Universidad de Los Andes
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas
Departamento de Computación
C.E.M.I.S.I.D.

Desarrollo de Un Simulador del Microprocesador RISC DMN-17N78 Programado en Lenguaje Java.

Autor: Alberto Enrique Dubuc Briceño.

Tutor: Dr. Gerard Páez Monzón

Mérida, Noviembre de 1996.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Resumen

El siguiente trabajo presenta un Simulador de la Arquitectura del Microprocesador RISC DMN 17N78, Microprocesador Académico, el cual es usado como caso de estudio en la Cátedra de Organización de Computadoras de la Escuela de Ingeniería de Sistemas. Dicho simulador fue implementado en el Lenguaje de Programación Java, y se usaron técnicas de modelado Orientado a Objetos para el diseño del sistema. Se han logrado dos importantes objetivos con este proyecto, el primero es que dicho simulador pudiese ser ejecutado en cualquier plataforma de Hardware y Sistema Operativo, sin necesidad de recodificar o recompilar los códigos fuentes, y como segundo objetivo se logró un simulador que ayudará a los alumnos de la Cátedra de Organización de Computadoras a entender la Organización y Arquitectura de un Microprocesador RISC.

- Microprocesadores - Simulación.
- JAVA Lenguaje de Programación.
- Simulación de Computadoras.
- Arquitectura del Computador.

Marcas Registradas

Java y las marcas basadas en Java son Marcas Registradas de Sun Microsystems Inc. en los Estados Unidos y en otros Países. La Universidad de Los Andes, o cualquiera de sus dependencias no tiene nada que ver con Sun Microsystems Inc.

Sparc y Ultra 1 son Marcas Registradas de Sun Microsystems Inc.

Netscape es Marca Registrada de Netscape Communications Inc.

IBM OS/2 Warp es Marca Registrada de International Bussines Machines Corp.

Windows 95 es Marca Registrada de Microsoft Corp.

Indy es Marca Registrada de Silicon Graphics Inc.

PowerMac es Marca Registrada de Apple Inc.

Introducción

Antecedentes

El procesador académico RISC[2][6] DMN 17N78 [11] fue desarrollado en el CEMISID usando como base la arquitectura Damian 17N78 descrita en [10]. Dicho diseño fue refinado durante diversos cursos de pre y posgrado en asignaturas de Arquitectura y Organización de Computadores, hasta llegar a la versión que representa este simulador denominada DMN 17N78 V7. Ya para la versión 6 de este procesador se estaba proyectando el diseño de un simulador de esta arquitectura, simulador este que servirá como base para comprobar la arquitectura del procesador antes de implementarlo en FPGA u otro proceso de fabricación de hardware, además se servir de apoyo al estudiante de las cátedras de Arquitectura Organización de Computadores en el estudio de esta Arquitectura en particular.

Objetivos del Proyecto

Como objetivos centrales, este proyecto persigue lo siguiente :

- Implementar un Simulador del Procesador RISC DMN 17N78 V7 que cumpla con las siguientes carácterísticas :

    1. Ambiente amigable para el usuario ;
    2. Altamente portable entre distintas plataformas ;
    3. Servir como apoyo docente en la compresión de la arquitectura DMN;
    4. Poseer un ensamblador para la arquitectura DMN

- Usar técnicas de diseño y programación Orientados a Objetos, para el diseño y la implementación del simulador;
- Usar el lenguaje de programación Java, el cual fue escogido gracias a que es :
    - Orientado a Objetos ;
    - Altamente Portable ;
    - Posee una extensa librería para implementar de una manera relativamente sencilla una GUI[1] para cualquier aplicación.

[1] GUI : Graphic User Interface : Interface Gráfica de Usuario.

Etapas del proyecto

Al tener los objetivos ya previstos se procedió a organizar las etapas de evolución del proyecto, las cuales fueron principalmente :

1. Estudio de la Arquitectura RISC DMN 17N78 [2][6][10][11];
2. Estudio del Lenguaje Java [4][5][9];
3. Diseño Orientado a Objetos del Sistema [3][8][12][13];
4. Diseño de la Interfaz Gráfica de Usuario del Simulador;
5. Diseño del Ensamblador DMN junto con su Interfaz Gráfica de Usuario[1];
6. Programación e integración de los distintos sistemas componentes ;
7. Depuración y Pruebas del Simulador ;
8. Publicación del Simulador en el WWW ;

Algunas de estas actividades fueron hechas en paralelo.

Como resultado final se obtuvo un simulador de la arquitectura RISC DMN 17N78, que es muy fácil de usar, altamente portable, diseñado y programado usando las técnicas del Modelo de Objeto dadas por Booch[3] y con ensamblador incorporado de la arquitectura DMN.

Capítulo I

Descripción de la Arquitectura DMN 17N78

Introducción

El procesador académico de arquitectura RISC, denominado DMN 17N78, ejecuta cada instrucción en el camino de datos. Concentra todas las instrucciones de movimiento, salto y Aritmético/Lógicas en su unidad ALL-U, o unidad Aritmético/Lógica/Control. La idea es normalizar la generación de señales de control en una sola unidad funcional. Esto se obtiene implementando un número colas de registros de diferente profundidad alrededor del camino de datos. Dicha arquitectura reduce aún mas la complejidad de la ejecución de un programa encauzado. Las características principales son: Arquitectura de Carga/Almacenamiento, encauzamiento de 4 etapas, aritmética entera de 8 bits en complemento a dos, 16 registros de 8 bits cada uno, memoria principal tipo Hardvard [2], es decir, separada la memoria de instrucción de la de datos, y 14 instrucciones en su repertorio.

Este procesador está orientado a ser un procesador genérico para juguetes basados en computadora. Incluiría interfaces especializadas para detectar la interacción con procesos a tiempo real.

Arquitectura DMN 17N78

La siguiente figura muestra la arquitectura del procesador :

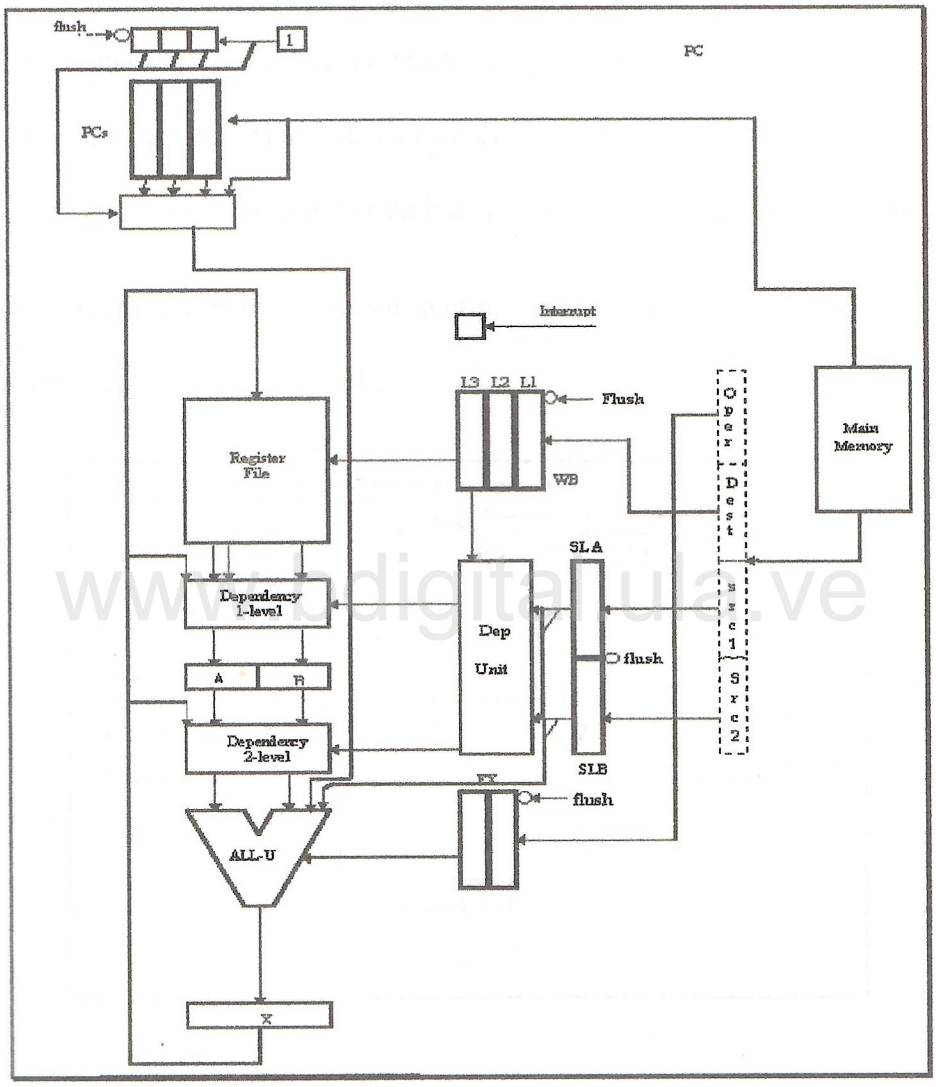

Figura 1-1 : Diagrama de la Arquitectura del DMN 17N78

Se puede notar en la figura 1-1 el archivo de registros, el cual contiene 12 registros de 8 bits, donde el R0 siempre es 0. Algunos de estos registros guardan información importante acerca del estado de la maquina. El conjunto de instrucciones ofrece 14 de ellas con una organización de 16 bits necesarias para controlar los distintos estados del procesamiento encauzado. El área de silicio del procesador integra una arquitectura de memoria principal del tipo Harvard, con un ciclo de acceso. Todas las referencias a la memoria principal son hechas a través de instrucciones de carga/almacenamiento.

En la figura 2 se puede observar la arquitectura de la ALLU, o unidad Aritmético Lógica, Memoria de Datos y Control :

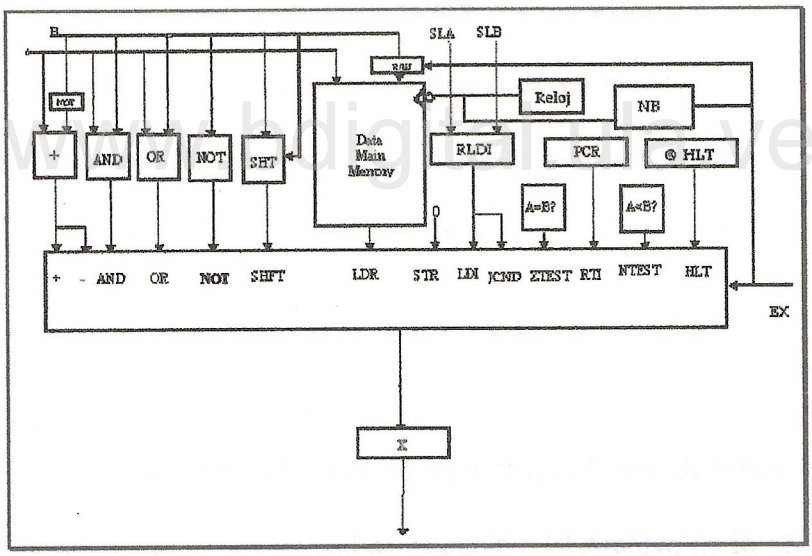

Figura 1-2 : Diagrama de la Unidad Aritmético/Lógica Integrada

Conjunto de Instrucciones

Se dividen en tres tipos de operaciones : de UAL[2], de Control y de referencia a Memoria. Cualquier registro puede ser escrito a excepción del R0 y R15, este ultimo cumple la función de Contador de Programa o PC. A su vez R0 mantiene su valor en 0. La carga de un registro puede ser hecha en dos modos, direccionamiento inmediato o direccionamiento por registro. El almacenamiento solo se puede hacer mediante el modo de direccionamiento por registro. La operaciones de UAL son : Suma, Resta, Y Lógico, O Lógico, Complemento y desplazamiento de bits en ambos sentidos. El conjunto de instrucciones de control ofrece la comparación y la capacidad de repetición. Todos los saltos son condicionales. La condición es el estado de un registro llamado Registro de Condición o CN. CN es afectado solo por las instrucciónes de prueba, que permite la evaluación de la condición, y los saltos son ejecutados por una instrucción separada de salto condicional.

El Camino de Datos

Está compuesto por un archivo de 16 registros, cada uno de 8 bits, un selector de operandos de dos entradas, y una unidad de ejecución. El Archivo de Registros incluye el contador de programas (PC) el Registro de Condición (CN), el Registro apuntador a la rutina de Interrupción (PS), y el registro de Paro de Ejecución (HA). Cada instrucción fluye a través de este camino de datos.

---

[2] UAL : Unidad Aritmético Lógica.

Las siguientes tablas resumen el conjunto de instrucciones del DMN 17N78 :

Instrucciones Aritmético/Lógicas		
Mnemonico	**Sintaxis**	**Semántica**
ADD	ADD Rd,Rs1,Rs2	Rd <= Rs1 + Rs2
SUB	SUB Rd,Rs1,Rs2	Rd <= Rs1 - Rs2
AND	AND Rd,Rs1,Rs2	Rd <= Rs1 ^ Rs2
OR	OR Rd,Rs1,Rs2	Rd <= Rs1 v Rs2
NOT	NOT Rd,Rs	Rd <= not(Rs)
SHT	SHT Rd,Rs1,Rs2	Rd <= Rs1 desp(Rs2)

Tabla 1-1 : Conjunto de Instrucciones Aritmético/Lógicas

Instrucciones de Control		
Mnemonico	**Sintaxis**	**Semántica**
HLT	HTL	HA <= FFh (detiene el procesamiento)
RTI	RTI	PC <= [PCR]
ZTS	ZTS Rc1,Rc2	CN <= (Rc1 == Rc2)
NTS	NST Rc1,Rc2	CN <= (Rc1 <= Rc2)
JCN	JCN @dirección	PC <= dirección ssi (CN <> 0)

Tabla 1-2 : Conjunto de Instrucciones de Control

Instrucciones de Carga/Almacenamiento de datos.		
Mnemonico	**Sintaxis**	**Semántica**
LDI	LDI Rd,#dato	Rd <= #dato
LDR	LDR Rd,Rs	Rd <= Mem[Rs]
STR	STR Rd,Rs	Mem[Rd] <= Rs

Tabla 1-3 : Conjunto de Instrucciones de Carga/Almacenamiento.

Formato de Instrucciones

La siguiente tabla muestra el formato binario en los 4 bits (Nibble) mas significativos (Código de Operación, u Opcode) de las Instrucciones del DMN 17N78

Binario	Hexadecimal	Mnemonico
0000	0	ADD
0001	1	SUB
0010	2	AND
0011	3	OR
0100	4	NOT
0101	5	SHT
0110	6	Reservado
0111	7	Reservado
1000	8	ZTS
1001	9	NTS
1010	A	JCN
1011	B	RTI
1100	C	HLT
1101	D	LDI
1110	E	LDR
1111	F	STR

Tabla 1-4 : Formato Binario del Código de Operación de las Instrucciones DMN.

Formato de Instrucciones UAL

Nibble 3		Nibble 2		Nibble 1		Nibble 0	
15	12	11	8	7	4	3	0
Opcode		R. Destino		R. Fuente 1		R. Fuente 2	

Formato de Instrucciones de Control

Instrucciones NTS y ZTS

Opcode	CN	R. Test 1	R. Test 2

Instrucción JCN

1010	PC	Dir. Dest. Nibble Alto	Dir. Dest. Nibble Bajo

Instrucción RTI

1011	PC	No importa	No importa

Instrucción HLT

1100	HA	No importa	No importa

Formato de Instrucciones de Carga Ejecución

Instrucción LDI

1101	R. Destino	Dato Nibble Alto	Dato Nibble Bajo

Instrucción LDR

1110	R. Destino	R. de Dirección a Mem. De Datos	No Importa

Instrucción STR

1111	R0	R. de Dirección a Mem. De Datos	R. Fuente

Ejecución de Instrucciones

El procesador DMN 17N78 ejecuta múltiples actividades en cada ciclo de reloj, dichas actividades están basadas en el siguiente encauzamiento de cuatro etapas, a saber :

Etapa 1	IF : (Instruction Fetch) Captura de Instrucción hacia el camino de datos.
Etapa 2	RS : (Register Select) Selección de Registros. RA <= RegS1 ; RB <= RegS2 ;
Etapa 3	EX : Ejecución de la Instrucción. X <= Resultado de Acuerdo al Opcode ;
Etapa 4	WB : (Write Back) Escritura del registro X en el archivo de registros. Rdest <= X ;

Tabla 1-5 :Etapas de Ejecución

Las Etapas en Detalle

IF : Captura de Instrucción, en este ciclo es traída la instrucción apuntada por el PC (registro 15) desde la memoria de instrucciones hasta el registro IR, el cual es un registro de 16 bits.

RS : Selección de Registros, en el siguiente ciclo de reloj la instrucción que se encuentra en el registro IR, es subdividida en sus 4 nibbles componentes. El mas significativo (nibble 3) es introducido en la cola de Ejecución (cola de registros EX en la figura 1-1). El segundo nibble mas significativo (nibble 2) se introduce en la cola de escritura (cola WB en figura 1-1), y por ultimo, los 2 nibbles menos significativos (nibble 1 y 0) son introducidos en los registros SLA y SLB, los cuales activan la lógica dentro de la unidad de dependencia de datos para seleccionar el valor de los registros correspondientes en el archivo de registros o, en caso de haber un problema de dependencia, el valor de X. Estos valores se depositarán en los registros RA y RB, los cuales servirán posteriormente para el ciclo de ejecución.

EX : Ejecución de la operación, en esta etapa se ejecutara la Operación de la instrucción. Aquí se resuelven las operaciones Aritmético/Lógicas, de control y referencia de memoria.

WB : Escritura del Registro X, en esta etapa el resultado de la etapa anterior queda en el registro X, por lo tanto dicho registro debe ser almacenado, y es almacenado de acuerdo a lo que indica el tope de la Cola WB.

Descripción Funcional de cada Instrucción

A continuación se detallaran los que hace cada instrucción en cada etapa dentro (o sobre) el camino de datos (la etapa IF se obviará, debido a que en todas las instrucciones se hace lo mismo en dicha etapa) :

Instrucciónes ADD, SUB, AND, OR

- **RS** : Se seleccionan los registros del archivo de registros, el contenido de cada registro se deposita en los registros A y B ;
- **EX** : Se opera sobre los registros A y B de acuerdo a la operación dada, y se deposita su resultado en el registro X ;
- **WB** : Se escribe el registro X en el archivo de registro que se encuentra direccionado por el Registro de Destino en el formato de la instrucción.

Instrucción NOT

- **RS** : Se selecciona el registro a operar y se deposita en el registro A ;
- **EX** : Se hace un complemento a uno del registro A, y su resultado se deposita en el registro X ;
- **WB** : Se escribe el registro X en el archivo de registro que se encuentra direccionado por el Registro de Destino en el formato de la instrucción.

Instrucción SHT

- **RS** : Se seleccionan los registros del archivo de registros, el contenido de cada registro se deposita en los registros A y B ;

- **EX** : Se hace desplazamiento lógico de los bits de B, de acuerdo al valor (en complemento a dos) de A, es decir, si A es positivo el desplazamiento es hacia la izquierda, pero si A es negativo este será hacia la derecha ;

- **WB** : Se escribe el registro X en el archivo de registro que se encuentra direccionado por el Registro de Destino en el formato de la instrucción.

Instrucciones ZTS y NTS

- **RS** : Se seleccionan los registros del archivo de registros, el contenido de cada registro se deposita en los registros A y B ;

- **EX** : Se efectúa la comparación de los registros A y B, de acuerdo al resultado de la comparación se le asigna al registro X un valor de FFh en caso de éxito o 00h en caso de fracaso ;

- **WB** : Se escribe el registro X en el archivo de registro, el cual debería estar direccionado hacia el registro CN.

Instrucción JCN

- **RS** : Aquí los registros A y B tomaran los valores dados por RSA y RSB, pero estos valores serán desechados posteriormente, ya que el valor que importa es el que forma el byte constituido por los nibbles RSA y RSB ;

- **EX** : en esta etapa se toman los valores de los nibbles RSA y RSB y con ellos se forma un Byte dentro de la ALL-U, el cual llamamos RLDI. Paralelamente se verifica el valor del registro CN, si CN es distinto de cero, entonces a X se le asigna el valor del registro RLDI.

- **WB** : Se escribe el registro X en el archivo de registro, el cual debería estar direccionado hacia el registro PC.

Instrucción RTI

- **RS** : Nuevamente aquí se seleccionaran los valores de los registros A y B, de acuerdo con RSA y RSB, pero en este caso ni esta selección, ni el contenido de RSA y RSB tiene importancia ;

- **EX** : Introduce en el registro X el valor del registro PCR ;

- **WB** : Se escribe el registro X en el archivo de registro, el cual debería estar direccionado hacia el registro PC.

Instrucción HLT

- **RS** : Igual que en la instrucción anterior ;

- **EX** : Coloca el valor de FF en el registro X ;

- **WB** : Se escribe el registro X en el archivo de registro, el cual debería estar direccionado hacia el registro HA.

