de la tabla de páginas y su estado.
- *Dirección virtual:* muestra en formato página virtual+desplazamiento una dirección virtual.
- *Dirección física:* muestra en formato página física+desplazamiento una dirección física.
- *Tiempo total:* muestra el tiempo en ciclos de la simulación en un momento dado.

La aplicación tiene distintos menús de forma que se pueden realizar con cada grupo de menús varias acciones (figura 5.2).

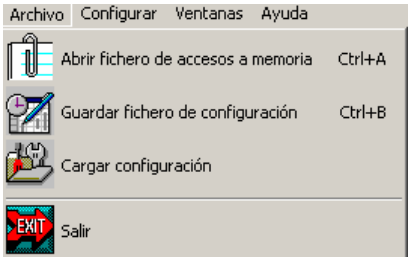

Figura 5.2 – Menú archivo.

El menú *archivo* permite abrir un archivo de accesos a memoria, mediante la opción abrir fichero de accesos a memoria (figura 5.3).

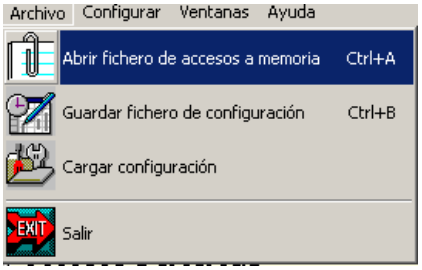

Figura 5.3 – Menú archivo opción abrir fichero de accesos a memoria.

Una vez seleccionada la opción en el menú se mostrará una ventana contextual que permitirá seleccionar un archivo de entrada (figura 5.4).

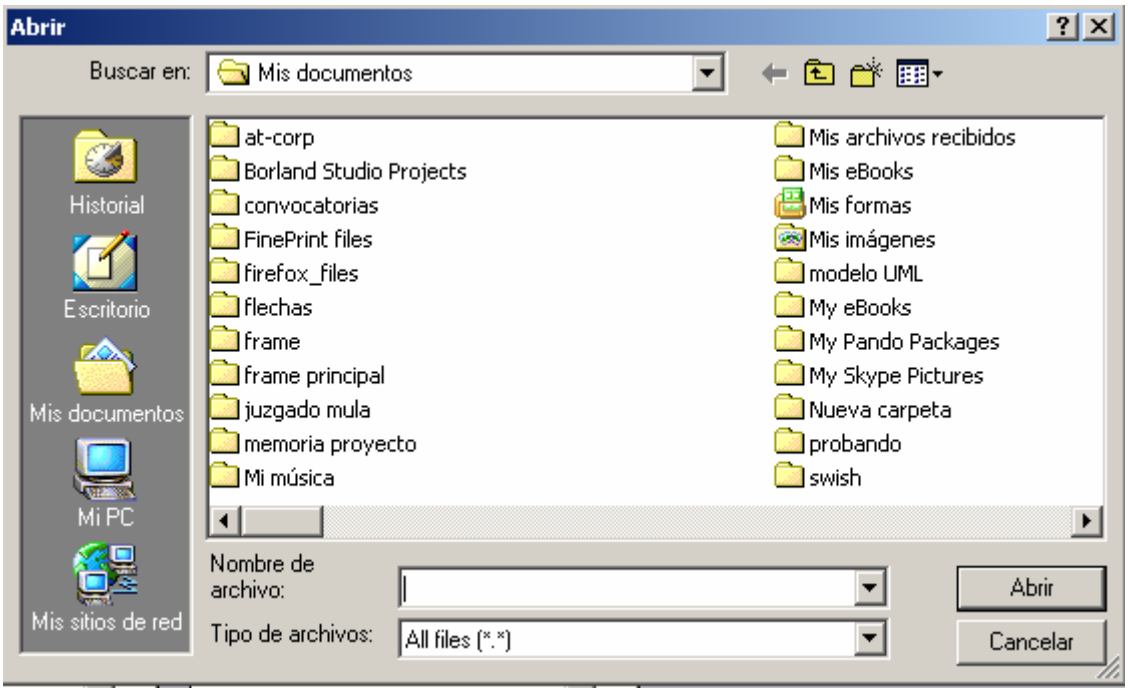

Figura 5.4 – Ventana para abrir un fichero de accesos a memoria.

Una vez cargado el fichero se mostrará la traza de accesos en la *ventana simulación*, en la parte *Accesos de memoria* (figura 5.5), y esta parte de la *ventana de simulación* quedará habilitada.

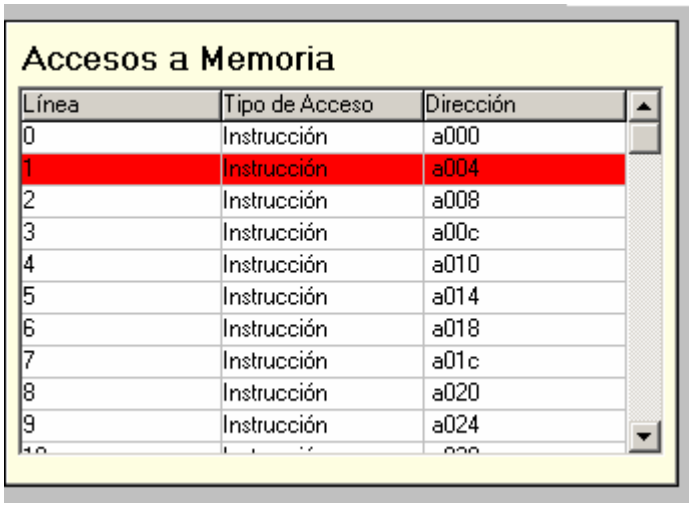

Figura 5.5 – Panel de accesos a memoria de la ventana principal.

En el menú archivo aparecen otras opciones (figura 5.6), que están referidas a los ficheros de configuración. Éstos se guardan por defecto con la extensión "cfg". Los datos que se guardan en estos archivos de configuración dependen de la configuración que previamente se haya creado, más adelante se explicará el funcionamiento de la configuración.

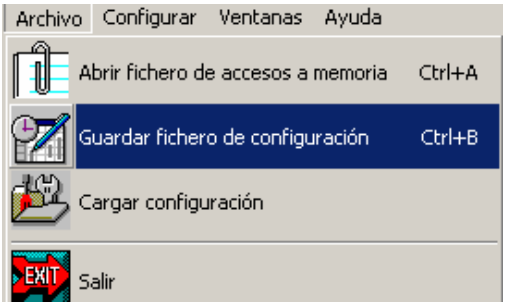

Figura 5.6 – Menú archivo opción guardar fichero de configuración.

Seleccionando en el menú *guardar archivo* se abrirá una ventana contextual (figura 5.7) guardar el fichero de configuración.

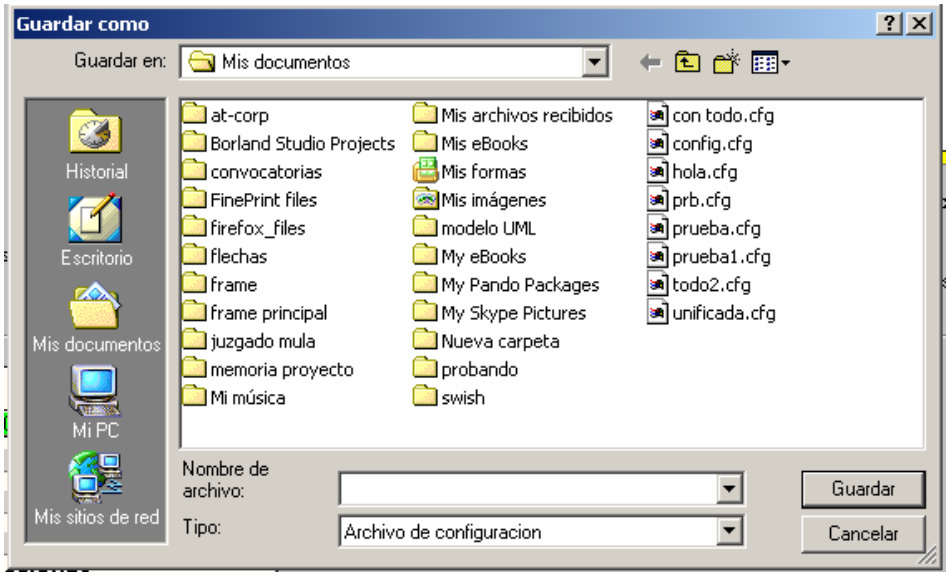

Figura 5.7 – Ventana para guardar un fichero de configuración.

De una forma parecida en el menú *archivo* (figuras 5.8 y 5.9) la opción *cargar configuración* carga una configuración previamente guardada.

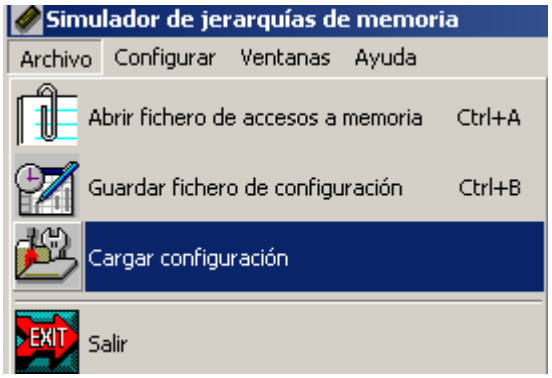

Figura 5.8 – Ventana para guardar un fichero de configuración.

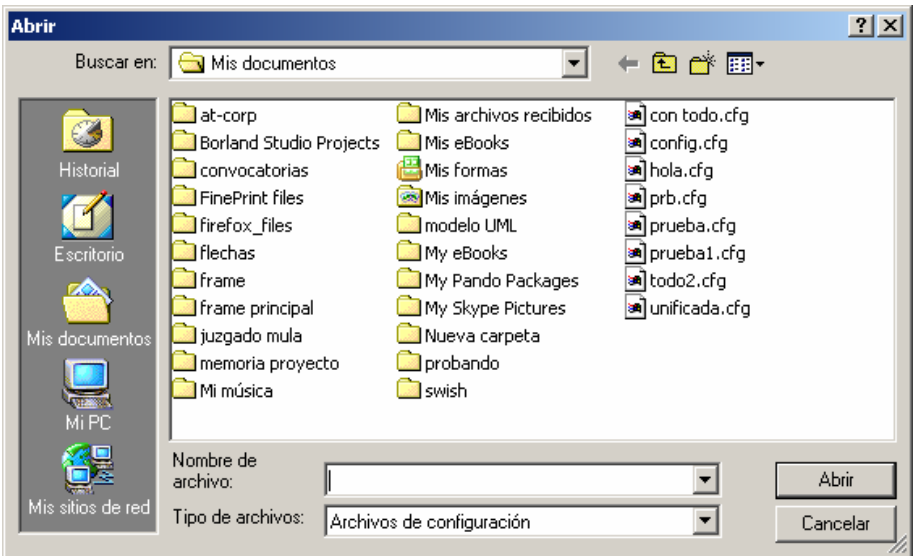

Figura 5.9 – Ventana para abrir un fichero de configuración.

El menú quizás más importante es el menú *configurar* (figura 5.10). Desde este menú y pulsando sobre la opción *jerarquía* se puede configurar toda la jerarquía de memoria.

Figura 5.10 – Menú de configuración.

Una vez escogida la opción en el menú, aparece una ventana en la que se muestra un círculo rojo con una P en su interior representa al procesador y un rectángulo amarillo que representa a la memoria principal. Se pueden añadir elementos

a la jerarquía pulsando sobre los botones que hay a la derecha. De esta forma se pueden añadir hasta 4 niveles de memoria cache, un TLB, que podrá ser de datos o instrucciones, y una tabla de páginas. La inclusión de ciertos elementos hace obligatoria la inserción de otros (por ejemplo un TLB necesita una tabla de páginas TP) aunque hay ciertas combinaciones que son opcionales.

A continuación se muestra una configuración normal sin TP y con dos niveles de cache separadas para instrucciones y datos (véanse figuras 5.11 y 5.12).

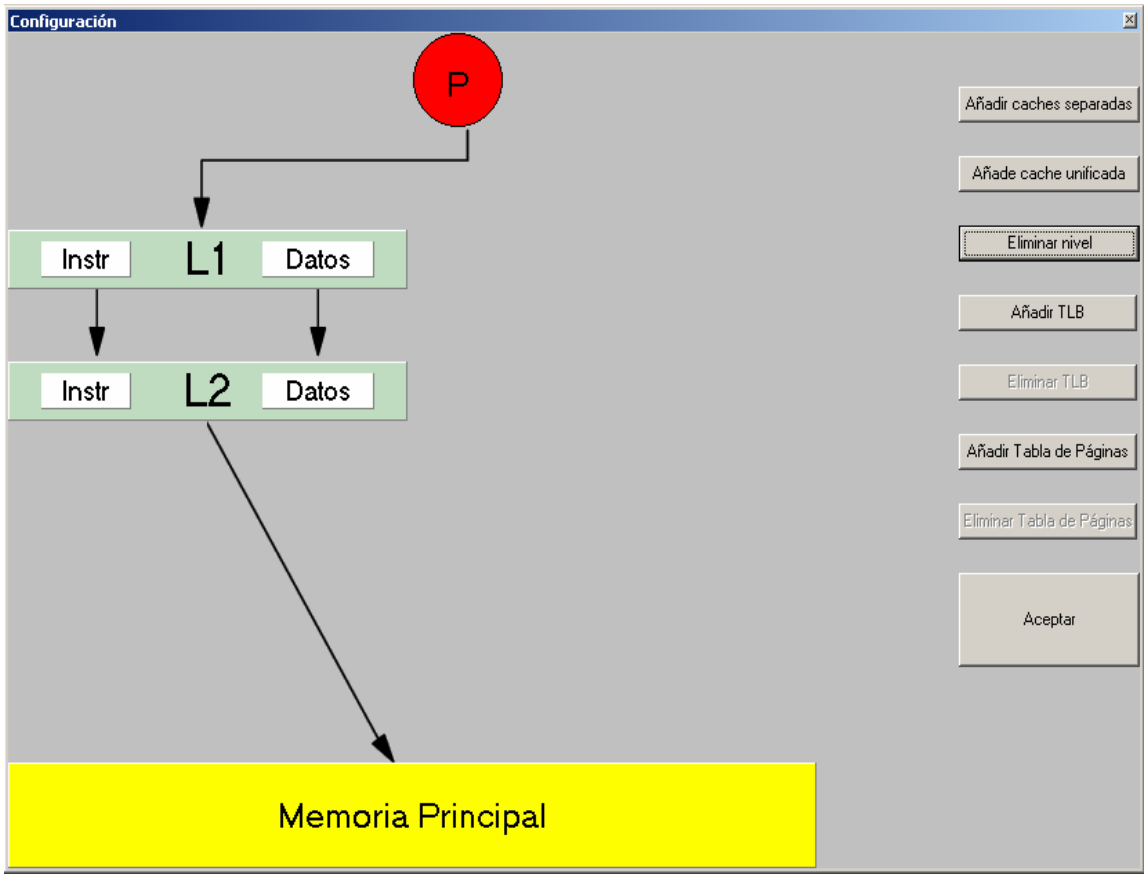

Figura 5.11 – Ventana de configuración.

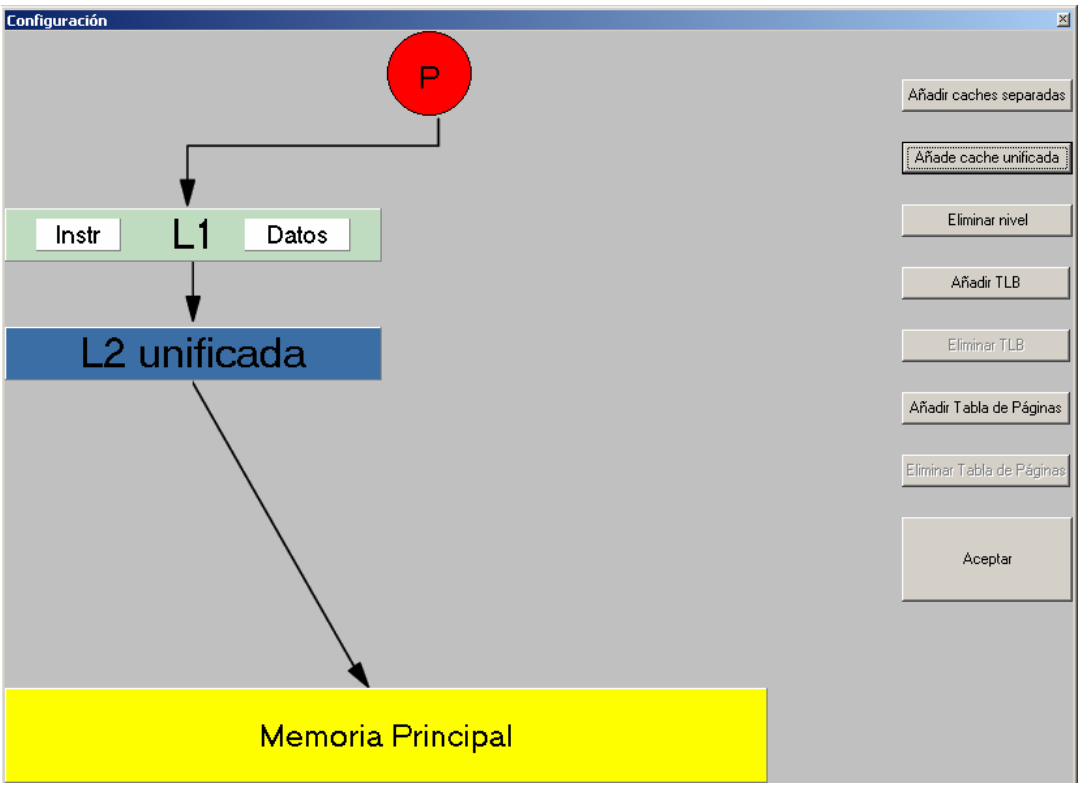

Figura 5.12 – Ventana de configuración.

