Un Ambiente de Programación para el Procesador DLX

Perna, Juan Ignacio; Grosso, Alejandro Leonardo; Berón, Mario Marcelo.
Departamento de Informática
Facultad de Ciencias Físico, Matemáticas y Naturales
Universidad Nacional de San Luis – Argentina
e-mail: {jiperna, agrosso, mberon}@unsl.edu.ar
Fax: 54-2652-430224

RESUMEN

El estudio de las arquitecturas de los procesadores es esencial en la ciencia de la computación. Conocer el funcionamiento del procesador permite construir programas que aprovechen eficientemente sus recursos. El procesador *DLX* (Deluxe) se ha tomado como base para el estudio de las arquitecturas porque es simple y reúne las características principales de los procesadores actuales.

En este trabajo se presenta una *herramienta multiplataforma* que provee un *Simulador* del Procesador DLX y un *ensamblador*, embebidos en un *ambiente de desarrollo gráfico*. El simulador permite mapear dispositivos, administrar el sistema de interrupciones y contiene una unidad de control multi-hebra que posibilita simulaciones concurrentes. El ensamblador reúne las características esenciales de este tipo de programas (administración de la tabla de símbolos, recuperación precisa de errores, etc.). La interfaz gráfica facilita el uso de la herramienta a través de la visualización de las distintas componentes del procesador, y la realización automática de operaciones de uso común (conversiones entre sistemas numéricos complementarios y signados, cálculos de desplazamientos relativos, etc.) durante el proceso de programación en lenguaje ensamblador.

La herramienta posee un diseño que hace simple la incorporación de diferentes versiones del procesador de estudio, permitiendo analizar el desempeño del mismo ante modificaciones arquitecturales.

Palabras Clave: Arquitectura del Procesador, DLX, Simulador, Programación Orientada a Objetos, Multiplataforma.

1. INTRODUCCIÓN

Los avances tecnológicos han facilitado la creación de procesadores con gran capacidad de procesamiento. Esto se basa principalmente en: el uso de nuevas estrategias de diseño y circuitos cada vez más pequeños y eficientes. De esta forma, para llevar a cabo un estudio profundo de las arquitecturas de los procesadores actuales [14], es necesario entender concienzudamente las bases de su construcción (diseño y componentes).

En esta línea de trabajo, se han construido numerosas herramientas que simulan el comportamiento de un procesador específico. Esta tarea, en general, ha sido motivada por: i) la imposibilidad económica de obtener versiones reales del procesador, ii) la existencia de diseños conceptuales que necesitan ser evaluados y, iii) su aplicación en el proceso de enseñaza-aprendizaje.

Un análisis del estado del arte en este ámbito revela que los simuladores [5] [6] [7] [8] actuales presentan diseños que: dificultan el cambio de la versión actual del procesador por otra con

modificaciones en su arquitectura, proporcionan pocas utilidades para el estudio del procesador en si mismo y de su desempeño y no son portables.

Teniendo en cuenta esta realidad, y tomando como objeto de estudio al procesador DLX, por su amplia utilización en la aplicación de técnicas de evaluación del desempeño de procesadores y enseñanza, y por constar con un diseño conceptual que sigue la tendencia de los procesadores actuales, se construyó una herramienta que simula su funcionamiento. La herramienta fue diseñada de forma tal que pueda funcionar con distintas versiones de DLX. Esto tiene la ventaja de que nuevas propuestas con cambios en la arquitectura de este procesador, puedan ser incorporadas a la herramienta con facilidad. Esta característica es de gran utilidad tanto en la enseñanza-aprendizaje como en el análisis del desempeño de diferentes versiones del DLX. Además el simulador posee una interfaz gráfica cuya finalidad es facilitar el uso de la herramienta y promover el estudio del procesador. Con este objetivo se proveen diferentes utilidades que han sido seleccionadas a partir de la experiencia en el estudio de procesadores, entre estas se encuentran: facilidades para visualizar las distintas componentes del procesador, conversiones numéricas, visualización de la memoria, cambio de los valores de los registros, ejecución paso a paso y total, puntos de corte, etc. Es importante notar que las componentes de software que conforman a la herramienta simulan fielmente cada una de las partes del procesador y emplean técnicas y estructuras de datos eficientes.

Por último es fundamental destacar que esta herramienta fue construida para ser utilizada en: la evaluación nuevas versiones del DLX que son sintetizadas en FPGAs, y en las materias Arquitectura del Procesador I y II de la carrera licenciatura en computación de la Universidad Nacional de San Luis.

El artículo esta organizado como sigue. En las secciones 2 y 3 se describen las características y requerimientos del sistema junto con el modelo subyacente. En la sección 4 se presenta una vista detallada de las componentes que conforman a la herramienta (la interfaz gráfica, el simulador, ensamblador y cargador). Finalmente se muestran las conclusiones obtenidas.

2. REQUERIMIENTOS DEL SISTEMA

Entre los requerimientos principales del sistema se encuentran: simplicidad de incorporación de nuevas versiones del DLX con el fin de evaluar el desempeño; facilidad de acceso y modificación de cada una de las componentes del procesador para simplificar su entendimiento y fomentar su estudio; y posibilidad de poder ejecutar la herramienta en diferentes plataformas.