Instrucción LDI

- **RS** : Igual que la etapa RS en la instrucción JCN ;
- **EX** : Aquí se le asigna al registro X el valor del registro RLDI, que como se dijo en la descripción de la instrucción JCN, este registro se toma de unir los valores de RSA y RSB en un solo byte ;
- **WB** : Se escribe el registro X en el archivo de registro que se encuentra direccionado por el Registro de Destino en el formato de la instrucción.

Instrucción LDR

- **RS** : Se selecciona el registro que direccionará la localidad en la memoria de datos y se deposita en el registro A ;
- **EX** : En el registro X se coloca el valor direccionado por el registro A
- **WB** : Se escribe el registro X en el archivo de registro que se encuentra direccionado por el Registro de Destino en el formato de la instrucción.

Instrucción STR

- **RS** : Igual que en la instrucción anterior, pero además en el registro B se deposita el Byte a introducir en la memoria de datos ;
- **EX** : En esta etapa se selecciona la localidad a escribir en la M. de D. y se escribe el valor que contiene el registro B. Nótese que esta etapa esta instrucción termina su ejecución.

Ejecución del Encauzamiento

El control de la Arquitectura encauzada está basado en un numero de colas de registros, estas colas de registros están representadas el la Figura 1-1 como WB, EX, SLA y SLB.

La diferencia en profundidad entre las colas refleja los estados de encauzamiento. Por ejemplo, la cola WB es de tres grados de profundidad, mientras que la cola EX presenta dos grados de profundidad.

La siguiente tabla muestra las etapas por las que pasa 4 instrucciones, y su diferencia en los ciclos de ejecución. Se puede observar que en el ciclo 2 la instrucción i está en la etapa de RS, mientras que en ese mismo ciclo la instrucción i+1 entra al camino de datos. En esta tabla se observa que el ciclo más crítico del procesador es el 4 donde en ese mismo ciclo está :

1) haciendo WB a la instrucción i ;

2) haciendo EX a la instrucción i+1 ;

3) haciendo RS en la instrucción i+2 ;

4) haciendo IF en la instrucción i+3

Ciclo	1	2	3	4	5	6	7
Inst. I	IF	RS	EX	WB			
Inst I + 1		IF	RS	EX	WB		
Inst I + 2			IF	RS	EX	WB	
Inst I + 3				IF	RS	EX	WB

Tabla 1-6 : Vista del Encauzamiento

Particularidades de la Arquitectura

La Arquitectura presenta cuatro particularidades, que en los siguientes apartados se describirá como fue que se resolvieron, dichas particularidades son :

1. Manejo de la Dependencia de Datos ;
2. Escritura en la Memoria de Datos ;
3. Bloqueo de escritura sobre el Registro PC ;
4. Manejo de Interrupciones.

Manejo de la Dependencia de Datos

Un problema de dependencia de datos puede ocurrir debido a que una operación no ha terminado de actualizar los valores de su registro destino cuando la siguiente operación está por usar este mismo registro como fuente en el siguiente ciclo de reloj, cuando todavía no se encuentra disponible. Este problema lo resuelve el DMN en dos niveles. Esos niveles son la distancia entre la dependencia de instrucciones. Por Ejemplo :

LDI **R2**,#78h

AND R3,**R2**,R6

SUB R4,R5,**R2**

La Instrucción AND muestra un primer nivel de dependencia de datos con respecto a la instrucción LDI, la cual carga un valor nuevo en el registro R2. Mientras que se puede observar que la instrucción SUB tiene dependencia de segundo nivel. La arquitectura DMN adelanta el valor correcto en ambos niveles antes de llegar a la etapa EX.

Escritura en Memoria de Datos

En la sección Descripción Funcional de cada Instrucción, se notó que la instrucción STR terminaba su ejecución durante la tercera etapa, es decir, escribe el valor sobre la M. de D. en la etapa EX, finalizando su ejecución. Esto trae como consecuencia un problema con las instrucciones JCN, HLT y RTI, como ejemplo mostraremos el siguiente código :

A4 : JCN @80

A5 : STR R1,R5

Si el valor del registro CN es distinto de cero la instrucción en la dirección A4h cambiará el flujo del programa a la dirección 80h, pero antes de que esto ocurra, es decir en el momento en que la instrucción A4 este en la etapa WB, la instrucción A5 (que se encuentra en la etapa EX) estará escribiendo el valor de R5 en la dirección apuntada por R1. Se supone que el programador no desea este comportamiento.

Para resolver este problema se diseñó la siguiente lógica dentro de la ALL-U que impide la escritura de la M. de D. en caso de que se presenten instrucciones de cambio en el flujo de programa, tales como JCN, HLT y RTI :

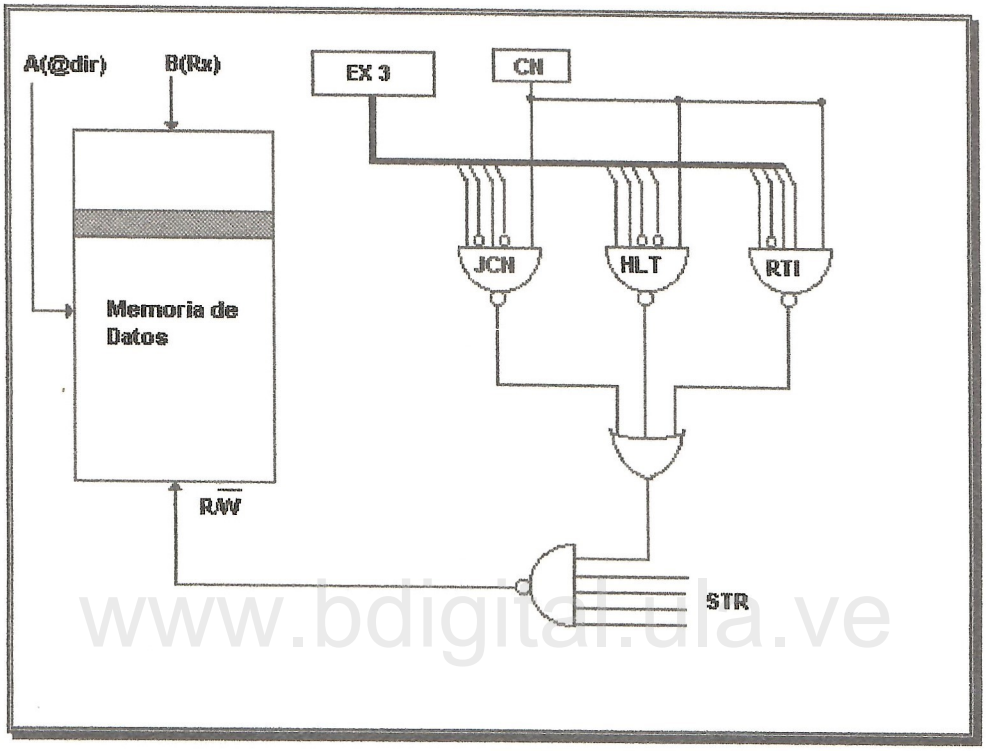

Figura 1-3 : Lógica de Bloqueo de Escritura en Memoria de Datos.

En la figura 1-3 se puede observar un registro llamado EX3, este registro es una extensión de la cola de control EX al nivel 3. La puerta NAND que activa la memoria en lectura/escritura, detecta que ocurra lo siguiente : el Opcode sea STR (1111 en binario) y que además exista una instrucción JCN o RTI, o HLT en EX3, y adicionalmente CN sea distinto de cero. Si una de estas condiciones ocurre esta puerta envía una señal en alto impidiendo la escritura en la M. de D.

Bloqueo de escritura sobre el Registro PC

Se debe impedir que cualquier instrucción, a excepción de JCN y RTI, escriba sobre el registro R15, que en nuestro caso es el registro PC, o contador de programas, por lo tanto dicho registro tiene la siguiente lógica dentro del archivo de registros, para evitar estos problemas :

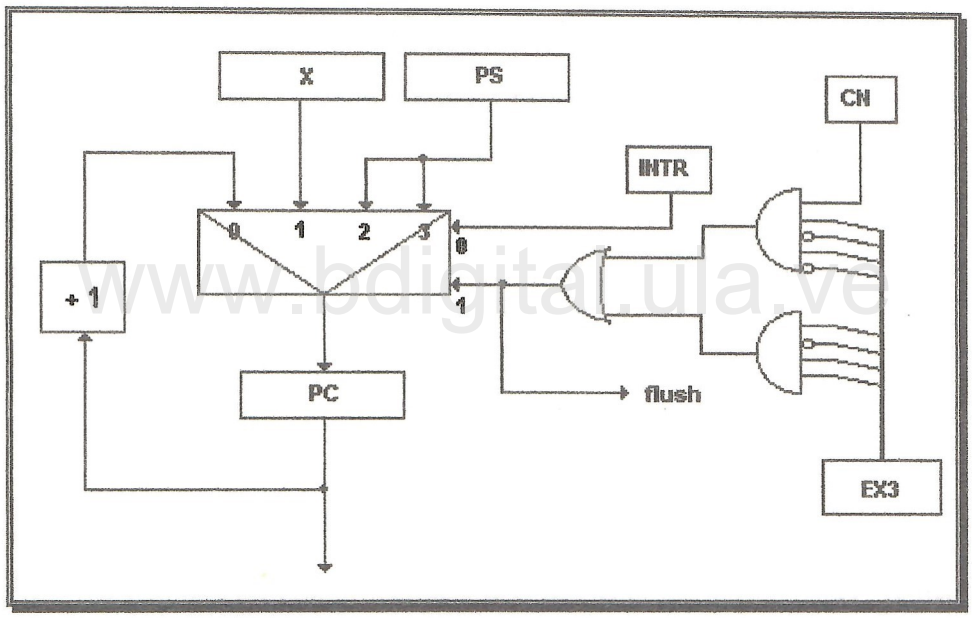

Figura 1-4 : Lógica para el bloqueo de escritura sobre el Registro PC (R15).

En la figura 1-4 se puede notar que el multiplexor controla si el registro PC es actualizado o bien por su valor actual + 1, o bien por el valor del registro X, o bien por el registro PS. El primer caso se da siempre que no ocurra ni una interrupción ni una instrucción JCN exitosa o un RTI en el nivel 3 de la cola EX. El segundo caso ocurre cuando hay una instrucción JCN, y esta es exitosa (CN distinto de 0), o cuando la

instrucción es RTI. Y el ultimo caso ocurre cuando es detectada una interrupción externa.

Manejo de Interrupciones

Todas las interrupciones son reconocidas en la etapa de WB. En caso de una interrupción el procesador cambia su flujo de control a una rutina para que pueda darle servicio a dicha interrupción. Para lograr esto se usa uno de los registros del archivo de registros como apuntador a la dirección de la rutina de servicio. Este registro es el R14, también llamado PS.

Cada vez que ocurre una interrupción se debe mantener el estado del programa que se está ejecutando, luego de servida la interrupción, se debe regresar al punto del programa donde ocurrió la interrupción.

Para lograr esto se implementó una cola de registros de 3 niveles donde se encolan los valores PC de las instrucciones que se ejecutan en el encauce, llamada PCQ (véase la figura 1-1), cada elemento de esta cola tiene a su lado una bandera, la cual va avanzando con la cola de PC. Al ocurrir un salto exitoso, se limpian las colas de control, y a su vez las banderas que están al lado de la cola de PC. Cuando es reconocida una interrupción una lógica activa el multiplexor que escoge el valor correcto del PC para ser resguardado en el registro PCR, de acuerdo a la tabla de verdad que se muestra a continuación :

PCR	B3	B2	B1	B0
PCq3	1	1	1	1
PCq2	0	1	1	1
PCq1	0	0	1	1
PC	0	0	0	1

Tabla 1-7 : Tabla de Verdad para escoger PCR

Es importante hacer notar que no se pueden servir interrupciones anidadas, no obstante cada vez que es recibida una interrupción se deshabilita la recepción de futuras interrupciones.

Evolución del Microprocesador DMN 17N78

Se tiene programada la siguiente evolución del Microprocesador DMN 17N78 :

- **DMN-T :** Microprocesador Orientado a Juguetes Programados. (DMN-7)
- **DMN-C :** Microprocesador DMN-T Avanzado, con las siguientes características :
    - Memoria Caché
    - Instrucciones de 32 bits
    - Predicción de Saltos
    - Instrucciónes de Carga sin Bloqueo
- **DMN-S :** Microprocesador con Arquitectura Super Escalar :

- Procesamiento de datos en 32 bits
- Emisión de 4 instrucciones por ciclo
- Ejecución Fuera de Orden
- Ejecución Especulativa
- Unidades Aritméticas de punto flotante

- **DMN-N :** Microprocesador para Redes Neuronales Artificiales : Arquitectura Super Escalar especializada para métodos neuronales

- **DMN-E :**Microprocesador de funciones ampliadas (Embbeded) :
  - Timers
  - BIU
  - Convertidores D /A

Capitulo II

El Simulador SimDMN

Lenguaje de Programación Java

¿Que es Java ?

Java es un lenguaje de programación desarrollado por James Gosling basandose en la sintaxis de C y C++, omitiendo las características semánticas que hacen C y C++ confuso e inseguro. Inicialmente se llamó Oak. El primer compilador de Java fue creado por Arthur van Hoff. Este lenguaje fue creado inicialmente para servir en pequeños sistemas de control hogareños, tales como controles remotos de televisores y termostatos de neveras y hornos. La potencialidad de este lenguaje fue debido a que se especificó para trabajar en cualquier sistema de microprocesador, independiente de la arquitectura. Esto se logra usando una maquina virtual, la cual se denomina Maquina Virtual Java o JVM. Sun Microsystems Inc. es el dueño actual de la marca Java.

La forma en que trabaja Java es muy simple : al compilar un archivo fuente en Java el compilador genera un código intermedio o **bytecode**, el cual es código binario que solo puede ser reconocido por una JVM. Este código debe ser ejecutado por un

interprete el cual transforma los bytecode a código maquina correspondiente a la arquitectura en la que se está ejecutando la aplicación.

Características del Lenguaje

- **Simple:** No requiere el uso de punteros para hacer referencia a objetos como en C++. No existen los punteros. Maneja todo por si mismo: memoria, manejo de errores, y múltiples tareas , sin que el programador tenga que ocuparse de esos detalles.
- **Familiar:** Ya que es parecido, en las estructuras básicas de control del lenguaje, a C y C++
- **Orientado a Objetos:** Es una característica muy importante en su filosofía, y la aplica mejor que C++
- **Innovador:** Es uno de los primeros lenguajes en proporcionar concurrencia per se, además, está su alta cohesión con el Web, es decir, con Java se puede hacer programación para la Internet.
- **Amplia Librería de Objetos :** posee una gran librería de objetos por omisión, esta librería cuenta con Clases para el manejo de ambientes gráficos, comunicaciones en red, entrada y salida, utilitarias, y para la integración de

código nativo[3] al lenguaje Java. En la actualidad esta librería de Clases se está expandiendo aún mas para que sea mas flexible y completa.

- **Fuertemente tipificado** : esta característica permite un mejor manejo entre los diversos Objetos de Clases componentes en una aplicación.

Tipos de Programas Java

- Aplicaciones "Stand-Alone" que poseen una interfaz basada en texto
- Aplicaciones "Stand-Alone" que poseen una interfaz gráfica
- Applets que corren en un ambiente habilitado de Java, tal como un Web-Browser con JVM incorporado, tal como el Netscape Navigator.
- Extensiones a páginas Web escritas con un lenguaje de "script" construido sobre Java, como el **JavaScript**, que junta los elementos de una página Web con el lenguaje Java
- Manejadores de contenidos y protocolos, que se auto añaden a los Web-Browsers para extender las capacidades del mismo.

[3] Código Nativo : se refiere a código específico de la arquitectura en la que se hace la aplicación.

Desarrollo del Simulador

Para el desarrollo del simulador, el trabajo se dividió en dos partes, a saber : diseño funcional del simulador, y diseño de la interfaz gráfica de usuario. Esta metodología sirvió para un mejor y mas rápido desarrollo, ya que de esta manera al ir desarrollando el simulador, se iban probando sus partes componentes, de manera totalmente independiente al desarrollo de la interfaz de usuario. Dicho desarrollo de la interfaz fue evolucionando de acuerdo a los requerimientos exigidos, para que dicha interfaz estuviera de acuerdo como un sistema altamente didáctico.

Diseño Funcional del Simulador

Inicialmente se estudió de una manera profunda los componentes del sistema a simular, es decir, el Procesador DMN 17N78. Una vez estudiado todo el funcionamiento se define cual es el contexto en el que se va a modelar el Sistema. Dicho contexto es un Sistema Digital, cuyos componentes básicos son dispositivos de lógica combinatoria y secuencial, los cuales trabajan de manera sincrónica [15]. Una vez definido este contexto se aplicó la metodología de Booch para definir su Diseño Orientado a Objetos.

Dentro de este contexto en el sistema a simular se observan las siguientes particularidades :

- El componente básico de intercambio de información en este Sistema es el bit. Pero por si solo un bit no puede transmitir suficiente información, por lo tanto se define *la Palabra o Word*[4], que para este caso en particular lo modelamos como un conjunto de 16 bits. También se toman en cuenta que existen conjuntos de bits mas pequeños, valga decir, el Byte, conjunto de 8 bits, y el Nibble, o conjunto de 4 bits, siendo este último el mas pequeño de todos.

- Este componente básico debe estar agrupado de una manera organizada, una de estas organizaciones es la *memoria*. En el sistema se observaron tres tipos de memoria : memoria de datos, memoria de instrucciones y memoria del archivo de registros.

- El flujo de control de la arquitectura se hace a través de colas de registros de capacidad limitada, y a su vez esta capacidad varía de acuerdo a la funcionalidad de la cola .

- Se deben evitar las dependencias de datos entre los datos que fluyen por la arquitectura, de esta manera se identifica una unidad de dependencia de datos.

- Por último, el corazón de todo el sistema es su Unidad Aritmético/Lógica y de Control integrada.

De esta manera se identifican las abstracciones claves que poseen los componentes funcionales de la arquitectura. Estas abstracciones se agrupan en tres

---

[4] A partir de este punto siempre llamaremos al conjunto de 16 bits como Word.

grandes paquetes que, empezando a aplicar la metodología de Booch, podríamos clasificar como las principales *Categorías de Clases* :

1.***Contenedor Binario ;***

2.***Memoria ;***

3.***Simulador ;***

Existen 2 abstracciones adicionales que están fuera del contexto del presente análisis, pero que se retomaran mas adelante, estas son :

- El Ensamblador ;
- La Interfaz Gráfica de Usuario ;

Modelado de las Clases Componentes del Simulador

En la metodología de Booch se definen cuatro notaciones para el modelado de un sistema usando el Modelo de Objetos como base. Dichas notaciones representan los estados estáticos y dinámicos de un Sistema, así como también dan una visión tanto *lógica como Física del mismo. Estas notaciones responden a las siguientes interrogantes* durante el proceso de diseño de un Sistema :

- ¿Que Clases existen en el Sistema ? y ¿Como están relacionadas entre si ?

- ¿Que mecanismos son usados para regular la colaboración entre objetos ?

- ¿Donde se debe declarar cada clase u objeto ?

- ¿A que procesador se le debe asignar un proceso ? y para un procesador dado ¿como se deben habilitar los tiempos de múltiples procesos[5] ?

---

[5] *Esto es en caso de que el modelo presente concurrencia.*

Entonces estos diagramas se denominan :

1. Diagrama de Clases ;

2. Diagramas de Objetos ;

3. Diagramas de Módulos ;

4. Diagramas de Procesos ;

En este apartado solo se observaran los Diagramas de Clases que conforman el diseño del sistema.

Empezaremos con las Categorías de Clases definidas en la sección anterior, y su relación entre ellas :

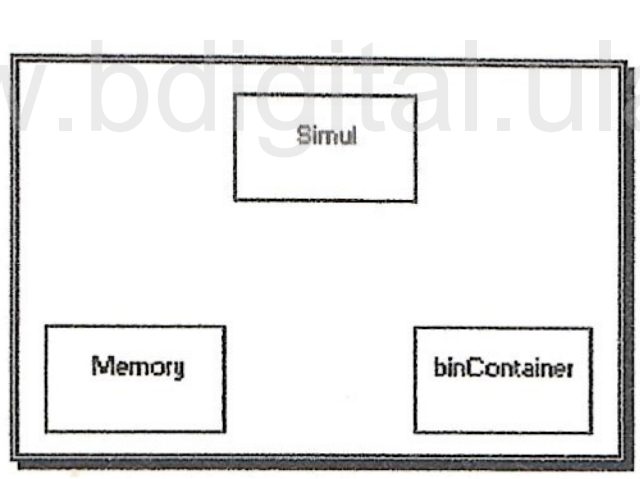

Figura 2-1 : Categorías de Clases del Sistema SimDMN.