Pulsando sobre cualquier elemento de la ventana, excepto en la "P", que representa al procesador, se puede acceder a una ventana contextual que permite configurar ciertos parámetros, según el elemento (figura 5.13).

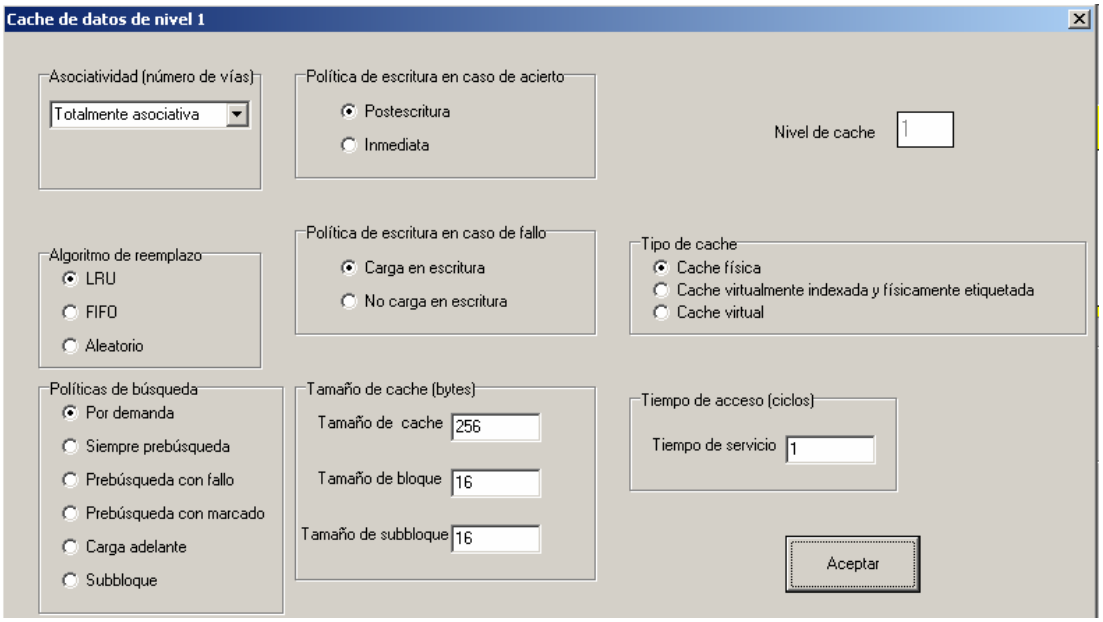

Figura 5.13 – Ventana configuración de cache de datos de nivel 1.

Dentro de cualquiera de las *ventanas de cache* se pueden configurar todos los datos para una cache: asociatividad, algoritmos, políticas de búsqueda, políticas de escritura, tiempo de acceso y tipo de cache (figura 5.14).

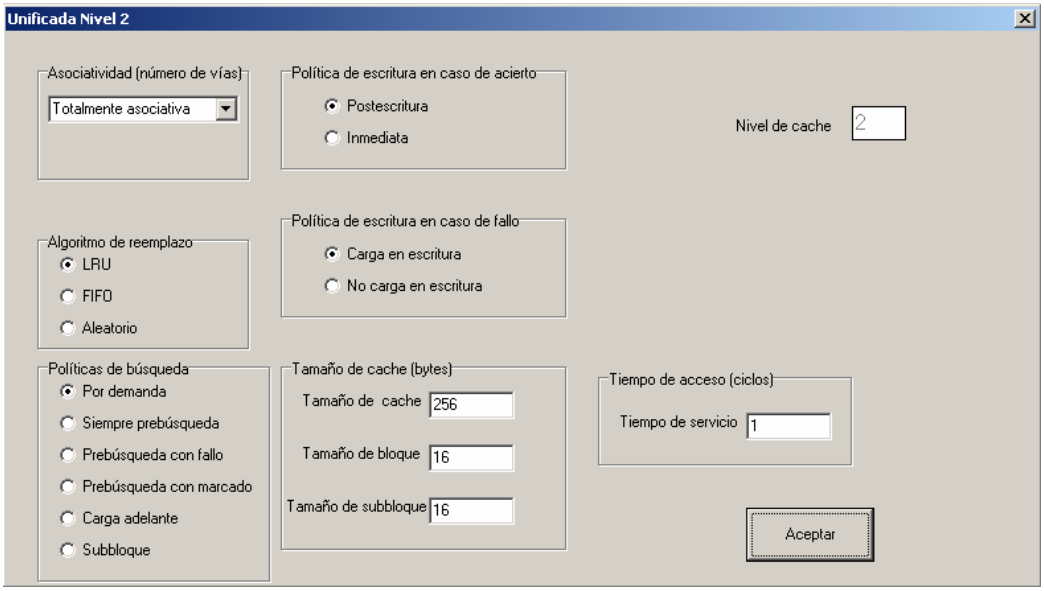

Figura 5.14 – Ventana configuración de cache unificada de nivel 2.

La opción de configuración de *tipo de cache* sólo aparece en la configuración de cache de nivel 1 (véanse figuras 5.13 y 5.14). Esta opción determina **la forma en que se va a acceder a la cache**.

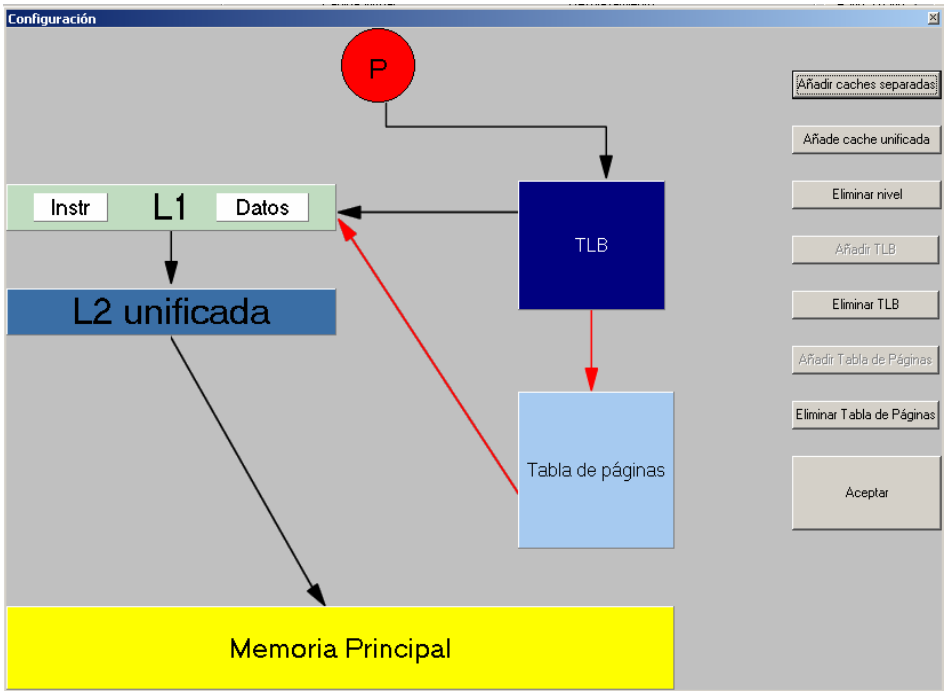

Figura 5.15 – Ventana configuración con elementos de memoria virtual.

Según este *tipo de acceso* la **forma de mostrar la jerarquía** en esta ventana cambiará. Además en el caso de una cache separada (datos e instrucciones), se mostrarán los distintos tipos por una *opción* que aparecerá en el ventana.

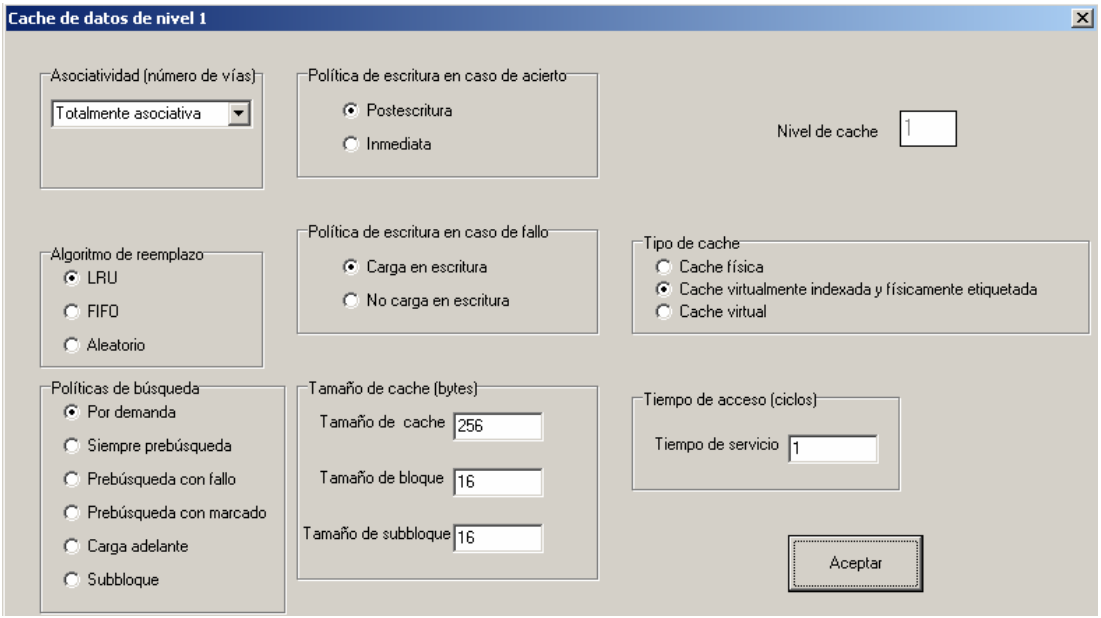

Figura 5.16 – Configurando cache con distintos tipos de acceso.

A continuación se muestran distintos tipos de acceso:

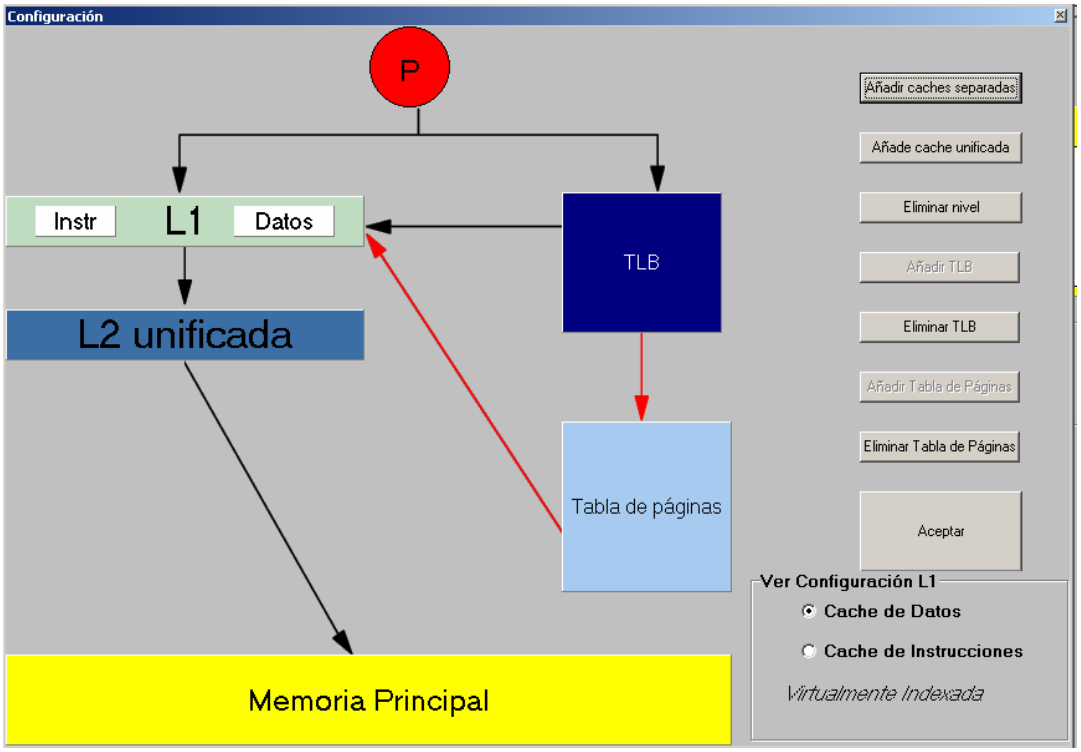

Figura 5.17 – Cache virtualmente indexada pero físicamente etiquetada.

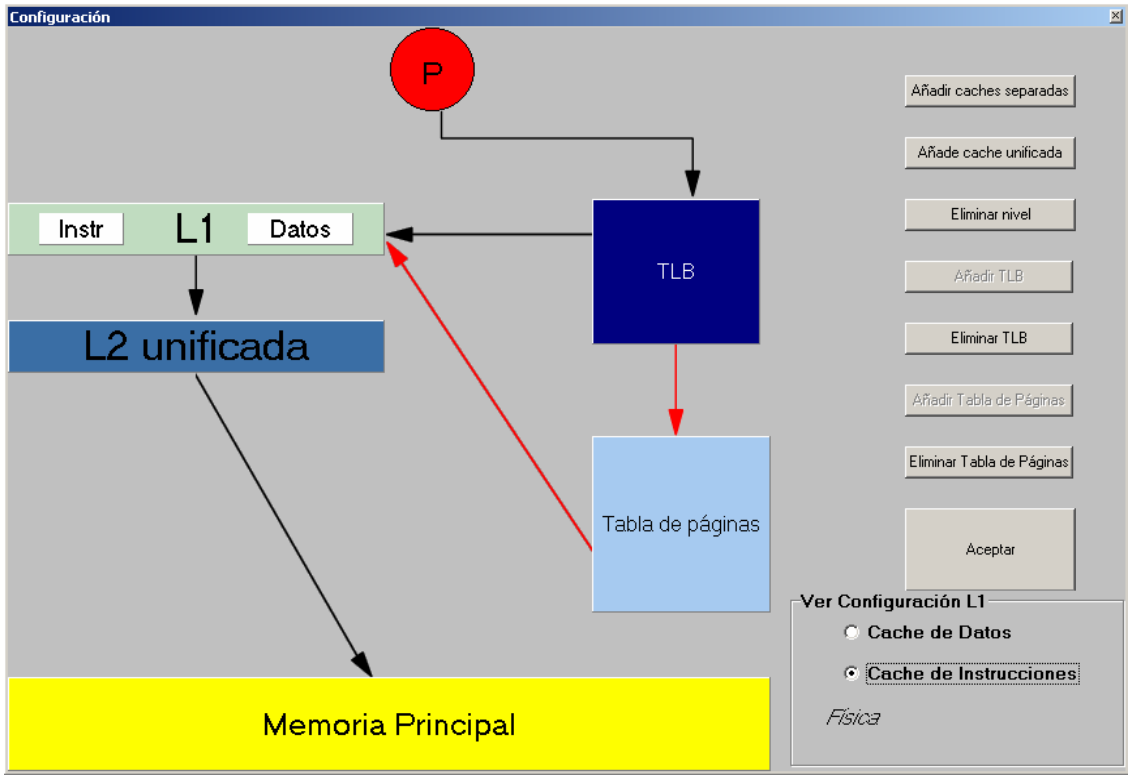

Figura 5.18 – Cache física.