Para satisfacer el primer requerimiento la herramienta está provista una jerarquía de clases, con interfaces bien definidas, que hace posible la incorporación de una nueva arquitectura con solo con cambiar la clase que implementa a la máquina DLX.

Para cumplir con el segundo requerimiento, la herramienta permite, entre otras funcionalidades:

- Visualizar e interpretar el contenido de los registros del procesador y cualquier dirección de memoria en distintos sistemas numéricos.
- Modificar el contenido de direcciones de memoria y/o registros.
- Visualizar y modificar en todo momento el modo de operación del simulador (modo supervisor o usuario) así como de otras características del procesador (habilitación de interrupciones, mapeo de dispositivos, etc.).
- Utilizar diferentes modos de ejecución (paso a paso, parcial y total).
- Administrar las líneas de interrupción del procesador.
- Administrar manualmente y por lotes los dispositivos del sistema.
- Estipular el momento en el que los mismos producen interrupciones y obtienen/procesan los datos requeridos.
- Escoger entre dispositivos de entrada/salida preestipulados o de personalizar dispositivos genéricos y adaptarlos a requerimientos particulares.

- Realizar el proceso de ensamble y carga del código de máquina generado fácilmente.

Finalmente, la aplicación fue construida con software portable (librerías Qt y C++), siendo necesaria la compilación de la misma para que pueda ejecutar en distintas plataformas.

3. MODELO DEL SISTEMA: ARQUITECTURA DE DOS CAPAS

De los requerimientos planteados para el sistema, es posible concluir que la aplicación es en realidad, la conjunción de dos funcionalidades claramente diferentes (la *traducción del lenguaje ensamblador a código máquina* y la *simulación*) unificadas por una interfaz gráfica que brinda acceso a dichas características.

Aunque estos tres componentes del sistema son inherentemente independientes, es importante mantenerlos unificados dentro de una misma aplicación por lo que se opta por una implementación en modelo arquitectural de dos capas. Según esta división, la capa inferior provee las funcionalidades tanto de simulación de la máquina DLX como de traducción de programas en lenguaje ensamblador (con el cargador de programas como interfaz entre ellos). La capa superior, administra la interfaz gráfica y transforma los eventos generados por el usuario en invocaciones a métodos de la capa inferior. Este patrón de diseño de la aplicación (ver figura 1), posibilita un desarrollo modular así como la incorporación de nuevas funcionalidades que sean requeridas en un futuro.

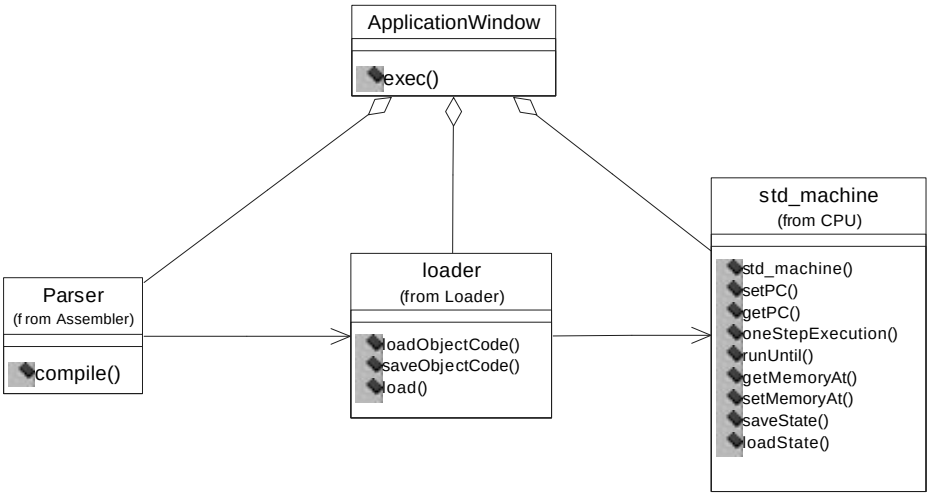

Figura 1: Vista general de la arquitectura del sistema

Para posibilitar esta división lógica de la aplicación, fue necesario definir una interfaz clara, precisa y simple entre las partes, de manera tal de que la comunicación entre ellas fuese no sólo eficiente, sino además, intuitiva para su uso por parte del desarrollador (característica necesaria para posibilitar futuras modificaciones).

4. COMPONENTES DEL SISTEMA

En esta sección se describen, las componentes principales del simulador dando una visión detallada de su funcionamiento y estructura interna.

4.1 La interfaz gráfica

Tomando en consideración los requerimientos de amigabilidad y portabilidad se seleccionaron las librerías Qt dada la necesidad de contar con interfaces gráficas visualmente atractivas que provean funcionalidades multiplataforma. La opción por estas librerías se basó fundamentalmente en que están orientadas a objetos (es una característica necesaria dado que el resto del sistema también fue implementado en este paradigma), que proveen funcionalidades de hebras (threads) y

que son fácilmente integrables a los códigos fuentes de la aplicación (basta con incluir los encabezados (headers) de las librerías para poder desarrollar el entorno visual).