Ahora observaremos el Diagrama de Clases de la Categoría de Clases **binContainer :**

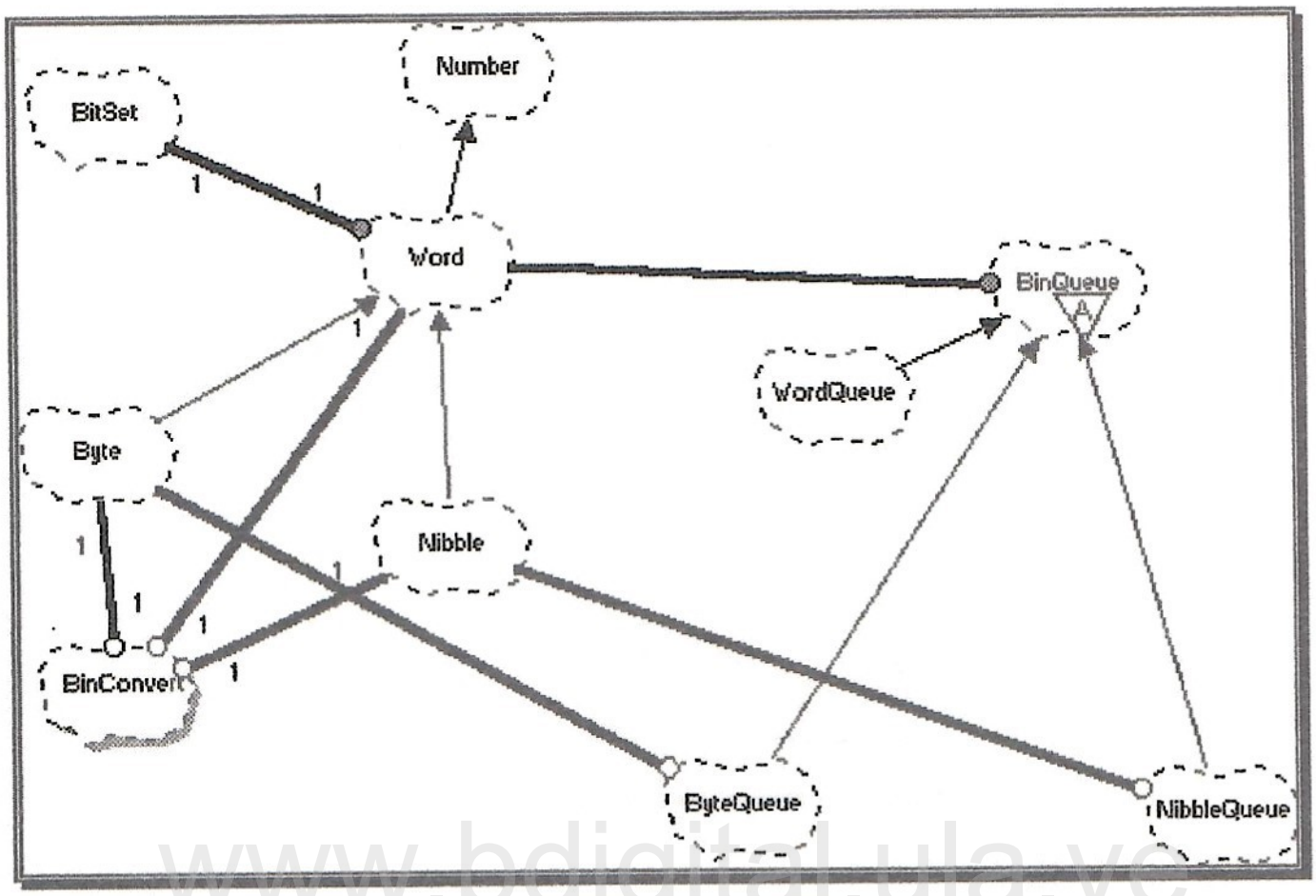

Figura 2-2 : Diagrama de Clases de la Categoría binContainer.

Esta Categoría de Clases contiene seis Clases componentes, una Clase Abstracta y una Clase Utilitaria. Las Clases BitSet y Number deben ser de alguna categoría externa. Las Clases BinQueue, WordQueue, ByteQueue y NibbleQueue implementan las abstacciones de colas que fueron identificadas en el análisis del Sistema.

La siguiente Figura se observa el Diagrama de Clases de la Categoría de Clases

Memory :

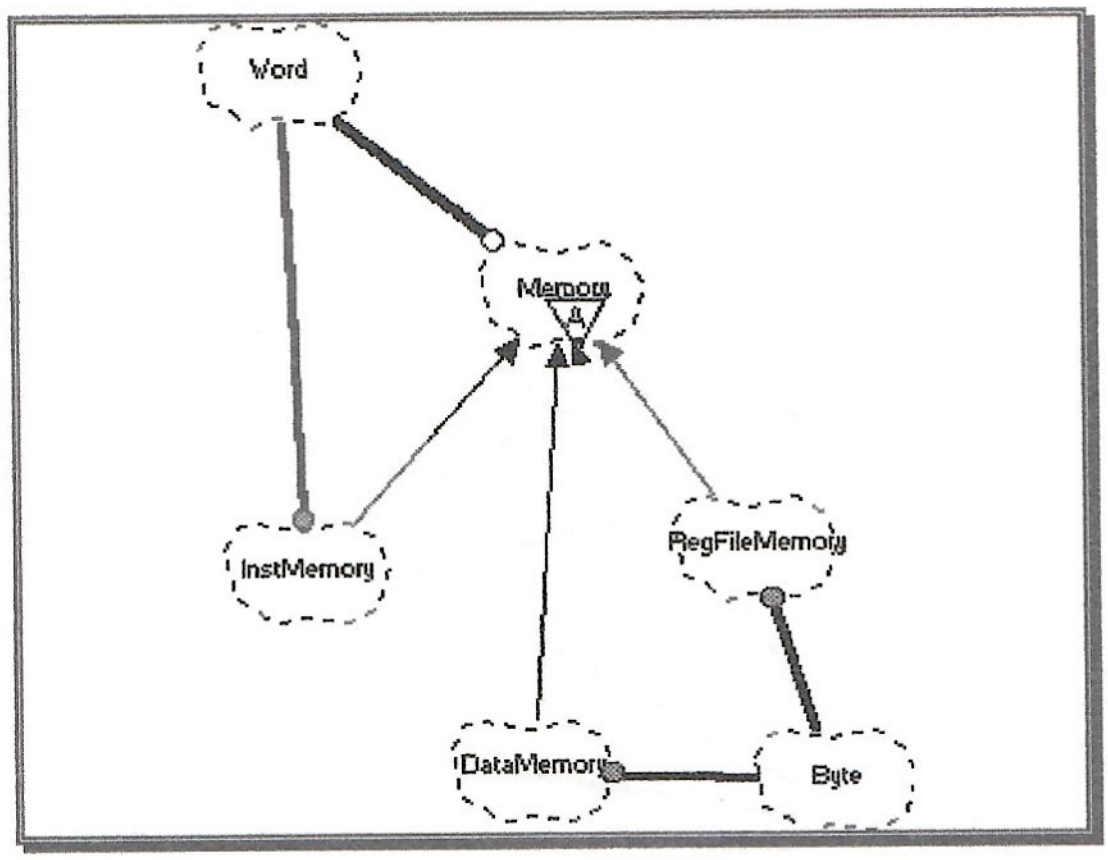

Figura 2-3 : Diagrama de Clases de la Categoría Memory.

Esta Categoría está compuesta de tres Clases componentes y una Clase Abstracta. Las Clases Word y Byte son importadas de la Categoría binContainer. Las Clases componentes DataMemory, InstMemory y RegFileMemory implementan las abstracciones de la Memoria de Datos, la Memoria de Instrucciones y la Memoria del Archivo de Registros.

A continuación se verá el Diagrama de Clases de la Categoría Simul, la cual es la categoría central de este modelado, ya que en ella se encuentran agrupadas todas las abstracciones definidas previamente.

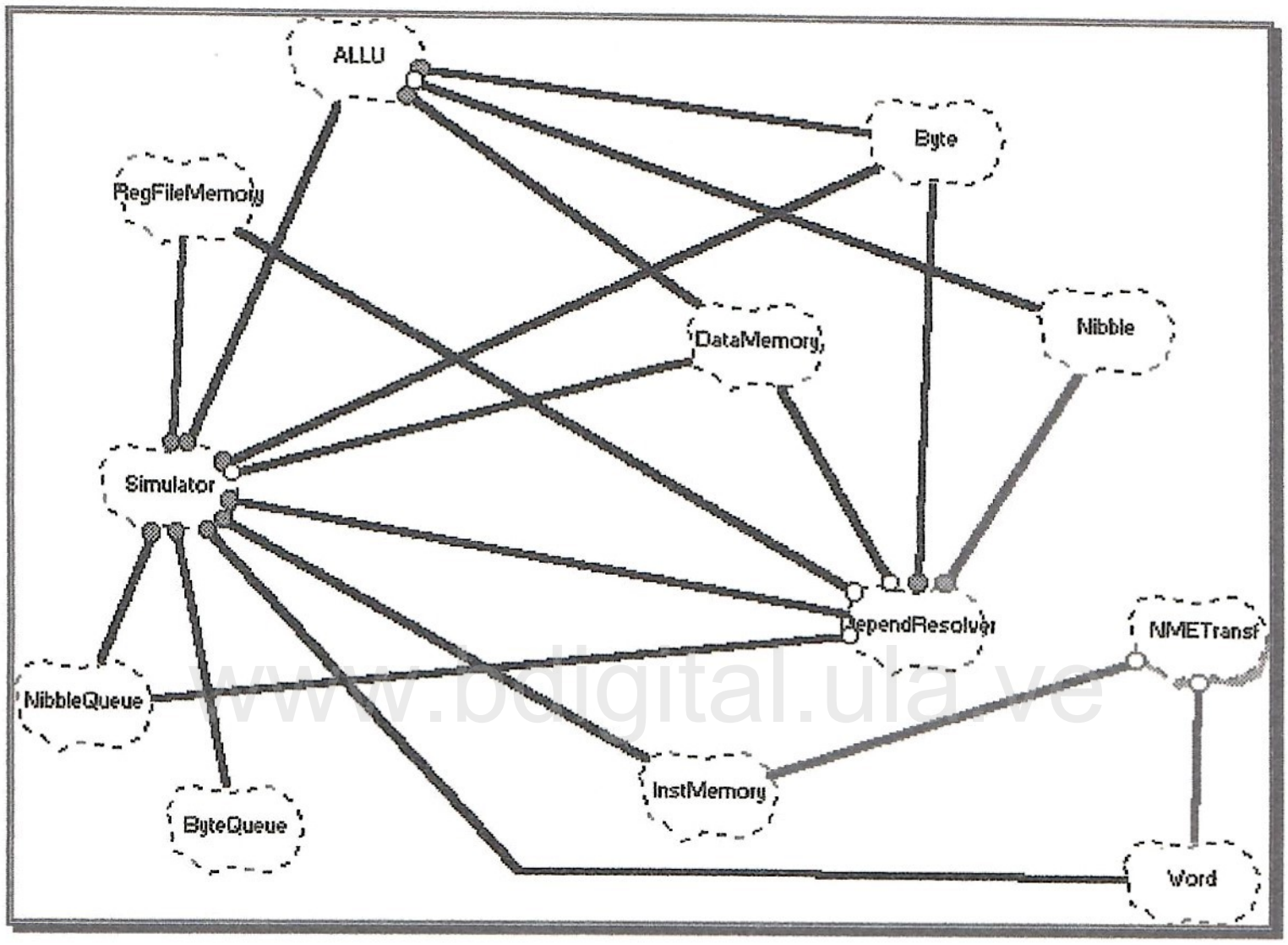

Figura 2-4 : Diagrama de Clases de la Categoría Simul.

En esta categoría se observan muchas de las clases ya definidas anteriormente. Las Clases que se definen en esta categoría son : Simulator, DependResolver y ALLU. Existe una Clase Utilitaria llamada NMETransf. Se puede Observar que la Clase Simulator es la que tiene mayor numero de enlaces, debido a que es esta la Clase central del proceso del Sistema.

Descripción de las Clases del Simulador

En este apartado se describirán las Clases que conforman el diseño del Simulador, así como también sus atributos y sus métodos.

Categoría binContainer

Clase Word

Descripción :

Clase básica de la categoría binContainer, implementa la abstracción de un contenedor binario de 16 bits, también implementa métodos para el acceso de valores binarios puros, conversiones a varias bases numéricas y aritmética en complemento a dos.

Atributos :

- bits : BitSet, bits integrantes de cualquier instancia de esta clase.
- nbits : Integer, numero de bits habilitados en una instancia de esta clase.

Métodos :

- add2Complement(Word sum) : Word, hace una adición en complemento a dos entre esta instancia y la que es pasada como parámetro;
- clearValue(), pone todos los bits en cero ;
- clone() : Object, copia una instancia de esta Clase en otra instancia de la misma Clase ;
- doubleValue() : Double, retorna el valor doble que contiene esta instancia ;
- floatValue() : Float, retorna el valor flotante que contiene esta instancia ;
- intValue() : Integer, retorna el valor entero que contiene esta instancia ;
- intValue2Complement() : Integer, retorna el valor entero en complemento a dos que contiene esta instancia ;
- longValue() : Long, retorna el valor entero largo que contiene esta instancia ;

- lt2Complement(Word sum) : Boolean, hace comparación en complemento a dos entre esta instancia y la que se está pasando como parámetro, retorna verdadero si esta instancia es menor ;
- setBit(Integer nbit), habilita un bit específico de la instancia ;
- setValue(Integer value), coloca en la instancia el valor binario dado por el parámetro. No acepta valores negativos ;
- setValue2Complement(Integer value), coloca en la instancia el valor binario en complemento a dos dado por el parámetro ;
- sub2Complement(Word sum) : Word, hace una substracción en complemento a dos entre esta instancia y la que es pasada como parámetro ;
- toBinaryString() : String, devuelve el contenido de esta instancia en una cadena con representación binaria ;
- toHexString() : String, devuelve el contenido de esta instancia en una cadena con representación hexadecimal ;
- toString() : String, devuelve el contenido de esta instancia en una cadena con representación decimal

Clase Byte

Descripción :

Esta clase es descendiente de la clase Word, la única diferencia es que habilita 8 bits en vez de 16, e implementa un método para la conversión entre objetos de la clase Byte a Nibble.

Atributos :

- Los hereda de la Clase Word ;

Metodos :

- toNibble() : Nibble, convierte el valor de este Byte a una instancia de la Clase Nibble, haciendo truncamiento binario del valor ;
- El resto de los métodos los hereda de la clase Word.

Clase Nibble

Descripción :

Esta clase es descendiente de la clase Word, la única diferencia es que habilita 4 bits en vez de 16, e implementa un método para la conversión entre objetos de la clase Nibble a Byte.

Atributos :

- Los hereda de la Clase Word ;

Metodos :

- toByte() : Byte, convierte el valor de este Nibble a una instancia de la Clase Byte;
- El resto de los métodos los hereda de la clase Word.

Clase BinQueue

Descripción :

La Clase BinQueue es una clase abstracta que implementa la definición para colas de Word, Byte o Nibble, de longitud finita.

Atributos :

- queue [] :Word [], el contenido de la cola, está declarada como Word debido a que es la superclase de los contenedores binarios.
- size : Integer, tamaño fijo de la cola ;

Métodos :

- pushQueue(Word w), introduce el valor de w en la cola, al hacer esto los demás valores en la cola se mueven hacia adelante, eliminándose el que se encuentra en el tope de la misma ;
- clearQueue(), Elimina todos los elementos de la cola, dejando la cola vacia ;

Clase WordQueue

Descripción :

Clase descendiente de la Clase BinQueue. Implementa una Cola de capacidad limitada, cuyos elementos son del tipo Word.

Atributos :

- Los hereda de BinQueue.

Métodos :

- seeElement(int pos) : Word, devuelve el valor apuntado por pos, donde, si pos = 0, se está viendo el final de la cola, y si pos = size -1, se está viendo el tope. Si pos cae fuera de este rango, el método siempre retorna como si fuera pos = 0 ;
- seeTopQueue() : Word, devuelve el valor al tope de la cola ;
- El resto de los Métodos los hereda de BinQueue

Clase ByteQueue

Descripción :

Clase descendiente de la Clase BinQueue. Implementa una Cola de capacidad limitada, cuyos elementos son del tipo Byte.

Atributos :

- Los hereda de BinQueue.

Métodos :

- seeElement(int pos) : Byte, devuelve el valor apuntado por pos, donde, si pos = 0, se está viendo el final de la cola, y si pos = size -1, se está viendo el tope. Si pos cae fuera de este rango, el método siempre retorna como si fuera pos = 0 ;
- seeTopQueue() : Byte, devuelve el valor al tope de la cola ;
- El resto de los Métodos los hereda de BinQueue

Clase NibbleQueue

Descripción :

Clase descendiente de la Clase BinQueue. Implementa una Cola de capacidad limitada, cuyos elementos son del tipo Nibble.

Atributos :

- Los hereda de BinQueue.

Métodos :

- seeElement(int pos) : Nibble, devuelve el valor apuntado por pos, donde, si pos = 0, se está viendo el final de la cola, y si pos = size -1, se está viendo el tope. Si pos cae fuera de este rango, el método siempre retorna como si fuera pos = 0 ;
- seeTopQueue() : Nibble, devuelve el valor al tope de la cola ;
- El resto de los Métodos los hereda de BinQueue

Clase BinConvert

Descripción :

Esta Clase es utilitaria, es decir, no se hacen instancias de ella, solo define 4 métodos estáticos para la conversión de Word a Byte y de Byte a Nibble.

Atributos :

- No tiene, es una Clase utilitaria ;

Métodos :

- wordToUpperByte(Word w) : Byte, devuelve el Byte alto que contiene el parámetro w ;
- wordToLowerByte(Word w) : Byte, devuelve el Byte bajo que contiene el parámetro w ;
- byteToUpperNibble(Byte b) : Nibble, devuelve el Nibble alto que contiene el parámetro b ;
- byteToLowerNibble(Byte b) : Nibble, devuelve el Nibble bajo que contiene el parámetro b ;

Categoría Memory

Clase Memory

Descripción :

Clase Abstracta que implementa los métodos para simular una organización de memoria y el acceso a la misma usando como elementos Word, Byte o Nibbles.

Atributos :

locs[] : Word[], instancias que representan las localidades de memoria ;
accessPointer : Integer, puntero de dirección de lectura/escritura de la memoria ;
capacity : capacidad de la Memoria en cantidad de localidades ;

Métodos :

- readMemory() :Word, lee una localidad de memoria apuntada por el atributo accessPointer ;
- readMemory(Integer pos) :Word, lee una localidad de memoria apuntada por el parámetro pos, y adicionalmente cambia la posición del accessPointer ;
- writeMemory(Word w), escribe un Word en la localidad apuntada en accessPointer ;
- writeMemory(Word w, Integer pos), escribe un Word en la localidad apuntada por el parámetro pos, y adicionalmente cambia la posición del accessPointer ;
- setAccessPointer(Integer pos), cambia la posición del accessPointer con el valor del parámetro pos ;
- getCapMemory() :Integer, devuelve la capacidad de la memoria ;

Clase RegFileMemory

Descripción :

Esta Clase es descendiente de la Clase Memory, la particularidad que tiene es que tiene una capacidad fija de 16 localidades del tipo Byte, y la localidad 0 siempre tiene como valor cero. Implementa el modelo de la memoria del Archivo de Registros del DMN. Además implementa un método adicional a su superclase, para al limpieza de *todas las localidades en un solo mensaje.*

Atributos :

- Los hereda todos de la Clase Memory.

Métodos :

- clearMemory(), coloca todas las localidades de esta memoria en 0 ;
- El resto de los métodos de esta clase son heredados de la Clase Memory ;

Clase InstMemory

Descripción :

Esta Clase es descendiente de Memory, tiene 256 localidades del tipo Word. Implementa el modelo de Memoria de Instrucciones del Procesador DMN.

Tanto los Atributos como los Métodos de esta Clase son todos descendientes de su superclase, no se implementas métodos distintos.

Clase DataMemory

Descripción :

Esta Clase es descendiente de Memory, tiene 256 localidades del tipo Byte. Implementa el modelo de Memoria de Datos del Procesador DMN.

Tanto los Atributos como los Métodos de esta Clase son todos descendientes de su superclase, no se implementas métodos distintos.

Categoría Simul

Clase DependResolver

Descripción :

Esta clase implementa la lógica de resolución de dependencias de datos del DMN, a su vez contiene las instancias de los Bytes A y B, que serán los operandos de la Unidad Aritmético/Lógica/Control, y además los Nibbles SLA y SLB.

Atributos :

- slA, slB : Nibble, valores de selección en el Archivo de Registro ;
- aReg, bReg : Byte, valores de A y B, que son seleccionados por slA, y slB del Archivo de Registros ;

Métodos :

- fetchSelector(Byte b), divide el Byte b en su parte alta y baja, la alta se deposita en slA y la baja en slB.
- reset(), coloca en cero todos los atributos de esta instancia ;
- resolve(RegFileMemory rf, Byte xr, Byte rldi, NibbleQueue wbq, Byte sel), este es el método que resuelve la dependencia de datos, al invocarlo, los valores de aReg y bReg deben estar de acuerdo a lo que dicta la lógica de resolución de datos del DMN.

Clase ALLU

Descripción :

Esta Clase implementa toda la funcionalidad de la Unidad Aritmético/Lógica/Control del DMN. Además de esto implementa dentro de ella la instancia de Memoria Principal de Datos, el Registro de Retorno de interrupción PCR, y el registro de resultados X.

