Desarrollo de un sistema simulador de la arquitectura SPARC mediante el sistema operativo Mach y el ambiente NeXTSTEP

Tomás Balderas Contreras
e-mail: balderas@optima.cs.buap.mx
Hugo García Monroy
e-mail: gmonroy@servidor.unam.mx

Departamento de aplicación de microcomputadoras
Instituto de Ciencias
Universidad Autónoma de Puebla
Apdo. Postal 461, C.P. 72000, Puebla, Pue. México
Teléfono: (22) 43–67–11
Fax: (22) 43–63–30
Mayo, 1999

Resumen

A través de los años, los científicos e investigadores han elaborado diferentes herramientas para reproducir diversos fenómenos o bien para simular y analizar el comportamiento de diferentes sistemas, en particular los dispositivos electrónicos. El desarrollo de estas herramientas ha cambiado la forma de trabajar de los diseñadores así como la clase de habilidades requeridas para realizar su labor. Anteriormente, un diseñador de circuitos necesitaba construir un prototipo y verificarlo en un laboratorio, ahora requiere de ciertos dotes para tratar con herramientas de software que le permiten editar un diseño y observar su comportamiento, adicionalmente debe conocer el modo de resolver los problemas que se presenten. Además de auxiliar en el proceso de diseño y manufactura de dispositivos, la simulación tiene otros campos de aplicación. Por medio de ella es posible construir modelos de cierto tipo de sistemas mediante el uso de plataformas distintas. El uso de este tipo de aplicaciones de la simulación tiene por objetivo contribuir en la conclusión de proyectos de desarrollo en ausencia del equipo real. En este documento describimos las características de un sistema simulador de la arquitectura de procesadores SPARC versión ocho, la forma en que se realizó su diseño e implantación y las herramientas utilizadas, entre ellas un ambiente de trabajo y desarrollo orientado a objetos denominado NeXTSTEP. El sistema simulador de la arquitectura SPARC está concebido como una aplicación que brinda soporte a la codificación, ejecución y depuración de programas escritos en lenguaje ensamblador SPARC, además proporciona estadísticas sobre la ejecución de las instrucciones. Su desarrollo obedece a un objetivo académico para dar a conocer el enfoque RISC.

Frases y palabras clave: Simulación de arquitecturas de computadoras, arquitecturas RISC, arquitectura SPARC, ambiente de trabajo NeXTSTEP, programación orientada a objetos, lenguaje C Objetivo.

1 Introducción

Desde su introducción en la segunda mitad de la década de los ochenta hasta la actualidad, la arquitectura de computadoras SPARC ha evolucionado considerablemente y se ha enriquecido con los nuevos avances tecnológicos logrados a lo largo de poco más de diez años [Sun, 1998]. En nuestros días, la arquitectura SPARC es una de las plataformas RISC más importantes en el mercado, pues es el estándar de las estaciones de trabajo desarrolladas por Sun Microsystems y el soporte para la ejecución del sistema operativo Solaris y de un buen número de aplicaciones y herramientas de desarrollo. La situación ideal en la que todos los estudiantes de una carrera de ciencias y/o ingeniería en computación puedan estudiar una arquitectura RISC mediante el contacto directo con una máquina de esta naturaleza y con sus herramientas de software desafortunadamente está aún muy lejos de nuestra realidad, debido a que una de sus características principales es la de ser sumamente costosas. Sin embargo, mediante la simulación es posible subsanar un poco estas carencias. El objetivo de este proyecto es proporcionar a todas las personas interesadas

en las arquitecturas RISC de computadoras, un ambiente para la simulación de la ejecución de programas, ya sea que se utilice con fines didácticos o como un soporte al desarrollo de software de sistemas como compiladores o ensambladores en la ausencia del equipo real.

Para llevar a buen término el desarrollo de este proyecto, se ha decidido utilizar como herramientas principales para la construcción del sistema las siguientes. Primero, una versión del sistema operativo Mach compatible con BSD 4.3 que además de proporcionar las funciones tradicionales de Unix, nos permite la creación de tareas, múltiples hilos de ejecución y los medios para comunicarlos. Segundo, un ambiente de desarrollo totalmente orientado a objetos e implantado sobre Mach conocido como NeXTSTEP, el cual no sólo proporciona una interfaz gráfica para Unix, sino que además brinda a los programadores un poderoso conjunto de herramientas y varios grupos de clases para la elaboración de aplicaciones de manera confiable y en poco tiempo. Es importante señalar las razones de la elección de NeXTSTEP como plataforma de software para nuestro sistema. Primero, NeXTSTEP es Unix, lo que es una garantía de confianza, pues detrás de Unix existen casi 30 años de investigación y desarrollo. Segundo, el conjunto de herramientas ofrecidas por el sistema junto con la programación orientada a objetos en lenguaje C Objetivo [NeXT, 1993] permiten el desarrollo en menos tiempo de aplicaciones también orientadas a objetos funcionalmente correctas. Tercero, los llamados al sistema que ofrece el micronúcleo de Mach, nos permiten diseñar y construir aplicaciones multihilos, algo bastante conveniente para realizar la simulación de ciertos rasgos característicos de las arquitecturas RISC, tales como el uso extenso de la segmentación o pipeline. Por último, el sistema NeXTSTEP se encuentra disponible en el Departamento de Aplicación de Microcomputadoras del Instituto de Ciencias de la U.A.P., ejecutándose sobre plataformas Intel 80486 y Pentium, también se dispone de su documentación completa y de la valiosa experiencia adquirida a lo largo de varios años por parte de los investigadores del departamento.

El resto de este artículo se encuentra organizado como sigue. En las secciones 2 y 3 se presentan los conceptos más importantes sobre la arquitectura SPARC y el ambiente NeXTSTEP, respectivamente. Las secciones 4, 5, 6 y 7 describen los componentes de la aplicación y su diseño. La sección 8 describe la forma en que se implantan los mecanismos de entrada y salida. Finalmente, las secciones 9 y 10 informan sobre el estado actual del proyecto y las extensiones pendientes.