Y cambiando el tipo de acceso a cache virtual:

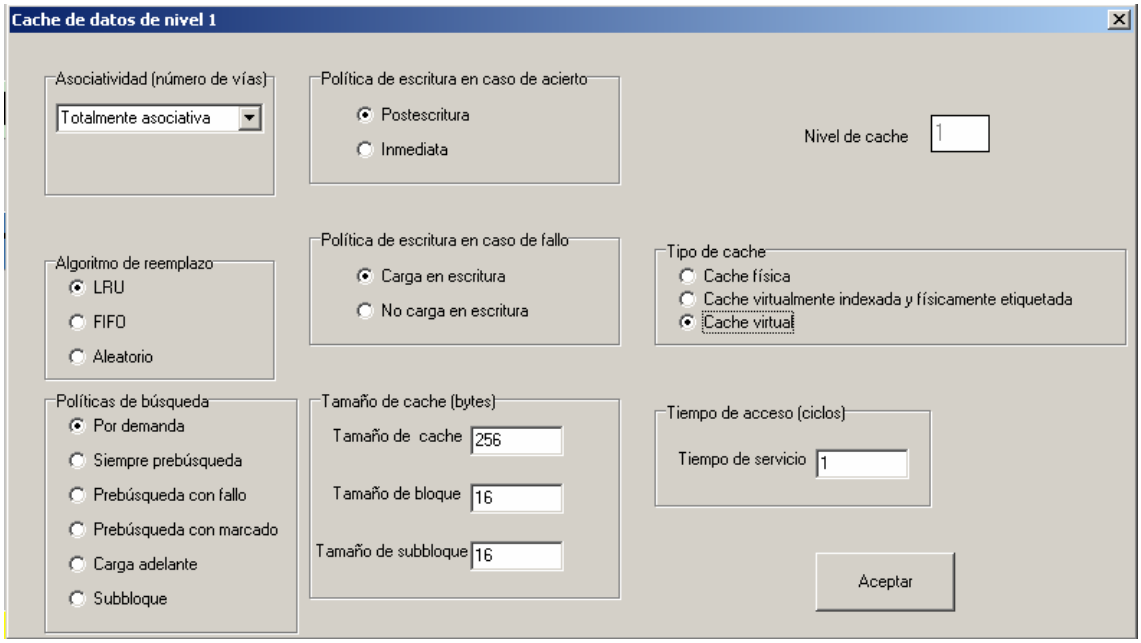

Figura 5.19 – Configurando cache virtual.

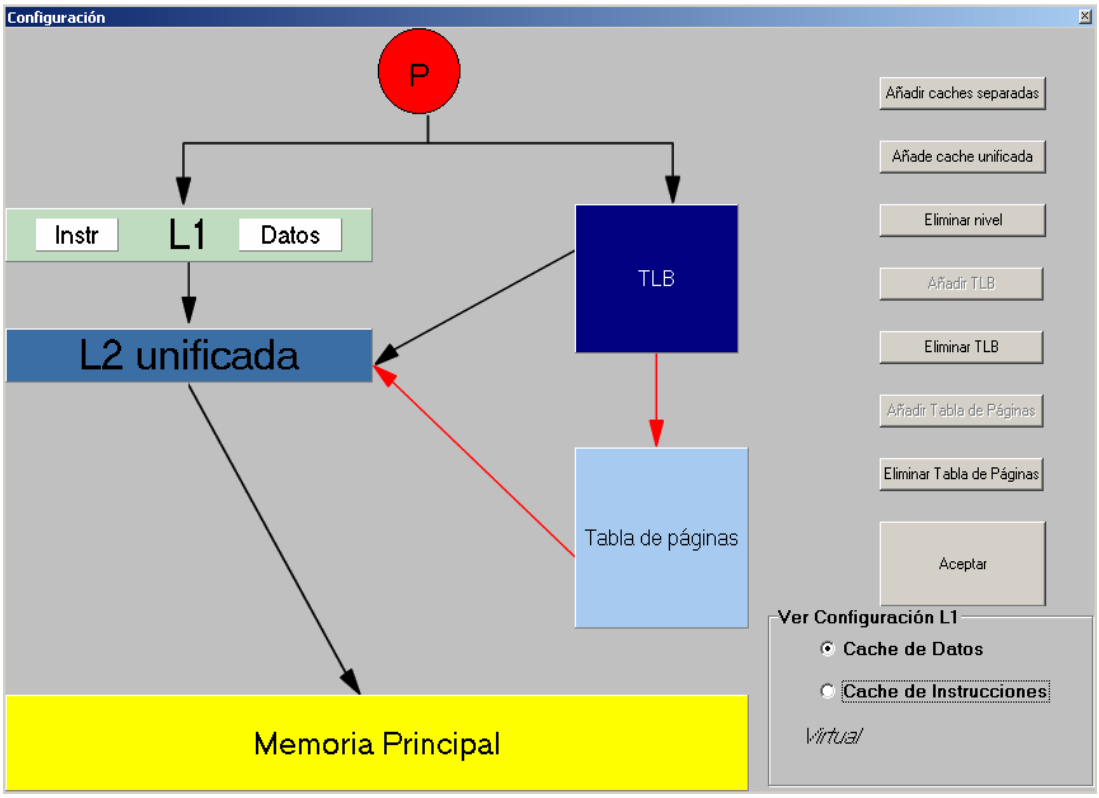

Figura 5.20 – Mostrando configuración cache virtual.

Elementos opcionales:

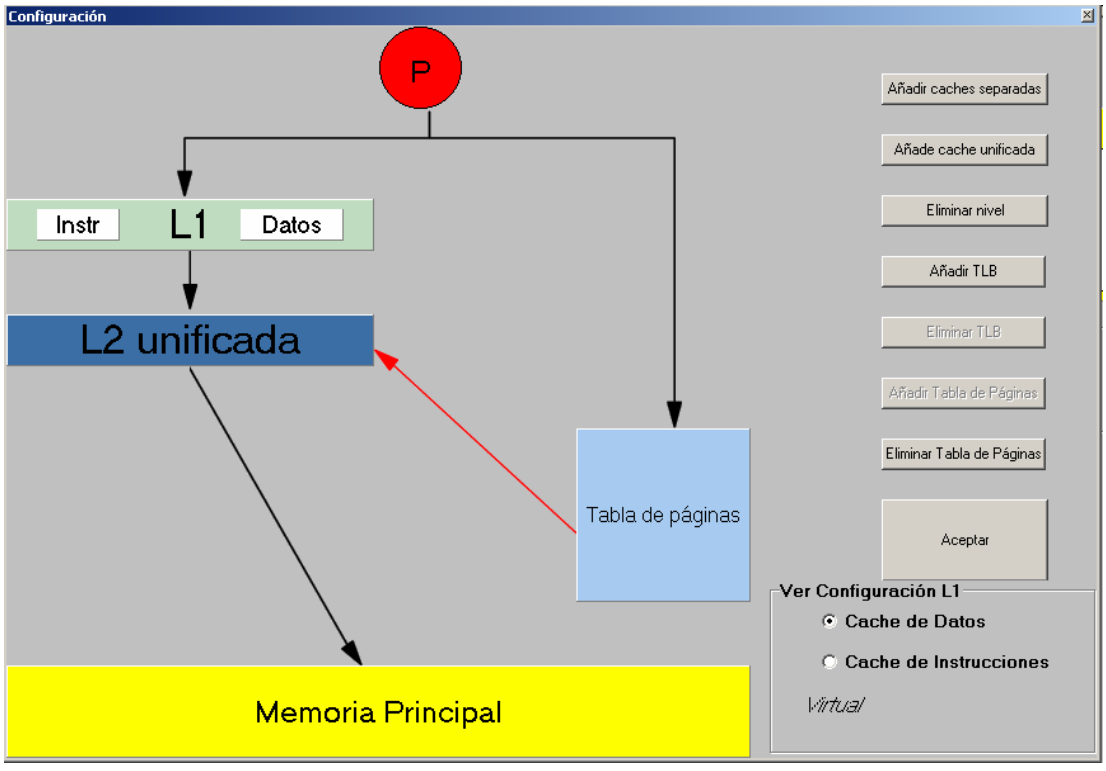

Figura 5.21 – Cache virtual sin TLB.

El resto de elementos son mucho más sencillos en su configuración (véanse desde la figura 5.22 a 5.25). De todos los componentes de la jerarquía se necesitan los tiempos de acceso así como el tamaño de los distintos elementos.

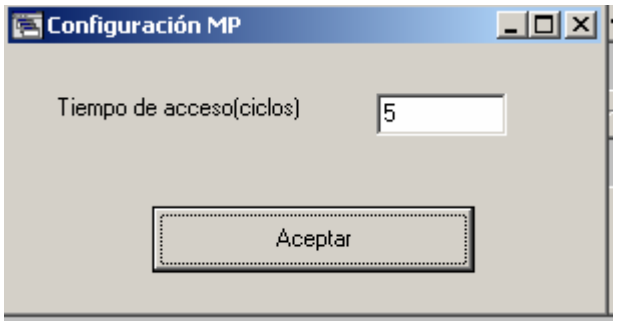

Figura 5.22 – Configuración de memoria principal.

Un TLB necesita además del tiempo de acceso el número de entradas y determinar su tipo (figura 5.23), si se accede de forma separada para datos e instrucciones o si se considera un único TLB tanto para datos como instrucciones.

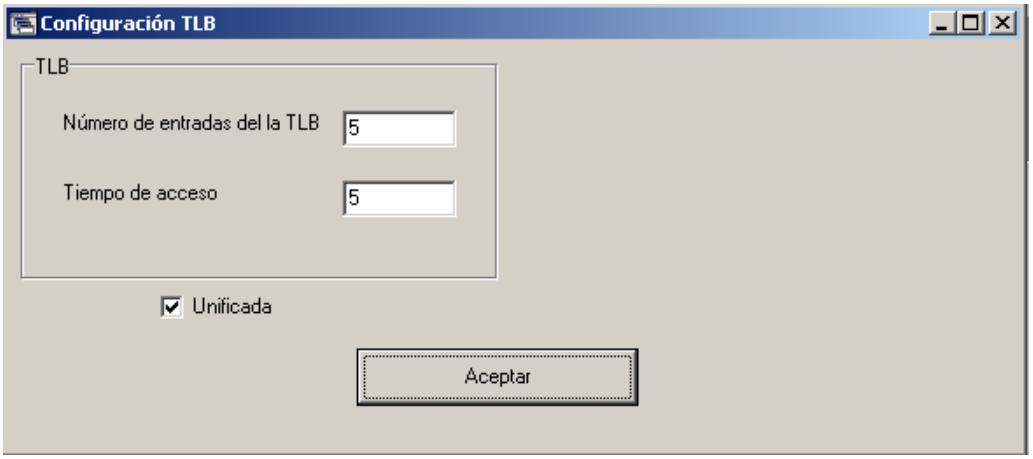

Figura 5.23 – Configuración de TLB unificado.

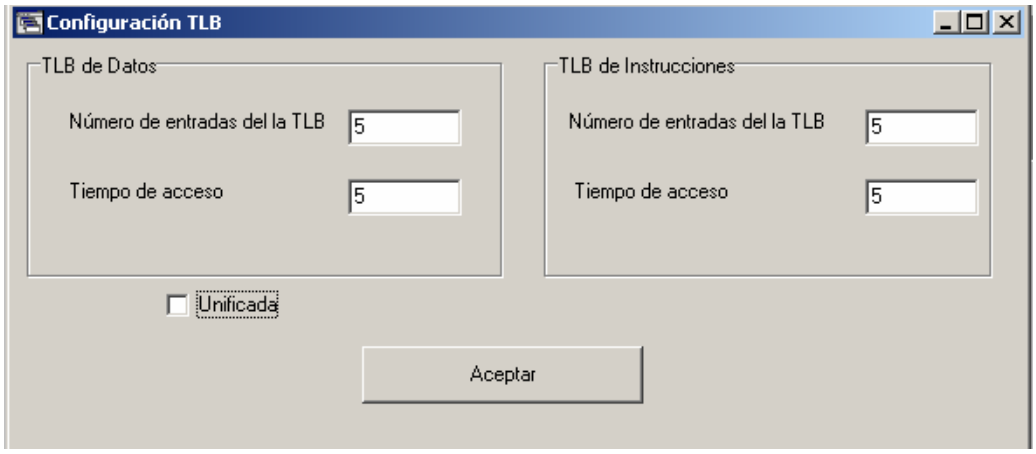

Figura 5.24 – Configuración de TLB separado para datos e instrucciones.

En la configuración de la tabla de páginas se necesita el tamaño de la tabla de páginas y los tiempos de acceso a la misma, y determinar el número de marcos físicos que se pueden alojar en memoria (figura 5.25).

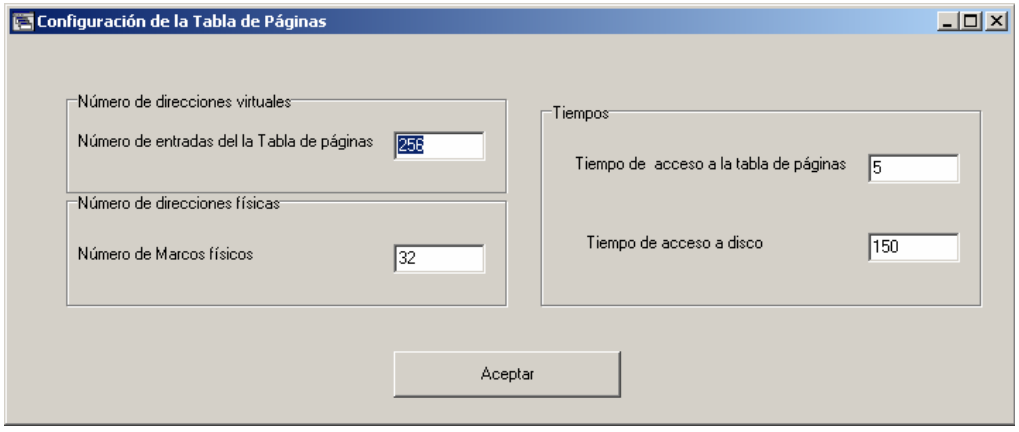

Figura 5.25 – Configuración de tabla de páginas.

Una vez completada la configuración ya se puede acceder a los elementos configurados de la jerarquía (figura 5.26). Los elementos no permiten modificación, sólo consulta, de forma que en todo momento se puede ver la evolución de los accesos a la jerarquía.

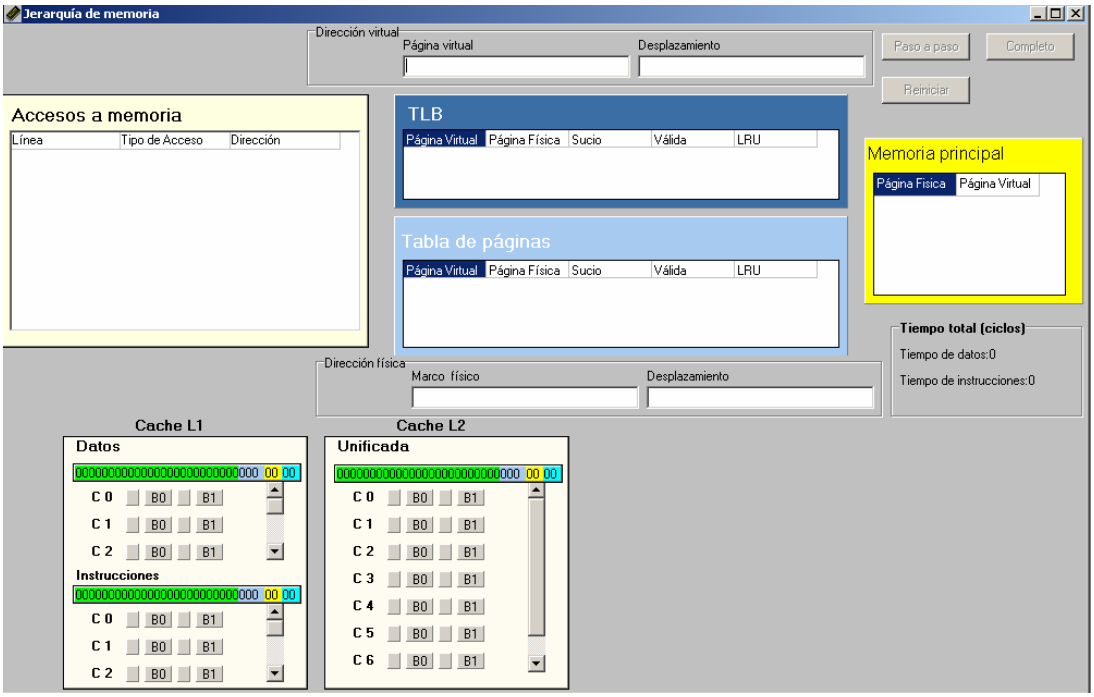

Figura 5.26 – Ventana simulación para la configuración de la figura 5.19.