Atributos :

- dMemory : DataMemory, instancia la Memoria de Datos del Procesador DMN ;
- flush : Boolean, bandera que es verdadera cuando ocurre un salto condicional exitoso ;
- PCR : Byte, dirección de retorno de una instrucción RTI ;
- rldi : Registro de almacenaje inmediato de datos ;
- xReg : Byte, Byte de resultados de la ALL-U ;

Métodos :

- exMux(Byte ra, Byte rb, RegFileMemory rf, Nibble exs, Nibble sla, Nibble slb) este es el método que ejecuta la operación básica de la ALL-U, opera sobre los Bytes ra y rb basado en la operación definida en exs, el resultado de la operación (de acuerdo a cual sea), se almacena en el atributo xReg ;
- reset(), limpia todos los atributos de una instancia de esta Clase ;

Clase Simulator

Descripción :

Clase Central del Simulador, en ella se implementan todas las unidades funcionales y de datos del mismo. Además en ella se implementan los métodos que hacen el funcionamiento del mismo de una manera totalmente sincrónica.

Atributos :

- alu : ALLU, implementación de la Unidad Aritmético/Lógica/Control
- depUnit : DependResolver, implementación de la unidad de Dependencia de Datos ;
- iMemory : InstMemory, implementación de la Memoria de Instrucciones ;
- instruction : Word, palabra de instrucción que se deposita en el ciclo de Captura de Instrucción (IF) ;
- pcQueue : ByteQueue : implementación de la cola de PC ;
- pcReg : Byte, registro que contiene el contador de programas del DMN ;
- pcState [] : Boolean [], array de banderas que acompañan al pcQueue en el control de la dirección de retorno de interrupción.
- rFile : RegFileMemory, implementación de la Memoria del Archivo de Registros ;
- wbQueue, exQueue : NibbleQueue, implementación de las colas de escritura en Archivo de Registros y Cola de Ejecución, respectivamente ;

Métodos :

- fecthInstruction(), este mensaje hace el proceso de IF del DMN, colocando la instrucción apuntada por pcReg, en el atributo instruction ;
- flushQueues(), vacia el contenido de las colas de Control ;
- intrAck(), método que habilita, sincroniza y reconoce una interrupción externa al procesador ;
- pushQueues(), este mensaje introduce en las colas de control sus respectivos valores ;
- reset(), re inicia todos los estados del procesador, a excepción de la memoria de datos e instrucciones ;
- step(), este mensaje es el que hace todo el proceso de simulación, en el se agrupan los cuatro ciclos del procesador : IF, RS, EX,WB. A continuación se lista el algoritmo que lo compone :

```
Si HA es distinto de 255
        fetchInstruction()
        pushQueues()
        depUnit.resolve()
        alu.exMux()
        writeBack()
        Si alu.flush es verdadero
                flushQueues()
        Fin Si
Fin Si
```

- writeBack(), este mensaje ejecuta el proceso de WB del DMN, colocando el valor de X en el lugar que indica el tope de la cola wbQueue ;

Breve Descripción del Diseño del Ensamblador

Introducción

El diseño del ensamblador se basó en las técnicas de ensamblado en dos pasadas que propone Aho y Ullman en [1]. Esta técnica consiste en que durante la primera pasada *se lee el archivo de entrada (o flujo de datos/tokens), se encuentran todos los* identificadores que denoten direcciones de memoria, tales como etiquetas. Al ser encontrada una etiqueta, la dirección que denota es guardada en una tabla de símbolos, junto con dicha etiqueta. Además de esto, se hace la verificación sintáctica del archivo fuente, si esta verificación es exitosa y en la sintaxis no existen referencias a etiquetas el equivalente binario de la instrucción es guardado en una tabla de compilación.

En la segunda pasada se revisa la tabla de símbolos, si existen símbolos, se revisa nuevamente el archivo fuente para observar donde se encuentran estos símbolos, cuando se encuentra un símbolo coincidente en el archivo fuente se termina de resolver el formato binario que es producido en esta operación y se actualiza la entrada correspondiente en la tabla de compilación.

Macro algoritmo del ensamblador

Seguidamente se lista el macro algoritmo de la primera pasada del ensamblador :

```
Repetir para cada línea en el Flujo de entrada del programa fuente
    Si la linea no es un comentario
        Si existe etiqueta
            Introducir etiqueta y dirección en tabla de símbolos
        Fin Si
        Verificar Sintaxis de la línea
        Si la sintaxis es correcta
            Si la sintaxis no requiere etiqueta
                Introduce equivalente binario de instrucción en tabla de
                Compilación
            De lo Contrario
                Introduce equivalente binario de instrucción en tabla de
                Compilación, pero sin resolver el valor de la direccion
                apuntada por la etiqueta
            Fin Si
            Incrementar dirección
        De lo Contrario
            Emitir Error para esta línea
        Fin Si
    Fin Si
Fin del lazo iterativo
```

Se observa a continuación el macro algoritmo de la segunda pasada :

```
Repetir para cada línea en el Flujo de entrada del programa fuente
        Si la sintaxis requiere etiqueta
                Se habilita bandera en falso
                Repetir para cada entrada en la tabla de símbolos
                        Verifica si la entrada en la tabla de símbolos coincide
                        con la etiqueta
                        Si coinciden
                                Resuelve el resto del valor binario en tabla de
                                compilación
                                Se habilita bandera en verdadero
                        Fin SI
                Fin del lazo iterativo
                Si la bandera se mantuvo en falso
                        Emitir Error para esta línea
                Fin Si
        Fin Si
Fin del lazo iterativo
```

Implementación

La metodología a emplear aquí fue en todo momento estructurada, debido principalmente a la forma de los algoritmos de compilación. Dado que Java es totalmente Orientado a Objetos, se debió implementar una Categoría de Clases, donde todas estas Clases son Clases Utilitarias. Por lo tanto, la Clase que implementa el Ensamblador, es una Clase Utilitaria de todo el Sistema.

Diseño de la Interfaz Gráfica de Usuario

En el diseño de la Interfaz gráfica de usuario, se usó mas las características del Lenguaje Java que un diseño propiamente de Objeto.

De esta manera se usaron diversas Clases propias de la librería de Clases del Lenguaje Java, esto con el propósito de hacer la aplicación lo mas portable posible. De esta librería se usaron ciertas Clases de los siguientes paquetes de clases :

- java.awt
- java.applet
- java.net
- java.io

Así entonces, la Interfaz Gráfica de Usuario se centra en una Clase llamada DMNFrame. En esta Clase se implementan tanto los objetos de interfaz gráfica de usuario, como instancias de las Clases del Simulador. Evidentemente DMNFrame no es la única Clase de la implementación de la Interfaz, existen 26 Clases adicionales que tienen diversas funciones como listas de datos especializados para los datos del simulador, Paneles de botones, etc. el lector curioso de como se implementó todo esto debe referirse a los listados del paquete GUI, y tener a la mano una copia del Java API [9].

Es de hacer notar que esta aplicación fue diseñada para que trabajara tanto como un Applet, como una Aplicación "Stand-Alone", es decir, en aquellos Sistemas Operativos que soporten Java El Simulador correrá como un programa normal, sin necesidad de usar el Web Browser.

Manual de Usuario del Simulador

A continuación se detallará el funcionamiento del SimDMN basándonos en la interfaz gráfica del Sistema.

Ambiente SimDMN, Generalidades

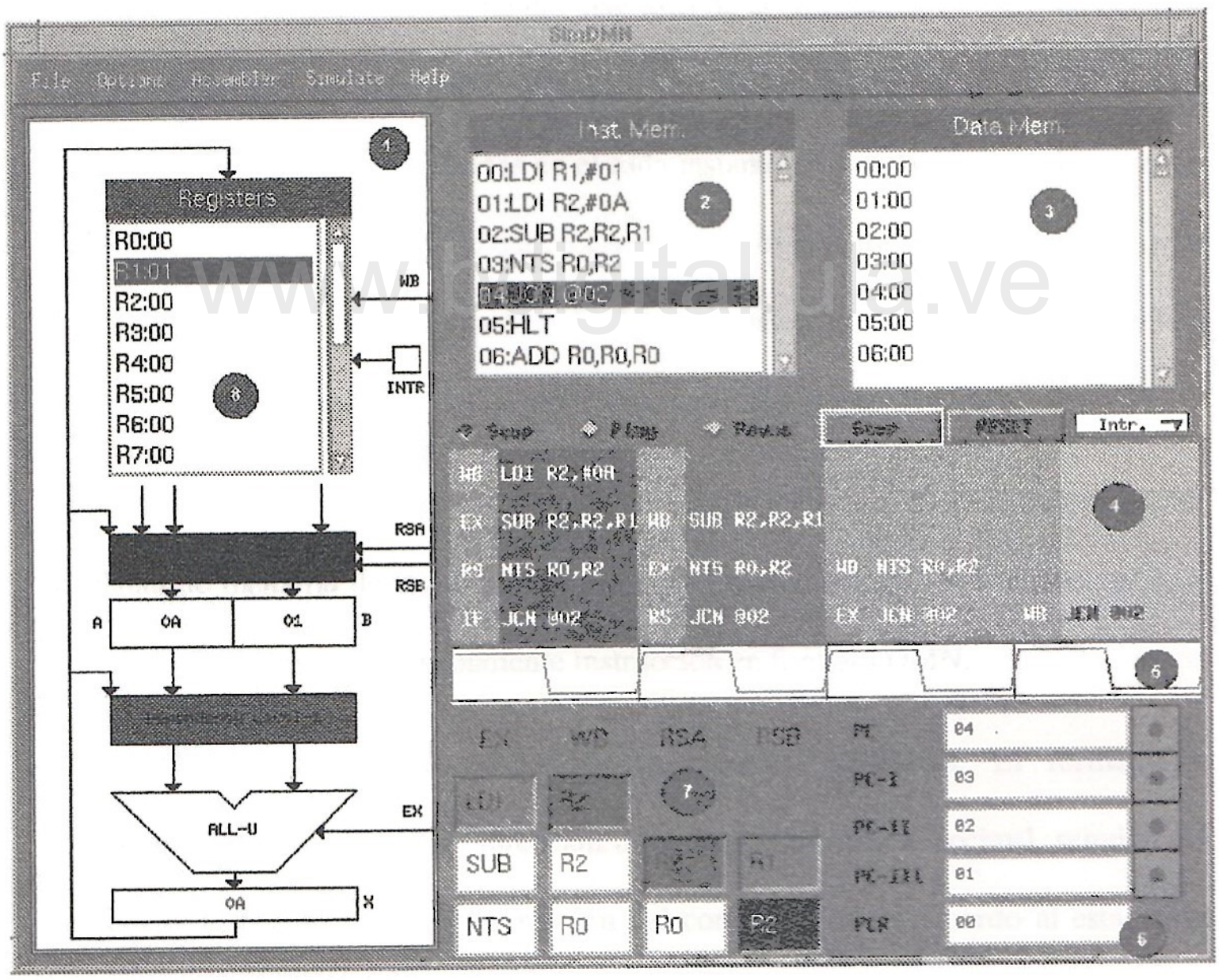

Figura 2-5 : Ventana Principal del SimDMN

La Figura 2-5 muestra la ventana principal del SimDMN. Esta ventana se observa de manera idéntica (aunque con pequeñas diferencias) en todo sistema Unix con X11-Motif como ambiente gráfico. Existirán diferencias notables en sistemas como Windows 95, OS/2 o Mac, principalmente en las formas de los menús, botones, tipos y tamaños de letras. Aún así, estos cambios no afectan para nada la funcionalidad del Sistema.

En la Figura 2-5 se pueden notar unos círculos numerados, estos círculos servirán para guiarnos a través de los distintos componentes del Simulador.

1. Esta es la gráfica de la zona central del diagrama de la Arquitectura DMN, que en este caso deja de ser estática para convertirse en diagrama dinámico, en el cual se modifican los valores representados en el cada instante de simulación. Se observan los multiplexores de dependencia de datos, los registros operandos de la ALL-U llamados A y B, el registro de salida del la ALL-U, X, y la bandera de interrupción. También se observa el Archivo de Registros introducido en este diagrama.

2. En esta lista se puede observar el contenido de la memoria de instrucciones, cada localidad de memoria de instrucciones aparece con su dirección en hexadecimal, y al lado de la dirección la correspondiente instrucción en formato DMN.

3. Esta lista representa el contenido de la memoria de datos. El formato de representación es el siguiente : dirección de la localidad en hexadecimal, seguido del contenido de la misma, la representación del contenido es de acuerdo al estado del menú de representación, del cual se discutirá mas adelante. Para introducir un dato en esta memoria basta con localizar la dirección donde se desea introducir el dato y hacer

doble click sobre la posición, Posteriormente aparecerá un diálogo que preguntará sobre el cambio de valor.

4. Panel de Encauzamiento y botones de control del Simulador. Este panel representa uno de los conceptos propios de la arquitectura RISC. Se observan como una "escalera" de listados, estos listados representan las etapas por las que van a pasar las instrucciones en 4 ciclos del procesador. Es decir, primer listado (de izquierda a derecha) representan las instrucciones que están siendo procesadas en el presente ciclo o ciclo i, el segundo listado representan las instrucciones a procesarse en el ciclo i + 1, el tercero en el ciclo i + 2, y el último listado es la instrucción que se procesara en el ciclo i + 4. Nótese que en este ultimo listado la instrucción que aparece es la misma que está entrando al procesador en este instante, es decir, la instrucción que en este instante está en la etapa IF, dentro de tres ciclos más estará en WB. Encima de este panel se encuentran seis botones cuya utilidad es la de controlar el proceso de simulación. Los tres primeros (nuevamente de izq. a der.) son mutuamente excluyentes, y representan el estado en que se encuentra el simulador. El botón que sigue (Step) sirve para que el simulador solo avance un solo ciclo de reloj. El botón reset, re inicia el Simulador sin borrar las memorias de datos ni instrucciones. Y El último botón Intr (que tiene una banderilla) es el botón que habilita una interrupción en el Simulador.

5. Representación gráfica del reloj del sistema, se pueden observar cuatro ciclos, los cuales están alineados con los listados del Panel de Encauzamiento. Esto quiere decir que el ciclo mas hacia la izquierda es el ciclo actual.

6. PC, Cola de PC y PCR. En esta parte de la interfaz se observa directamente el registro que sirve de PC para el DMN, así como también la cola de PC en sus tres niveles. También se observa el registro PCR. Tanto en el Registro PC, como en la cola de PC se observan unas cajas con un punto, este representa la bandera de PCs, cuando el punto se encuentra de color rojo indica que la bandera está activa.

7. Colas de Control del DMN. Aquí se observan las cuatro colas de control principales de la arquitectura DMN, como son : RSA, RSB, WB y EX, la profundidad de estas colas son : 2 niveles para EX, 3 para WB, y 1 para RSA y RSB. El resto de la representación (el cual se encuentra del mismo color de fondo de la aplicación) no forman parte de la cola, se colocan para observar cual es la consecuencia de una dependencia de datos.

8. Por último tenemos la representación del archivo de registros, el cual es una lista de 16 localidades, donde aparecen denotados los registros R0 al RB (B es 11 en hexadecimal) , y posteriormente los registros especiales CN, HA, PS y PC.

Menú Principal

Consta de cinco Sub Menús, a saber :

- **File,**
- **Options,**
- **Assembler,**
- **Simulate y**
- **Help .**

Sub Menú File

La siguiente Figura muestra el contenido del Menú File :

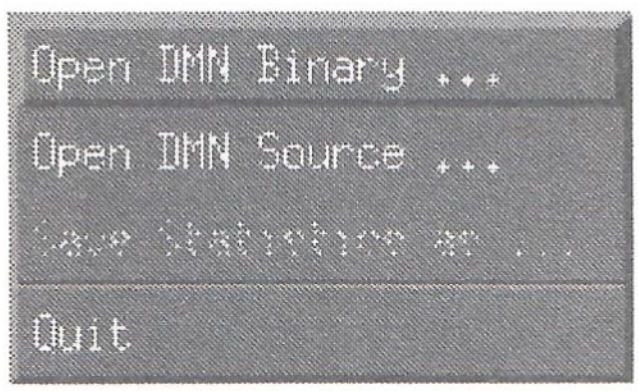

Figura 2-6 : Sub Menú File

Opción Open DMN Binary : Con esta opción se abren archivos en formato binario del DMN, es decir ejecutables que contienen instrucciones de DMN ya procesadas por algún otro ensamblador de DMN.

Opción Open DMN Source : En esta Opción se abre un archivo ASCII que contenga instrucciones de ensamblador para DMN. Al abrir un archivo exitosamente, invoca de manera inmediata al ensamblador integrado del SimDMN.

Nota : Como se discutió previamente Java tiene la capacidad de hacer varios tipos de aplicaciones, a saber, aplicaciones Stand Alone, ó Applets. El SimDMN fue implementado de ambas formas, es decir, trabaja tanto como un Applet dentro de un Browser como una aplicación Stand Alone. Por lo tanto, cuando el SimDMN funciona como aplicación Stand Alone, al usar una de las opciones anteriores (Open DMN Binary y Open DMN Source), aparece un diálogo como el siguiente :

Figura 2-7 : Diálogo para Apertura de Archivos cuando SimDMN es una aplicación Stand Alone

Sin Embargo cuando el SimDMN trabaja como un Applet, el diálogo que aparece es como el que se muestra a continuación :

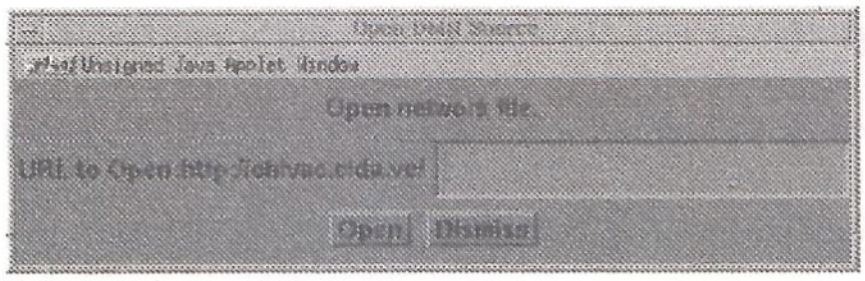

Figura 2-8 : Diálogo para Apertura de Archivos cuando SimDMN es un Applet

Para una descripción detallada de cual es la razón de esta diferencia véase el Capítulo III, sección : Problema inherente al Lenguaje Java.

Opción Save Statistics As : No fue implementado.

Opción Quit : Sale del SimDMN, presentando previamente el siguiente diálogo :

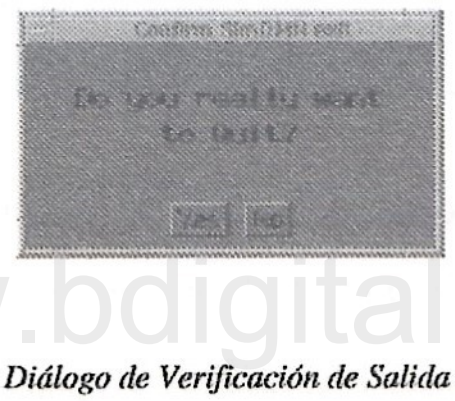

Figura 2-9 : Diálogo de Verificación de Salida del SimDMN

Sub Menú Options :

Se observan los siguientes menús :

Figura 2-10 : Sub Menú de Options, junto con su Sub Menú Data Rep.

Opción Show Mnemonics : Cuando esta Opción está seleccionada indica que tanto la lista de Memoria de Instrucciones, como las Colas de Control representaran sus datos usando mnemonicos, en vez de su contenido numérico.

Opcion Data Representation : Al seleccionar esta opción aparece el sub menú que aparece a la derecha de la Figura 2-8, indicando que una de las tres representaciones debe estar activa. Por omisión se encuentra activa la representación Hexadecimal. Al cambiar alguna de las representaciones, se actualizan todos los campos de manera automática.

Sub Menu Assembler

La siguiente figura muestra el sub menú Assembler :

Figura 2-11 : Sub Menu Assembler.

Opción Edit New DMN Program : Habilita el Ensamblador DMN con el editor vacío para editar un programa nuevo.

Opción Edit Current DMN Program : Habilita el Ensamblador DMN, pero dentro del editor se coloca el programa que se encuentra en la memoria de Instrucciones para ser modificado.

Sub Menú Simulate

Se observa el sub menú Simulate :

Figura 2-12 : Sub Menú Simulate.

Este Menu tiene la misma funcionalidad que la Barra de Botones de Control que está encima del Panel de Encauzamientos (No. 4 en la Figura 2-5).

Sub Menú Help

A continuación se presenta el sub menú Help :

Figura 2-13 : Sub Menu Help.