2 La arquitectura SPARC

Sun Microsystems definió la arquitectura SPARC (*Scalable Processor Architecture*) entre los años 1984 y 1987. El desarrollo de esta arquitectura estuvo basado en los trabajos sobre RISC realizados en la Universidad de California en Berkeley entre 1980 y 1982 [Patterson, 1985; Tabak, 1990]. Los científicos de Sun, quienes poseían amplia experiencia en sistemas operativos, compiladores y diseño de hardware, realizaron varias modificaciones con el propósito de mejorar los diseños producidos en Berkeley.

La visión de la compañía fue establecer un ambiente de computación abierto y estándar. Sun Microsystems proporcionó licencias a distintos fabricantes para explotar la arquitectura y desarrollar implantaciones propias de la misma, con el propósito de ofrecer productos distintos en precio y rendimiento, pero capaces todos de ejecutar las aplicaciones binarias existentes para la arquitectura SPARC. Cabe señalar que los derechos correspondientes a una implantación específica de la arquitectura son propiedad de su respectivo desarrollador. Con el paso del tiempo se instituyó el consorcio SPARC International, dedicado a dirigir la evolución y la estandarización de la arquitectura SPARC. El consorcio registra la evolución de la arquitectura en documentos tales como *El manual de la arquitectura SPARC* [SPARC, 1992]. Tales documentos son empleados por los fabricantes de hardware y desarrolladores de software de aplicación o de sistemas con la finalidad de que sus productos cumplan los estándares establecidos por SPARC International y de esta forma asegurar la compatibilidad. Entre las industrias que conforman dicha organización podemos mencionar a Fujitsu, Cypress Semiconductor, Texas Instruments, Phillips y Sun Microsystems.

A través de los años la arquitectura ha evolucionado notablemente. Los cambios han sido registrados en los manuales de las diferentes versiones de la arquitectura. La versión más reciente de la arquitectura SPARC es la versión 9. Las implantaciones de esta versión (UltraSPARC-I, UltraSPARC-II y UltraSPARC-IIi, por ejemplo) tienen, entre otras características, frecuencias superiores a los 200 MHz. Es necesario aclarar que también existe compatibilidad entre las diferentes versiones de la arquitectura SPARC.

2.1 Componentes del procesador

En la arquitectura SPARC V8 se encuentran definidas las siguientes unidades para el procesador.

- **La unidad entera.** Lleva a cabo la ejecución de las instrucciones aritméticas enteras y lógicas, calcula las direcciones requeridas y contiene el archivo de registros enteros de propósito general de 32 bits. El número de tales registros es dependiente de la implantación pero se encuentra acotado por 40 y 520. El archivo de registros se encuentra organizado en ventanas solapadas [SPARC, 1992; Stallings, 1996; Garner, 1990] (entre 2 y 32 dependiendo de la implantación) para realizar de manera eficiente el paso de parámetros.
- **La unidad de punto flotante.** Contiene un conjunto de 32 registros de 32 bits para almacenar

operandos de punto flotante. Los formatos de tales operandos así como las operaciones están conformes al estándar ANSI/IEEE 754-1985 [Goldberg, 1991].

- **Coprocesador.** Este es opcional y no se encuentra definido por la arquitectura pues es dependiente de la implantación.

Además define un espacio lineal de direcciones virtuales de 32 bits, en el cual los datos escalares constituidos por múltiples bytes se almacenan mediante un esquema *big endian*. Los tipos de datos soportados por la arquitectura incluyen enteros con y sin signo de 64, 32, 16 y 8 bits, datos de punto flotante de precisión simple (32 bits), precisión doble (64 bits) y precisión extendida (128 bits), además de tipos de datos enteros con etiqueta.

2.2 Tipos de instrucción

El repertorio de instrucciones se encuentra organizado en las categorías siguientes.

- **Carga y almacenamiento.** Son las únicas instrucciones que pueden hacer referencia a memoria, permiten almacenar y recuperar datos de tipo entero y de punto flotante. Especifican la dirección mediante dos formas, suma de dos registros ($\texttt{reg}_1 + \texttt{reg}_2$) o mediante la suma de un registro y un inmediato ($\texttt{reg}_1 + \texttt{simm13}$).

- **Aritméticas y lógicas.** Realizan operaciones aritméticas, de corrimiento y lógicas con operandos contenidos en los registros. Existen versiones de tales instrucciones que modifican los códigos de condición en el registro de estado (acarreo, cero, sobreflujo y negativo) y otras que no. Las instrucciones de multiplicación y división pueden tomar operandos de 64 bits de longitud o producir un resultado de dicha longitud.

- **Transferencia de control.** Existen instrucciones de transferencia condicionales, incondicionales y para hacer llamados a subrutinas. Las de tipo condicional verifican el valor de los bits de condición mencionados anteriormente. El tipo de transferencia que se realiza es una *transferencia de control retardada*, es decir, las ramificaciones tienen efecto con un retardo de una instrucción. Es posible alterar este comportamiento anulando tal instrucción de retardo mediante la especificación de un bit para tal propósito.

- **Instrucciones de punto flotante.** Toman sus operandos del archivo de registros de punto flotante. hay instrucciones para realizar conversiones entre datos de distinta precisión y entre datos enteros. Las instrucciones de comparación modifican el valor de los códigos de condición para punto flotante.

Además existen instrucciones para realizar lecturas y escrituras a los registros de estado del procesador. Dependiendo de una implantación particular es posible especificar un coprocesador y sus instrucciones. Finalmente, todas las instrucciones se codifican mediante un patrón de 32 bits de longitud y empleando uno de tres formatos, dependiendo de la instrucción.

3 NeXTSTEP y el lenguaje C Objetivo

En 1989 NeXT Computer Corporation (hoy extinta, desafortunadamente) liberó los primeros modelos de sus estaciones de trabajo basadas en los procesadores Motorola 680x0. Los diseños de hardware de la compañía eran bastante inusuales pues contaban con dispositivos revolucionarios en aquella época como circuitos de procesamiento digital de señales y lectores de discos ópticos. Sin embargo, más inusual aún era la tecnología empleada en el desarrollo del sistema operativo orientado a objetos de tales máquinas, NeXTSTEP. NeXTSTEP es mucho más que una versión de Unix con una interfaz gráfica excelente y bastante elaborada. En términos más formales, NeXTSTEP es un ambiente de trabajo y de desarrollo, construido en base al sistema operativo Mach y un conjunto de tecnologías existentes orientadas a objetos.