La dirección virtual se muestra en la parte superior de la ventana (figura 5.27).

Figura 5.27 – Parte superior de la venta de simulación.

La dirección física se muestra en la parte central de la ventana (figura 5.28).

Figura 5.28 – Parte central de la venta de simulación.

En la parte inferior se muestran unos paneles que representan una cache, organizada en conjuntos y en bloques (véanse figura 5.29 y 5.30). Según el tipo se mostrará instrucciones, datos o unificada.

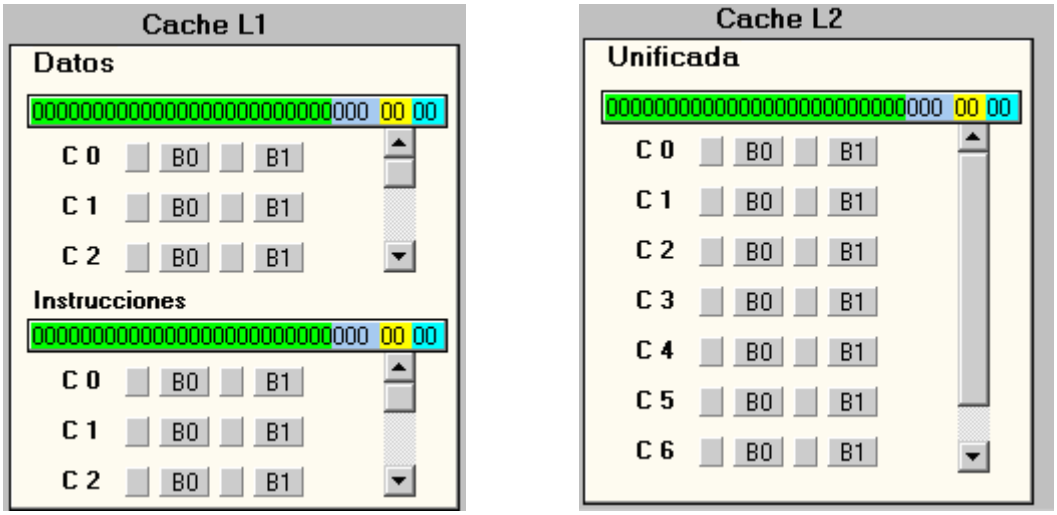

Figura 5.29 –Parte inferior de la ventana simulación.

En estos paneles se muestran paso a paso los accesos que se van produciendo en la jerarquía, así como el formato de dirección que utilizará cada cache y los bloques que han sido sustituidos. Para esto se utiliza un código de colores (figura 5.30).

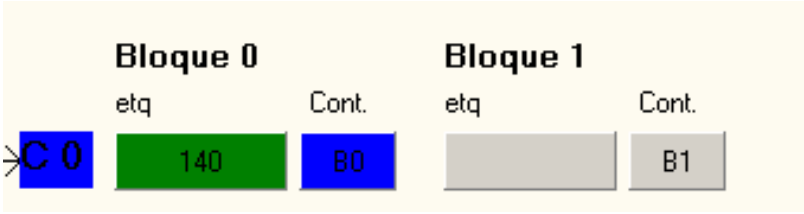

Figura 5.30 – Código de colores fallo en cache Figura 5.17.

Pulsando sobre cualquiera de las caches aparecerá una ventana (figura 5.31) que muestra de forma más detallada el estado de la misma, así como la operación que se ha producido.

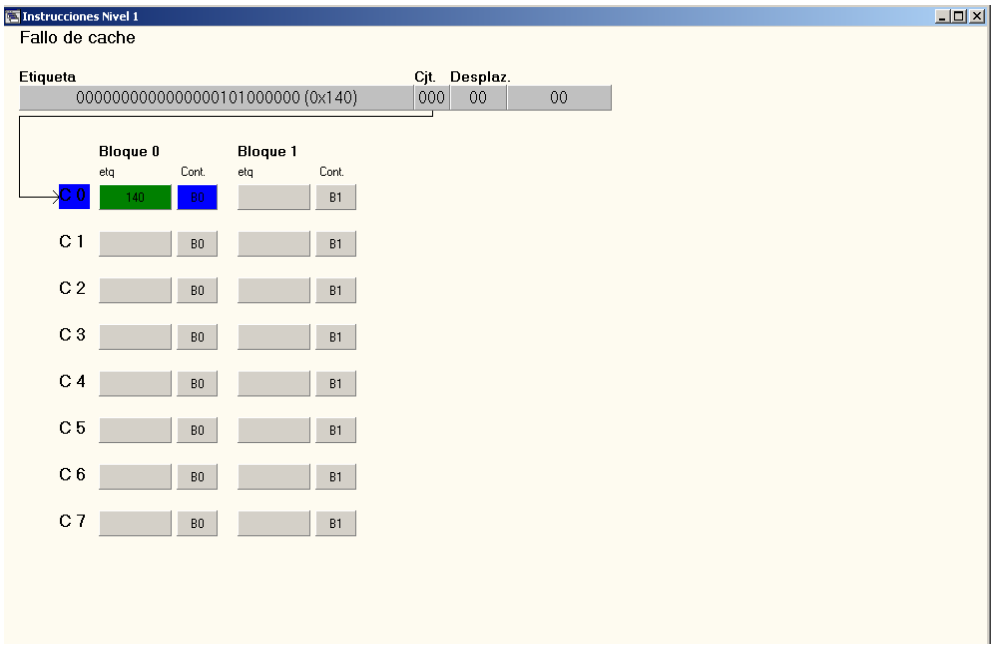

Figura 5.31 – Ventana de detalle de cache de instrucciones nivel 1.

Según el tipo de configuración se mostrará una cantidad distinta de datos. Así, en el caso de una cache de prebúsqueda se representarán todos los bloques traídos de cache utilizando una flecha doble y un color distinto para mostrar la diferencia entre los dos tipos de acceso (figura 5.32 y 5.33).

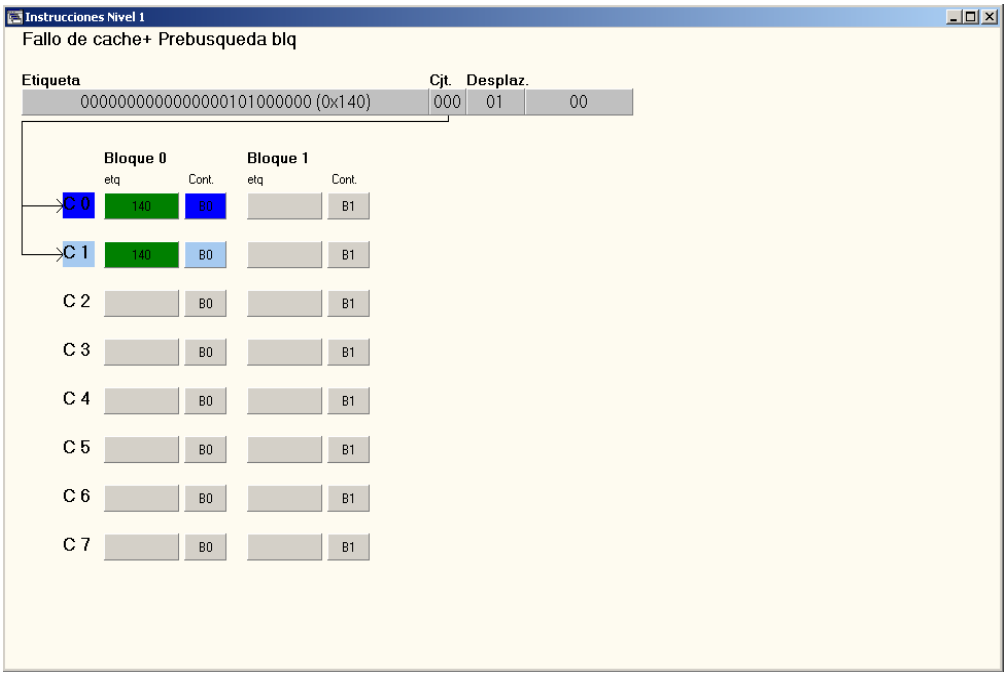

Figura 5.32 – Cache de prebúsqueda en detalle.

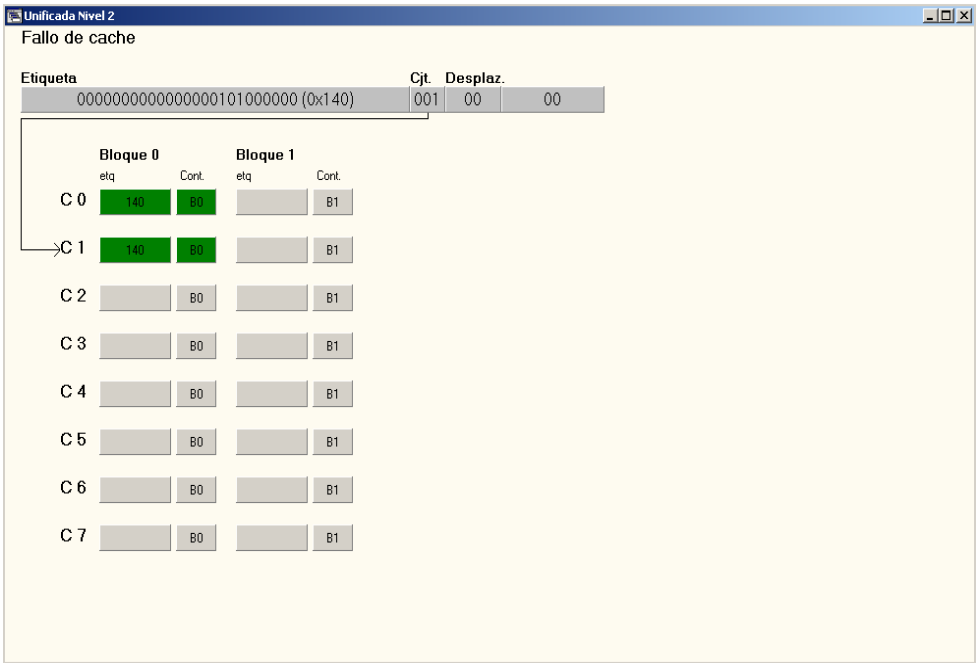

Figura 5.33 – Cache unificada de nivel 2 (ventana cache nivel 2).

Sobre estas ventanas también aparece el formato de dirección de esa cache de forma mucho más detallada (figura 5.34), indicando cuáles son los campos Etiqueta, conjunto y desplazamiento.

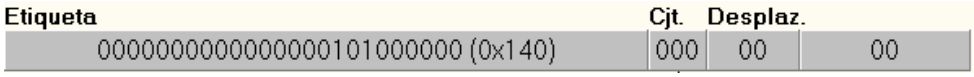

Figura 5.34 – Formato de dirección para cache Figura 5.17 (ventana cache nivel 1).

El caso del TLB o la tabla de páginas se muestran en forma de tabla, indicando el bit de validez si la página está o no, el de suciedad que ha sido modificada, y LRU el valor del contador LRU (figuras 5.35 y 5.36).

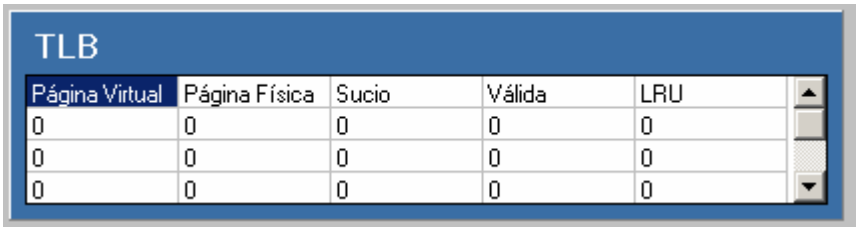

Figura 5.35 –Panel TLB (ventana simulación).

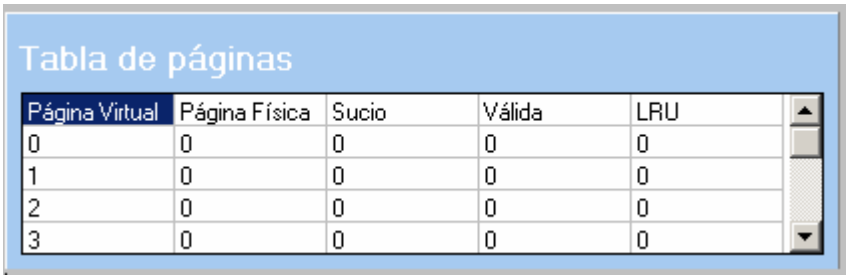

Figura 5.36 –Panel TP (ventana simulación).

Además, en todo momento se va mostrando el desarrollo de la simulación en tiempos en la parte de Tiempo Total (Ciclos) (figura 5.37).

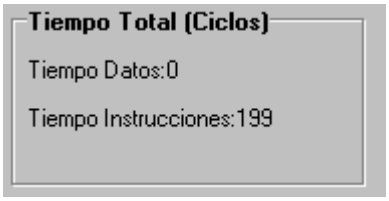

Figura 5.37 –Panel tiempos (ventana simulación).

Todos los elementos de esta ventana tienen acceso a una ventana contextual de ayuda, de forma que cualquiera que no conozca muy bien los elementos pueda obtener en todo momento un soporte más o menos rápido (figura 5.38).

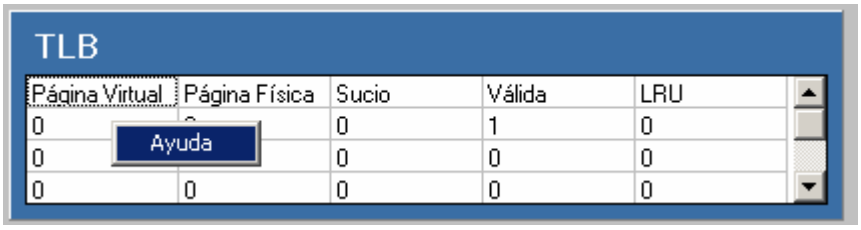

Figura 5.38 – Acceso a ayuda contextual TLB.

Una vez descrita toda la representación a grandes rasgos se pasa a describir de forma rápida los elementos de representación de resultados, así como las distintas opciones de presentación.

La representación de los resultados, a parte de lo que se muestra en la ventana de simulación, se puede consultar en la ventana estadísticas (figura 5.39) que muestra las *estadísticas* completas de todo el proceso de simulación. Se puede acceder a ella desde el menú en la opción estadísticas o desde el icono colocado en la barra de herramientas. Aquí se permite guardar o imprimir el informe.

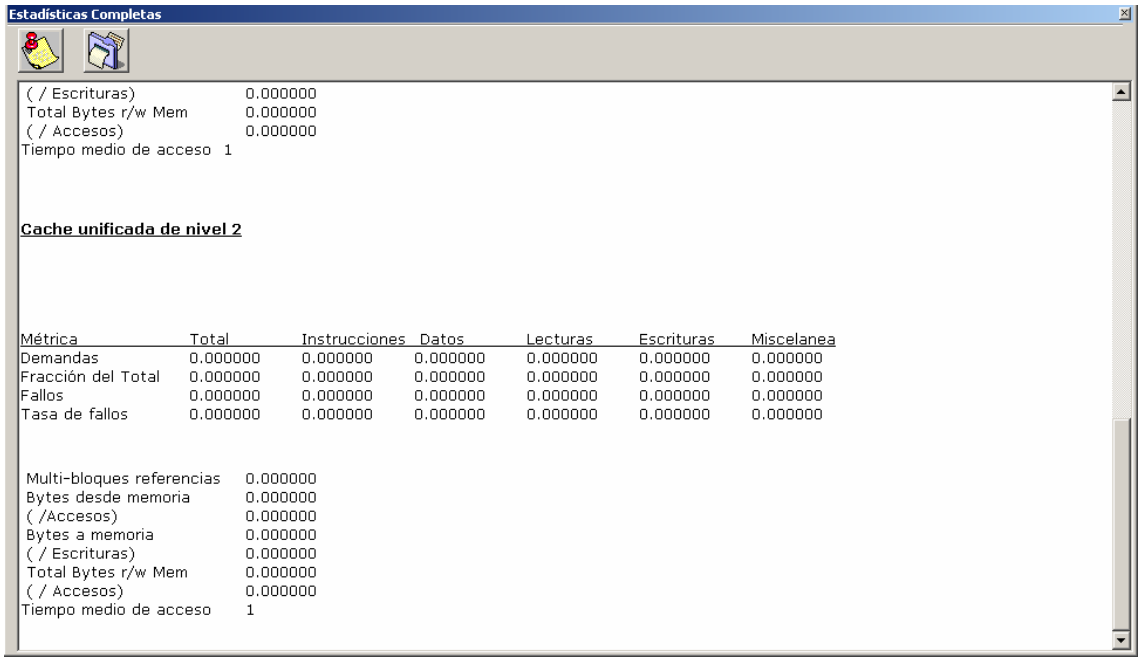

Figura 5.39 – Ventana de estadísticas.