Las dos opciones de este menú son auto explicativas, la primera activa una página de Ayuda del SimDMN, y la segunda muestra el siguiente diálogo :

Figura 2-14 : Diálogo About

Ambiente del Ensamblador

El ambiente del ensamblador se invoca mediante uno de los siguientes casos :

1. Cuando se abre un archivo fuente DMN en la opción *Open DMN Source* del Sub Menú *File* ;

2. Cuando se usa la opción *Edit New DMN Program* del Sub Menu *Assembler* ;

3. Cuando se usa la opción *Edit Current DMN Program* del Sub Menu *Assembler* ;

Una vez invocado la pantalla que se observa es la siguiente :

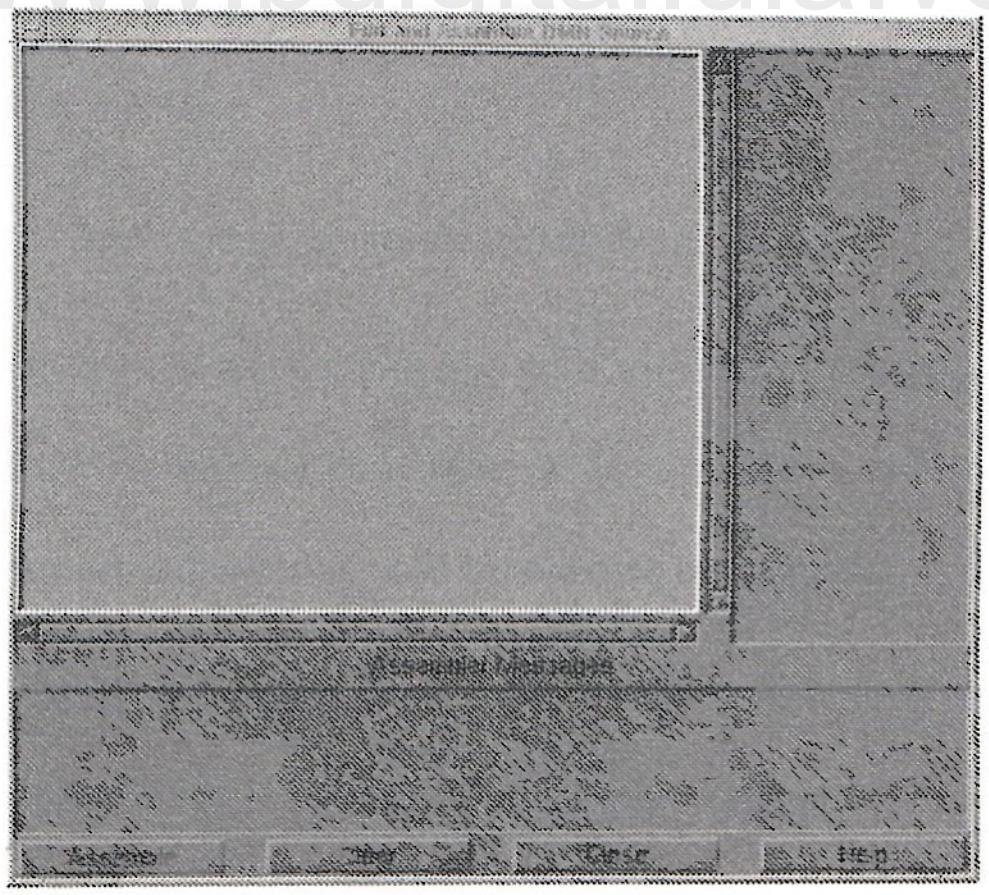

Figura 2-15 : Ventana principal del DMN Assembler.

En la Figura 2-13 se observa que este ambiente se compone de tres ventanas y una barra de botones de control. La ventana azul es el editor de programas DMN. A su derecha se encuentra la ventana de resultado de ensamblaje en la cual se muestra el código maquina generado por el ensamblador una vez ensamblado. Debajo de estas ventanas se observa la ventana de mensajes del ensamblador, en esta ventana se muestran los mensajes generados por el ensamblador en caso de errores en el código fuente, o inclusive cuando el ensamblaje es exitoso. Es de hacer notar que mientras el ensamblador se encuentre activo, la ventana del Simulador se desactiva del todo, es decir, todos los eventos son enviados a la ventana del ensamblador, impidiendo de esta manera que se interáctue con la interfaz del Simulador.

Los botones de control son cuatro :

1. **Assemble** : inicia el proceso de ensamblado del código que se encuentra en el cuadro de edición ;

2. **Clear** : Limpia todo el contenido del cuadro de edición ;

3. **Close** : Cierra la ventana principal del Ensamblador, devolviendo el control al SimDMN. Lo que está en el editor se pierde a menos que sea ensamblado y posteriormente depositado en la memoria de instrucciones.

4. **Help** : Muestra un diálogo de ayuda de los mnemonicos de la arquitectura DMN, y la sintaxis del ensamblador, la Figura 2-14 muestra este diálogo :

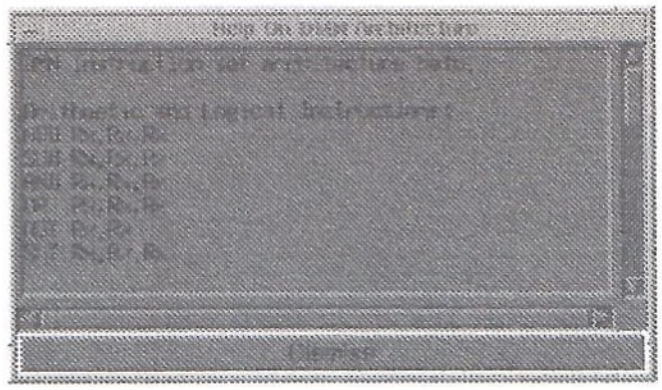

Figura 2-16 : Diálogo de Ayuda del Ensamblador DMN

Para generar código maquina en este ensamblador, solo se debe introducir los mnemonicos correspondientes siguiendo la sintaxis de la arquitectura DMN 17N78, y posteriormente se presiona el botón **Assemble**. Al presionar este botón se ensambla el código que se encuentra en el editor, si el ensamblaje es exitoso se muestra en la porción derecha del Ensamblador los códigos maquina (en la representación numérica activa) correspondientes al código editado. También aparece un diálogo para confirmar si se desea pasar los códigos maquina generados a la memoria de instrucciones del SimDMN. En la siguiente figura se puede observar el diálogo del que se ha hablado :

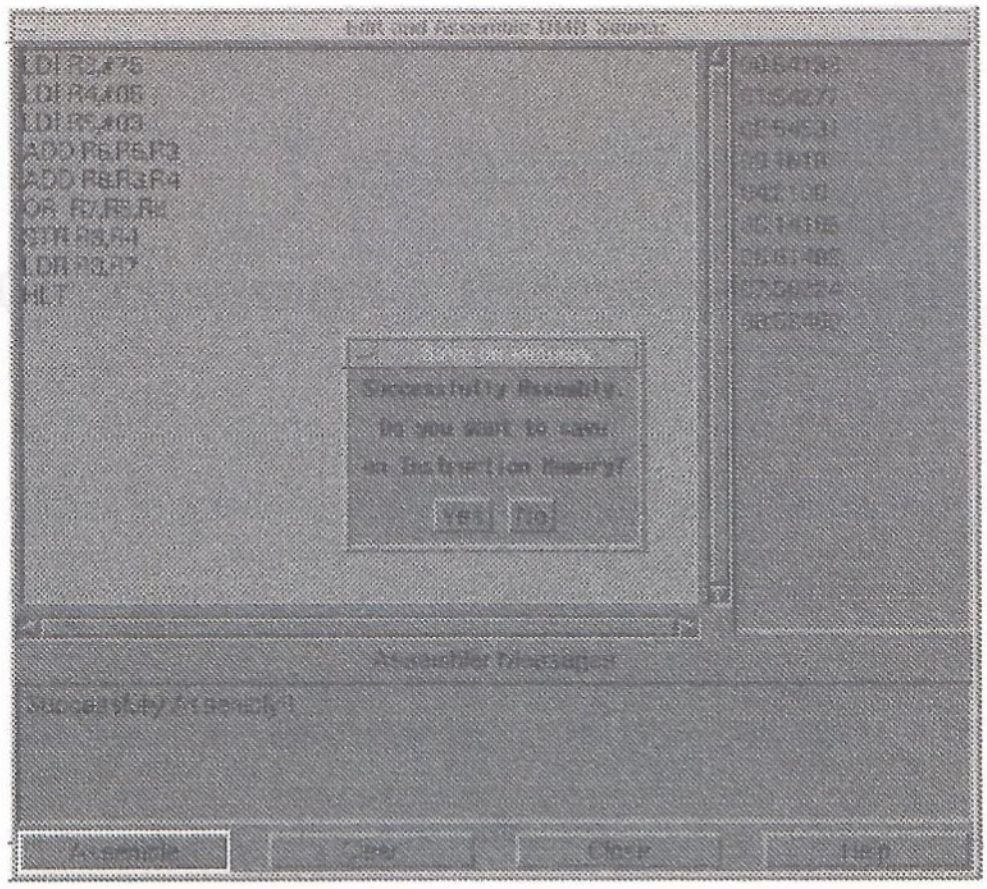

Figura 2-17 : El Ensamblador haciendo un ensamblaje exitoso.

De esta manera concluimos este pequeño manual de usuario del SimDMN.

Capítulo III

Conclusiones y Recomendaciones.

Como conclusión principal se puede decir que se pudieron lograr de una manera muy satisfactoria los objetivos centrales del proyecto :

- Se implementó un simulador de la arquitectura DMN 17N78 que cumple exactamente con todos los requisitos dados por la especificación del procesador
- Se usaron de manera totalmente satisfactoria las técnicas de diseño y programación Orientadas a Objetos.
- Se programó totalmente este Simulador en Leguaje Java.

Problema inherente al Lenguaje Java

En el desarrollo se presentó un problema que es totalmente inherente al lenguaje Java.

Lo que se ha observado es que cuando el programa se ejecuta como un Applet no se pueden leer archivos de la maquina local, es decir, supóngase que el simulador se ejecuta en una PC con Windows 95 y a través de la página Web

http://www.ing.ula.ve/~cemisid/SimDMN.html entonces si se desea cargar un archivo fuente DMN que se encuentra en el disco local de la PC referida previamente, el *Security Manager* (el cual es una clase adicional del lenguaje Java implementado solo en los Browsers) del Netscape no permite tal acción. La forma en que se resolvió este problema es usando el protocolo http, pero aún existe una restricción, ya que al usar este protocolo no se puede leer un archivo en una PC, debido a que ese mismo Security Manager impide que se pueda leer un archivo que no se encuentre en el mismo servidor desde donde se está trayendo la aplicación (en nuestro caso www.ing.ula.ve). Para resolver esta restricción entonces se debe copiar el archivo en cuestión al directorio public_html de la cuenta de la persona interesada (dicha cuenta debe estar en www.ing.ula.ve, en el caso ilustrado) darle permisos de lectura global, y luego accesar el archivo usando el protocolo http.

Ejemplo : en este ejemplo se corrió el simulador desde la dirección http://www.cida.ve/~dubuc, se obtiene la siguiente página Web usando Netscape Navigator Gold v3.0 :

Figura 3-1 : Página Web que Ejecuta el Simulador como un Applet.

Al ejecutar la aplicación supóngase que se desea leer un archivo llamado dmnsrc.asm el cual es un archivo en lenguaje ensamblador de la arquitectura DMN. Para poder leerlo se usó la opción Open DMN Source del Sub Menú File. Debe aparecer la siguiente caja de diálogo :

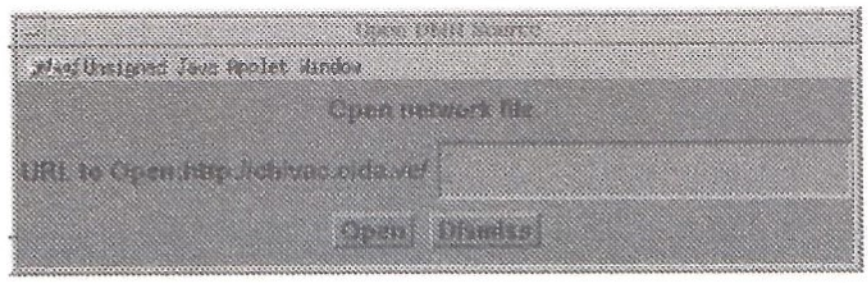

Figura 3-2 : Dialogo de Abrir archivo de Red usando protocolo http.

Como se usa el servidor http del CIDA[6] se puede observar que al abrir el diálogo de archivo aparece la definición previa del URL[7] del servidor donde se encuentra el Applet, en este caso http://www.cida.ve/ y el resto del dialogo espera por una dirección de directorio valido. En este ejemplo supóngase que el usuario es **xuser**, entonces se debe crear un directorio llamado public_html dentro del directorio raíz de la cuenta xuser, y darle permisos de lectura para todo el mundo, posteriormente copiar el archivo dmnsrc.asm en ese directorio. El usuario xuser debe entonces colocar en el espacio libre del diálogo mostrado en la Figura 3-2 lo siguiente para acceder a su archivo dmnsrc.asm :

~xuser/dmnsrc.asm

Y luego presionar el botón Open. Al hacer esto, automáticamente se debe invocar el Ensamblador DMN que contendrá el archivo en cuestión dentro de su ventana de edición.

---

[6] Durante el desarrollo de este programa el autor trabajaba en el CIDA, y por lo tanto hizo uso de sus servicios de Web para probar la funcionalidad del SimDMN.

[7] URL : Uniform Resource Locator, Localizador de Recursos Uniforme, denominación oficial de direcciones en el Web.

Portabilidad del Simulador

En este punto se puede decir que gracias al Lenguaje Java esto es algo totalmente garantizado, la Aplicación se probó sin ningún problema funcional en diversas plataformas Unix/X11, tales como Sun Sparc (modelos 4/110, Sparc 1, Sparc 2, Sparc 5, Sparc 10, Sparc 20, y Ultra 1) y Silicon Graphics Indy. Además de PC usando diversos S.O. como Windows 95, IBM OS/2 Warp y Linux. Así como también se pudo comprobar su satisfactorio funcionamiento en una PowerMac del Laboratorio de Edumática de La Escuela de Ingeniería de Sistemas.

Los únicos detalles que presentó esta aplicación con respecto a portabilidad fue debido al tamaño de los tipos de letras, que en cada ambiente eran distintos. Este detalle se puede obviar totalmente, ya que no afecta en nada la funcionalidad del simulador.

Un último detalle se debió a que este simulador debe ser ejecutado con una resolución gráfica mínima de 800x600 pixels. Este detalle si afecta la funcionalidad gráfica del simulador si se emplea una resolución menor que esa.

En las siguientes figuras se podrán observar las pequeñas diferencias del Simulador corriendo en ambiente PC, con IBM OS/2 Warp y Windows 95, compárese estas figuras con la Figura 2-5 para obtener una relación de ambientes Unix/Windows/Warp.

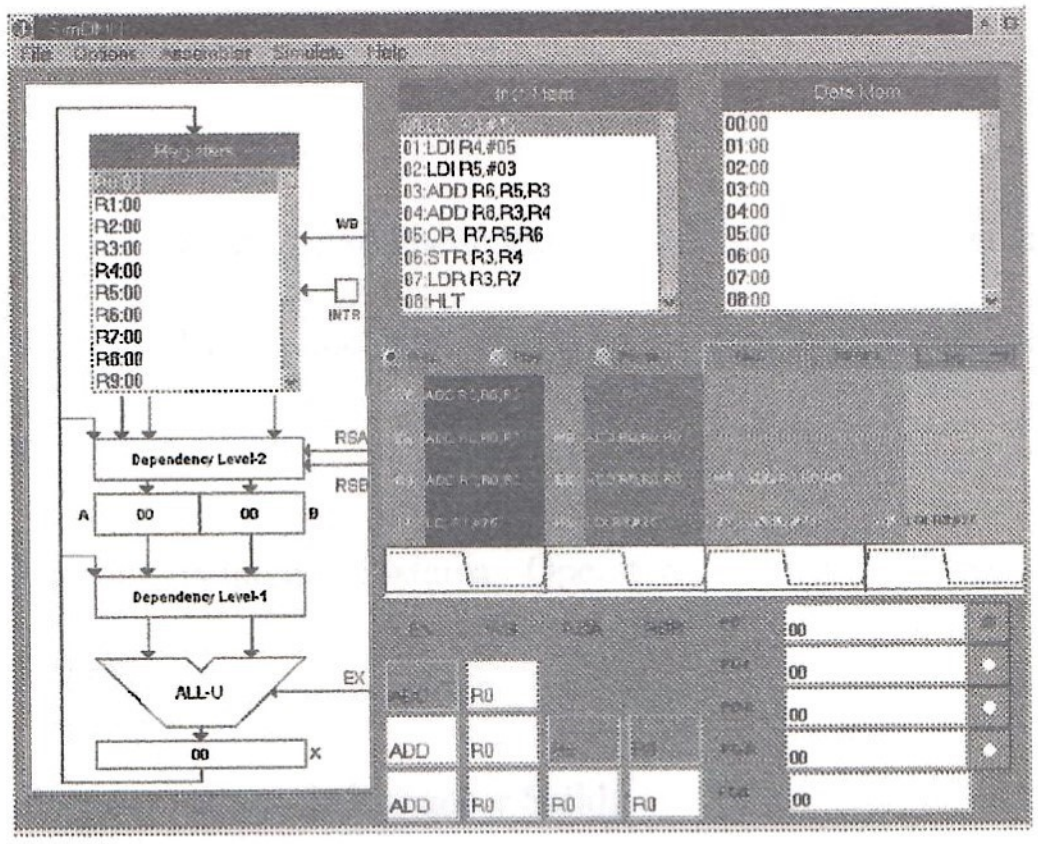

Figura 3-3 : SimDMN corriendo en IBM OS/2 Warp.

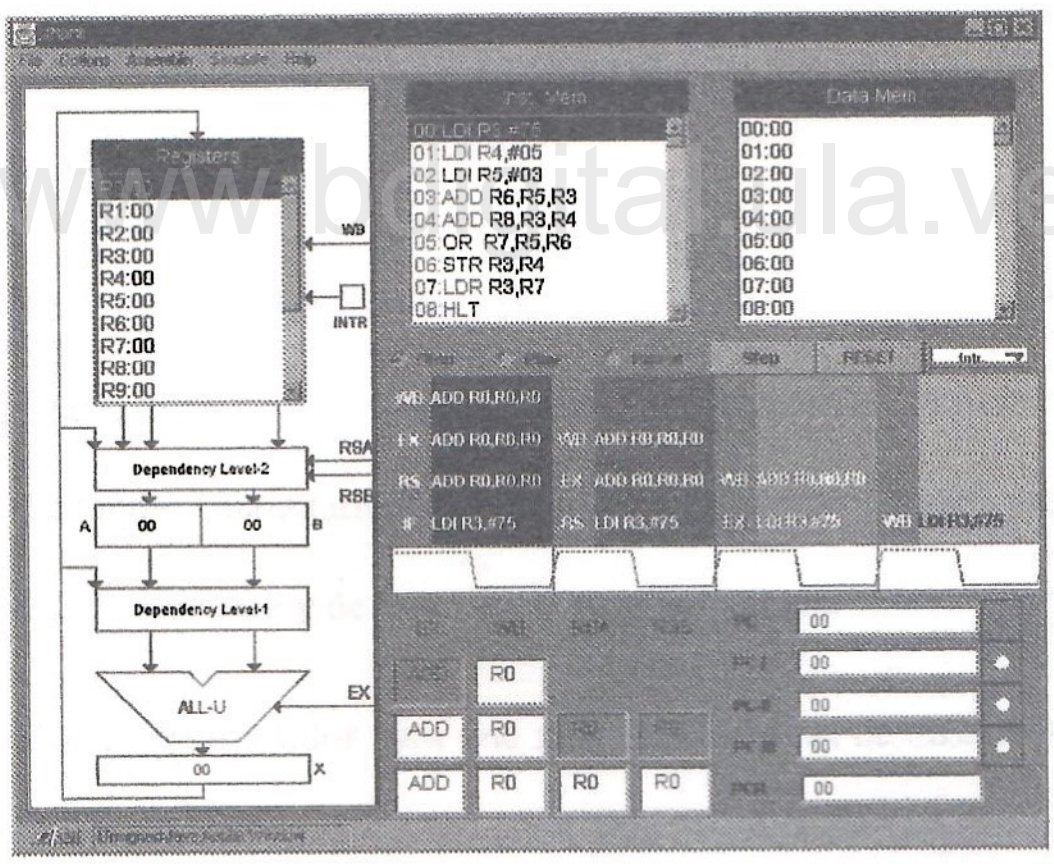

Figura 3-4 : SimDMN corriendo en Windows 95.

Mejoras del Simulador

Debido a la forma en que se encuentra diseñado el simulador, la amplia jerarquía de Clases que soporta el Lenguaje Java, así como su gran flexibilidad, permite un buen numero de mejoras, tales como :

- Diseñar un pequeño Sistema Operativo, idealmente basado en Micro Núcleo[15].
- Simular dispositivos de Entrada y Salida
- Que se puedan guardar archivos de programas desde el Ensamblador. Este punto en la actualidad no es posible debido a las restricciones nombradas en el apartado anterior.
- Diseñar un módulo de Estadísticas para observar el rendimiento del Procesador y aplicarle distintos *benchmarks*. Esto con el propósito de usar la metodología del Enfoque Cuantitativo[12] tanto en el diseño, como para optimar la arquitectura de este y de futuros microprocesadores.
- Ampliar el ensamblador para que soporte directivas de ensamblaje tales como definir espacio en memoria de datos para implementar segmentos de cabecera de programas, pila y espacio de datos. También directivas para la definición de subrutinas y código para las rutinas de servicio de interrupciones.