Mach es un sistema operativo desarrollado en la Universidad de Carnegie Mellon en la segunda mitad de la década de los ochenta y es compatible con la versión 4.3 de BSD. Se encuentra dividido en dos partes, un *micronúcleo* que proporciona las funcionalidades básicas de un sistema operativo y un *ambiente de soporte al sistema* que complementa al micronúcleo para ofrecer todas las capacidades de un verdadero sistema operativo. Mach es un sistema operativo multihilos (*multithreaded*) que puede ser empleado sin dificultad en sistemas de múltiples procesadores y es también adecuado para realizar computación distribuida.

El lenguaje híbrido C Objetivo es un superconjunto del lenguaje C que admite la programación orientada a objetos. Tal extensión se soporta teniendo como base los conceptos introducidos por el lenguaje Smalltalk [Xerox, 1981]. C Objetivo soporta las características propias de este paradigma tales como *objeto*, *clase*, *herencia*, *encapsulación* y *polimorfismo*. NeXT eligió este lenguaje debido a su gran dinamismo, que permite realizar tanto tipificación como enlace en tiempo de ejecución.

La versión de NeXTSTEP para el desarrollador proporciona un conjunto de herramientas para la elaboración de aplicaciones. De tales herramientas, las más comúnmente empleadas son las siguientes.

- *AppKit.* Es un conjunto de clases escritas en lenguaje C Objetivo. Cada clase define a un objeto que puede ser utilizado para construir la interfaz gráfica de una aplicación. Por ejemplo, ventanas, botones, objetos de texto, campos de texto, etc. A través del mecanismo de herencia es posible

extender las capacidades de tales objetos según lo requiera la aplicación.

- *Project Builder.* Es la herramienta principal de desarrollo. Se emplea para organizar los archivos de código y de soporte que componen la aplicación, proporciona acceso a las herramientas utilizadas para crear y/o modificar tales componentes y se encarga de construir e instalar la versión ejecutable de la aplicación.

- *Interface Builder.* Permite construir de una manera interactiva y gráfica la interfaz de la aplicación mediante la selección de los objetos desde una paleta y su colocación en el lugar apropiado dentro de la interfaz. Permite modificar las características de tales objetos y establecer enlaces entre ellos para que interaccionen unos con otros. Además permite la creación de instancias de clases definidas por el desarrollador.

Otro de los componentes de NeXTSTEP es *Window Server*, un proceso de bajo nivel que se ejecuta en segundo plano (*background*). Las funciones de este proceso son bastante importantes. Primero, se encarga de desplegar imagenes y texto en la pantalla, ya que cuenta con un intérprete del lenguaje PostScript, y segundo, es el responsable de canalizar los eventos producidos en el sistema (debido al manejo del ratón o el teclado) a las aplicaciones adecuadas.

4 El simulador de la arquitectura SPARC

Nuestra aplicación es un *simulador de conjunto de instrucciones* [Magnusson *et al.*, 1998]. Este tipo de sistemas reproducen dentro de una plataforma determinada el efecto de la ejecución de instrucciones de otra clase de computadora una instrucción a la vez. La denominación anterior enfatiza también que ciertos aspectos relativos a la organización del procesador son excluidos de la simulación. En contraste, un *simulador a nivel de sistema* es capaz de simular dispositivos, memoria física y otros componentes que le permiten ejecutar programas más complejos como un sistema operativo y sus aplicaciones.

Está destinado a todos los usuarios interesados en adquirir conocimiento sobre el funcionamiento de microprocesadores con diseños RISC, además, les proporciona un medio para la evaluación de productos de software de sistema tales como compiladores y ensambladores.

Para realizar de manera correcta la simulación de la ejecución de las instrucciones propias de un microprocesador SPARC, deben tenerse en cuenta los siguientes aspectos. La máquina virtual debe contar con una representación de los recursos inherentes a una máquina real, tales como memoria principal, ventanas de registros enteros, un archivo de registros de punto flotante, registros de estado, etc., además de que la aplicación debe contar con una interfaz con el usuario amigable y adecuada. Para construir un producto que cumpla con los requisitos anteriores de una manera adecuada, consideramos que NeXTSTEP y Mach son las mejores opciones.

Las metodologías de diseño orientado a objetos, así como las herramientas para la implantación de tales diseños han demostrado su efectividad en la actividad de desarrollo de varios sistemas de software. En efecto, la programación orientada a objetos se ha convertido en el paradigma dominante en el desarrollo de software de calidad. En particular, en el Departamento de Aplicación de Microcomputadoras han sido desarrollados algunos sistemas que explotan las características de orientación a objetos proporcionadas por NeXTSTEP, entre tales sistemas destaca un *editor de tarjetas de circuito impreso* [Briones *et al.*, 1997], el cual hace uso extenso de la programación orientada a objetos para modelar tanto los componentes de una tarjeta de circuito impreso como los módulos de control de la aplicación. El resultado es una herramienta bastante interactiva y con grandes capacidades de ruteo automático. En el caso de nuestro simulador, la programación orientada a objetos juega un papel relevante, hasta cierto grado. El papel principal lo desempeña un conjunto de módulos que se encargan de soportar toda la carga de procesamiento en el sistema. En nuestra aplicación, es la interfaz hacia el usuario la que mediante las propiedades orientadas a objetos de NeXTSTEP aísla al usuario de las particularidades de tal conjunto de módulos. Ambos componentes del sistema son descritos en la siguiente sección.