En ciertos momentos, el trasiego entre ventanas resulta muy pesado. Para evitar tener que recorrer gran parte de las ventanas cuando se quiera acceder a una en particular, o en el caso de que se quieran ver todas, más o menos a la vez, se ha añadido el menú de ventanas. Este menú permite organizar las ventanas o moverse entre ellas (figuras 5.40 y 5.41). Según la opción escogida la distribución de ventanas será distinta.

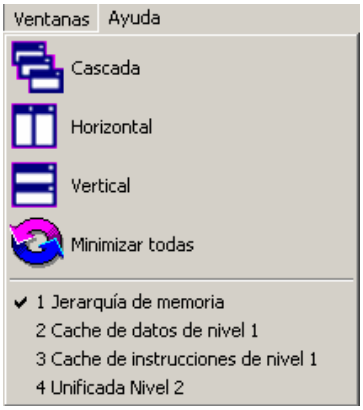

Figura 5.40 – Menú ventanas.

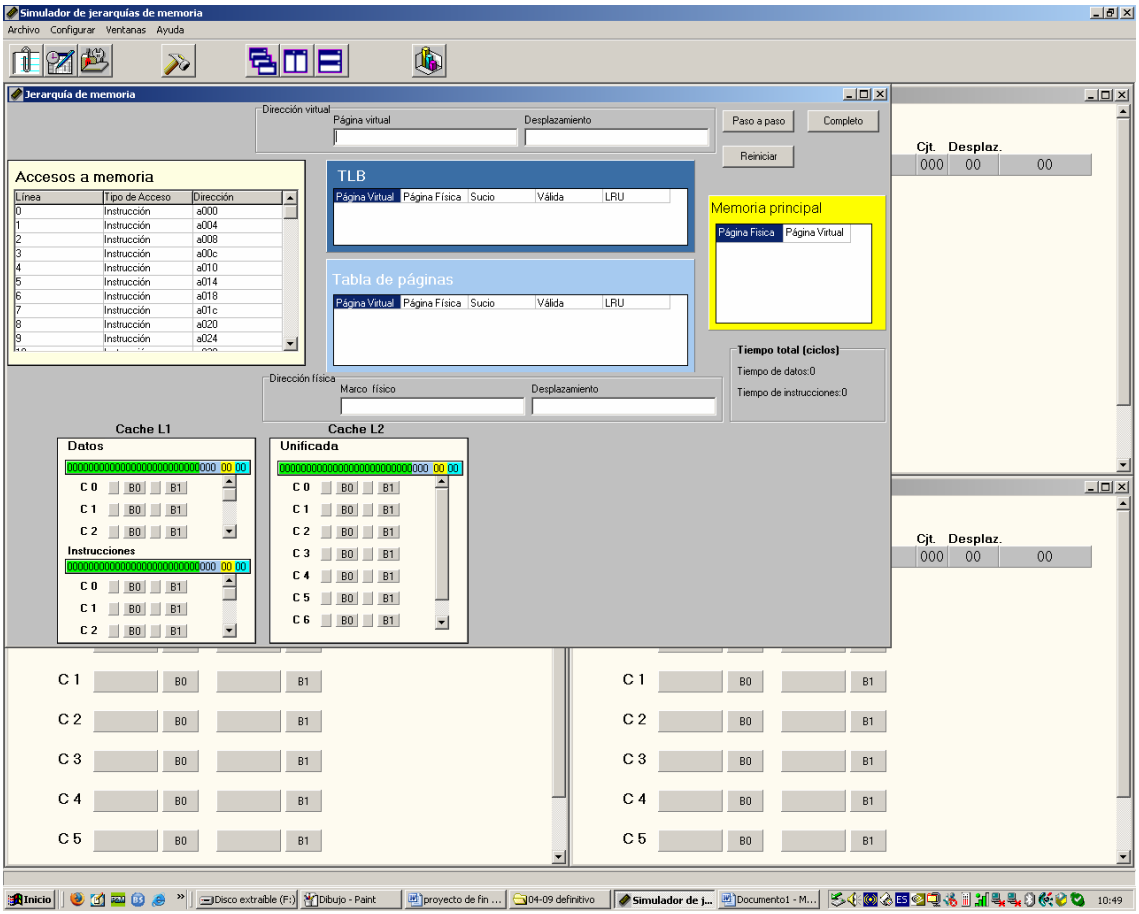

Figura 5.41 – Distribución de las ventanas en vertical.

Para terminar esta pequeña descripción, cabe mencionar que como muchas otras aplicaciones estilo Windows se incluye una barra de herramientas (figura 5.42) de forma que se puede acceder de forma rápida a todas las opciones de los menús.

Figura 5.42 **– Barra de herramientas del simulador.**

5.4. Otros Aspectos

Otro aspecto muy importante del que prácticamente no se ha comentado nada es el soporte al usuario que este simulador ha tratado de dar en todo momento, desde una ayuda contextual rápida hasta un completo archivo de ayuda con ejemplos paso a paso, teoría básica de jerarquías de memoria, y descripción detallada de todos los menús.

El simulador integra dos tipos de ayuda. Por un lado una ayuda general en formato de libros dividida en tres secciones (figurara 5.43). Esta ayuda trata de dar al usuario información general sobre el simulador, soporte teórico y soporte a nivel práctico con casos de ejemplo. Por otro lado se incluye una ayuda contextual rápida, pulsando sobre algunos elementos con el botón derecho y escogiendo el menú (figura 5.44).

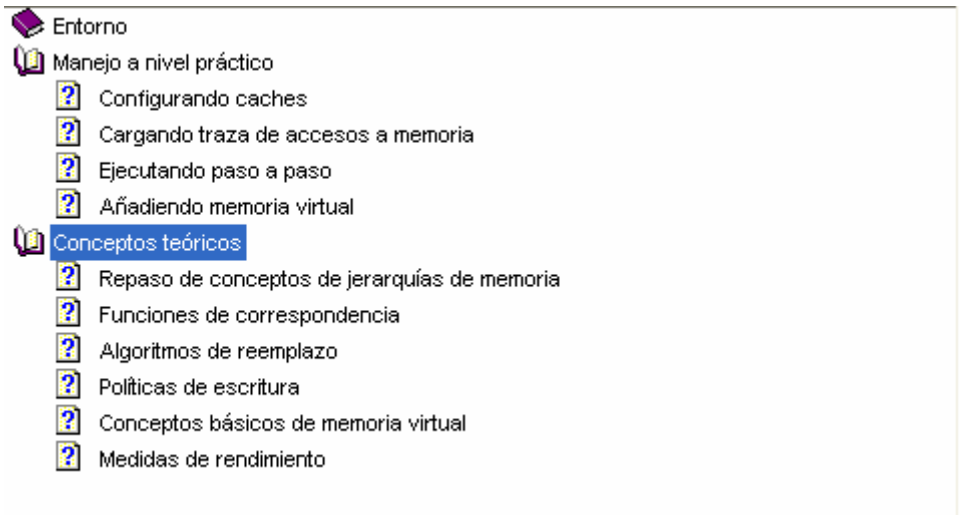

Figura 5.43 – Ayuda general.

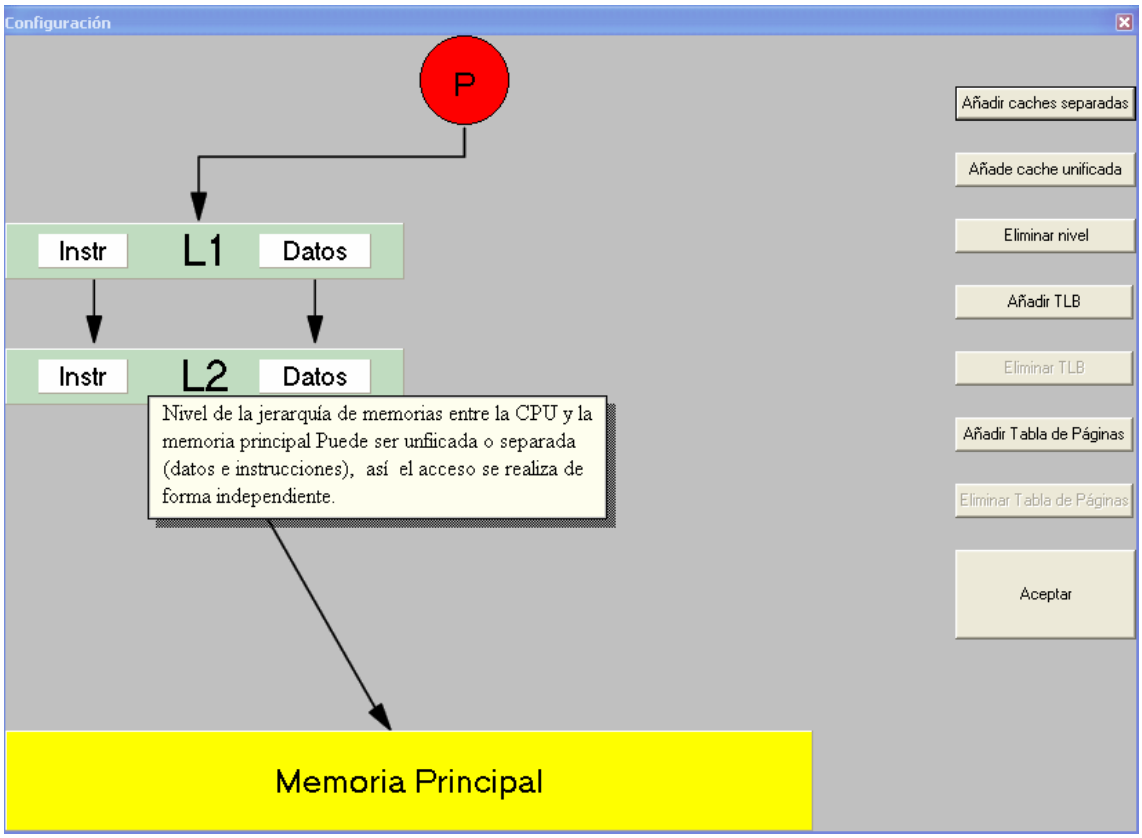

Figura 5.44 – Ayuda contextual.

Se trata de un aspecto bastante importante, aunque no es objetivo de esta vista rápida explicar todos los aspectos de la ayuda, ya que es el uso de la misma con el simulador el que puede decir lo buena o mala que es la misma.

5.5. Limitaciones

Como se comentó en la parte de diseño, el simulador tiene algunas limitaciones. Sin embargo, el propio diseño y la estructura modular del mismo permiten que en el futuro se puedan subsanar estos aspectos, y también ampliar este simulador con muchos más algoritmos, estructuras, ventanas de información o aquello que se crea necesario.

Algunas limitaciones han sido añadidas en la codificación del programa, tales como el tamaño máximo del fichero de entrada o el número de entradas de la tabla de páginas y del TLB. Estas restricciones han sido impuestas por razones de eficiencia ya que en principio tampoco aportan nada nuevo.

CAPITULO 6 PRUEBAS

6.1 Introducción

Las pruebas son las tareas que se realizan durante la construcción de un Sistema de Información y deben estar orientadas a la entrega de un producto de calidad. La Ingeniería del Software engloba un subconjunto del total de las pruebas, actividades a realizar específicamente orientadas a la consecución de este objetivo bajo la denominación “garantía de calidad” [4].

Existe más de un tipo de pruebas. Pueden clasificarse atendiendo a varios criterios. Desde la perspectiva de su ámbito se distinguen tres clases:

Pruebas unitarias: cada componente del sistema se prueba individualmente.

Pruebas de integración: se prueba la integración de los componentes del sistema para demostrar que encajan correctamente.

Pruebas del sistema: se prueba el sistema globalmente. Desde la perspectiva de la metodología empleada, las tareas pueden agruparse en dos procesos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas, por el cliente final.

Esta definición lleva al análisis de los términos verificación y validación y su relación con el plan de pruebas.

La verificación es el conjunto de actividades que aseguran que el software implementa correctamente una función específica. La validación se refiere al conjunto de pruebas que aseguran que el software construido se ajusta a los requisitos del cliente. Las estrategias de verificación y validación son un subconjunto de las empleadas para construir un software de calidad y complementarias al testing en sentido estricto. En los términos que utiliza Métrica 3 [7] la prueba del equipo lógico es el método usado más a menudo para determinar si funciona como debe. Se trata de una definición inclusiva que incluye, además de otras cosas, a las pruebas en sentido estricto que consisten en ejecutar los programas para encontrar errores [4].

Hay dos posibles estrategias puras para enfocar el diseño del plan de pruebas. Dado que el software consiste en código que realiza una función se puede poner el acento en el código o en la función. El procedimiento recomendado es empezar por las pruebas denominadas de caja negra y seguir con otro tipo al que se denomina de caja blanca con una estrategia combinada que se verá más adelante una vez analizadas las dos por separado [4].

Una vez descritas las alternativas a la hora de realizar las pruebas se pasará a describir la estrategia a seguir, el entorno y forma en que se han ejecutado. Todos los resultados se han incluido en el anexo II.

6.2 Objetivos de las pruebas

Desde el principio del capítulo se ha hecho hincapié en la importancia de las pruebas. La motivación de las pruebas es detectar posibles problemas que tenga el desarrollo. Una prueba que no revela ningún problema es una pérdida de tiempo.

En resumen, un plan de pruebas intenta:

- Facilitar las tareas técnicas de las pruebas.
- Mejorar la comunicación sobre las tareas y los procesos de las pruebas.
- Suministrar estructura para la organización, planificación y gestión del proyecto de prueba.

Habrá que seguir una adecuada metodología, planificación en el diseño y planificación de las pruebas, para hacer que el desarrollo tenga mucha más calidad y se ajuste mejor a los requisitos inicialmente impuestos. El plan de prueba propondrá un número de pruebas, así como una forma de ejecutarlas y documentarlas, de forma que se cumplan los requisitos y que se ofrezca una calidad determinada.

6.3. Planificación y documentación

Anteriormente se hablaba de las distintas alternativas a la hora de realizar las pruebas y sus objetivos. A continuación se concretará el tipo de pruebas que se van a realizar así como la forma de documentarlas y el proceso que se seguirá para ello.

6.3.1 Pruebas de caja negra

Una aproximación al problema consiste en visualizar el programa que se desea probar como una caja negra de forma que quien realiza las pruebas no se preocupa en absoluto por su contenido y concentra sus esfuerzos en encontrar circunstancias en las que el programa no se comporte de acuerdo con las especificaciones [7].

En este caso las pruebas se centrarán en la navegación por las distintas ventanas, rellenando datos y modificando distintos campos de las mismas. Se irá viendo si los resultados obtenidos son los esperados y considerando las posibles excepciones que puedan suceder. Para esto se asignará a cada ventana un nombre y se numerarán las pruebas, y dentro de éstas, resultados obtenidos y resultados deseados.

La documentación se hará indicando el nombre de la pantalla, valores introducidos y el resultado obtenido.

6.3.2 Pruebas de comparación

En los casos en que la aplicación es crítica se han realizado a veces dos versiones distintas del software que son probadas independientemente contra las especificaciones para asegurar que los resultados no sólo son correctos sino que son idénticos en cualquier circunstancia. Es lo que se denomina prueba de la comparación o prueba mano a mano [7].

Las pruebas de este tipo se centrarán en la parte de caches, ya que la misma es resultado de una adaptación con pequeñas modificaciones de DineroIV. En principio, los resultados que se obtengan de estas pruebas no tienen por qué ser exactos a los obtenidos por DineroIV, aunque tienen que ser muy parecidos, ya que las modificaciones, como se ha comentado antes, son mínimas.

La documentación de las mismas se hará comentando la configuración usada, el resultado obtenido, el esperado y la comparación con los resultados del simulador.

6.3.3 Pruebas de caja blanca

La otra alternativa es concentrarse en el código. Esta es la estrategia de caja blanca o “logic driven”. El esfuerzo se concentra en encontrar casos de prueba que permitan comprobar cómo funciona el código en cualquier situación, lo que supone, como mínimo, recorrer todos sus caminos con un conjunto tan amplio de datos de prueba como sea necesario. En resumen, se trataría de recorrer exhaustivamente todos los caminos posibles. Esto supone una tarea inabordable incluso para módulos relativamente simples [4].