En lo que respecta a la implementación de la interfaz gráfica de la aplicación, consta de una ventana principal (ver clase applicationWindow en figura 2) que brinda, principalmente, las funcionalidades para la creación y edición de códigos en lenguaje ensamblador. Por otra parte, también se encuentran disponibles las opciones de configuración del simulador y se provee de cuadros de diálogo para la compilación; ejecución; visualización/edición de la memoria y los registros; y mapeo/configuración de dispositivos.

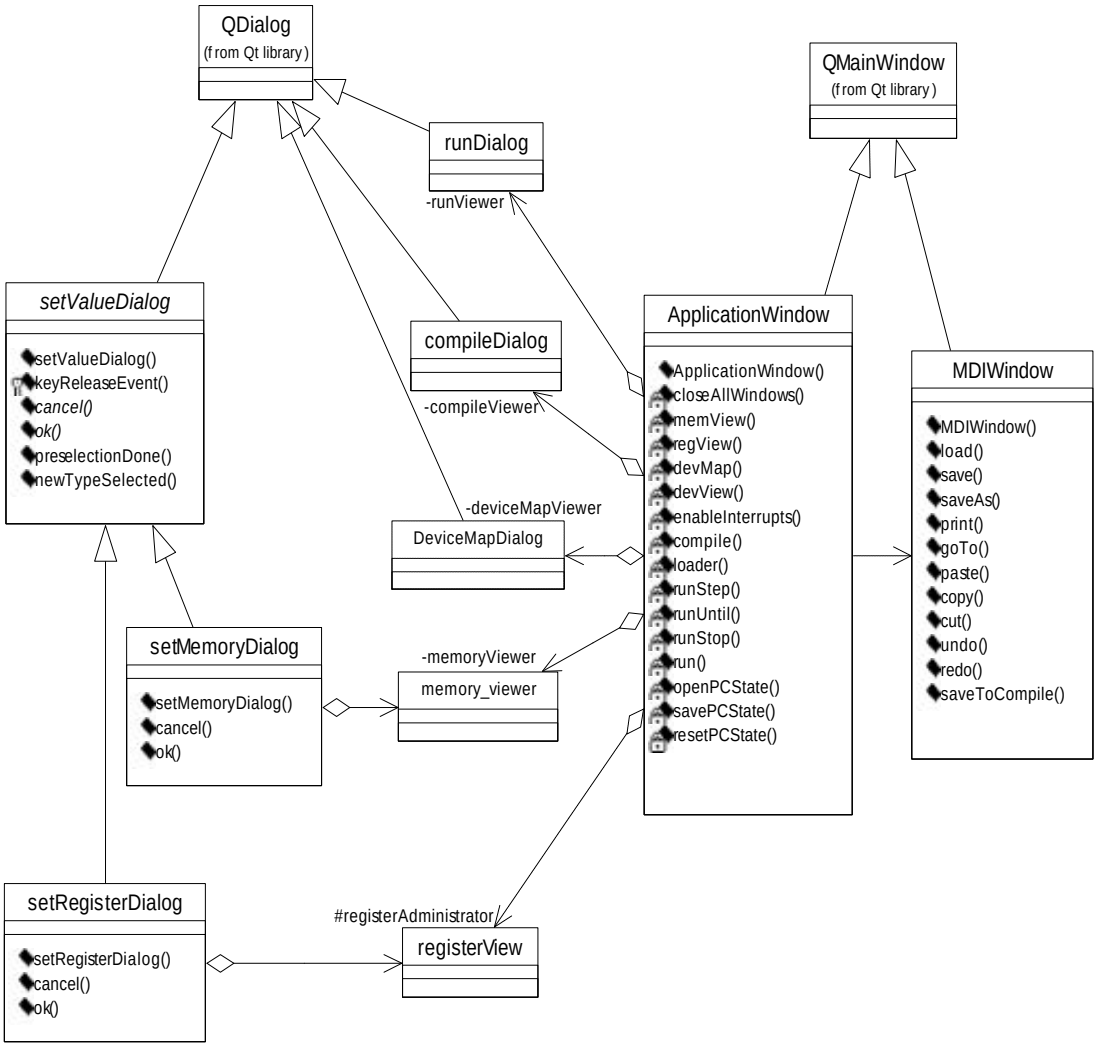

Figura 2: Modelo de implementación de la interfaz gráfica

4.2 El simulador

En lo que respecta al simulador de la máquina DLX en particular, se requiere que presente todas las características que son descriptas por Patterson y Hennesy en “Computer Architecture, a quantitative approach” [4]. Tomando esto como base y considerando la necesidad de brindar un diseño que posibilite la experimentación con componentes alternativos, se decidió que la arquitectura del sistema debería asemejarse lo más posible al modelo teórico. De ésta forma, se logra que cada módulo del simulador represente a una unidad funcional del procesador real en su totalidad (figura 3), facilitando el cambio de componentes.