- Crear una aplicación del tipo cliente/servidor para el correcto manejo de archivos usando Applets, esto con la finalidad de salvar el obstáculo que representa el Security Manager en los Applets.

Como último comentario se puede decir que el proyecto cumplió con todas las expectativas. Además este sistema puede crecer de manera abierta permitiendo una amplia experimentación académica dada su flexibilidad.

Además de esto el lenguaje de programación que se usó, Java, está en continuo crecimiento, y adicionalmente ya existen Sistemas Operativos que soportan los bytecodes de Java sin usar un Browser o un interprete (de manera explícita). Dichos Sistemas actualmente son IBM OS/2 Warp versión 4, y Linux con kernel 2.0 o superior. Se espera que en un futuro muy próximo muchos S.O. soporten este formato binario, y además Sun Microsystems está desarrollando un microprocesador llamado Java Chip [17].

Bibliografía

1. Aho, Alfred V. et al. *Compiladores, Principios, técnicas y herramientas.* Adison-Wesley, 1990.

2. Baron, Robert y Higbie, L. *Computer Architecture.* Adison-Wesley, 1992.

3. Booch, Grady. *Object Oriented Analysis and Design.* Bejamin/Cummings, 1994.

4. Boone, Barry. *Java Essentials for C and C++ Programmers.* Adison-Wesley, 1996.

5. Campione, Mary y Walrath, Kathy.*The Java Tutorial : Object Oriented Programming for the Internet,* Adison-Wesley, 1996.

6. Dasgupta, Subrata. *Computer Architecture, a modern synthesis.* John Wiley & Sons, 1989.

7. Day, Robert A. *How to Write and Publish a Scientific Paper.* Cambridge University Press, 1988.

8. Goldberg, A y Robson, D. *Smalltalk-80 : The Language and Its Implementation.* Adison-Wesley-1983.

9. Gosling, James et. al. *The Java ApplicationProgramming Interface, V. I and II.* Adison-Wesley, 1996.

10. Páez Monzón, Gerard. *Arquitectura y Organización del Computador*. Editorial Senda Sol, 1993.

11. Páez, G. y Páez, Ch. *The RISC Processor DMN-6 : A Unified Data-Control Flow Architecture*. Computer Architecture News, Sept. 1996.

12. Patterson, D. y Hennessy, J. *Computer Architecture, a Quantitative Approach.* Morgan Kauffmann Publ. Inc. 1995. II Edición.

13. Schildt, Herbert. *Turbo C++ manual de referencia.* Mc Graw Hill, 1992.

14. Sommerville, Ian. *Ingeniería de Software segunda edición.* Adison-Wesley, 1988.

15. Tenenbaum, Andrew S. *Sistemas Operativos Modernos*. Prentice Hall, 1992.

16. Wakerly, John F. *Diseño Digital Principios y Prácticas*. Prentice Hall, 1992.

17. Wayner, Peter. *Sun Gambles on Java Chips*. Byte Magazine, Vol 21 No. 11, Noviembre 1996, pp 79-88.

Apéndices

Apéndice A

Palabras Claves Básicas de Java.

Este apéndice cubre las palabras claves básicas del lenguaje de programación Java, es de hacer notar que algunas de estas palabras claves son iguales a las de C, y C++.

Organización de Clases

- **package** especifica el paquete de rutinas al que pertenecen las clases en un archivo fuente.

- **import** hace que las clases especificadas sean importadas en la aplicación.

Definición de Clases

- **interface** define un grupo de métodos y datos globales que pueden ser compartidos entre clases.

- **class** define una colección de comportamientos y datos relacionados.

- **extends** indica cual clase servirá de pariente (o padre) de la clase que se está definiendo.

- **implements** indica las interfaces para la cual una nueva clase debe suplir sus métodos.

Palabras Claves para Clases y Variables

- **abstract** especifica que la clase no puede ser instanciada directamente
- **public** significa que la clase, método o variable puede ser accedida de cualquier parte.
- **private** significa que solo la clase que define el método o la variable puede accederla
- **protected** significa que la clase que define el método y todas sus subclases pueden acceder al método o variable.
- **static** especifica un método o variable de clase, es decir un método o variable que siempre tiene el mismo valor en distintas instancias de una clase.
- **synchonized** indica que solo un objeto o clase puede acceder esta variable o método a la vez.

- **volatile** le dice al compilador que esta variable puede cambiar asincrónicamente debido a las trazas de ejecución.
- **final** significa que esta variable o método no puede ser cambiado por otra subclase.
- **native** enlaza un método a código nativo.

Tipos de Datos Simples

- **long** es un valor entero de 64 bits.
- **int** es un valor entero de 32 bits.
- **short** es un valor entero de 16 bits.
- **byte** es un valor entero de 8 bits.
- **double** es un valor flotante de 64 bits.
- **float** es un valor flotante de 32 bits.
- **char** es un carácter de en formato Unicode de 16 bits.
- **boolean** es un valor verdadero o falso.
- **void** indica un método el cual no retorna ningún valor.

Valores y variables

- **false** es un valor Lógico, indica falso.
- **true** es un valor Lógico, indica verdadero.
- **this** se refiere a la instancia actual en un método de instancia (es decir un método que no es estático).
- **super** se refiere a la inmediata superclase en un método de instancia.
- **null** representa una instancia no existente.

Manejo de Excepciones

- **throw** crea una excepción
- **try** marca la pila de ejecución, de tal manera que si se crea una excepción se regresará hasta este punto.
- **catch** captura una excepción.
- **finally** hace la ejecución de este bloque de código independientemente del flujo de manejo de errores.

Creación y Pruebas de Instancias

- **new** crea nuevas instancias de una clase.
- **instanceof** prueba si una instancia pertenece, o deriva de una clase en particular.

Flujo de Control

- **switch** prueba una variable.
- **case** ejecuta un bloque particular de instrucciones de acuerdo con el valor probado en la palabra clave **switch**.
- **default** significa que se ejecuta este bloque de código por omisión si ningún valor en **case** fue encontrado.
- **break** rompe con el flujo de ejecución dentro de un bloque de código.

- **continue** continúa con la siguiente iteración de un lazo.
- **return** retorna de un método, y adicionalmente regresa un valor.
- **do** ejecuta una línea de código o un bloque de código.
- **if** prueba una condición y realiza alguna acción si el resultado es cierto.
- **else** realiza una acción si la prueba anterior da como resultado falso.
- **for** significa iteración.
- **while** ejecuta alguna acción mientras el resultado sea cierto.

Apéndice B

Descripción del Ambiente de Programación JDK bajo Sistema Operativo Linux.

Para este proyecto se usó el JDK (Java Developers Kit) en su versión 1.0.1 bajo el Sistema Operativo Linux, kernel versión 1.3.13, y como ambiente gráfico se usó X11R6 con FVWM como administrador de ventanas. La plataforma de Hardware usada fue un PC 486 DX/4 100Mhz con 24 MB de RAM y 1.5 GB en Disco Duro. Es de hacer notar que el ambiente de programación típico de toda plataforma Unix es principalmente la línea de comandos del Sistema Operativo y editores tales como el vi o el emacs.

El JDK se consigue para varias plataformas. Actualmente su sitio ftp oficial es ftp.javasoft.com y se encuentra bajo el directorio /pub. En este sitio se encuentra el JDK para SunOS, Windows 95/NT, y muy recientemente para MacOS. Este paquete de Software es gratis para aplicaciones del tipo académico o no comerciales.

Para Linux el JDK se encuentra actualmente en ftp.webconn.com, bajo el directorio /pub/java/blackdown. Se deben bajar dos archivos :

- linux.jdk-1.0.1-try1.common.tar.gz
- linux.jdk-1.0.1.-try3.static.motif.tar.gz

Se recomienda instalarlo bajo el directorio /usr/local.

El JDK se compone de lo siguiente :

- El compilador de línea de comandos : javac ;
- Un interprete de clases : java ;
- Un visor de Applets : appletviewer ;
- Un generador de documentación : javadoc ;
- Un depurador de línea de comandos : jdb ;
- Muchos Applets de ejemplo.
- Un archivo comprimido llamado src.zip que contiene todos los fuentes de las la librería de clases de Java. Al descomprimir este archivo se ocupan como 8MB de espacio en disco.

Para compilar un programa en Java se debe hacer el siguiente comando :

```
javac ClaseACompilar.java
```

Luego si la aplicación es del tipo "Stand-Alone" (es decir, contiene el método main), esta se puede ejecutar con el comando :

```
java ClaseACompilar
```

Sin embargo, si la aplicación es un Applet (que debe correr usando un Browser tal como Netscape versión 2.0 o superior), se debe usar el siguiente comando :

appletviewer ClaseACompilar.html

Es de hacer notar que cuando se tiene un Applet, se debe hacer un archivo en formato html, que contenga la etiqueta <APPLET, para poder ejecutar el Applet. Tomese como ejemplo el siguiente código en html para ejecutar un Applet :

```
<applet CODE="HelloWorld.class"  WIDTH=150 HEIGHT=25>
```

Para coordinar la compilación de los 32 archivos que componen este proyecto se usó la ayuda de una herramienta muy común en los sistemas Unix, como es el comando make. Este comando requiere la ayuda de un archivo llamado **Makefile** el cual le indica al comando make que debe compilar y como, es decir, en que orden. Usando esta herramienta fue muy sencillo compilar los 32 archivos, ubicados en 6 distintos directorios, de una sola invocación.

Para mayor información sobre el comando make y los Makefiles puede referirse a las páginas de manual en línea de todo sistema Unix, es decir el comando man.

Apéndice C

Listados de las Clases que Conforman el Simulador.

Clases Iniciales

Estas Clase son las que se usan para poner a funcionar el ambiente del simulador, de manera "Stand-Alone" o a través del Web como un Applet de Java.

Clase DMNApplet

```
//$Id: DMNApplet.java,v 1.7 1996/09/25 00:44:03 dubuc Exp dubuc $

/*
$Log: DMNApplet.java,v $
Revision 1.7  1996/09/25 00:44:03  dubuc
Nueva Revision 1.7.x para unificar todo el codigo

*/

import GUI.*;

import java.awt.Button;
import java.awt.Event;
import java.awt.Font;
import java.awt.BorderLayout;
import java.applet.Applet;
import java.net.URL;

public class DMNApplet extends Applet {
        DMNFrame dmnSim;

        URL urlBase;

        static String STARTSIM = "Start SimDMN";

        public void init() {
                setLayout(new BorderLayout(2,2));
                resize(200,50);
                Button but = new Button(STARTSIM);
                but.setFont(new Font("Helvetica", Font.BOLD, 14));
                add("Center",but);
                urlBase = this.getDocumentBase();
        }

        public boolean handleEvent(Event e) {
                if (STARTSIM.equals(e.arg)) {
                        dmnSim = new DMNFrame();
                        dmnSim.inAnApplet = true;
                        dmnSim.urlBase = urlBase;
                        dmnSim.init();
                        dmnSim.pack();
                        dmnSim.show();
                }
                return super.handleEvent(e);
        }
}
```

Clase SimDMN

```
//$Id: SimDMN.java,v 1.5 1996/11/13 13:42:03 dubuc Exp $

/*
$Log: SimDMN.java,v $
Revision 1.5  1996/11/13 13:42:03  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

import GUI.*;

public class SimDMN {

        public static void main(String [] args) {
                DMNFrame f = new DMNFrame();
                f.init();
                f.pack(); // Esta instruccion empaqueta los componentes visuales
                f.show();
                f.repaint();
        }

}
```

Clases del Paquete binContainer

Este conjunto de Clases representa el manejo básico de datos a través del simulador.

Clase Word

```
//$Id: Word.java,v 1.5 1996/11/13 13:42:03 dubuc Exp $

/*
$log$
*/

package binContainer;

import java.util.BitSet;

/**
 * Clase Basica del paquete binContainer, implementa los metodos de la
 * clase abstracta java.lang.Number el resto de las clases en este
 * paquete dependen altamente de esta clase.
 * @version   1.0, Oct 1996
 * @author    Alberto E. Dubuc B.
 */

public class Word extends Number implements Cloneable {
    /**
                * El conjunto de bits que conforma el contenedor binario.
                */
               protected BitSet bits;

    /**
                * La cantridad de bits que conforma el contenedor binario.
                */
               protected int nbits;

    /**
     * Construye un objeto Word de 16 bits inicializado a cero.
                               */
               public Word() {
                               nbits = 16;
                               bits = new BitSet(nbits);
               }

    /**
     * Construye un objeto Word de 16 bits inicializado con el valor
                               * int especificado por val.
                               * @param     val               el valor inicial del objeto Word.
                               */
               public Word(int val) {
                               nbits = 16;
                               bits = new BitSet(nbits);
                               this.setValue(val);
               }

    /**
     * Construye un objeto Word de 16 bits inicializado con el valor
                               * long especificado por val.
                               * @param     val               el valor inicial del objeto Word.
                               */
               public Word(long val) {
                               nbits = 16;
                               bits = new BitSet(nbits);
                               this.setValue(val);
               }

    /**
     * Construye un objeto Word de nb bits inicializado con el valor
                               * int especificado por val.
                               * @param     val               el valor inicial del objeto Word.
                               * @param     nb                          la cantidad de bits habilitados al objeto Word.
```

```
                */
        protected Word(int val,int nb) {
                nbits = nb;
                bits = new BitSet(nbits);
                this.setValue(val);
        }

/**
 * Construye un objeto Word de nb bits inicializado con el valor
                * long especificado por val.
                * @param   val             el valor inicial del objeto Word.
                * @param   nb                      la cantidad de bits habilitados al objeto Word.
                */
        protected Word(long val,int nb) {
                nbits = nb;
                bits = new BitSet(nbits);
                this.setValue(val);
        }

/**
  * Convierte el contenido del objeto Word en un objeto String cuya
  * representacion es binaria (base 2).
  */
        public String toBinaryString() {
                String str = "";

                for(int i = 0; i < nbits; i++)
                        if (bits.get(i))
                                str = "1" + str;
                        else
                                str = "0" + str;
                return str;
        }

/**
  * Habilita un bit especifico del objeto Word.
  * @param bit                      la pocision especifica del bit (desde 0 hasta nbits - 1)
  */
        public void setBit(int bit){
                if(bit < nbits)
                        bits.set(bit);
        }

/**
  * Retorna el valor de este objeto Word como un double.
  */
        public double doubleValue() {
                double res = 0;
                for(int i = 0; i < nbits; i++)
                        if(bits.get(i))
                                res += Math.pow(2.0,(double)i);
                return res;
        }

/**
  * Retorna el valor de este objeto Word como un float.
  */
        public float floatValue() {
                return (float)this.doubleValue();
        }

/**
  * Retorna el valor de este objeto Word como un int.
  */
        public int intValue() {
                return (int)this.doubleValue();
        }

/**
  * Retorna el valor de este objeto Word como un long.
  */
        public long longValue() {
                return (long)this.doubleValue();
        }

/**
  * Retorna el valor de este objeto Word como un objeto String con
  * representacion hexadecimal (base 16).
  */
        public String toHexString() {
                StringBuffer sb = new StringBuffer();
                sb.append(Long.toString(this.longValue(),16));
                int nnib = (nbits / 4) - sb.length();
                for(int i = 0; i < nnib; i++) {
                        sb.insert(0,"0");
                }
                return sb.toString().toUpperCase();
        }
```

```
/**
 * Retorna el valor de este objeto Word como un objeto String con
 * representacion decimal (base 10).
 */
public String toString() {
        return (new Long(this.longValue())).toString();
}

/**
 * Pone todos los bits a cero, limpiando el Word.
 */
public void clearValue() {
        for(int i = 0; i < nbits; i++)
                bits.clear(i);
}

protected void setValue2Complement(int value){
        int nsig = Math.abs(value);
        if(nsig < (int)Math.pow(2.0,(double)(nbits - 1))) {
                this.setValue(nsig);
                if(value < 0){
                        for(int i = 0; i < nbits; i++)
                                if(bits.get(i))
                                        bits.clear(i);
                                else
                                        bits.set(i);
                        nsig = this.intValue() + 1;
                        this.setValue(nsig);
                }
        }
        else this.clearValue();
}

/**
 * Coloca el valor long especificado por value en el objeto Word.
 * @param  value                valor al cual se le asignara el objeto.
 */
public void setValue(long value) {
        this.clearValue();
        if((value > 0) && (value < (long)Math.pow(2.0,(double)nbits))) {
                long tempv = value;
                int i = 0;
                while(tempv != 1) {
                        if((tempv % 2) == 1)
                                bits.set(i);
                        tempv /= 2;
                        i++;
                }
                bits.set(i);
        }
}

public int intValue2Complement(){
        double res = 0;
        for(int i = 0; i < nbits - 1; i++)
                if(bits.get(nbits - 1)){
                        if(!bits.get(i))
                                res += Math.pow(2.0,(double)i);
                } else
                        if(bits.get(i))
                                res += Math.pow(2.0,(double)i);
        if(bits.get(nbits - 1)) {
                res++;
                res = -res;
        }
        return (int) res;
}

public Word add2Complement(Word sum) {
        int s1 = this.intValue2Complement();
        int s2 = sum.intValue2Complement();
        Word w = new Word();
        w.setValue2Complement(s1 + s2);
        return w;
}

public Word sub2Complement(Word sus) {
        int s1 = this.intValue2Complement();
        int s2 = sus.intValue2Complement();
        Word w = new Word();
        w.setValue2Complement(s1 - s2);
        return w;
}

public boolean lt2Complement(Word cmp) {
        int s1 = this.intValue2Complement();
        int s2 = cmp.intValue2Complement();
```

```
                return (s1 < s2);
        }

    /**
     * Coloca el valor int especificado por value en el objeto Word.
     * @param  value                    valor al cual se le asignara el objeto.
     */
    public void setValue(int value) {
                this.setValue((long)value);
    }

    /**
     * Convierte una instancia de una subclase de la clase Word a una instancia
     * de la clase Word.
     */
    public Word toWord() {
                Word w = (Word) this.clone();
                w.nbits = 16;
                return w;
    }

    /**
     * Hace una copia del objeto Word.
     */
    public Object clone() {
                try {
                        Word w = (Word)super.clone();
                        w.nbits = nbits;
                        w.bits = (BitSet)bits.clone();
                        return w;
                } catch (CloneNotSupportedException e) {
                        throw new InternalError();
                }
        }
}
```

Clase Byte

```
//$Id: Byte.java,v 1.5 1996/11/13 14:20:39 dubuc Exp $

/*
$Log: Byte.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

import java.util.BitSet;

public class Byte extends Word {
        public Byte(){
                super(0,8);
        }

        public Byte(int n){
                super(n,8);
        }

        public Byte(long l){
                super(l,8);
        }

        public Nibble toNibble() {
                Nibble n = new Nibble();
                for(int i = 0; i < n.nbits; i++)
                        if(this.bits.get(i))
                                n.bits.set(i);
                return n;
        }

        public Byte add2Complement(Byte sum) {
                int s1 = this.intValue2Complement();
                int s2 = sum.intValue2Complement();
                Byte w = new Byte();
                w.setValue2Complement(s1 + s2);
                return w;
        }

        public Byte sub2Complement(Byte sus) {
                int s1 = this.intValue2Complement();
                int s2 = sus.intValue2Complement();
                Byte w = new Byte();
                w.setValue2Complement(s1 - s2);
                return w;
```

```
		}
}
```

Clase Nibble

```
//$Id: Nibble.java,v 1.5 1996/11/13 14:20:39 dubuc Exp $

/*
$Log: Nibble.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

import java.util.BitSet;

public class Nibble extends Word {

		public Nibble() {
				super(0,4);
		}

		public Nibble(int n) {
				super(n,4);
		}

		public Nibble(long n) {
				super(n,4);
		}

		public Byte toByte() {
				Byte b = new Byte();
				b.nbits = 8;
				b.bits = (BitSet)this.bits.clone();
				return b;
		}

		public Nibble add2Complement(Nibble sum) {
				int s1 = this.intValue2Complement();
				int s2 = sum.intValue2Complement();
				Nibble w = new Nibble();
				w.setValue2Complement(s1 + s2);
				return w;
		}

		public Nibble sub2Complement(Nibble sus) {
				int s1 = this.intValue2Complement();
				int s2 = sus.intValue2Complement();
				Nibble w = new Nibble();
				w.setValue2Complement(s1 - s2);
				return w;
		}
}
```

Clase BinConvert

```
//$Id: BinConvert.java,v 1.5 1996/11/13 14:20:58 dubuc Exp $

/*
$Log: BinConvert.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

public class BinConvert {

		public static Byte wordToUpperByte(Word w) {
				Byte b = new Byte();
				for(int i = 0; i < 8; i++)
						if(w.bits.get(i + 8))
								b.bits.set(i);
				return b;
		}

		public static Byte wordToLowerByte(Word w) {
				Byte b = new Byte();
				for(int i = 0; i < 8; i++)
						if(w.bits.get(i))
								b.bits.set(i);
```

```
                return b;
        }

        public static Nibble byteToUpperNibble(Byte b) {
                Nibble n = new Nibble();
                for(int i = 0; i < 4; i++)
                        if(b.bits.get(i + 4))
                                n.bits.set(i);
                return n;
        }

        public static Nibble byteToLowerNibble(Byte b) {
                Nibble n = new Nibble();
                for(int i = 0; i < 4; i++)
                        if(b.bits.get(i))
                                n.bits.set(i);
                return n;
        }
}
```

Clase BinQueue

```
//$Id: BinQueue.java,v 1.5 1996/11/13 14:20:14 dubuc Exp $

/*
$Log: BinQueue.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

public abstract class BinQueue {

        protected int size;
        protected Word [] queue;

        public void pushQueue(Word w) {
                for(int i = 0; i < (size - 1); i++)
                        queue[size - 1 - i].setValue(queue[size - 2 - i].intValue());
                queue[0].setValue(w.intValue());
        }

        public void clearQueue() {
                for(int i = 0; i < size; i++)
                        queue[i].clearValue();
        }

}
```


Clase WordQueue

```
//$Id: WordQueue.java,v 1.5 1996/11/13 14:20:00 dubuc Exp $

/*
$Log: WordQueue.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

public class WordQueue extends BinQueue {

        public WordQueue(int s) {
                size = s;
                queue = new Word[size];
                for(int i = 0; i < size; i++)
                        queue[i] = new Word();
        }

        public Word seeTopQueue() {
                return queue[size - 1];
        }

        public Word seeElement(int pos) {
                if ((pos >= 0) && (pos < size))
                        return queue[pos];
                else
                        return queue[0];
        }
```

```
}
```

Clase ByteQueue

```
//$Id: ByteQueue.java,v 1.5 1996/11/13 14:20:45 dubuc Exp $

/*
$Log: ByteQueue.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

public class ByteQueue extends BinQueue {

        public ByteQueue(int s) {
                size = s;
                queue = new Byte[size];
                for(int i = 0; i < size; i++)
                        queue[i] = new Byte();
        }

        public Byte seeTopQueue() {
                return (Byte) queue[size - 1];
        }

        public Byte seeElement(int pos) {
                if ((pos >= 0) && (pos < size))
                        return (Byte) queue[pos];
                else
                        return (Byte) queue[0];
        }

}
```

Clase NibbleQueue

```
//$Id: ByteQueue.java,v 1.5 1996/11/13 14:21:03 dubuc Exp $

/*
$Log: NibbleQueue.java,v $
Revision 1.5  1996/11/13 14:20:39  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package binContainer;

public class NibbleQueue extends BinQueue {

        public NibbleQueue(int s) {
                size = s;
                queue = new Nibble[size];
                for(int i = 0; i < size; i++)
                        queue[i] = new Nibble();
        }

        public Nibble seeTopQueue() {
                return (Nibble) queue[size - 1];
        }

        public Nibble seeElement(int pos) {
                if ((pos >= 0) && (pos < size))
                        return (Nibble) queue[pos];
                else
                        return (Nibble) queue[0];
        }

}
```

Clases del Paquete memory

Clases para el Manejo de Memoria del Simulador DMN.