En el caso de que sean necesarias, se aplicarán pruebas unitarias que se han realizado al crear cada módulo o cada componente. Las pruebas de integración se realizan cuando se han unido varias componentes y se comprueba que las interfaces son correctas de forma que tanto las llamadas como la entrega de los datos se realiza correctamente. Dado que los módulos están relacionados unos con otros se podrá suponer que están organizados jerárquicamente,

Las componentes de planificación son:

- Definición de objetivos con la descripción del alcance de las pruebas.
- Revisión de las tendencias de gestión que se aplicarán durante la ejecución de las pruebas.
- Descripción de los controles aplicables.
- Establecimiento de los criterios de terminación de las pruebas.
- Definición de responsabilidades.
- Elaboración de la biblioteca de casos de pruebas y normas conteniendo los requerimientos que deben ser probados, la identificación y denominación de cada prueba, el elemento que está siendo probado, el método de prueba utilizado y el criterio de aceptación.
- Definición de las herramientas que van a emplearse.
- Estimación de los recursos (humanos, de hardware y de comunicaciones) necesarios para la realización de las pruebas.

6.4 Especificación de las pruebas

Los elementos que se utilizarán para las pruebas serán:

- Máquinas donde funcionará la aplicación.
- Sistema operativo.
- Todos los módulos que componen la aplicación.

6.4.1 Pruebas de funciones

El objetivo de estas pruebas es encontrar problemas en el procesamiento, configuración o cualquier otro aspecto de la interfaz. Estas pruebas serán de caja negra (tabla 6.1).

Objetivo de la técnica	Realizar entradas en la interfaz de usuario y comprobar si el sistema proporciona las salidas esperadas.
Técnica:	Ejecutar los flujos o las funciones de los casos de uso individuales, usando datos válidos e inválidos, para verificar: -Los resultados esperados ocurren cuando los datos legítimos son usados. -Advertir que los mensajes sean exhibidos cuando los datos de configuración no son correctas. -Cada regla es correctamente aplicada
Herramientas necesarias:	-Monitorización -Generación de datos
Criterio de éxito	Se encuentra algún tipo de error
Consideraciones	El proceso se desarrollará de forma manual.

Tabla 6.1 – Pruebas de caja negra (I).

6.4.2 Pruebas de comparación

El objetivo de estas pruebas es encontrar problemas en el procesamiento, configuración o cualquier otro aspecto de la interfaz. Estas pruebas serán de caja negra (tabla 6.2).

Objetivo de la técnica	Realizar entradas en la interfaz de usuario y comprobar si el sistema proporciona las salidas esperadas.
Técnica:	Ejecutar los flujos o las funciones de la parte dedicada a caches: -Los resultados son los esperados una vez utilizado DineroIV.
Herramientas necesarias:	-Monitorización -Generación de datos
Criterio de éxito	Se encuentra algún tipo de error
Consideraciones	El proceso se desarrollará de forma manual.

Tabla 6.2 – Pruebas de caja negra (II).

6.4.3 Pruebas de carga

El objetivo de estas pruebas es medir el rendimiento a niveles diferentes de carga de trabajo y la capacidad de continuar funcionando apropiadamente bajo esos niveles diferentes (tabla 6.3).

Objetivo de la técnica	Observar el comportamiento con distintos niveles de carga de trabajo
Técnica:	Simular la carga de usuarios con usuarios reales.
Herramientas necesarias:	-Herramientas de control y monitorización -Herramientas de generación de datos
Criterio de éxito	Se encuentra algún tipo de error
Consideraciones	En el caso de que se vaya a probar con usuarios por lo menos tendrá que haber tantos usuarios como vaya a existir en un aula real.

Tabla 6.3 – Pruebas de caja negra (III).

6.4.4 Elementos necesarios para las pruebas

En esta sección se presentan los recursos necesarios para llevar a cabo el Plan de Pruebas (tablas 6.4 y 6.5).

Sistema Hardware Base

Recursos del Sistema		
Recurso	Cantidad	Nombre o Tipo
PCs para pruebas del cliente	1	hardware

Tabla 6.4 – Recursos para pruebas.

Elementos de Software Base en el Plan de Entorno

Elemento software	Versión	Tipo y notas de interés
Windows 2000	Última	Sistema Operativo
Simulador de jerarquías de memoria	6.0	Software desrarrollado
Borland C++ builder	6.0	Compilador/depurador
DineroIV	Última	Simulador dineroIV
RedHat Linux	8.0	Sistema Operativo

Tabla 6.5 – Entorno de pruebas.

CAPITIULO 7 CONCLUSIONES Y TRABAJO FUTURO

7.1 Conclusiones

Al iniciar esta memoria se propusieron una serie de objetivos, entre ellos uno principal: obtener una herramienta de simulación docente completa y de fácil manejo que permita poner en práctica los conocimientos fundamentales de un sistema jerárquico de memoria.

Se puede considerar que este objetivo ha sido plénamente logrado, con la construcción de una herramienta que puede resultar muy interesante para ofrecer un soporte al aprendizaje del funcionamiento de la jerarquía de memoria. Así el uso que se pretende del mismo es como herramienta didáctica, lo que no quiere decir que en un fututo cercano y con algunas mejoras no pueda convertirse en un simulador mucho más completo ofreciendo resultados mucho más fiables en términos de rendimiento y velocidad para distintas configuraciones de memoria.

Otro objetivo era dar al usuario soporte en todo el proceso de simulación. Objetivo que ha sido cumplido con la realización de la ayuda y el manual de usuario que han sido integrados en la herramienta.

También se han conseguido los objetivos parciales:

- **Estudio de los actuales diseños del sistema de memoria de los computadores**: se han estudiado y comparado distintos sistemas, incluyendo partes de los mismos en el proyecto.
- **Establecimiento de los requisitos del sistema a desarrollar**: los requisitos han sido expuestos y plasmados con claridad.
- **Realización del análisis y diseño del sistema**: se ha conseguido realizar un diseño conforme a los requisitos.
- **Implementación de la herramienta**: la herramienta ha sido plénamente implementada.
- **Comprobación del correcto funcionamiento del simulador**: se han realizado baterías de pruebas sobre el sistema final para comprobar su correcto funcionamiento y que, el mismo, se ajusta a los requisitos.

Aunque la presentación de las estadísticas de datos como los resultados obtenidos podrían ser mucho más completos, las limitaciones del tiempo de desarrollo, así como los limitados plazos han determinado que estas presentaciones no sean demasiado atractivas a un posible usuario, cosa que sí que se ha cuidado en el resto de partes de la interfaz de la herramienta.

A parte de que el producto realizado sea más o menos completo, cabe mencionar, que en la realización del proyecto se han utilizado conocimientos, herramientas y metodologías que han sido adquiridas en distintas asignaturas de la carrera. Los mismos han sido desarrollados en mucha más profundidad que en cualquier asignatura de la carrera.

7.2 Ayuda para el aprendizaje

Es una realidad que el aprendizaje es mucho más sencillo si se realiza de forma práctica, y si éste es asistido, en todo momento, por un apoyo de forma que se puedan resolver las dudas que vayan surgiendo sin demasiado esfuerzo por parte del alumno. En este marco todas las ayudas que se han incorporado al proyecto tratan de ser lo más completas posibles y con una cantidad de ejemplos considerable, explicando las distintas diferencias entre los distintos tipos de jerarquías.

7.3 Trabajo futuro

Desde el principio del proyecto el entrono desarrollado ha pretendido cubrir el pequeño hueco que otras herramientas dejan, porque no implementan cierta funcionalidad, no muestran los resultados de una forma muy intuitiva o simplemente no convence su interfaz de usuario.

Aunque esos han sido los objetivos principales, en todo momento el diseño se ha hecho de la forma más modular posible y tratando de documentar cada una de las fases del desarrollo. Por todo eso, prácticamente con unos conocimientos más o menos normales del entorno de desarrollo utilizado para la implementación, cualquiera podría ponerse manos a la obra e implementar sus propias estructuras que interactúen con las del programa, pudiendo también crearse una representación propia a añadir a las demás. Está claro que lo mejor habría sido permitir una incorporación automática de estructuras

a la aplicación, ésta sería una alternativa muy interesante aunque escapa a los objetivos de este proyecto.

Además como se ha comentado anteriormente la representación de los resultados de las simulaciones podría ser mucho más atractiva en el sentido de que podrían mostrarse gráficas 3D o informes en PDF completos sobre una simulación completa.

Como se ha visto, los campos más interesantes a la hora de realizar un posible trabajo futuro pasan por.

- Ampliación del simulador
- Mejora de la representación de los resultados y estadísticas.

ANEXO I DINEROIV

1. Dinero IV a nivel de código

1.1 Estructura de cache

DineroIV utiliza para simular una jerarquía de caches una lista de elementos cada uno de los cuales emula a una memoria cache, siendo el superior la memoria principal y los inferiores todos los demás niveles de la jerarquía que irán de mayor a menor enlazados (véanse figuras I.1 e I.2). Dentro de estas estructuras se guarda desde la configuración hasta los datos estadísticos, así como los datos de los bloques que hay en cache. La estructura principal se describe en la tabla I.1.

d4cache: simula una cache completa, dependiendo de la configuración de la misma los punteros a funciones que están dentro de esta estructura son inicializados de una forma, así el puntero (*ref) tendrá un valor distinto según el algoritmo de reemplazo que se haya configurado. Se utiliza en *cdmain.c* y *ref.c* . La función que inicializa una cache es *d4_new* y se configura una cache con *d4_Setup()*.

```
char *name;                                   /* para mostrar */
int cacheid;                                  /* unico en cada cache*/
int flags;
d4stackhead *stack;                           /* de aquí cuelgan todos los
bloques que tiene la cache */

d4pendstack *pending;                         /* aqui se guardan los bloques

        pendientes de actualizar en los niveles superiores dependiendo del algoritmo
hará unas cosas etc. */

struct d4_cache_struct *link;                 /* enlace con las demás caches
/*
* parametros de la cache
*/
```

```
int lg2blocksize;
int lg2subblocksize;
int lg2size;
int assoc;

int numsets;                                    /* número de conjuntos, derivado
de los

        anteriores*/

struct d4_cache_struct *downstream;             /* apunta al nivel superior

cache, se suele utilizar para recorrer las caches*/

void (*ref)(struct d4_cache_struct *, d4memref);        /* d4ref función que se usará
para realizar las referencias*/

/*Mas funciones que ejecutan las distintas politicas de reemplazo, dependen de la
configuración usada, pueden ser completamente distintas para dos caches, aquí
tambien se utilizan punteros para las funciones.


d4stacknode   *(*replacementf) (struct d4_cache_struct *, int stacknum,
d4memref, d4stacknode *ptr);

                                        /* indica la politica de reemplazo  */

d4pendstack   *(*prefetchf) (struct d4_cache_struct *, d4memref,
                                            int miss, d4stacknode *ptr);

                                        /* indica la política de escritura en caso de acierto
*/
int             (*wallocf) (struct d4_cache_struct *, d4memref);

                                        /* política de escritura en caso de fallo*/

int             (*wbackf) (struct d4_cache_struct *, d4memref, int,
                                        d4stacknode *ptr, int);

int             prefetch_distance;
```

```
int             prefetch_abortpercent;
char            *name_replacement;
char            *name_prefetch;
char            *name_walloc;
char            *name_wback;

#ifdef D4CACHE_USERHOOK
                D4CACHE_USERHOOK    /* allow additional stuff for user policies */
#endif

/*
int             nranges;
int             maxranges;
d4range                 *ranges;


/*Aqui se guardan las estadisticas de acceso
 */
double fetch        [2 * D4NUMACCESSTYPES];
double miss         [2 * D4NUMACCESSTYPES];
double blockmiss    [2 * D4NUMACCESSTYPES];
double comp_miss    [2 * D4NUMACCESSTYPES];     /* fallos forzosos*/
double comp_blockmiss [2 * D4NUMACCESSTYPES];
double cap_miss           [2 * D4NUMACCESSTYPES]; /* fallos de capacidad */
double cap_blockmiss  [2 * D4NUMACCESSTYPES];
double conf_miss    [2 * D4NUMACCESSTYPES];     /* fallos de conflicto */
double conf_blockmiss [2 * D4NUMACCESSTYPES];
double multiblock;
double bytes_read;
double bytes_written;
```

Tabla I.1 – Descripción de la estructura d4cache.

La organización

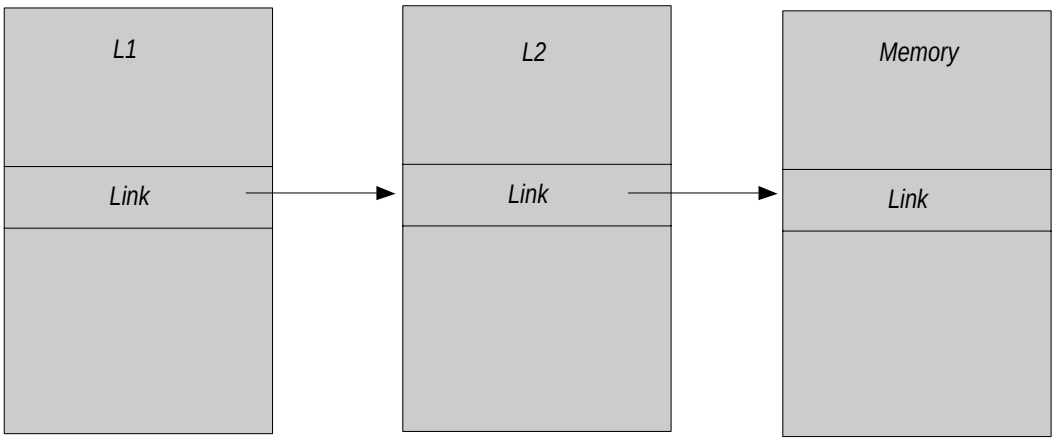

Figura I.1 – Organización de las estructuras en memoria.

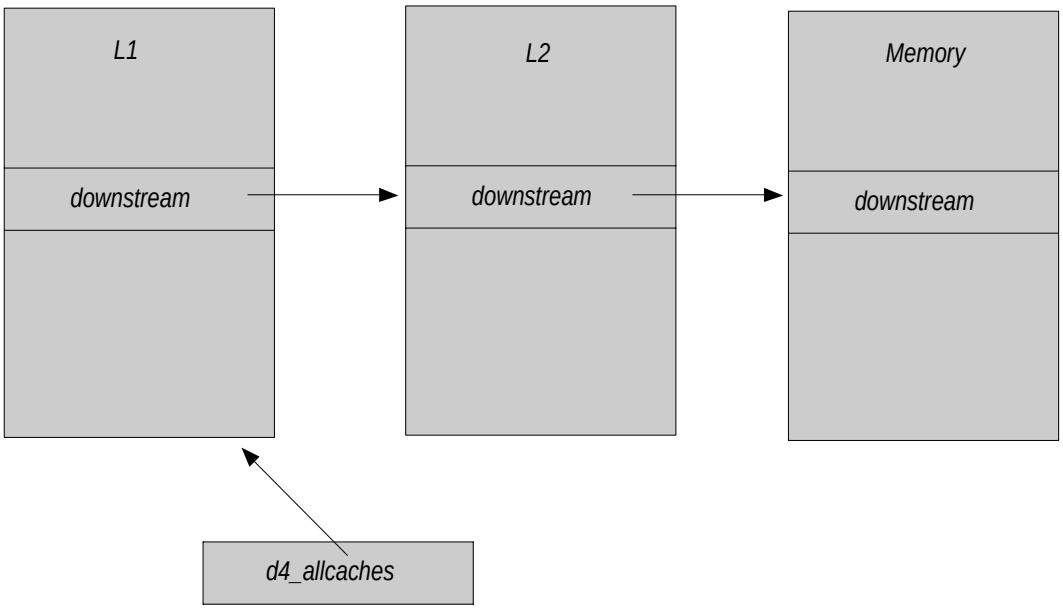

Figura I.2 – Puntero principal de las caches.

Dentro de cada una de estas estructuras de cache la forma de representar los bloques que están en cache es utilizando una lista (véase tabla I.3). Esta listas es una lista circular doblemente enlazada. Hay un un puntero Top que indica cual es el primer elemento de la lista (véase figura I.3). Esta lista siempre tiene un elemento más que el número de bloques de esa cache o conjunto, de forma que se utilizará como variable auxiliar en las operaciones de la lista.