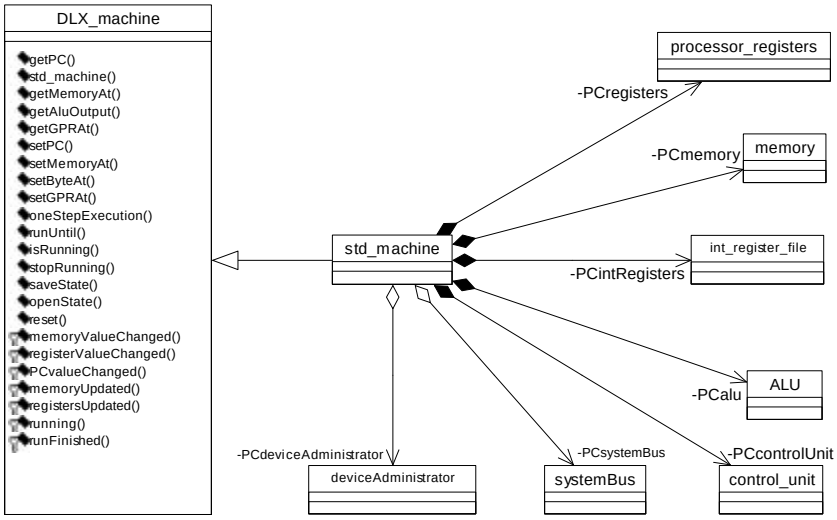

Figura 3: Vista arquitectural del simulador

4.2.1 El Sistema de Ejecución

Para la implementación del sistema de ejecución del procesador, se optó por crear una clase por cada instrucción del repertorio de instrucciones del mismo (ver figura 4).

Si bien se presenta la desventaja de la creación y destrucción de una instancia para cada instrucción dentro del programa en ejecución, ésta implementación muestra ser la más conveniente si se toman en consideración aspectos tales como:

- Mayor modularidad: todas las etapas del proceso de ejecución son conocidas y llevadas a cabo por cada instrucción. Esto hace explícita la semántica de cada instrucción y facilita la modificación/incorporación de instrucciones.

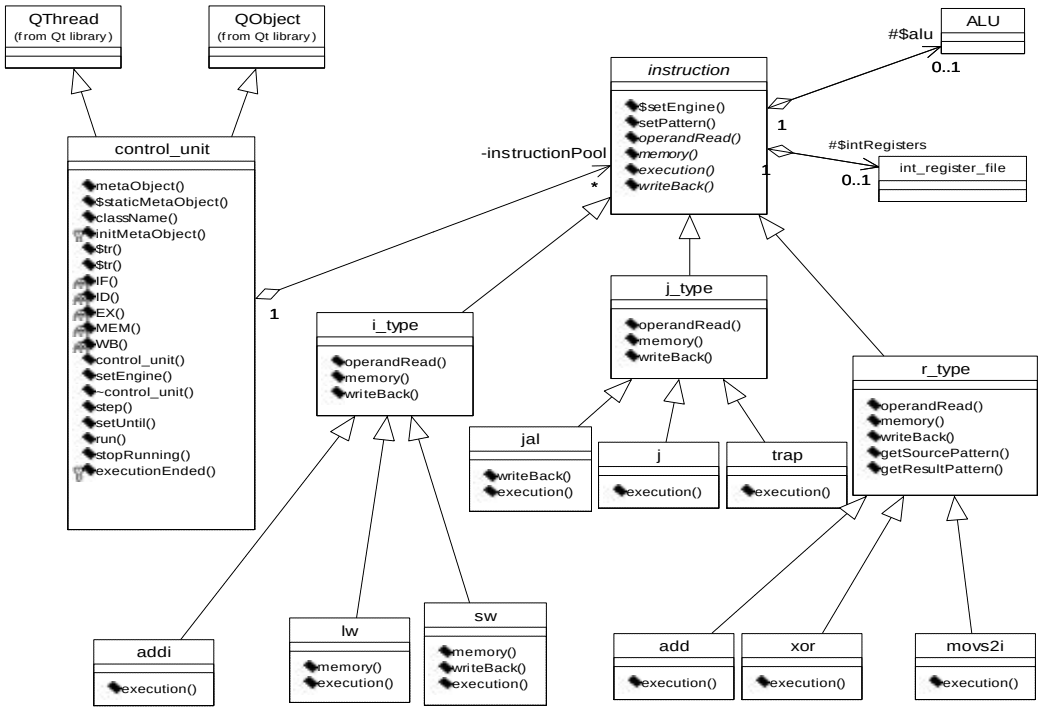

Figura 4: Modelo del subsistema de ejecución del simulador DLX

- Simplificación de la lógica de la unidad de control del simulador: el control de ejecución esta contenido en las instrucciones, por lo que sólo es necesario enviar los mensajes correspondientes a cada etapa de ejecución a la instrucción corriente. En particular, el proceso de decodificación (en el cual se selecciona la instrucción que se va a ejecutar) se realiza mediante el envío de un mensaje con el patrón de bits a todas las instrucciones (que se encuentran inactivas en un pool dentro de la unidad de control). Todas las instrucciones reciben el mensaje y sólo aquella cuyo formato coincide con el patrón enviado, es activada y seleccionada para continuar con el proceso de ejecución.
- Soporte para la incorporación de técnicas para optimización del tiempo de ejecución: enfoques con pipeline podrían ser implementados con facilidad en el simulador, ya que se basan principalmente, en la partición del ciclo de ejecución en etapas y en el solapamiento/detención de las mismas. En base a esta observación, sería suficiente con modificar la unidad de control para que despache una instrucción por ciclo de reloj (en vez de una cada cuatro/cinco ciclos), detecte los posibles conflictos que pudiesen ocurrir entre instrucciones consecutivas y el procesador opere con pipeline.