5 La arquitectura del sistema simulador

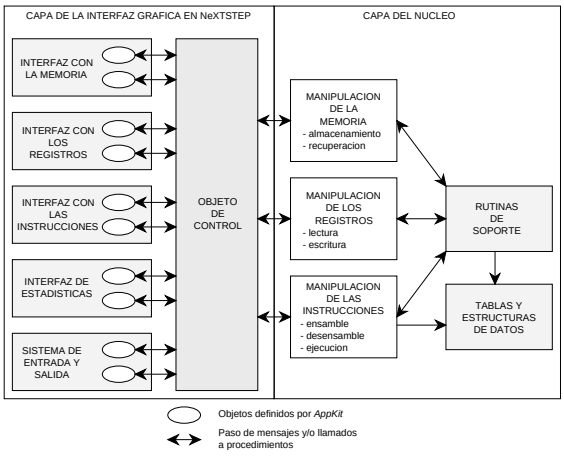

Figura 1: La arquitectura en capas del simulador.

Uno de los objetivos definidos por el equipo de desarrollo es trasladar el sistema simulador hacia plataformas distintas (tanto en software como en hardware) a la empleada para el desarrollo de la primera versión. Un poco

de análisis en este punto de la portabilidad nos muestra la imposibilidad de transportar la totalidad del código del sistema hacia un ambiente distinto de NeXTSTEP. Lo anterior debido a que el diseño de la interfaz es completamente dependiente de la tecnología orientada a objetos empleada en el diseño de NeXTSTEP, tecnología no presente en otros ambientes operativos. Sin embargo, podemos estructurar el proyecto de tal manera que sea posible transportar algún porcentaje del total de código sin dificultad, dejando que la parte restante sea escrita en el nuevo ambiente operativo. De acuerdo al principio anterior, establecimos que la estructura del programa se encuentre dividida en dos partes importantes, el *núcleo* y la *interfaz*. Esta división se realiza tomando en cuenta un modelo de capas, tal como lo muestra la figura 1.

Mediante este modelo el usuario únicamente observa la capa superior, la interfaz, que es la capa a través de la cual envía sus solicitudes y mediante la cual recibe los resultados de dichas peticiones. Para poder proporcionar las respuestas necesarias a los usuarios, la capa de interfaz se vale de la capa inferior, el núcleo, dentro de la cual se realiza el procesamiento que el usuario ha solicitado. A continuación se describen las características de ambas capas.

- La capa del núcleo está concebida como un conjunto de módulos que permiten realizar las operaciones relacionadas con la ejecución de las instrucciones (que involucran carga o almacenamiento de datos, procesamiento aritmético y lógico, etc.), manejo de los recursos como registros o memoria, además de proporcionar los medios necesarios para ofrecer servicios como ensamble y desensamble de instrucciones. Los módulos contenidos en esta capa deben ser portables de manera inmediata, pues se encuentran escritos en lenguaje C. La capa del núcleo por sí sola no es funcional, necesita de un medio de control que invoque a sus componentes a petición del usuario.

- La capa de interfaz está dedicada a la interacción con el usuario. Permite ordenar la modificación de los recursos, la ejecución de los programas, el ensamble y/o desensamble de instrucciones, etc. Como todas las aplicaciones dentro de NeXTSTEP, la interfaz está constituida por un conjunto de objetos controlados por otro objeto "maestro" y que son capaces de responder a los eventos que les son enviados por el ambiente. Obviamente, la naturaleza de NeXTSTEP obliga a la adopción de un modelo orientado a objetos para llevar a cabo la implantación de esta capa. Lo anterior también provoca que la implantación de la interfaz no pueda transportarse a un ambiente distinto.

6 Estructura de la capa del núcleo

La capa del núcleo está constituida como un conjunto de módulos, los cuales están organizados en grupos. Además cuenta con ciertas estructuras de datos y las funciones necesarias para manipularlas. Por último, contiene algunas funciones de soporte que brindan varios servicios. Cada grupo de módulos está dedicado a proporcionar los medios necesarios para realizar actividades específicas. Por ejemplo, el manejo de la memoria, las operaciones con los registros, el ensamble, desensamble y ejecución de instrucciones. A continuación se describen brevemente las características de cada grupo.

- **Manipulación de la memoria.** En el simulador la memoria se representa por un arreglo lineal de datos de 8 bits. Para realizar las operaciones de almacenamiento hacia y recuperación desde la memoria es necesario tener en cuenta el esquema de almacenamiento *big endian* empleado en SPARC V8. Las rutinas dentro de este grupo se encargan de transferir datos hacia y desde la memoria considerando los esquemas de almacenamiento de SPARC V8 y de la plataforma de hardware en que se ejecuta el simulador.

- **Manipulación de los registros.** Las funciones dentro de este grupo son empleadas para realizar un manejo adecuado de las estructuras de datos usadas para simular los registros enteros de propósito general. Son necesarias puesto que toman en cuenta la organización en ventanas solapadas del archivo de registros.

- **Manipulación de instrucciones.** Estos procedimientos llevan a cabo las tareas solicitadas al sistema para realizar el ensamble, desensamble y ejecución de instrucciones. El número de módulos implantados dentro de este grupo es grande, pues existen funciones para análisis sintáctico, análisis semántico y ensamble de instrucciones en lenguaje ensamblador así como funciones para el desensamble y la ejecución de instrucciones binarias.

- **Módulos de soporte.** Aquí se encuentran rutinas de conversión, de manipulación de cadenas y rutinas aritméticas y lógicas sobre datos enteros de longitud arbitraria.

6.1 Estructuras de datos

Existen tres estructuras de datos definidas dentro del núcleo. Su misión es simplificar los procedimientos de ensamble, desensamble y ejecución. Son descritas a continuación.

- La *tabla de símbolos* es una herramienta fundamental para el proceso de ensamble. Por cada mnemónico reconocido existe una entrada que consta del mismo mnemónico, un apuntador a la función de ensamble correspondiente y un índice para la tabla de códigos.

- La *tabla de códigos* contiene información de gran importancia para los módulos de ensamble, desensamble y ejeucución. Entre los campos de cada entrada se encuentran el mnemónico de una instrucción, el tipo de parámetros de la instrucción, una máscara para construir el código binario y dos apuntadores a las funciones para desensamble y ejecución.