Clase Memory

```
//$Id: Memory.java,v 1.5 1996/11/13 14:09:19 dubuc Exp $

/*
$Log: Memory.java,v $
Revision 1.5  1996/11/13 14:09:19  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package memory;

import binContainer.*;

public abstract class Memory {

        protected Word [] locs,
        protected int accessPointer, capacity;

        public Word readMemory() {
                return locs[accessPointer];
        }

        public Word readMemory(int loc) {
                if(loc < capacity) {
                        accessPointer = loc;
                        return locs[accessPointer];
                }
                else return locs[capacity - 1];
        }

        public void writeMemory(Word w) {
                locs[accessPointer] = (Word)w.clone();
        }
        public void writeMemory(Word w, int loc) {
                if(loc < capacity) {
                        accessPointer = loc;
                        locs[accessPointer] = (Word)w.clone();
                }
        }

        public void setAccessPointer(int loc) {
                if(loc < capacity)
                        accessPointer = loc;
        }

        public int getAccessPointer() {
                return accessPointer;
        }

        public int getCapMemory() {
                return capacity;
        }
}
```

Clase RegFileMemory

```
//$Id: RegFileMemory.java,v 1.5 1996/11/13 14:09:19 dubuc Exp $

/*
$Log: RegFileMemory.java,v $
Revision 1.5  1996/11/13 14:09:19  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package memory;
```

```
import binContainer.*;

public class RegFileMemory extends Memory {

        public RegFileMemory(){
                capacity = 16;
                locs = new Byte[capacity];
                accessPointer = 0;
                for(int i = 0; i < capacity; i++)
                        locs[i] = new Byte();
        }

        public Word readMemory() {
                locs[0].clearValue();
                return super.readMemory();
        }

        public Word readMemory(int loc) {
                locs[0].clearValue();
                return super.readMemory(loc);
        }

        public void writeMemory(Word w) {
                if(accessPointer != 0)
                        super.writeMemory(w);
        }

        public void writeMemory(Word w, int loc) {
                if(loc != 0)
                        super.writeMemory(w,loc);
        }

        public void clearMemory() {
                for(int i = 0; i < capacity; i ++)
                        locs[i].clearValue();
        }

}
```

Clase InstMemory

```
//$Id: InstMemory.java,v 1.5 1996/11/13 14:09:19 dubuc Exp $

/*
$Log: InstMemory.java,v $
Revision 1.5  1996/11/13 14:09:19  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package memory;

import binContainer.*;

public class InstMemory extends Memory {

        int [] inst = {0xd375,0xd405,0xd503,0x0653,0x0834,0x3756,0xf034,0xe370,0xcd00};
        public int nInst = 9;

        public InstMemory() {
                capacity = 256;
                locs = new Word[capacity];
                accessPointer = 0;
                for(int i = 0; i < capacity; i++)
                        locs[i] = new Word();
        }

        public void initialize() {
                for(int i = 0; i < nInst; i++)
                        locs[i].setValue(inst[i]);
        }

}
```

Clase DataMemory

```
//$Id: DataMemory.java,v 1.5 1996/11/13 14:09:19 dubuc Exp $

/*
$Log
*/

package memory;

import binContainer.*;
```

```
public class DataMemory extends Memory {

        public DataMemory(){
                capacity = 256;
                locs = new Byte[capacity];
                accessPointer = 0;
                for(int i = 0; i < capacity; i++)
                        locs[i] = new Byte();
        }

}
```

Clases del Paquete Simul

Estas Clases forman la integración del Sistema de Simulación.

Clase Simulator

```
//$Id: Simulator.java,v 1.6 1996/11/13 14:07:26 dubuc Exp $

/*
$Log: Simulator.java,v $
Revision 1.6  1996/11/13 14:07:26  dubuc
Peque~no Cambio

Revision 1.5  1996/11/13 13:50:27  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

Revision 1.1  1996/10/30 00:45:14  dubuc
Initial revision

*/

package Simul;

import memory.*;
import binContainer.*;

public class Simulator {

        public static int PCREG = 15;
        public static int PSREG = 14;
        public static int HAREG = 13;
        public static int CNDBT = 12;

        public String [] nmeMem;

        Word instruction;
        Byte lInst, hInst;
        public Byte xReg;
        public Byte pcReg;
        public NibbleQueue wbQueue, exQueue;
        public Nibble wb4;
        public RegFileMemory rFile;
        public InstMemory iMemory;
        public DataMemory dMemory;
        public ALLU alu;
        public DependResolver depUnit;
        public ByteQueue pcQueue;
        public boolean [] pcState = new boolean[4];
        public boolean wmem;
        public int mempos;
        private ByteQueue rIdi;

        public Simulator() {
                instruction = new Word();
                lInst = new Byte();
                hInst = new Byte();
                wbQueue = new NibbleQueue(3);
                exQueue = new NibbleQueue(2);
```

```
                wb4 = new Nibble();
                pcQueue = new ByteQueue(3);
                rFile = new RegFileMemory();
                iMemory = new InstMemory();
                alu = new ALU();
                depUnit = new DependResolver();
                xReg = alu.xReg;
                dMemory = alu.dMemory;
                rldi = alu.rldi;
                pcReg = new Byte();
                nmeMem = new String[iMemory.getCapMemory()];
                pcState[0] = true;
                wmem = false;
}

public synchronized void intrAck(){
                boolean flag = true;
                alu.intr = true;
                int i = 4;
                do{
                                i--;
                                if(pcState[i]) {
                                                alu.PCR.setValue(pcQueue.seeElement(i-1).intValue());
                                                flag = false;
                                }
                }while((i > 0) && (!pcState[i]));
                if (flag)
                                alu.PCR.setValue(pcReg.intValue());
}

public synchronized void reset() {
                lInst.clearValue();
                hInst.clearValue();
                pcQueue.clearQueue();
                pcReg.clearValue();
                rFile.clearMemory();
                flushQueues();
                alu.reset();
                wb4.clearValue();
                wmem = false;
}

public void nmeConvMem() {
                NMETransf.convMemory(iMemory,nmeMem);
}

private void fetchInstruction() {
                instruction = iMemory.readMemory(pcReg.intValue());

                hInst = BinConvert.wordToUpperByte(instruction);
                lInst = BinConvert.wordToLowerByte(instruction);
}

private void pushQueues() {
                exQueue.pushQueue(BinConvert.byteToUpperNibble(hInst));
                wb4.setValue(wbQueue.seeTopQueue().intValue());
                wbQueue.pushQueue(BinConvert.byteToLowerNibble(hInst));
}

private void writeBack() {
                int topWb = wbQueue.seeTopQueue().intValue();
                if(topWb != 15)
                                rFile.writeMemory(xReg,topWb);

                pcQueue.pushQueue(pcReg);
                pcReg.setValue(rFile.readMemory(PCREG).intValue());

                for(int i = 3; i > 0; i--)
                                pcState[i] = pcState[i-1];
                pcState[0] = true;
                rFile.writeMemory(pcReg,PCREG);
}

public synchronized void step() {
                if(rFile.readMemory(HAREG).intValue() != 255) {
                                mempos = alu.mempos;
                                wmem = alu.wmem;
                                this.fetchInstruction();
                                this.pushQueues();
                                this.depUnit.resolve(rFile,alu.xQueue.seeElement(0),rldi,wbQueue,lInst);
                                this.alu.exMux(depUnit.aReg[1],depUnit.bReg[1],rFile,exQueue.seeTopQueue(),depUnit.sla,depUnit.slb);
                                this.writeBack();
                                if(alu.flush)
                                                flushQueues();
                }
}
```

```
        private void flushQueues() {
                wbQueue.clearQueue();
                exQueue.clearQueue();
                depUnit.reset();
                resetPCState();
        }

        private void resetPCState() {
                for(int i = 1; i < 4; i++)
                        pcState[i] = false;
        }
}
```

Clase ALLU

```
//$Id: ALLU.java,v 1.5 1996/11/13 13:50:27 dubuc Exp $

/*
$Log: ALLU.java,v $
Revision 1.5  1996/11/13 13:50:27  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package Simul;

import memory.*;
import binContainer.*;

public class ALLU {

        Byte
                rA, rB;

        RegFileMemory
                rFile;

        public Byte
                xReg, PCR;

        public ByteQueue
                rIdi, xQueue;

        Nibble
                exSelector, wbQ3,
                selectA, selectB,

public Nibble
        wbQ3p;

public DataMemory dMemory;

public boolean intr, flush, wmem;
public int mempos;

ALLU() {
        dMemory = new DataMemory();
        xReg = new Byte();
        PCR = new Byte();
        rIdi = new ByteQueue(2);
        xQueue = new ByteQueue(2);
        wmem = intr = flush = false;
        mempos = 0;
        wbQ3 = new Nibble();
        wbQ3p = new Nibble();
}

public void reset() {
        wbQ3.clearValue();
        wbQ3p.clearValue();
        PCR.clearValue();
        xReg.clearValue();
        rIdi.clearQueue();
        xQueue.clearQueue();
}

public void setParameters(
        Byte ra,
        Byte rb,
        RegFileMemory rf,
        Nibble exs,
        Nibble sa,
        Nibble sb)
{
        rA = ra;
        rB = rb;
```

```
                rFile = rf;
                exSelector = exs;
                selectA = sa;
                selectB = sb;
}
public void exMux(
                Byte ra,
                Byte rb,
                RegFileMemory rf,
                Nibble exs,
                Nibble sa,
                Nibble sb)
{
                this.setParameters(ra,rb,rf,exs,sa,sb);
                this.exMux();
}

public void exMux() {
                if((rA != null)&&(rB != null)&&(rFile != null)&&(exSelector != null)&&(selectA != null)&&(selectB != null)) {
                                wmem = flush = false;
                                rldi.pushQueue(new Byte(selectA.intValue()*16 + selectB.intValue()));
                                int ex = exSelector.intValue();
                                switch (ex) {
                                case 0x00 : { // ADD
                                                xReg.setValue((rA.add2Complement(rB)).intValue());
                                }
                                break;
                                case 0x01 : { // SUB
                                                xReg.setValue((rA.sub2Complement(rB)).intValue());
                                }
                                break;
                                case 0x02 : { // AND
                                                xReg.setValue(rA.intValue() & rB.intValue());
                                }
                                break;
                                case 0x03 : { // OR
                                                xReg.setValue(rA.intValue() | rB.intValue());
                                }
                                break;
                                case 0x04 : { // NOT
                                                xReg.setValue(Math.abs(~rA.intValue()));
                                }
                                break;
                                case 0x05 : { // SHT
                                int tmp = rA.intValue();
                                int desp = rA.intValue2Complement();
                                if (desp < 0)
                                                xReg.setValue(tmp << Math.abs(desp));
                                else
                                                xReg.setValue(tmp >> Math.abs(desp));
                                xReg.setValue(rA.intValue() << rB.intValue());
                                }
                                break;
                                case 0x06 : { // <Reservado>
                                                xReg.setValue(0);
                                }
                                break;
                                case 0x07 : { // <Reservado>
                                                xReg.setValue(0);
                                }
                                break;
                                case 0x08 : { // ZTST
                                                if(rA.intValue() == rB.intValue())
                                                                rFile.writeMemory(new Byte(255),Simulator.CNDBT);
                                                else
                                                                rFile.writeMemory(new Byte(0),Simulator.CNDBT);
                                                xReg.setValue(0);
                                }
                                break;
                                case 0x09 : { // NTST
                                                if(rA.lt2Complement(rB))
                                                                rFile.writeMemory(new Byte(255),Simulator.CNDBT);
                                                else
                                                                rFile.writeMemory(new Byte(0),Simulator.CNDBT);
                                                xReg.setValue(0);
                                }
                                break;
                                case 0x0a : { // JCND
                                                if(rFile.readMemory(Simulator.CNDBT).intValue() != 0)
                                                                xReg.setValue((rldi.seeTopQueue()).intValue());
                                }
                                break;
                                case 0x0b : { // RTI
                                                xReg.setValue(PCR.intValue());
                                                PCR.clearValue();
                                }
                                break;
                                case 0x0c : { // HLT
```

```
                                        xReg.setValue(255);
                                }
                                break;
                                case 0x0d : { // LDI
                                        xReg.setValue((rIdi.seeTopQueue()).intValue());
                                }
                                break;
                                case 0x0e : { // LDR
                                        Byte b = (Byte) dMemory.readMemory(rA.intValue());
                                        xReg.setValue(b.intValue());
                                }
                                break;
                                case 0x0f : { // STR
                                        int wbq = wbQ3.intValue();
                                        if ((wbq != 0x0a)&&(wbq != 0x0c)&&(wbq != 0x0b)){
                                                wmem = true;
                                                mempos = rA.intValue();
                                                dMemory.writeMemory(rB,rA.intValue());
                                        }
                                        xReg.setValue(0);
                                }
                                break;
                                }
                                xQueue.pushQueue(xReg);
                                xReg.setValue((xQueue.seeTopQueue()).intValue());
                                this.incrPC();
                                wbQ3p = (Nibble) wbQ3.clone();
                                wbQ3.setValue(exSelector.intValue());
                        }
}

private void incrPC() {
                Byte pc = (Byte) rFile.readMemory(Simulator.PCREG);
                int wbq = wbQ3.intValue();
                if (intr) {
                                pc.setValue(rFile.readMemory(Simulator.PSREG).intValue());
                                intr = false;
                                flush = true;
                }
                else if ((wbq == 0x0b) || ((wbq == 0x0a)&&(rFile.readMemory(Simulator.CNDBT).intValue() != 0))) {
                                pc.setValue(xReg.intValue());
                                rFile.readMemory(Simulator.CNDBT).clearValue();
                                flush = true;
                }
                else {
                                if(pc.intValue() > 255)
                                                pc.setValue(0);
                                else
                                                pc.setValue(pc.intValue() + 1);
                }
        }
}
```


Clase DependResolver

```
//$Id: DependResolver.java,v 1.5 1996/11/13 13:50:27 dubuc Exp $

/*
$Log: DependResolver.java,v $
Revision 1.5  1996/11/13 13:50:27  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package Simul;

import binContainer.*;
import memory.*;

public class DependResolver {

        RegFileMemory rFile;
        public Byte [] aReg = new Byte[2];
        public Byte [] bReg = new Byte[2];
        public Nibble sla, slb;
        ByteQueue rIdi;
        NibbleQueue wbQueue;
        Byte xReg;
        public boolean depL1, depL2;
        public boolean depAL1, depAL2;
        public boolean depBL1, depBL2;

        public DependResolver() {
                sla = new Nibble();
                slb = new Nibble();
                for(int i = 0; i < 2; i++) {
```

```
                    aReg[i] = new Byte();
                    bReg[i] = new Byte();
                }
                depL1 = depL2 = false;
                depAL1 = depAL2 = depBL1 = depBL2 = false;
}

public void reset() {
                sla.clearValue();
                slb.clearValue();
                for(int i = 0; i < 2; i++) {
                    aReg[i].clearValue();
                    bReg[i].clearValue();
                }
                depL1 = depL2 = false;
                depAL1 = depAL2 = depBL1 = depBL2 = false;
}

public void fetchSelector(Byte b) {
                sla.setValue((BinConvert.byteToUpperNibble(b)).intValue());
                slb.setValue((BinConvert.byteToLowerNibble(b)).intValue());
}

public void setParameters(
                RegFileMemory rf,
                Byte xr,
                ByteQueue rldiq,
                NibbleQueue wbq,
                Byte sByte)
{
                this.setParameters(rf,xr,rldiq,wbq);
                this.fetchSelector(sByte);
}

public void setParameters(
                RegFileMemory rf,
                Byte xr,
                ByteQueue rldiq,
                NibbleQueue wbq)
{
                this.rFile = rf;
                this.xReg = xr;
                this.rldi = rldiq;
                this.wbQueue = wbq;
}
public void resolve() {
                depL1 = depL2 = false;
                depAL1 = depAL2 = depBL1 = depBL2 = false;
                // A[1] (Dependencia de Nivel 1 en A)
                if((wbQueue.seeTopQueue()).intValue() != (BinConvert.byteToUpperNibble(rldi.seeElement(0))).intValue()){
                    aReg[1].setValue(aReg[0].intValue());
                }
                else {
                    aReg[1].setValue(xReg.intValue());
                    depAL2 = true;
                }

                // B[1] (Dependencia de Nivel 1 en B)
                if((wbQueue.seeTopQueue()).intValue() != (BinConvert.byteToLowerNibble(rldi.seeElement(0))).intValue()){
                    bReg[1].setValue(bReg[0].intValue());
                }
                else{
                    bReg[1].setValue(xReg.intValue());
                    depBL2 = true;
                }

                depL2 = depAL2 || depBL2;

                // A[0] (Dependencia de Nivel 2 en A)
                if((wbQueue.seeTopQueue()).intValue() != sla.intValue()){
                    aReg[0].setValue((rFile.readMemory(sla.intValue())).intValue());
                }
                else{
                    aReg[0].setValue(xReg.intValue());
                    depAL1 = true;
                }

                // B[0] (Dependencia de Nivel 2 en B)
                if((wbQueue.seeTopQueue()).intValue() != slb.intValue()){
                    bReg[0].setValue((rFile.readMemory(slb.intValue())).intValue());
                }
                else{
                    bReg[0].setValue(xReg.intValue());
                    depBL1 = true;
                }

                depL1 = depAL1 || depBL1;
}
```

```
	public void resolve(
			RegFileMemory rf,
			Byte xr,
			ByteQueue rldiq,
			NibbleQueue wbq)
	{
			this.setParameters(rf,xr,rldiq,wbq);
			this.resolve();
	}

	public void resolve(
			RegFileMemory rf,
			Byte xr,
			ByteQueue rldiq,
			NibbleQueue wbq,
			Byte sByte)
	{
			this.setParameters(rf,xr,rldiq,wbq);
			this.fetchSelector(sByte);
			this.resolve(),
	}
}
```

Clase NMETransf

```
//$Id: NMETransf.java,v 1.5 1996/11/13 13:50:27 dubuc Exp $

/*
$Log: NMETransf.java,v $
Revision 1.5  1996/11/13 13:50:27  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package Simul;

import binContainer.*;
import memory.*;

public class NMETransf {

	public static String [] NMES = {
			"ADD","SUB","AND","OR ",
			"NOT","SHT","<R>","<R>",
			"ZTS","NTS","JCN","RTI",
			"HLT","LDI","LDR","STR"
};

public static String bin2Nme(Word w) {

		StringBuffer tmpStr = new StringBuffer(),
		Byte hByte, lByte;

		hByte = BinConvert.wordToUpperByte(w);
		lByte = BinConvert.wordToLowerByte(w);

		int opcode = (BinConvert.byteToUpperNibble(hByte)).intValue();
		tmpStr = tmpStr.append(NMES[opcode]);

		if((opcode >= 0) && (opcode <= 5)){
			tmpStr = tmpStr.append(" R"+(BinConvert.byteToLowerNibble(hByte)).toHexString());
			tmpStr = tmpStr.append(",R"+(BinConvert.byteToUpperNibble(lByte)).toHexString());
			tmpStr = tmpStr.append(",R"+(BinConvert.byteToLowerNibble(lByte)).toHexString());
		}
		else if((opcode == 8) || (opcode == 9)) {
			tmpStr = tmpStr.append(" R"+(BinConvert.byteToUpperNibble(lByte)).toHexString()),
			tmpStr = tmpStr.append(",R"+(BinConvert.byteToLowerNibble(lByte)).toHexString()),
		}
		else if(opcode == 0x0a) {
			tmpStr = tmpStr.append(" @" + lByte.toHexString()),
		}
		else if(opcode == 0x0d) {
			Nibble n = BinConvert.byteToLowerNibble(hByte);
			if(n.intValue() == 0xe)
				tmpStr = tmpStr.append(" PS");
			else
				tmpStr = tmpStr.append(" R"+n.toHexString());
			tmpStr = tmpStr.append(",#" + lByte.toHexString());
		}
		else if(opcode == 0x0e) {
			tmpStr = tmpStr.append(" R"+(BinConvert.byteToLowerNibble(hByte)).toHexString());
			tmpStr = tmpStr.append(",R"+(BinConvert.byteToUpperNibble(lByte)).toHexString());
		}
		else if(opcode == 0x0f) {
			tmpStr = tmpStr.append(" R"+(BinConvert.byteToUpperNibble(lByte)).toHexString()),
			tmpStr = tmpStr.append(",R"+(BinConvert.byteToLowerNibble(lByte)).toHexString()),
```

```
                }
                return tmpStr.toString();
        }

        public static void convMemory(InstMemory iMem, String [] nmeMem) {

                int size = iMem.getCapMemory();

                for(int i = 0; i < size; i++)
                        nmeMem[i] = bin2Nme(iMem.readMemory(i));
        }

}
```

Clases del Paquete Assembler

Estas Clases fueron implementadas en un solo archivo .java por lo que al listarlo aparecerá como si fuera una sola clase, cuando en realidad lo conforman seis clases.