4.2.2 Soporte de Ejecuciones Concurrentes

El proceso de ejecución es independiente del resto de la aplicación. La motivación para esta decisión de diseño radica en la posibilidad de que se intente simular la ejecución de un número infinito de instrucciones (posiblemente, por una iteración que nunca termine). Ante esta situación, el sistema completo quedaría bloqueado y sería imposible para el usuario recuperar el control.

La forma en que se logró la ejecución concurrente del proceso de simulación del procesador DLX, junto con la administración de la interfaz gráfica y demás componentes de la aplicación, fue mediante la incorporación de hebras. En particular, los métodos que llevan a cabo la ejecución dentro de la unidad de control son los que fueron seleccionados para iniciar nuevas hebras según las demandas del usuario (notar que la clase control_unit hereda de la clase QThread de las librerías Qt en el esquema mostrado en la figura 4).

4.2.3 Entrada Salida e Interrupciones

El simulador consta con la capacidad de manipular dispositivos de entrada/salida así como de administrarlos mediante un sistema de interrupciones. Es necesario tener en cuenta que éstas características no fueron detalladas en la obra de Patterson y Hennesy y en los programas de simulación actuales su presencia es obscura.

Dada la falta de especificaciones y de precedentes en el tema [9] [10] [11], se optó entonces por utilizar un esquema de cuatro líneas de interrupciones no priorizadas con una única dirección fija para la rutina de atención de las mismas. Esta decisión se fundamenta en la sencillez del esquema y en la flexibilidad que provee a los usuarios para implementar (por software) los distintos esquemas de prioridades y de atención de interrupciones [17].

Por otra parte, es necesaria la administración de excepciones para poder solucionar y/o manejar situaciones anómalas producidas durante la ejecución. Es importante destacar que éste aspecto si es descripto por el modelo teórico y en el se contemplan cuatro tipos de excepciones: IAV (violación de dirección de instrucción), DAV (violación de dirección de datos), ICO (código de operación ilegal) y OVR (desbordamiento matemático) [3].

En lo que respecta a la implementación, el simulador consta de dos administradores: uno para las excepciones y otro para las interrupciones, tal como puede observarse en la figura 5.

El manejador de excepciones provee la interfaz entre las excepciones y el resto de la máquina y contiene una instancia de alguno de los tipos de excepciones (si es que ha ocurrido alguna). Tanto la modificación de la información de estado del procesador (palabra de estado, dirección de la instrucción que produjo el error, etc.) como la acción a tomar en cada caso, es determinada por la

clase que modela ese tipo de excepción (cuya instancia se encuentra en el exceptionHandler). Este modelo posibilita la incorporación de nuevas excepciones y la modificación de las existentes con gran facilidad, ya que sólo es necesario definir la clase que modela la nueva situación anómala, o extender el funcionamiento de alguna ya existente para que la nueva funcionalidad quede incorporada al sistema.

De manera similar, el sistema de interrupciones es administrado mediante cuatro instancias de la clase interruptionLine que son administradas por el manejador de interrupciones. Cada línea de interrupciones mantiene una lista de los dispositivos que se encuentran conectados a ella, y es la encargada de emitir las señales de actualización de estado en caso de recibir la señal de interrupción por parte de alguno de los dispositivos.

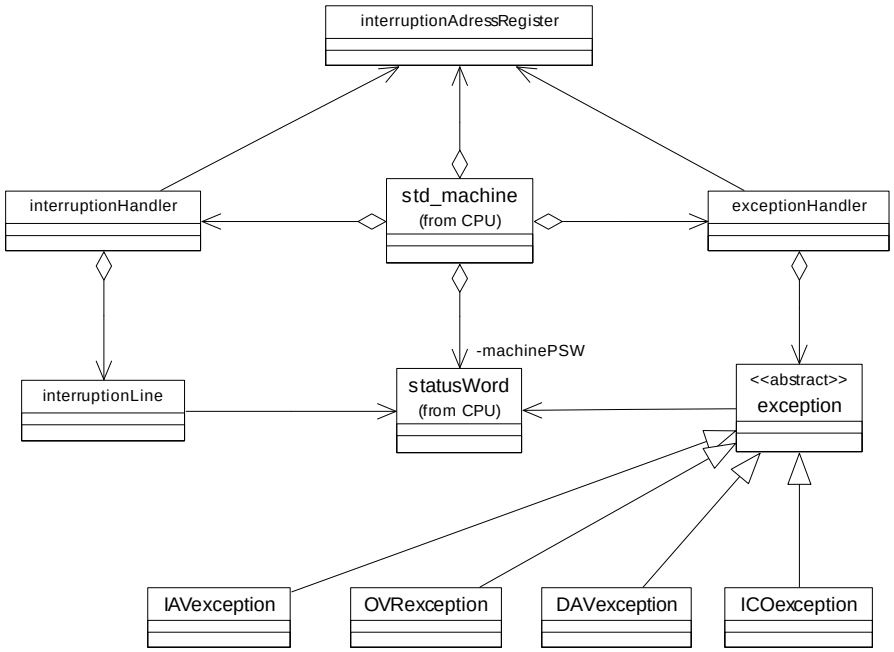

Figura 5: Vista de los sistemas de administración de interrupciones y excepciones

4.2.4 La Memoria

Del modelo presentado en la figura 3, sobresale una componente del sistema, que es parte del modelo teórico y requiere técnicas y estructuras de datos eficientes para su manipulación: una memoria principal de 4 Gb.