- La *tabla de registros* es empleada por los módulos de ensamble y desensamble. Cada entrada contiene el identificador de cada registro y si es necesario, un identificador equivalente, además contiene la dirección de cada registro entero.

SPARC V8 define un archivo de registros enteros circular y dividido en ventanas. Tal archivo de registros se modela mediante un arreglo lineal de enteros de 32 bits. Los módulos de lectura y escritura se encargan de manipular el valor del *apuntador a la ventana actual* (contenido en uno de los registros de estado) junto con la dirección del registro para obtener el índice correcto en el arreglo.

La característica más importante es que todos los módulos se encuentran escritos en lenguaje C. Debido a nuestras expectativas de portabilidad del núcleo, sus componentes realizan únicamente llamados al sistema, cualquier tipo de interacción directa del usuario con el núcleo es imposible, esta comunicación debe establecerse a través de una interfaz conveniente que debe ser implantada para cada ambiente. La portabilidad a otros sistemas se asegura si el núcleo es escrito en un lenguaje de programación estándar como C y si sus componentes utilizan los servicios proporcionados por el sistema operativo Unix, o bien, por las funciones de biblioteca equivalentes en los compiladores para DOS, por ejemplo.

6.2 El proceso de ensamble

Existen varias funciones de ensamble dentro del núcleo, cada una de las cuales se encarga de llevar a cabo el proceso para un grupo de instrucciones con la misma sintaxis. Sin embargo, todas ellas se invocan mediante el mismo proceso e implantan esencialmente la misma estrategia. Para llevar a cabo el ensamble de una instrucción en lenguaje ensamblador se realizan los siguientes pasos.

1. Se obtiene el primer elemento sintáctico de la cadena que contiene la instrucción. Tal elemento corresponde al mnemónico de la instrucción.

2. Se realiza la búsqueda del mnemónico obtenido en la tabla de símbolos.

3. Si la búsqueda es exitosa se invoca a la función de ensamble correspondiente. El llamado se realiza utilizando el tercer campo de la entrada hallada en la tabla, el cual es un apuntador a la función apropiada.

4. Una vez en ejecución el módulo de ensamble adecuado, este invoca a otro procedimiento que realiza el análisis sintáctico de la cadena y la separación de los elementos sintácticos restantes, es decir, los operandos.

5. A continuación entra en acción otra función, cuyo objetivo es determinar si los operandos son válidos. En caso de ser así, este modulo habrá obtenido también los números de los registros a los que hace referencia la instrucción, en caso de que los operandos sean identificadores de registros. También obtiene un valor numérico si existe algún valor inmediato como operando. Si la verificación de los operandos fracasa, el módulo de ensamble termina regresando un código de error.

6. Si los operandos son correctos se procede a construir el código binario de la instrucción. En base al mnemónico de la instrucción y al tipo de sus operandos se obtiene de la tabla de códigos un número entero de 32 bits (una máscara). Dependiendo del formato de la instrucción que se pretende ensamblar y del tipo de los operandos, se realizan operaciones `or` entre los valores obtenidos en el paso anterior y la máscara para producir finalmente la instrucción.

6.3 El proceso de desensamble

Del mismo modo que en el caso anterior, los distintos módulos dedicados al desensamble de instrucciones se comportan de la misma manera. Los pasos que se llevan a cabo para realizar el desensamble de una instrucción son los siguientes.

1. Se accesa cada entrada en la tabla de códigos. Para cada iteración se efectúa una operación `and` entre la instrucción y un valor que se determina según la posición actual en la tabla. Si el resultado de tal operación es igual al valor de la máscara contenido en la entrada actual de la tabla se procede con el siguiente paso. De otro modo, se avanza a la siguiente entrada. Si la tabla termina y ninguna función fue invocada entonces el código de la "instrucción" es inválido y el proceso termina regresando un código de error.

2. Se invoca a la función de desensamble adecuada mediante el campo apuntador correspondiente dentro de la entrada actual.

3. Una vez en ejecución, el módulo de desensamble verifica el tipo de operandos que contiene la instrucción, esta información la obtiene de un campo específico dentro de la tabla de códigos.

4. En base a la información anterior, se obtienen de la instrucción los operandos, que pueden ser direcciones de registros, valores inmediatos, desplazamientos, etc.

5. Mediante la tabla de códigos, la tabla de registros, los módulos de soporte y los valores obtenidos en el paso anterior se construye la cadena de caracteres que representa la instrucción en lenguaje ensamblador SPARC.

6.4 El proceso de ejecución

Una vez discutidos los pasos llevados a cabo para desensamblar una instrucción, podemos abordar el proceso de ejecución. Existen varios módulos dentro del núcleo destinados a simular la ejecución de las instrucciones. Como se ilustró anteriormente, la labor de las instrucciones varía mucho entre las diferentes catogorías. El paso 1 del algoritmo para desensamble se aplica idéntico para el caso de la ejecución. La invocación de la función correcta se realiza a través del apuntador a la función de ejecución contenido en uno de los campos de las entradas de la tabla de códigos. Una vez que el procedimiento ha sido llamado, este es responsable de sus actos. Dos módulos de ejecución diferentes tienen muy poco o nada en común.

Como un ejemplo; consideremos la instrucción `add %g5, %l4, %o6`. El proceso de ejecución sobre el código binario de esta instrucción provoca que sea invocado el módulo *Ejecutar_Add* dentro del núcleo. Esta función se encarga de verificar el tipo de operandos dentro de la instrucción, de realizar la recuperación de los valores desde el archivo de registros, de realizar la suma de tales valores y finalmente de escribir el resultado de la operación en el registro apropiado.

7 La capa de interfaz

Como se explicó anteriormente, la capa de la interfaz es muy importante para el funcionamiento de la aplicación ya que es el medio a través del cual el usuario solicita los servicios de la capa del núcleo. Aunque el núcleo puede ser transportado a otro sistema operativo, la implantación de la interfaz puede no serlo. En el caso de interfaces gráficas puede ser necesario diseñar e implantar una nueva interfaz para cada ambiente operativo al que se traslade el núcleo. La comunicación, que se establece entre el usuario y el núcleo a través de la interfaz, se encuentra coordinada por un objeto que se encuentra dentro de esta capa, el *objeto controlador del sistema*, el cual recibe los mensajes enviados por los objetos en la interfaz, hace llamados al núcleo para llevar a cabo el procesamiento deseado y finalmente envía mensajes de vuelta a los objetos de la interfaz para mostrar resultados.