Clase Assembler

```
//$Id: Assembler.java,v 1.5 1996/11/13 13:44:46 dubuc Exp $

/*
$Log: Assembler.java,v $
Revision 1.5  1996/11/13 13:44:46  dubuc
Finalizacion de proyecto en esta revision, se igualan todas las revisiones
a la 1.5

*/

package Assembler;

import java.util.StringTokenizer;
import java.util.Vector;
import java.awt.*;
import binContainer.*;
import memory.*;

public class Assembler {

        StringTokenizer stok,
        String [] lines,
        int clines,numLab,
        OpcodeTable [] opcodes;
        LabelTable []  labels;
        static Opcodes oplist = new Opcodes();
        OpcodeTable regOp;

        public int numErr, numOp,
        public boolean assembOk, debug,
        public ErrorMessage [] errors,

        final int       DEC = 1;
        final int       HEX = 2;
        final int       BIN = 3,

        public Assembler() {
                this(false);
        }

        public Assembler(boolean deb) {
                this.debug = deb;
                this.assembOk = false,
        }

        public boolean assemble(String st) {

                assembOk = false;
                initLinesBuffer(st),
                makeTable();
                resolveLabels();

                if(debug) showBin(),

                if(numErr > 0){
                        if (debug) showErr();
                        return true;
                }
                else {
                        assembOk = true,
                        return false;
                }
        }

        private void showErr(){
                for(int i = 0 ; i < numErr, i++)
```

```
                        System.out.println("Error on Line "+(errors[i].linNo+1)+":"+errors[i].msg);
}

public String [] errMsgArray(){
        if(!assembOk){
                String [] stArr = new String[numErr];
                for(int i = 0 ; i < numErr; i++)
                        stArr[i] = "Error on Line "+(errors[i].linNo+1)+":"+errors[i].msg;
                return stArr;
        }
        else
                return new String[0];
}

public String [] binArray(int dataRep){
        if(assembOk){
                String [] stArr = new String[numOp];
                for(int i = 0 ; i < numOp; i++){
                        Byte b = new Byte(opcodes[i].dir);
                        Word w = new Word(opcodes[i].opcode);
                        switch(dataRep) {
                        case HEX:
                                stArr[i] = b.toHexString()+":"+w.toHexString();
                                break;
                        case DEC:
                                stArr[i] = b.toHexString()+":"+w.toString();
                                break;
                        case BIN:
                                stArr[i] = b.toHexString()+":"+w.toBinaryString();
                                break;
                        default:
                                stArr[i] = "";
                        }
                }
                return stArr;
        }
        else
                return new String[0];
}

public void putOnMemory(InstMemory iMem) {
        if(assembOk){
                for(int i = 0; i < numOp; i++)
                        if(i <= 0xff)
                                iMem.writeMemory(new Word(opcodes[i].opcode),i);
                if(numOp <= 0xff)
                        for(int i = numOp; i <= 0xff; i++)
                                iMem.writeMemory(new Word(0),i);
        }
}

private void showBin() {
        for(int i = 0, i < numOp; i++) {
                Word w = new Word(opcodes[i].opcode);
                System.out.println(opcodes[i].dir + ":nemon:"+ w.toHexString());
        }
}

private void initLinesBuffer(String st) {
        stok = new StringTokenizer(st,"\n");
        clines = stok.countTokens();
        lines = new String[clines];
        opcodes = new OpcodeTable[clines];
        labels = new LabelTable[clines];
        errors = new ErrorMessage[clines];
        numOp = numLab = numErr = 0;
        int i = 0;
        while (stok.hasMoreTokens()){
                lines[i] =  stok.nextToken().toUpperCase();
                i++;
        }
}


private void makeTable() {
        String tmpSt, nenmo;
        StringBuffer tmpStb;
        int Dir = 0, cnt1, cnt2, llin, x;

        for(cnt1 = 0, cnt1 < clines;cnt1++) {
                llin = lines[cnt1].length();
                tmpSt = new String(lines[cnt1]);
                tmpSt = tmpSt.replace('\t',' ');
                tmpStb = new StringBuffer(elimChar(tmpSt,' '));
                if(tmpStb.charAt(0) != ';') {
                        stok = new StringTokenizer(tmpStb.toString(),":");
                        tmpSt = stok.nextToken();
```

```
                if(tmpSt.compareTo(tmpStb.toString()) != 0){
                        tmpSt = elimChar(tmpSt,' ');
                        tmpSt = cleanSpaces(tmpSt);
                        labels[numLab] = new LabelTable(tmpSt,Dir);
                        numLab++;
                        try {
                                tmpSt = stok.nextToken();
                        } catch (java.util.NoSuchElementException e){
                                tmpSt = new String();
                        }
                }
                if(tmpSt.length() > 0) {
                        stok = new StringTokenizer(tmpSt," ");
                        nenmo = stok.nextToken();
                        this.searchNenm(nenmo,cnt1,Dir);
                        if((regOp.opval != 0xb)&&(regOp.opval != 0xc)&&(regOp.opval != 16)){

                                tmpSt = elimChar(tmpSt,' ');

                                cnt2 = 0;
                                while((cnt2<tmpSt.length())&&(tmpSt.charAt(cnt2)!=' '))
                                        cnt2++;
                                if(cnt2 < tmpSt.length()){
                                        tmpSt = tmpSt.substring(cnt2);

                                        this.resolveOperands(tmpSt,cnt1);
                                }
                                else { // Error
                                        errors[numErr] = new ErrorMessage(4,cnt1);
                                        numErr ++;
                                }
                        }
                        else if (regOp.opval == 0xc) {
                                regOp.opcode += 0xd00;
                        }
                        opcodes[Dir] = regOp;
                        Dir++;
                }
        }
    }
}


private String elimChar(String stIn,char ch){
        int cnt2= 0;
        while((cnt2 < stIn.length())&&(stIn.charAt(cnt2) == ch)) cnt2++;
        if(cnt2 < stIn.length())
                return stIn.substring(cnt2);
        else
                return stIn;
}

private String fromThisChar(String stIn,char ch){
        int cnt2= 0;
        while((cnt2 < stIn.length())&&(stIn.charAt(cnt2) != ch)) cnt2++;
        if(cnt2 < stIn.length())
                return stIn.substring(cnt2);
        else
                return stIn;
}

private void resolveLabels() {
        for(int i = 0; i < numOp; i ++)
                if(opcodes[i].label != null) {
                        boolean finded = false;
                        for(int j = 0; j < numLab; j++){
                                if(opcodes[i].label.compareTo(labels[j].label) == 0){
                                        opcodes[i].opcode += labels[j].dir;
                                        finded = true;
                                }
                        }
                        if(!finded) {
                                errors[numErr] = new ErrorMessage(2,opcodes[i].nlin);
                                numErr ++;
                        }
                }
}

 private void resolveOperands(String opSt, int nlin) {
          switch(regOp.opval) {
                        case 0x0: //ADD
                        case 0x1: //SUB
                        case 0x2: //AND
                        case 0x3: //OR
                        case 0x5: //SHT
                        {//ADD
                                opSt = elimChar(opSt,' ');
```

```
                if((opSt.charAt(0) != ',')&&(opSt.length() != 0)){
                        String tmp;
                        int nOp = 0;
                        stok = new StringTokenizer(opSt,",");
                        opSt = stok.nextToken();
                        stok = new StringTokenizer(opSt,", ");
                        for (nOp = 0; nOp < 3; nOp++){
                                try {
                                        tmp = stok.nextToken();
                                        int regv = regNameVerif(tmp,nlin);
                                        if(nOp == 0)
                                                regOp.opcode += regv << 8;
                                        else if (nOp == 1)
                                                regOp.opcode += regv << 4;
                                        else if (nOp == 2)
                                                regOp.opcode += regv;
                                } catch (java.util.NoSuchElementException e) {
                                        // Tengo menos de tres Operandos
                                        errors[numErr] = new ErrorMessage(1,nlin);
                                        numErr ++;
                                }
                        }
                        if(stok.hasMoreTokens()){ //Tengo mas de Tres Operandos
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                        }
                }
                else {
                        errors[numErr] = new ErrorMessage(1,nlin);
                        numErr ++;
                }
}break;
case 0x4: //NOT
case 0x8: //NTS
case 0x9: //ZTS
{
                opSt = elimChar(opSt,' ');
                if((opSt.charAt(0) != ',')&&(opSt.length() != 0)){
                        String tmp;
                        int nOp = 0;
                        stok = new StringTokenizer(opSt,",");
                        opSt = stok.nextToken();
                        stok = new StringTokenizer(opSt,",");
                        for (nOp = 0; nOp < 2; nOp++){
                                try {
                                        tmp = stok.nextToken();
                                        int regv = regNameVerif(tmp,nlin);
                                        if(nOp == 0){
                                                if(regOp.opval == 4)
                                                        regOp.opcode += regv << 8;
                                                else
                                                        regOp.opcode += regv << 4;
                                        }
                                        else if (nOp == 1){
                                                if(regOp.opval == 4)
                                                        regOp.opcode += regv << 4;
                                                else
                                                        regOp.opcode += regv;
                                        }
                                } catch (java.util.NoSuchElementException e) {
                                        // Tengo menos de dos Operandos
                                        errors[numErr] = new ErrorMessage(1,nlin);
                                        numErr ++;
                                }
                        }
                        if(stok.hasMoreTokens()){ //Tengo mas de dos Operandos
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                        }
                }
                else {
                        errors[numErr] = new ErrorMessage(1,nlin);
                        numErr ++;
                }
}break;
case 0xe: //LDR
case 0xf: //STR
{
                opSt = elimChar(opSt,' ');
                if((opSt.charAt(0) != ',')&&(opSt.length() != 0)){
                        String tmp;
                        int nOp = 0;
                        stok = new StringTokenizer(opSt,",");
                        opSt = stok.nextToken();
                        stok = new StringTokenizer(opSt,", ");
                        for (nOp = 0; nOp < 2; nOp++){
                                try {
                                        tmp = stok.nextToken();
```

```
                                                int regv = regNameVerif(tmp,nlin);
                                                if(nOp == 0){
                                                        if(regOp.opval == 0xe)
                                                                regOp.opcode += regv << 8;
                                                        else
                                                                regOp.opcode += regv << 4;
                                                }
                                                else if (nOp == 1){
                                                        if(regOp.opval == 0xe)
                                                                regOp.opcode += regv << 4;
                                                        else
                                                                regOp.opcode += regv;
                                                }
                                        } catch (java.util.NoSuchElementException e) {
                                                // Tengo menos de dos Operandos
                                                errors[numErr] = new ErrorMessage(1,nlin);
                                                numErr ++;
                                        }
                                }
                                if(stok.hasMoreTokens()){ //Tengo mas de dos Operandos
                                        errors[numErr] = new ErrorMessage(1,nlin);
                                        numErr ++;
                                }
                        }
                        else {
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                        }
}break;
case 0xd: //LDI
{
                opSt = elimChar(opSt,' ');
                if((opSt.charAt(0) != ',')&&(opSt.length() != 0)){
                        String tmp;
                        int nOp = 0;
                        stok = new StringTokenizer(opSt,",");
                        opSt = stok.nextToken();
                        stok = new StringTokenizer(opSt,", ");
                        for (nOp = 0; nOp < 2; nOp++){
                                try {
                                        tmp = stok.nextToken();
                                        if(nOp == 0){
                                                int regv = regNameVerif(tmp,nlin),
                                                regOp.opcode += regv << 8;
                                        }
                                        else if (nOp == 1){
                                                opSt = fromThisChar(opSt,'#');
                                                if(opSt.charAt(0) == '#'){
                                                        opSt = opSt.substring(1);
                                                        opSt = cleanSpaces(opSt);
                                                        try {
                                                                int dir =

Integer.parseInt(opSt,16);

                                                                if((dir < 0) || (dir > 0xff)) {

                errors[numErr] = new ErrorMessage(8,nlin);

                                                                        numErr ++;
                                                                }
                                                                else


                regOp.opcode += dir;

                                                        }catch

(java.lang.NumberFormatException e) {

                                                                errors[numErr] = new

ErrorMessage(1,nlin);

                                                                numErr ++;
                                                        }
                                                }
                                                else {
                                                        errors[numErr] = new

ErrorMessage(1,nlin);

                                                        numErr++;
                                                }
                                        }
                                } catch (java.util.NoSuchElementException e) {
                                        // Tengo menos de dos Operandos
                                        errors[numErr] = new ErrorMessage(1,nlin);
                                        numErr ++;
                                }
                        }
                        if(stok.hasMoreTokens()){ //Tengo mas de dos Operandos
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                        }
                }
                else {
                        errors[numErr] = new ErrorMessage(1,nlin);
                        numErr ++;
```

```
                        }
                }break;
                case 0xa: //JCN
                {
                        opSt = elimChar(opSt,' ');
                        if((opSt.charAt(0) != ',')&&(opSt.length() != 0)){
                                stok = new StringTokenizer(opSt,",");
                                opSt = stok.nextToken();
                                opSt = fromThisChar(opSt,'@');
                                if(opSt.charAt(0) == '@'){
                                        opSt = opSt.substring(1);
                                        opSt = cleanSpaces(opSt);
                                        try {
                                                int dir = Integer.parseInt(opSt,16);
                                                if((dir < 0) || (dir > 0xff)) {
                                                        errors[numErr] = new ErrorMessage(7,nlin);
                                                        numErr ++;
                                                }
                                                else
                                                        regOp.opcode += (0xf00 + dir);
                                        }catch (java.lang.NumberFormatException e) {
                                                errors[numErr] = new ErrorMessage(6,nlin);
                                                numErr ++;
                                        }
                                }
                                else {
                                        opSt = elimChar(opSt,' ');
                                        opSt = cleanSpaces(opSt);
                                        regOp.opcode += 0xf00;
                                        regOp.label = opSt;
                                }
                        }
                        else {
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                        }
                }break;
                }
}

private String cleanSpaces(String st){
        int cap = st.length();
        Vector v = new Vector();
        for(int i=0; i<cap; i++)
                if(st.charAt(i) != ' '){
                        v.addElement(new Character(st.charAt(i)));
                }
        StringBuffer sb = new StringBuffer();
        for(int i=0;i < v.size();i++)
                sb = new StringBuffer(sb.toString() + ((Character)v.elementAt(i)).toString());
        return sb.toString();
}

private int regNameVerif(String rSt, int nlin){
        rSt = elimChar(rSt,' ');
        if((rSt.length() > 1)&&(rSt.charAt(0) == 'P')&&(rSt.charAt(1) == 'S')){
                return 14;
        }
        else if(rSt.charAt(0) == 'R'){
                rSt = rSt.substring(1);
                        try {
                                int dir = Integer.parseInt(rSt,16);
                                if((dir < 0) || (dir > 0xe)) {
                                        errors[numErr] = new ErrorMessage(3,nlin);
                                        numErr ++;
                                        return 0;
                                }
                                else
                                        return dir;
                        }catch (java.lang.NumberFormatException e) {
                                errors[numErr] = new ErrorMessage(1,nlin);
                                numErr ++;
                                return 0;
                        }
        }
        else {// Error
                        errors[numErr] = new ErrorMessage(3,nlin);
                        numErr ++;
                        return 0;
        }
}

private void searchNemm(String nemmo, int lnum, int dir) {
        int i = 0;
        while((i < 14) && (nemmo.compareTo(oplist.oplst[i].mnemon) != 0)) i++;
        if(i == 14) {
                errors[numErr] = new ErrorMessage(0,lnum);
                regOp = new OpcodeTable(dir,16,lnum,null);
```

```
                                numErr++;
                        }
                        else {
                                regOp = new OpcodeTable(dir,oplist.oplst[i].opval,lnum,null);
                        }
                        numOp ++;
                }
}

class OpcodeTable {
                public int dir;
                public int opcode,
                public int opval;
                public String label;
                public int nlin;

                public OpcodeTable(int d, int o, int n, String s) {
                        dir = d;
                        opval = o;
                        opcode = opval << 12;
                        nlin = n;
                        label = s;
                }
}

class LabelTable {
                public int dir;
                public String label;

                public LabelTable(String l, int d){
                        label = l;
                        dir = d;
                }
}

class ErrorMessage {
                public String msg;
                public int linNo,

                private static String [] msgs = {
                        "No such nenmonic in DMN architecture.",      // 0
                        "Syntax Error.",                              // 1
                        "No such Label in JCN instrucction.",         // 2
                        "No such register in DMN Architecture.",      // 3
                        "Operands missings in instruction.",          // 4
                        "Too much operands in instruction.",          // 5
                        "Invalid address format in JCN instrucction.", // 6
                        "Address out of range in JCN instrucction.",  // 7
                        "Value out of range in LDI instrucction."     // 8
                };

                public ErrorMessage(int errNo, int ln) {
                        try{
                                msg = this.msgs[errNo];
                        } catch (java.lang.ArrayIndexOutOfBoundsException e) {
                                msg = this.msgs[0];
                        }
                        linNo = ln;
                }
}

class Opcode {
                public String mnemon,
                public int opval;
                public int oppos,

                Opcode(String st, int ov, int op){
                        mnemon = st;
                        opval = ov;
                        oppos = op;
                }
}

class Opcodes {
                public static Opcode [] oplst = new Opcode[14];

                public Opcodes() {
                        oplst[0] = new Opcode("ADD",0x0,1);
                        oplst[1] = new Opcode("SUB",0x1,2);
                        oplst[2] = new Opcode("AND",0x2,3);
                        oplst[3] = new Opcode("OR",0x3,4);
                        oplst[4] = new Opcode("NOT",0x4,5);
                        oplst[5] = new Opcode("SHT",0x5,6);
                        oplst[7] = new Opcode("ZTS",0x8,7);
                        oplst[6] = new Opcode("NTS",0x9,8);
```

```
            oplst[8]  = new Opcode("JCN",0xa,9);
            oplst[9]  = new Opcode("RTI",0xb,10);
            oplst[10] = new Opcode("HLT",0xc,11);
            oplst[11] = new Opcode("LDI",0xd,12);
            oplst[12] = new Opcode("LDR",0xe,13);
            oplst[13] = new Opcode("STR",0xf,14);
        };
}
```

Clases del Paquete GUI

Estas clases son el soporte de la interfaz gráfica, no se listaran por motivos de espacio, en su lugar se refiere al lector interesado a los listados del Disquete anexo.

FECHA DEVOLUCION	Nº CARNET	FECHA DEVOLUCION	Nº CARNET
17-9-97	I5974	6-6-03	I958
12-1-98	I768	1-06-04	I958
25-2-98	I6560		
21-9-00	I958		
23-10-00	I958		
20-11-00	I958		
11 01 01	I958		
~~8-02-01~~	I		
8-02-01	I089		
21-02-01	I958		
5-401	I458		
15-5-01	I958		
23-5-01	I958		
01-06-01	I958		
29/01/08	B11700		