La principal dificultad a la hora de dotar al simulador con está capacidad de memoria radica en el excesivo tamaño de la información a almacenar, esto hace que sea imposible satisfacer esta demanda de capacidad con una estrategia que almacene todos los bytes de la memoria principal del procesador.

La solución por la que se optó consiste en brindar al usuario la ilusión de una memoria completa pero que en realidad almacene el contenido de aquellas celdas que contengan información útil (es decir, que hayan sido escritas alguna vez), y generar datos “basura” para el resto de las posiciones de memoria (dado que el contenido de las mismas es indeterminado en lo que respecta a una simulación en particular). Esta idea no elimina por completo la posibilidad de generar una estructura de gran tamaño (este caso se podría producir si el usuario escribiese sistemáticamente todas las celdas de memoria) pero acota el tamaño de la porción de memoria almacenada a los

pedidos del usuario. De esta forma, simulaciones que impliquen una utilización baja o media de la memoria conllevarán representaciones de la misma de un tamaño manejable.

La implementación de la solución planteada anteriormente revela que las estructuras necesarias para mantener la información referente al uso de cada celda de memoria, necesitan un tamaño que pertenece al mismo orden que el espacio necesario para almacenar la memoria del simulador directamente. Se optó entonces por registrar el uso no de celdas individuales sino de grupos de celdas, lo que naturalmente arribó a una administración al estilo de un sistema de paginado [1] [15]. Por otra parte, la paginación posibilita la transferencia de las estructuras que almacenan el contenido de la memoria del DLX a un archivo de intercambio en memoria secundaria (característica beneficiosa para los casos donde el tamaño de las mismas comienza a ser considerable).

El registro de utilización fue implementado mediante un vector de bits [12]. Con este dispositivo, una lectura a memoria se transforma primero en una consulta al vector de páginas y, si la página fue escrita anteriormente, ésta es cargada en memoria y se resuelve el pedido como se haría normalmente en un sistema con paginado. Por otra parte, si la lectura hace referencia a una posición de memoria que no fue escrita, se retorna un dato basura producido por un generador de datos pseudo aleatorios.

Para la indexación del archivo de intercambio se optó por la utilización de árboles digitales de búsqueda. Estos difieren principalmente respecto de los árboles tradicionales en que el recorrido de los nodos del árbol es guiado no de acuerdo a la comparación de la clave completa de búsqueda, sino que a partir de subselecciones dentro del patrón de bits que la conforma. Esta estructura es particularmente eficiente si se cuenta con claves únicas y bajo estas condiciones es posible demostrar [13] que poseen un costo promedio de búsqueda es de orden logarítmico y un peor caso de orden lineal, ambos respecto del número de bits en la clave.

En el caso particular de los bloques de memoria del simulador, es natural tomar como clave de búsqueda la dirección base de la página (que puede ser obtenida fácilmente mediante operaciones de enmascaramiento aplicadas a la dirección de la palabra buscada) y trabajar seleccionando bits individuales dentro de la misma.

Por otro lado, si se toma en consideración la localidad de referencia espacial [16], se deduce que existe una alta probabilidad de realizar operaciones sobre páginas que se encuentran consecutivas en memoria. Es por esto que es deseable que direcciones contiguas sean diferenciables rápidamente por el mecanismo de búsqueda (y, por consiguiente, por los mecanismos de inserción y consulta que lo invocan). Basados en la última observación, en la implementación los bits de la clave se seleccionaron de derecha a izquierda (el inverso a la forma en que se leen generalmente), característica que permite que patrones consecutivos sean evaluados como diferentes después de efectuar unas pocas consultas.

La implementación de los componentes de la memoria, así como los mecanismos para la indexación y optimización de búsquedas en el archivo de intercambio se muestran esquematizados en la figura 6.

4.3 El ensamblador

El ensamblador disponible en la herramienta es de dos pasadas [2]. En la primera se realiza el análisis del código fuente se detectan y relevan todos los símbolos y/o etiquetas con su respectivas direcciones absolutas. Mientras que en la segunda pasada, se efectúa el chequeo sintáctico y se utiliza la información obtenida en la primera pasada para la generación del código de máquina.