7.1 Componentes de la interfaz

A continuación se describen los componentes de la interfaz, a través de los cuales el usuario interacciona con la aplicación.

- **Interfaz con los registros.** A través de esta ventana el usuario puede consultar y/o modificar los valores contenidos en los registros enteros, de punto flotante, contadores de programa, banderas de condición y otros registros de estado. Además puede desplazarse a lo largo de las ventanas de registros. En la figura 2 se muestra esta ventana.

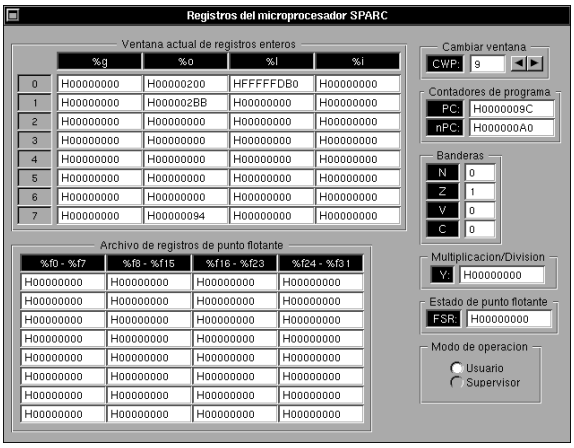

Figura 2: La interfaz con los registros.

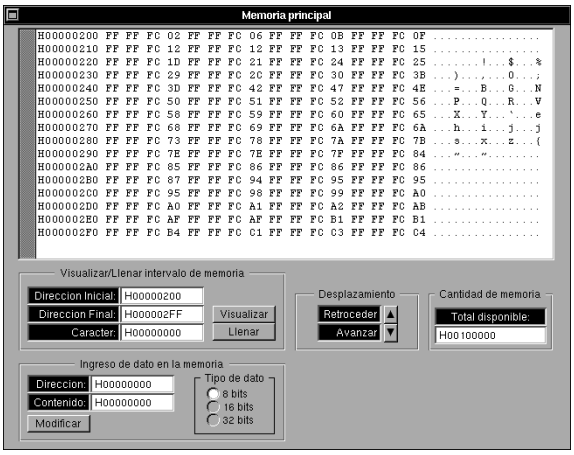

Figura 3: La interfaz con la memoria.

- **Interfaz con la memoria.** Mediante esta interfaz el usuario puede examinar intervalos de memoria, modificar el valor contenido en alguna dirección, llenar con un caracter un intervalo y desplazarse a través del contenido de la memoria. La estructura de esta interfaz se muestra en la figura 3.

- **Interfaz para los códigos de instrucciones.** Con esta ventana es posible invocar el ensamble de instrucciones desde un archivo o desde un objeto campo de texto en la misma interfaz. Es posible invocar la ejecución de las instrucciones contenidas en un intervalo de memoria especificado, modificar tal intervalo, trazar una instrucción o desensamblar un conjunto de instrucciones en memoria. En la figura 4 se muestra la interfaz.

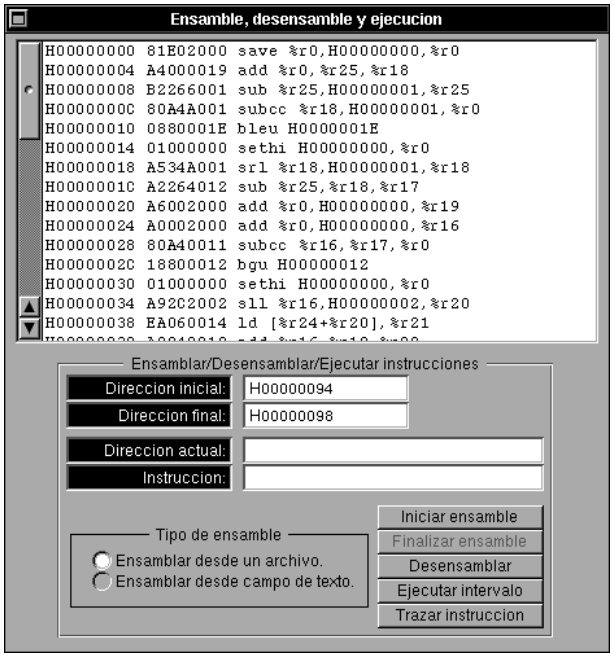

Figura 4: La interfaz con los códigos de instrucciones.

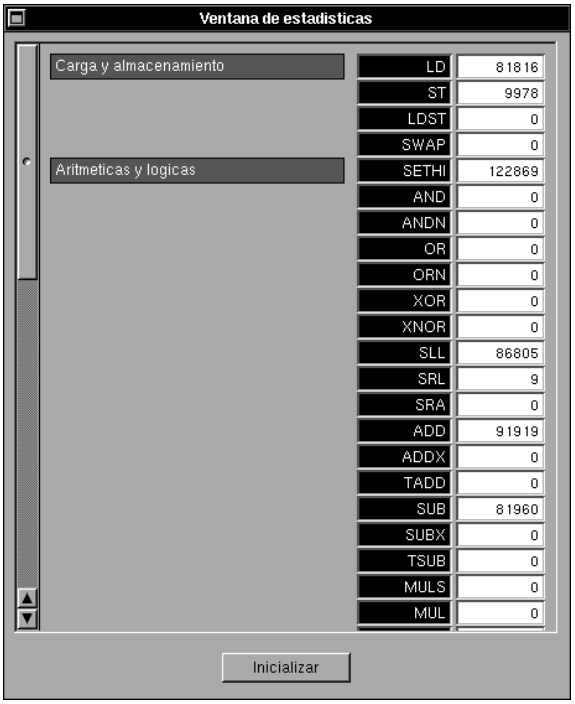

Figura 5: La ventana de estadísticas.