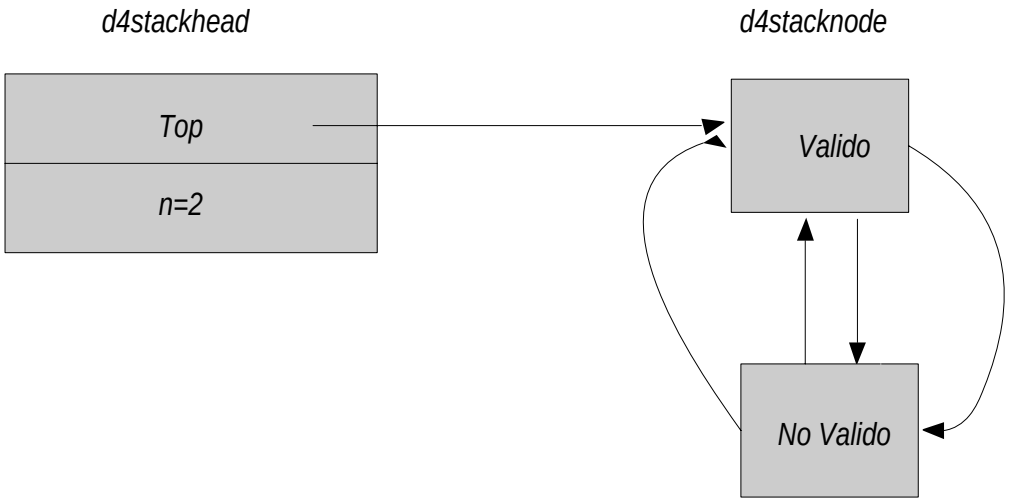

Figura I.3 – Estructura interna de una memoria cache.

d4stackhead: se utliza para conocer el primer nodo de la lista de bloques en ref.c.

d4stacknode *top;	// apunta al primer nodo de la lista de bloques de cache
int n;	//Tamaño de la lista de bloques 1+asociatividad

Tabla I.2 – Descripción de la d4_stackhead.

d4stacknode: representa a un bloque dentro de una cache, los nodos van formando una lista cuya organización dependerá del algoritmo de reemplazo (tabla I.3).

d4addr	blockaddr;	/* dirección del bloque */
unsigned int	valid;	/* bit de validez */
unsigned int	referenced;	/* bit de referencia */
unsigned int	dirty;	/* dirección de suciedad */
int	onstack;	/* número de conjunto */

struct d4_cache_struct *cachep;	/* puntero a la cache de la que forma parte */
struct d4_stacknode_struct *down;	/* nodo menos utilizado */
struct d4_stacknode_struct *up;	/* nodo más utilizado*/
struct d4_stacknode_struct *bucket;	/* auxiliar para las colisiones*/

Tabla I.3 – Descripción de la d4stacknode.

1.2 Organización y estructuras de datos dentro de una cache

La organización si el tipo de memoria es asociativa por conjuntos se realiza con un vector de punteros de los que cada uno tendrá una cantidad de nodos (número de bloques por conjunto+1) (véase tabla I.2 y figura I.4) y un puntero a lista que representa a esos nodos

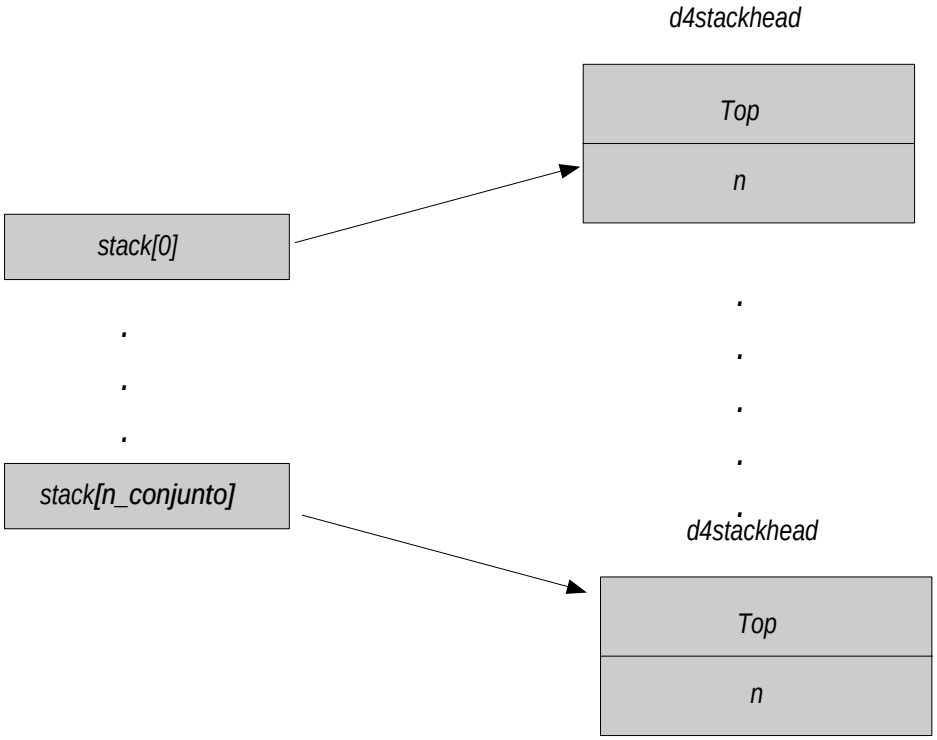

Figura I.4 – Estructura memoria asociativa.

El caso anterior (figura I.4) se **trata de memoria directa**. En este caso el puntero stack se usa como un vector que contiene en el puntero top el nodo que ha sido el último en utilizarse. Los nodos que simbolizan los bloques de memoria se unen

formando una lista circular, que siempre tendrá un elemento más para evitar tener que reorganizar la lista cada vez que se produzca una inserción. De esta forma cuando se produce un fallo basta con marcar el bloque actual (en este caso en memoria asociativa es distinto) como no válido, añadir los datos que sean necesarios al nodo que estaba marcado como no válido (véase figura 1.5) y modificar el puntero top dentro de *d4stackhead*.

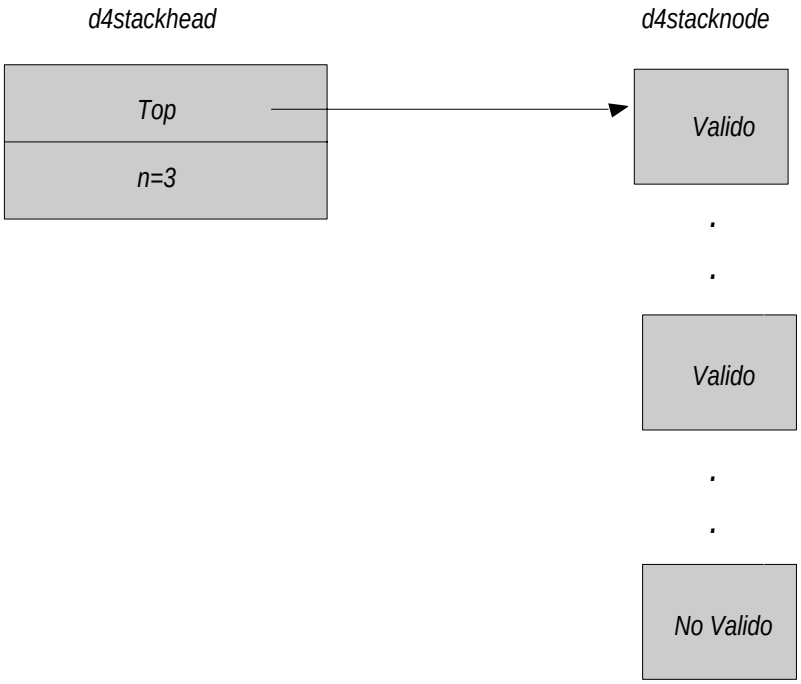

Figura I.5 – Estructura memoria asociativa de 2 vías.

Cuando se trata de memoria asociativa se tienen en cada entrada del vector stack tantas entradas como conjuntos y dentro de cada conjunto un número de nodos en la lista que será igual a la asociatividad+1 de forma que permite a la lista circular no tener que reorganizarse (véase figura I.6).

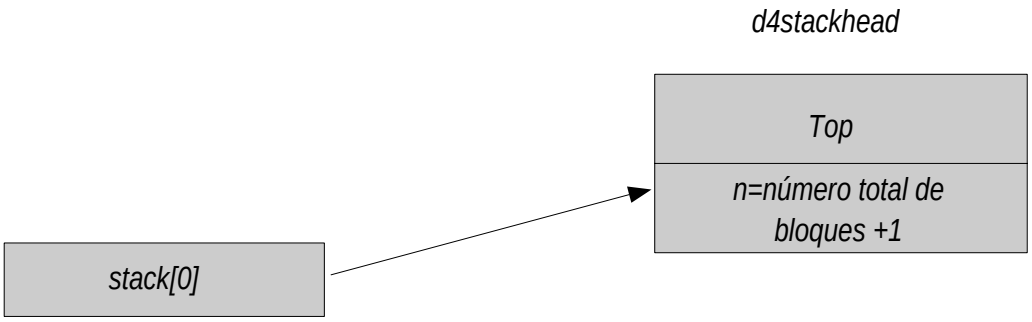

Figura I.6 – Estructura d4stackhead.

En el caso de una memoria totalmente asociativa sólo habrá una lista circular así que stack sólo tendrá una posición posible.

2. Estructuras de configuración

2.1 Creando caches

Los argumentos para configurar las caches que se introducen por lista de parámetros de entrada, según han sido introducidos éstos, son extraídos uno a uno y añadidos por separado a cada una de las caches, aunque antes se almacenan en la variable arglist (todas las definiciones necesarias se encuentran en config.h), el formato es una estructura de forma que cada campo indica una configuración distinta para un nivel específico de cache..Una vez se tienen los mismos se pasa a crear los niveles según la configuración de cache (véase tabla I.4).

```
case 0: cd = ci = c = d4new (ci); break;          /* crear una cache unificada*/
case 1:     ci = c = d4new (ci); break;           /* crear una cache de datos*/
case 2:     cd = c = d4new (cd); break;           /* crear una cache de instrucciones*/
c->link = d4_allcaches;                           /* enlazando caches*/
d4_allcaches = c;                                 /* creando lista*/
```

Tabla I.4 – Creación de caches.

Una vez creadas se pasa a la inicialización de las mismas. Este proceso se expondrá en el siguiente apartado.

2.2 Inicializar una cache

Las caches se inicializan con la función **init_1cache (d4cache *c, int lev, int idu).** Esta función se encarga de configurar una cache según los datos pasados que previamente habían sido guardados en arglist.

Luego se comprueban las configuraciones con **d4setup()** que utiliza el **puntero d4_allcaches** (véase figura I.1).

El siguiente paso es reservar memoria para los punteros d4stackhead (véase tabla I.5).

```
c->stack = calloc (c->numsets+((c->flags&D4F_CCC)!=0),
                                                    sizeof(d4stackhead));
        /* reserva de memoria*/
```

Tabla I.5 – Creación de caches: reserva para los conjuntos.

Se reserva memoria para los nodos que formarán la lista de bloques (véase tabla I.5).

```
        nnodes = c->numsets * (1 + c->assoc) +
                                        (c->numsets * c->assoc + 1) * ((c-
>flags&D4F_CCC)!=0);

        nodes = calloc (nnodes, sizeof(d4stacknode));
        /* creando la lista de bloques*/
```

Tabla I.6 – Creación de caches: reserva para los bloques.

Se organizan los nodos como una lista circular (véase tabla I.7 y figura I.2).

```
                        for (i = 0;  i < c->numsets+((c->flags&D4F_CCC)!=0);  i++) {
                                int j, n;
```

```
                                        n = 1 + c->assoc * ((i < c->numsets) ? 1 : c-
>numsets);
                                        c->stack[i].top = ptr;
                                        c->stack[i].n = n;
                                        for (j = 1;  j < n-1;  j++) {
                                                ptr[j].onstack = i;
                                                ptr[j].down = &ptr[j+1];
                                                ptr[j].up = &ptr[j-1];
                                        }
                                        ptr[0].onstack = i;
                                        ptr[0].down = &ptr[1];
                                        ptr[0].up = &ptr[n-1];
                                        ptr[n-1].onstack = i;
                                        ptr[n-1].down = &ptr[0];
                                        ptr[n-1].up = &ptr[n-2];
                                        ptr += n;
                                }

        /* se guardan en una tabla hash para acceso más rápido*/
                                assert (ptr - nodes == nnodes);
        #if D4_HASHSIZE == 0
                                d4stackhash.size += c->numsets * c->assoc;
        #endif
        /* Se guarda el número total de nodos*/
                d4nnodes += nnodes;
```

Tabla I.7 – Creación de caches: organización circular.

Y finalmente se hace para la memoria principal (véase tabla 1.8).

```
        d4stackhash.table = calloc (d4stackhash.size, sizeof(d4stacknode*));
```

Tabla I.8 – Creación de caches: reserva para los bloques.

3. Ejecución

3.1 Inicio

Una vez se han configurado todas las estructuras de forma correcta ya se puede iniciar una ejecución. Para obtener los datos que guían la simulación (traza) se utiliza una función:

next_tace_item que según la configuración obtiene los parámetros de entrada de forma distinta.

Según el tipo de acceso que se utilice en la traza se utiliza una u otra (véase tabla 1.9) para aplicar la referencia (si es instrucción o dato u otros).

```
r = next_trace_item();                                  /* generar una referencia*/

switch (r.accesstype) {                                 /* según el tipo de acceso
pasar a una cache distinta*/
                case D4XINSTRN:     d4ref (ci, r);  break;
                case D4XINVAL:      d4ref (ci, r);  /* fall through */
                default:            d4ref (cd, r);  break;
                }
```

Tabla I.9 – Estructura de referencias.

3.2 Buscando un bloque

3.2.1 Buscando en el primer nivel

La función **d4ref** es la que realiza todo el trabajo duro generando las lecturas e imitando la jerarquía de memoria.

Al principio llena los campos que formarán parte del bloque:

```
const d4addr blockaddr = D4ADDR2BLOCK (c, mr.address);          /* dirección*/
const d4memref m = d4_splitm (c, mr, blockaddr);                /* convertir a
referencia*/
```

```
const int atype = D4BASIC_ATYPE (m.accesstype);   /*extrayendo el tipo de acceso*/
const int setnumber = D4ADDR2SET (c, m.address);   /* número de conjunto*/
const int ronly = D4CUSTOM && (D4VAL (c, flags) & D4F_RO) != 0;
 /*política de escritura*/
const int walloc = !ronly && atype == D4XWRITE && D4VAL (c, wallocf) (c, m);
/* post escritura*/
const int sbbits = D4ADDR2SBMASK (c, m);
/* dirección de subbloque, sólo en prebúsqueda por subbloques*/
```

Tabla I.10– Accediendo a un bloque.

Un parámetro bastante importante será **setnumber** que indica el conjunto al que irá ese bloque y que se usará bastante más adelante.

Luego se buscará el bloque en este nivel y en el conjunto donde debe estar (véase tabla I.11), aunque antes se comparará con el bloque que está en la parte más alta de la pila que es el último que se ha insertado.

```
ptr = c->stack[setnumber].top;
        if (ptr->blockaddr == blockaddr && ptr->valid != 0)
                ; /* found it */
        else if (!D4CUSTOM || D4VAL (c, assoc) > 1)
                ptr = d4_find (c, setnumber, blockaddr);
        else
                ptr = NULL;
```

Tabla I.11– buscando un bloque (I).

Luego se comprueba qué bloque debe ser sustituido según la política de esa cache (véase tabla I.12) que se ha fijado anteriormente y se modifican las partes necesarias para que el bloque quede colocado en memoria.

```
ptr = D4VAL (c, replacementf) (c, setnumber, m, ptr);
```

Tabla I.12– Comprobando política (II).

En el caso de que se haya producido un acierto, no hay mucho que comentar, simplemente se actualizarán algunos datos estadísticos y se retorna de la función. En el caso de que se produzca un fallo será necesario acceder a niveles superiores, lo cual se explica en la siguiente sección.

3.2.1 Accediendo a niveles superiores

En el caso de que se haya producido un fallo se añade a la cache el campo **pending** que indicará las lecturas o escrituras que deben pasarse al nivel superior (véase tabla I.13). Normalmente suele contener una sola dirección de bloque aunque si se hace postescritura se suele crear una lista. Aquí la **variable c** es la cache.

```
        if (!ronly && atype == D4XWRITE && !wback) {
                        d4pendstack *newm = d4get_mref();
                        newm->m = m;
                        newm->next = c->pending;
                        c->pending = newm;
                }

        if (miss && (ronly || atype != D4XWRITE ||
                            (walloc && m.size != D4REFNSB (c, m) << D4VAL (c,
lg2subblocksize)))) {
                        d4pendstack *newm = d4get_mref();
                        /* note, we drop prefetch attribute */
                        newm->m.accesstype = (atype == D4XWRITE) ? D4XREAD :
atype;
                        newm->m.address = D4ADDR2SUBBLOCK (c, m.address);
                        newm->m.size = D4REFNSB (c, m) << D4VAL (c,
lg2subblocksize);
```

```
                newm->next = c->pending;
                c->pending = newm;
        }

/*Cambios en las estadisticas a partir de aquí*/
```

Tabla I.13– Generando referencias pendientes.