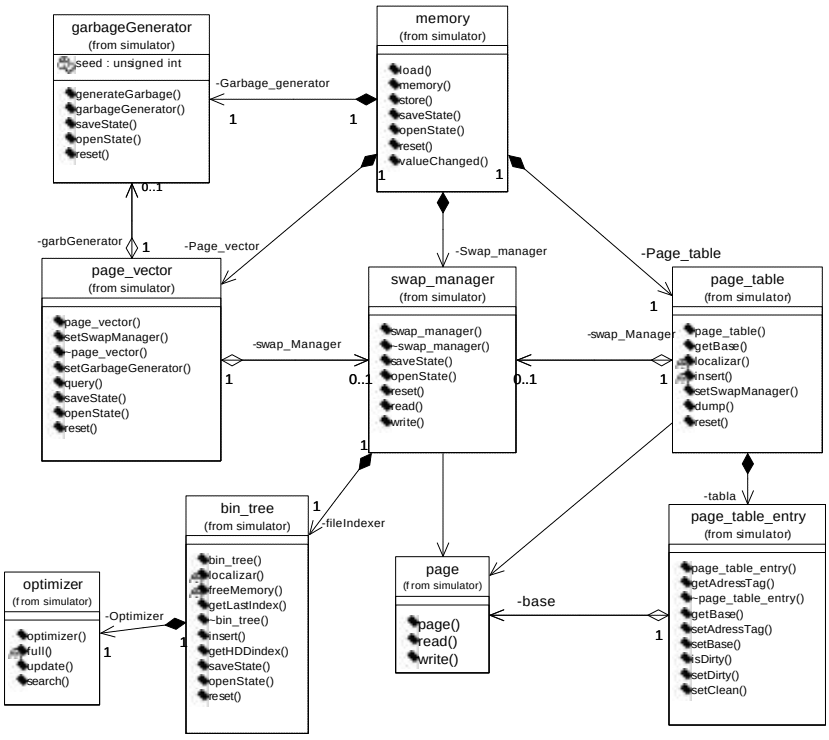

Figura 6: Modelo de implementación para el sistema de memoria

4.3.1 El Analizador Sintáctico

En lo que respecta a la implementación del analizador sintáctico en particular, se parte del modelo descendente recursivo, con la salvedad de que, al estar implementado en el paradigma de la programación orientada a objetos, los no terminales de la gramática (LL(1)) [17] son representados mediante clases (a diferencia de los planteos convencionales en los que cada no terminal es expandido por un llamado a subrutina) tal como se muestra en la figura 7.

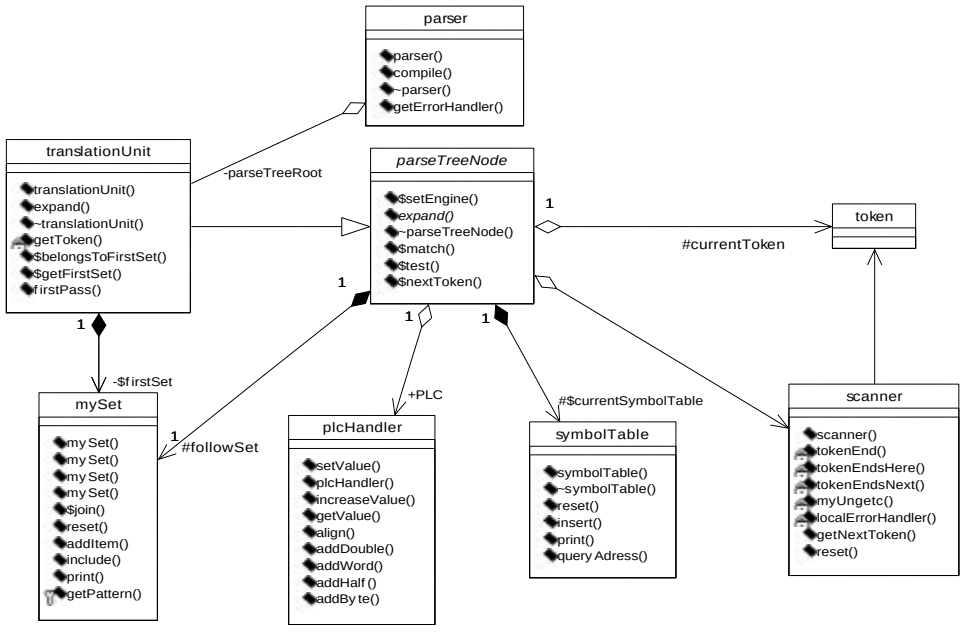

Figura 7: Vista de la implementación del ensamblador

Nuevamente se presenta el argumento de que esta implementación genera (y destruye) una significativa cantidad de objetos durante el proceso de análisis del código assembler. Sin embargo, se vuelve a optar por este modelo porque se ajusta más al de la programación orientada a objetos, es semánticamente más expresivo (ya que cada no terminal, instancia de una clase del analizador sintáctico, realiza el análisis sintáctico de la forma sentencial que se genera a partir de su expansión) y porque permite (en caso de necesidad) la incorporación de nuevas construcciones al analizador sintáctico sin mayores complicaciones (es una característica particularmente deseable dada las grandes posibilidades de experimentación que surgen a partir de las posibles extensiones realizables a la aplicación).

4.4 El Cargador

El código objeto es generado por el analizador sintáctico durante el tiempo de análisis del código fuente y almacenado en bloques del cargador. Estos últimos son de un tamaño fijo (256 bytes) y esto conlleva el problema de la especificación de códigos y/o datos que abarquen múltiples bloques dispersos a lo largo de todo el espacio de direcciones. Es por esto que se implementó el cargador como un árbol ya que es una estructura que optimiza el tiempo de búsqueda de bloques individuales, y que puede expandirse según las necesidades de cada caso en particular (ver figura 8).