- **Ventana de estadísticas.** Después de realizada la ejecución de las instrucciones contenidas en un intervalo, la interfaz de estadísticas se actualiza para mostrar el número de instrucciones de cada categoría que fueron ejecutadas. Un esquema de esta ventana se muestra en la figura 5.

La interfaz está diseñada tomando en cuenta un modelo de descomposición orientado a objetos. La forma de control es a través de un modelo centralizado. El objeto controlador es único y recibe mensajes desde las interfaces descritas anteriormente. Todas las clases que definen a los objetos en la interfaz son proporcionadas por el *Application Kit* (*AppKit*), la clase que define al objeto controlador fue diseñada e implantada en lenguaje C Objetivo especialmente para esta aplicación. Los objetos que componen la interfaz reciben y responden a los eventos generados por el usuario y algunos interaccionan con el objeto de control a través del envío de mensajes. El objeto de control responde a los mensajes que recibe e invoca la acción requerida (método), la cual puede involucrar interacción con la capa del núcleo. Además, el objeto de control debe comunicarse nuevamente con los otros objetos para enviarles información que debe ser mostrada al usuario como resultado del proceso que fue solicitado.

7.2 La clase SimuladorSPARC

Esta clase define al objeto controlador. Se encuentra colocada en la jerarquía de clases como una subclase de *Object*. Algunas de sus variables instancia son de tipo entero y se emplean para representar direcciones y datos en memoria, las restantes son apuntadores a objetos que se conocen como *conectores* (*outlets*). Mediante *Interface Builder* es posible establecer conexiones en ambos sentidos entre los objetos de la interfaz y el objeto de control. Una vez definidos los enlaces, los objetos envían mensajes al objeto de control cuando el usuario interacciona con ellos, estos mensajes indican el tipo de acción que se debe llevar a cabo. De la misma forma el objeto de control a través de sus conectores puede enviar mensajes a las instancias en la interfaz. Los métodos definidos e implantados dentro de la clase se encargan de invocar a los módulos dentro del núcleo para realizar operaciones como modificación de registros, ensamble, desensamble y ejecución de instrucciones, modificación del contenido de la memoria, etc.

8 El sistema de entrada y salida

Durante el desarrollo de la aplicación nos dimos cuenta de la conveniencia de contar con un mecanismo para que el usuario alimentara con datos a un programa en ejecución. De la misma forma, mediante este mecanismo el programa debe ser capaz de mostrar sus resultados. También se observó la necesidad de respaldar la información importante en medios de almacenamiento secundario. La solución consistió en incluir un conjunto de funciones que permitan realizar entrada y salida durante la ejecución de un programa, a tal conjunto lo denominamos como *sistema de entrada y salida*.

El problema inmediato fue hallar la forma de invocar los módulos de entrada y salida durante la ejecución de un programa. La solución propuesta para lograrlo fue definir un conjunto de instrucciones cuya ejecución

Mnemonico	Descripcion	Codigo de operacion
Entrada desde la consola		
inputi	Valor entero	111000
inputf	Valor de punto flotante de simple precision	111010
inputd	Valor de punto flotante de doble precision	111100
inputs	Cadena de caracteres	111110
inputc	Caracter	101000
Salida a la consola		
outpui	Valor entero	111001
outputf	Valor de punto flotante de simple precision	111011
outputd	Valor de punto flotante de doble precision	111101
outputs	Cadena de caracteres	111111
outputc	Caracter	101010
Entrada y salida en archivos		
create	Crear archivo	101011
open	Abrir archivo	101100
read	Leer de un archivo	101101
write	Escribir a un archivo	101110
close	Cerrar un archivo	101111
lseek	Modificar el apuntador del archivo	101001

Figura 6: Descripción de las instrucciones en el sistema de entrada y salida.

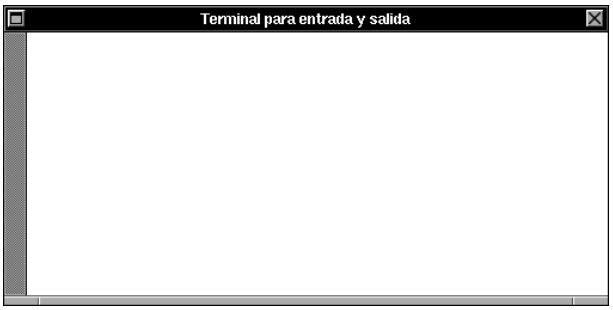

Figura 7: Ventana de la "consola" para entrada y salida.

provocara el llamado a las funciones de entrada y salida. En este punto es necesario realizar las siguientes observaciones.

- El sistema de entrada y salida es una extensión definida por nosotros al repertorio de instrucciones de la arquitectura SPARC V8, esta extensión no se encuentra definida por la arquitectura y no hay forma de que un procesador SPARC reconozca estas instrucciones.

- La ejecución de estas instrucciones se lleva a cabo dentro de la capa de interfaz. El núcleo puede ensamblarlas y desensamblarlas, pero la ejecución la delega a la capa de interfaz puesto que para llevarla a cabo es necesario interaccionar con los objetos que conforman la aplicación. El núcleo no conoce sobre la implantación de la interfaz y no tiene por que hacerlo.

En la figura 6 se resumen las instrucciones de las que se compone este sistema. Las instrucciones se ensamblan empleando el formato de instrucciones 3 [SPARC, 1992], y se tuvo especial cuidado en escoger códigos de operación sin uso dentro de este formato. Las instrucciones se agrupan en las siguientes categorías.

- **Instrucciones de entrada.** Permiten al usuario ingresar datos desde una ventana como la que se muestra en la figura 7. Puede hacerse una analogía con la ventana principal de la aplicación *Terminal* de NeXTSTEP. Se tiene una instrucción de entrada por cada uno de los siguientes tipos de datos. Entero, punto flotante de simple y doble precisión, cadena de caracteres y caracter.

- **Instrucciones de salida.** Muestran el valor de una variable en la consola. De la misma forma que las instrucciones de entrada, se tienen diferentes instrucciones de salida para cada tipo de dato.