Una vez acabado esto hay que actualizar los niveles superiores. Para eso se llama para cada referencia que cuelga de c->pending a la función d4_ref() de la que se ha estado hablando(véase tabla I.14).

```
d4_dopending (c, newm)

        c->pending = newm->next;

……


        c->downstream->ref (c->downstream, newm->m);
```

Tabla I.14– Generando referencias pendientes a todos los niveles.

ANEXO II. PRUEBAS

1. Introducción

El propósito de este anexo es describir las pruebas realizadas al desarrollo, así como el resultado de las mismas. Todas las pruebas han sido descritas en el capítulo 6, que puede ser consultado para obtener más información.

2. Resultados y descripción

A continuación se pasa a identificar las distintas pruebas, véase tabla II.2.1 y siguientes.

2.1. Prueba de funciones

En esta sección se verán las pruebas sobre los casos de uso que se mostraron en el capítulo 4. Cada tabla identificará una batería de pruebas realizada sobre una cierta funcionalidad.

2.1.1 Caso de uso *gestión de la configuración*

Id	Acción	Flujo
Configuración_1	Acceso a la configuración	Un usuario accede a la ventana de configuración, debe mostrarse la configuración actual en todo momento y en el caso de que no exista ninguna debe dejar cambiarla.

Configuración_2	Añadir/eliminar elementos a la configuración	Añadir/eliminar elementos
Configuración_3	Configurar elementos	Configurar algunos elementos
Configuración_4	Estudio de los valores límite	Inserción de valores en los cuadros de texto no numéricos o mayores a los permitidos.
Configuración_5	Guardar la configuración	Guardar una configuración
Configuración_6	Cargar una configuración	Cargar una configuración y comprobar que realmente es la que previamente se había guardado
Configuración_7	Visualizar la configuración	Una vez introducida la configuración comprobar que se muestra la configuración correctamente.

Tabla II.1 – Pruebas *gestión de configuración*.

Id	Resultado
Configuración_1	Satisfactorio
Configuración_2	Fallido. Resuelto problema con ciertos botones que aparecían activos aunque no pudieran añadirse o eliminarse algunos elementos
Configuración_3	Satisfactorio. Pulsando sobre cualquier elemento permite configurar y lo hace bien.
Configuración_4	Satisfactorio. Se ha observado la limitación en TLB y MP, ningún campo numérico permite introducir caracteres.
Configuración_5	Satisfactorio. La configuración se guarda con extensión "cfg".
Configuración_6	La configuración previamente guardada se carga correctamente
Configuración_7	En la ventana de simulación se muestran bien los datos

Tabla II.2 – Resultados pruebas *gestión de configuración*.

2.1.2 Caso de uso *cargar traza*

Id	Acción	Flujo
Cargar_1	Cargar una traza de accesos a memoria	Acceso a la opción de menú y carga de la traza
Traza_2	Visualizar	Visualizar la traza en la ventana de simulación
Traza_3	Ver paso a paso	La traza se va actualizando paso a paso.

Tabla II.3 – Pruebas *cargar de traza*.

Id	Resultado
Cargar_1	Satisfactorio. La traza se carga correctamente
Traza_2	Satisfactorio. La traza se visualiza en la ventana de simulación
Traza_3	Satisfactorio. Paso a paso la ventana simulación va mostrando la línea ejecutada de la traza.

Tabla II.4 – Resultados pruebas *cargar de traza*.

2.1.3 Caso de uso *simular*

Id	Acción	Flujo
Simular_1	Simular paso a paso	Se ejecuta una simulación paso a paso y se comprueba que los resultados son correctos
Simular_2	Simulación completa	Se ejecuta una simulación completa y se comprueba que los resultados son correctos
Simular_3	Visualizar	Se comprueba que los elementos están mostrándose correctamente

Tabla II.5 – Pruebas *simular*.

Id	Resultado
Simular_1	Satisfactorio. Todas las estructuras se van actualizando y se muestran los datos correctos.
Simular_2	Satisfactorio.
Simular_3	Satisfactorio

Tabla II.6 – Resultados pruebas *simular*.

2.1.4 Caso de uso mostrar estadísticas

Id	Acción	Flujo
Estadisticas_1	Mostrar estadísticas	Ejecutar la ventana para mostrar las estadísticas y comprobar que los resultados sean los correctos.

Tabla II.7 – Pruebas *estadísticas*.

Id	Resultado
Estadisticas_1	Satisfactorio. Las estadísticas se muestran correctamente.

Tabla II.8 – Resultados pruebas *estadísticas*.

2.1.5 Pruebas específicas de la memoria virtual

A continuación se describen las pruebas que se han realizado sobre la memoria virtual. Estas pruebas se han realizado configurando distintos tamaños de TLB y tabla de páginas, y comprobando la ejecución con distintas trazas. En todo momento se ha comprobado que la sustitución de páginas se realiza correctamente, la política de postescritura se realiza correctamente y se actualizan las estadísticas.

Id	Configuración del TLB	Configuración de la tabla de páginas
vm_1	**32 entradas**	**256 entradas**

Tabla II.9 – Resultados pruebas de memoria virtual (I).

Id	Número de marcos físicos	Configuración del TLB	Configuración de la tabla de páginas	Variante de la traza de entrada	Número de pruebas	Resultados
vm_2	**16**	**2 Entradas**	**32**	**Provocando escrituras y lecturas sobre 3 páginas distintas**	**4**	Satisfactorio.

Tabla II.10 – Resultados pruebas de memoria virtual (II).

Id	Número de marcos físicos	Configuración de la TLB	Configuración de la tabla de páginas	Variante de la traza de entrada	Número de pruebas	Resultados
vm_3	**12**	**2 Entradas**	**32**	**Provocando desalojo de páginas en TP.**	**4**	Satisfactorio.

Tabla II.11 – Resultados pruebas de memoria virtual (III).

2.2 Prueba de comparación

Id	Acción	Flujo
Cmp_1	Comprobar una cache	Comprobar resultados estadísticos para distintas configuraciones con un solo nivel de cache.
Cmp_2	Comprobar varios niveles con distintas configuraciones	Igual que el anterior pero con distintos niveles
Cmp_3	Comparar ejecuciones parciales de una misma traza para una misma ejecución	Se van añadiendo líneas a la traza, se ejecuta y se comprueba.

Tabla II.12– Pruebas de comparación.

Para estas pruebas se ha utilizado el simulador DineroIV, comparando directamente los datos obtenidos en el mismo con los que obtiene este desarrollo, utilizando las mismas configuraciones y traza de entrada. A continuación se muestra una lista de las simulaciones ejecutadas, junto con la lista de configuraciones de DineroIV, los resultados obtenidos en este simulador.

Id	Políticas de búsqueda probadas	Políticas de escritura probadas	Políticas de reemplazo probadas	Número de niveles probados	Número de pruebas	Resultados
Cmp_1	**Prebúsqueda Por demanda**	**Todas**	**Todas**	**1**	12	Fallido. Los resultados no son totalmente iguales cuando utilizamos un algoritmo de reemplazo aleatorio.

Tabla II.13 – Resultados pruebas de comparación (I).

A continuación se incluyen las distintas configuraciones para DineroIV utilizadas en esta parte.

Antes se describirán las opciones de DineroIV que se han utilizado:

−l*N*−*T***sbsize** *P:* establece el tamaño de subbloque, siendo el nivel de cache de 1 a *N* , T el tipo de cache(unificada, datos o instrucciones) y el número de bytes del subbloque *P*.
−l*N*−*T***size** *P:* donde *P* es el tamaño de la cache para un nivel específico *N*.
−l*N*−*T***assoc** *U : U* establece la asociatividad para un nivel *N de cache.*
−l*N*−*T***repl** *C:*establece la politica de reemplazo para un nivel *N de* cache (**l**=LRU, **f**=FIFO,**r**=random).
−l*N*−*T***fetch** *C:* establece la política de búsqueda para un nivel específico de cache N, donde C es: **d**=demanda, **a**=siempre prebúsqueda, **m**=prebúsqueda con fallo, **t**=prebúsqueda con marcado, **l**=carga adelante, **s**=subbloque.
−l*N*−*T***walloc** *C:* establece la política de escritura (**a**=carga en escritura, **n**=no carga en escritura).
−l*N*−*T***wback** *C: establece la política de postescritura* (**a**=siempre, **n**=nunca).

./dineroIV -l1-dassoc 1 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 2 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d <trazar1

./dineroIV -l1-dassoc 8 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback n -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

/dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback n -l1-dwalloc a -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback n -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar

./dineroIV -l1-dassoc 4 -l1-drepl f -l1-dsize 256 -l1-dbsize 16 -l1-dfetch m -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 1024 -l1-dbsize 32 -l1-dfetch a -l1-dsbsize 16 -l1-dwback n -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl f -l1-dsize 512 -l1-dbsize 32 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl r -l1-dsize 256 -l1-dbsize 16 -l1-dfetch a -l1-dsbsize 16 -l1-dwback n -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

./dineroIV -l1-dassoc 4 -l1-drepl r -l1-dsize 256 -l1-dbsize 16 -l1-dfetch a -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 16 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -informat d<trazar1

Id	Políticas de búsqueda	Políticas de escritura probadas	Políticas de reemplazo probadas	Tamaños de cache utilizados	Número de niveles probados	Número de pruebas	Resultados
Cmp_2	**Prebúsqueda Por demanda**	**LRU FIFO**	**Todas**	**Variable**	De 1 a 4	**7**	Satisfactorio.

Tabla II.14 – Resultados pruebas de comparación (II).

A continuación se incluyen las distintas configuraciones para DineroIV utilizadas en esta parte.

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 1024 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc a -l1-iassoc 1 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -l2-dassoc 16 -l2-drepl l -l2-dsize 256 -l2-dbsize 16 -l2-dfetch d -l2-dsbsize 16 -l2-dwback a -l2-dwalloc a -l2-iassoc 16 -l2-irepl l -l2-isize 256 -l2-ibsize 16 -l2-ifetch d -l2-isbsize 16 -l3-uassoc 4 -l3-urepl l -l3-usize 1024 -l3-ubsize 16 -l3-ufetch d -l3-usbsize 16 -l3-uwback a -l3-uwalloc a -l4-uassoc 2 -l4-urepl l -l4-usize 2048 -l4-ubsize 16 -l4-ufetch d -l4-usbsize 16 -l4-uwback a -l4-uwalloc a -informat d<reducida traza1

./dineroIV -l1-uassoc 1 -l1-urepl l -l1-usize 256 -l1-ubsize 16 -l1-ufetch d -l1-usbsize 16 -l1-uwback n -l1-uwalloc a -l2-uassoc 4 -l2-urepl l -l2-usize 512 -l2-ubsize 16 -l2-ufetch d -l2-usbsize 16 -l2-uwback n -l2-uwalloc n -l3-uassoc 4 -l3-urepl f -l3-usize 1024 -l3-ubsize 16 -l3-ufetch d -l3-usbsize 16 -l3-uwback a -l3-uwalloc n -informat d<reducida traza1

./dineroIV -l1-dassoc 1 -l1-drepl l -l1-dsize 1024 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 2 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -l2-uassoc 32 -l2-urepl f -l2-usize 512 -l2-ubsize 16 -l2-ufetch a -l2-usbsize 16 -l2-uwback a -l2-uwalloc a -l3-uassoc 2 -l3-urepl l -l3-usize 256 -l3-ubsize 16 -l3-ufetch m -l3-usbsize 16 -l3-uwback n -l3-uwalloc a -informat d<reducida traza1

./dineroIV -l1-dassoc 4 -l1-drepl l -l1-dsize 256 -l1-dbsize 16 -l1-dfetch m -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 2 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -l2-uassoc 128 -l2-urepl l -l2-usize 2048 -l2-ubsize 16 -l2-ufetch a -l2-usbsize 16 -l2-uwback a -l2-uwalloc n -informat d<reducida traza1

./dineroIV -l1-uassoc 1 -l1-urepl f -l1-usize 512 -l1-ubsize 16 -l1-ufetch m -l1-usbsize 16 -l1-uwback n -l1-uwalloc a -l2-uassoc 2 -l2-urepl f -l2-usize 2048 -l2-ubsize 16 -l2-ufetch a -l2-usbsize 16 -l2-uwback a -l2-uwalloc n -informat d<reducida traza1

./dineroIV -l1-uassoc 8 -l1-urepl r -l1-usize 128 -l1-ubsize 16 -l1-ufetch d -l1-usbsize 16 -l1-uwback n -l1-uwalloc a -l2-uassoc 4 -l2-urepl f -l2-usize 256 -l2-ubsize 16 -l2-ufetch a -l2-usbsize 16 -l2-uwback n -l2-uwalloc a -informat d< reducida traza1

./dineroIV -l1-dassoc 4 -l1-drepl f -l1-dsize 512 -l1-dbsize 16 -l1-dfetch d -l1-dsbsize 16 -l1-dwback a -l1-dwalloc n -l1-iassoc 2 -l1-irepl l -l1-isize 256 -l1-ibsize 16 -l1-ifetch d -l1-isbsize 16 -l2-dassoc 128 -l2-drepl l -l2-dsize 2048 -l2-dbsize 16 -l2-

dfetch d -l2-dsbsize 16 -l2-dwback n -l2-dwalloc n -l2-iassoc 2 -l2-irepl l -l2-isize 256 -l2-ibsize 16 -l2-ifetch d -l2-isbsize 16 -informat d<reducida traza1

3. Pruebas de carga

Se han realizado usando trazas muy grandes de 20000 entradas, ejecutando varias veces, cambiado configuraciones, y luego volviendo a cargar, cambiar y ejecutar.

4. Valoración de las pruebas

Se puede decir a grandes rasgos que las pruebas han sido satisfactorias, aunque también viendo la ejecución de las pruebas, que han sido limitadas.

Por otro lado el equipo encargado de las pruebas es muy limitado y, aún conociendo en profundidad la implementación, no ha podido entrar en muchos detalles.

Se ha detectado una diferencia en las pruebas frente a DineroIV. Se trata de la función aleatoria que el mismo utiliza y que por razones de compatibilidad fue cambiada en este desarrollo. Esto hace que los resultados para la política de reemplazo aleatorio no sean los mismos. De todas formas el resto de pruebas han demostrado que el funcionamiento es el mismo de DineroIV.

Además, las pruebas han demostrado que el funcionamiento es bastante estable y que el uso del simulador no tiene por qué dar ningún problema.

Algunas pruebas han sido limitadas, pero han tratado de ser totalmente exhaustivas.

BIBLIOGRAFÍA.

[1] G. Booch , James Rumbaugh, Ivar Jacobson . El lenguaje unificado de modelado. Traducción José Sáez Martínez ; supervisión de la traducción y revisión técnica Jesús J. García Molina (2000).

[2] Jan Edler, Mark D. Hill. Dinero IV. Univ. of Wisconsin Computer Sciences. URL: http://www.cs.wisc.edu/~markhill/DineroIV.

[3] Herbert Grünbacher. Xcache. Vienna University of Technology.

[4] Glenford J. Myers. The art of testing. John Wiley & Sons. 1979

[5] Israel Koren. Computer architecture educational tools. URL:http://www.ecs.umass.edu/ece/koren/architecture/

[6] David A. Patterson, John L. Hennessy. Estructura y diseño de computadores. Editorial reverté, 2000.

[7] Roger S. Pressman. Ingeniería del Software. Un enfoque práctico. Mc Graw Hill 3ª Edicion.

[8] Miguel A. Vega Rodriguez. SMPCACHE.

[9] Miguel A. Vega Rodríguez, Juan M. Sánchez Pérez, Juan A. Gómez Pulido. Innovación Docente en la Asignatura de Arquitectura de Computadores mediante el Uso de Simuladores. VI Jornadas sobre la Enseñanza Universitaria de la Informática. JENUI'2000.

[10] Miguel A. Vega Rodríguez, Juan M. Sánchez Pérez, Juan A. Gómez Pulido. Mejorando la Docencia en las Prácticas de la Asignatura Arquitectura e Ingeniería de Computadores. XII Jornadas de Enseñanza Universitaria de la Informática. JENUI'2006.