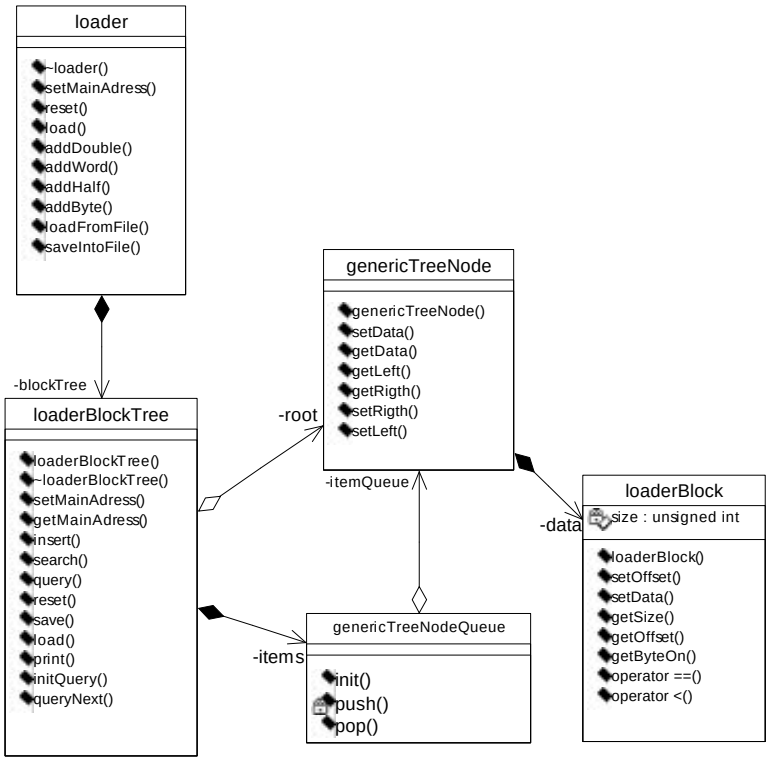

Figura 8: Vista de la arquitectura del cargador

5. CONCLUSIONES Y VISION DE FUTURO

En este trabajo se presentó un ambiente de programación para el procesador DLX. El ambiente consta de un: simulador, ensamblador, cargador, e interfaz gráfica. La finalidad de la herramienta es educativa y con proyecciones en aplicaciones de investigación referidas al análisis de desempeño de posibles modificaciones en la arquitectura del procesador DLX.

Es necesario destacar tres aspectos relevantes que surgieron durante el proceso de diseño e implementación de la herramienta: i) Si bien los fundamentos de funcionamiento del procesador son conocidos, la construcción de un simulador del mismo, requiere de una ardua y minuciosa tarea de programación que permite integrar conceptos variados dentro del área de las ciencias de la computación, ii) el desarrollo de la herramienta permitió detectar componentes del procesador carentes de especificación o subespecificadas que debieron ser diseñadas por el equipo de desarrollo, iii) la necesidad de proveer una documentación adecuada y un diseño fácilmente modificable condujo a la utilización de técnicas y métodos ingenieriles que mostraron ser de gran utilidad como soporte en el proceso de desarrollo.

Se piensa que en el futuro el simulador pueda constar con comandos que permitan: contabilizar la cantidad de instrucciones, calcular los ciclos por instrucción promedio, calcular el tiempo de ejecución para un programa, etc., con el fin de poder comparar la aceleración ganada, en la ejecución de programas sobre diferentes propuestas de DLX, como por ejemplo aquellas que incorporan pipeline y/o los algoritmos de Tomasulo y Scoreboarding.

6. BIBLIOGRAFÍA Y REFERENCIAS

[1] Deitel, H. *"Operating Systems"*. Segunda Edición. Ed. Addison Wesley. 1990.
[2] Gear, William C. *"Computer Organization and Programing"*. Ed. Mc. Graw Hill.
[3] Hayes J. "*Computer Architecture and Organization*". Ed. Mc. Graw Hill. 1988
[4] Hennessy, John L.; Patterson, David. *"Computer Architecture: A Quantitative Approach"*. Segunda Edición. Ed. Morgan and Kaufmann. 1990.
[5] http://heather.cs.ucdavis.edu/~matloff/DLX/Report.html
[6] http://www.ashenden.com.au/designers-guide/DG-DLX-material.html
[7] http://www.cse.ucsc.edu/~elm/Software/Dlxos/
[8] http://www.csee.umbc.edu/courses/undergraduate/411/spring96/dlx.html
[9] http://www.freevbcode.com/ShowCode.Asp?ID=4777
[10] http://www.iro.umontreal.ca/~bergeret/EBEL-DLX/ebel-dlx.txt
[11] http://www.ndsu.nodak.edu/instruct/tareski/ee774f96/notes/windlx/wdlxtut.htm
[12] Knuth, Donald E. *"The Art of Computer Programming. Volume 3: Sorting and searching"*. Segunda edición. Ed. Adisson Wesley. 1998.
[13] Robert Sedgewick. *"Algorithms in C*". Tercera edición. Ed. Addison Wesley. 1998.
[14] Stallings, William. *"Computer Organization and Architecture"*. Cuarta edición. Prentice Hall. 1996.
[15] Tanenbaum, Andrew S. *"Modern Operating Systems"*. Ed. Prentice Hall.1992.
[16] Ullman J., Sethi R., Aho A. "Compilers: Principles, Techniques and Tools". Ed. Addison Wesley. 1986.
[17] Wakerley, John. F. "Microcomputer Architecture and Programming". Ed. John Wiley and Sons. 1981.