- **Entrada y salida en archivos.** Se tiene una instrucción de este tipo por cada llamado al sistema en Unix para la administración de archivos. Estas instrucciones se implantan simplemente mediante tales servicios del sistema operativo.

9 Resultados

La implantación del núcleo se llevó a cabo inicialmente utilizando un equipo IBM PS/1 con procesador Intel 80486 a 25 MHz bajo el sistema operativo DOS, una interfaz mediante comandos de línea también fue construida. Este sistema preliminar fue probado exitosamente en dicho ambiente y posteriormente fue transportado sin problema alguno a una estación de trabajo Sun Ultra 10 con procesador UltraSPARC-IIi a 300 MHz y ambiente operativo Solaris 2.6. Al mismo tiempo se construía y probaba el sistema simulador en el ambiente NeXTSTEP empleando una computadora Gateway 2000 con procesador Intel Pentium a 60 MHz. La transportación del núcleo se llevó a cabo sin dificultades.

La aplicación ha sido probada de modo satisfactorio mediante la ejecución de programas escritos en lenguaje ensamblador SPARC. Tales programas incluyen la implantación de algoritmos de búsqueda y ordenamiento, obtención del factorial, cálculo de potencias, operaciones sobre matrices y eliminación de Gauss con pivoteo parcial y sustitución regresiva. El proyecto se encuentra aún en fase de verificación puesto que la clase de problemas que pueden ser tratados es muy amplia.

10 Conclusiones

Durante el desarrollo de este proyecto tuvimos la oportunidad de verificar las palabras de varios autores que califican al ambiente NeXTSTEP como una poderosa herramienta de desarrollo [Redman y Silbar, 1995; Mahoney y Garfinkel, 1993]. El tiempo invertido en diseñar e implantar la interfaz hacia el usuario fue realmente corto, las facilidades que proporciona el ambiente nos permitieron implantar la clase *SimuladorSPARC* sin complicaciones de importancia. Aunque para ser justos debemos decir que la implantación del sistema de entrada y salida requirió de la familiarización detallada del mecanismo de

respuesta a eventos, lo que consumió una mayor cantidad de tiempo que las otras actividades.

Aunque el resultado obtenido hasta el momento es satisfactorio, el proyecto no está terminado en su totalidad. A continuación se enuncian las características que pueden ser adicionadas al sistema.

- Inclusión de módulos para implantar aritmética de punto flotante en precisión extendida (128 bits).
- Simulación de la ejecución de instrucciones mediante *pipeline*, empleando los servicios de Mach para la creación y comunicación de hilos de ejecución.
- Implantación de un compilador del lenguaje REC (*Regular Expression Compiler*) [Cisneros y McIntosh, 1986] que genere secuencias de instrucciones SPARC que puedan ser analizadas en nuestra aplicación.

Referencias

[Briones *et al.*, 1997] J. Alfonso Briones García, Silvana Bravo Hernández y Hugo García Monroy. Diseño e implantación de un editor de tarjetas de circuitos impresos con capacidades de ruteo automático en el ambiente NeXTSTEP. No publicado. 1997.

[Wilkes, 1990] Maurice Wilkes. It's all software, now. En *Communications of the ACM*. Páginas 19–21. Octubre 1990.

[Patterson, 1985] David A. Patterson. Reduced Instruction Set Computers. En *Communications of the ACM*. Páginas 9–15. Enero 1985.

[Redman y Silbar, 1995] James M. Redman y Richard R. Silbar. Selecting an operating system part II: NeXTSTEP. En *Computers in physics*. Páginas 261–266. Mayo/junio 1995.

[Ibarra y Vergara, 1995] Enrique Ibarra Anaya y Gabriel Vergara. Evaluación del sistema operativo NeXTSTEP en el Grupo Financiero InverMéxico. En *Soluciones Avanzadas*. Páginas 5–11. Febrero 1995.

[SPARC, 1992] SPARC International, Inc. The SPARC Architecture Manual, Version 8. Prentice-Hall. 1992.

[Mahoney y Garfinkel, 1993] Michael K. Mahoney y Simson L. Garfinkel. NeXTSTEP Programming. Step one: Object oriented applications. Springer-Verlag. 1993.

[NeXT, 1992] NeXT Computer, Inc. NeXTSTEP Operating system software. Addison- Wesley. 1992.

[NeXT, 1993] NeXT Computer, Inc. The Objective C languaje. On-line documentation. 1993.

[Sun, 1998] Sun Microsystems, Inc. The UltraSPARC-IIi Processor. Technology White Paper. 1998.

[Stallings, 1996] William Stallings. Computer organization and architecture, designing for performance. Prentice-Hall. 1996.

[Garner, 1990] Robert B. Garner. SPARC: Scalable Processor Architecture. En *The Sun technology papers*. Springer-Verlag. Páginas 75–100. 1990.

[Tabak, 1990] Daniel Tabak. RISC Systems. John Wiley & Sons Inc. 1990.

[Xerox, 1981] The Xerox Learning Research Group. The Smalltalk 80 system. En *Byte*. Páginas 36–48. Agosto 1981.

[Bedichek, 1990] Robert C. Bedichek. Some efficient architecture simulation techniques. En *Proceedings of Winter '90 USENIX Conference*. Páginas 53–63. 1990.

[Magnusson *et al.*, 1998] Peter S. Magnusson, Håkan Grahn, Fredrik Dahlgren, Magnus Karlsson, Andreas Moestedt, Fredrik Larsson, Fredrik Lundholm, Jim Nilsson, Bengt Werner, Per Stenström. SimICS/sun4m: A virtual workstation. En *USENIX annual technical conference*. Junio 1998.

[Goldberg, 1991] David Goldberg. What every computer scientist should know about floating-point arithmetic. En *ACM computing surveys*. Páginas 5–48. Marzo 1991.

[Cisneros y McIntosh, 1986] Gerardo Cisneros y Harold V. McIntosh. Notas sobre los lenguajes REC y Convert. Departamento de Aplicación de Microcomputadoras. Instituto de Ciencias. Universidad Autónoma de Puebla. 1986