Universidad De Oriente

Núcleo De Anzoátegui

Escuela De Ingeniería Y Ciencias Aplicadas

Departamento De Electricidad

DESARROLLO DEL SIMULADOR DE COMPUTADORES UDO-2008 BASADO EN HERRAMIENTAS COMPUTACIONALES DE SOFTWARE LIBRE

Miguel Eduardo Barrios Canache

Trabajo de Grado presentado ante la Universidad De Oriente como requisito parcial para optar al título de:

Ingeniero Electricista

Barcelona, Febrero de 2009

Universidad De Oriente

Núcleo De Anzoátegui

Escuela De Ingeniería Y Ciencias Aplicadas

Departamento De Electricidad

DESARROLLO DEL SIMULADOR DE COMPUTADORES UDO-2008 BASADO EN HERRAMIENTAS COMPUTACIONALES DE SOFTWARE LIBRE

Ing. Luís Parraguez

Asesor Académico

Barcelona, Febrero de 2009

Universidad De Oriente

Núcleo De Anzoátegui

Escuela De Ingeniería Y Ciencias Aplicadas

Departamento De Electricidad

DESARROLLO DEL SIMULADOR DE COMPUTADORES UDO-2008 BASADO EN HERRAMIENTAS COMPUTACIONALES DE SOFTWARE LIBRE

JURADO CALIFICADOR

El Jurado hace constar que asignó a esta Tesis la calificación de:

_____

Ing. Luís Parraguez

Asesor Académico

_____	_____
Danilo Navarro	Margarita Heraoui
Jurado Principal	Jurado Principal

Barcelona, Febrero de 2009

RESOLUCIÓN

De acuerdo al artículo 44 del Reglamento de Trabajos de Grado: “Los Trabajos de Grado son de exclusiva propiedad de la Universidad y sólo podrán ser utilizados para otros fines con el consentimiento del Consejo de Núcleo respectivo, quién lo participará al Consejo Universitario”.

DEDICATORIA

A Mi Hijo, Miguel De Jesús

Bienaventurado el hombre que halla la sabiduría,
Y que obtiene la inteligencia;
Porque su ganancia es mejor que la ganancia de la plata,
Y sus frutos más que el oro fino.
Más preciosa es que las piedras preciosas;
Y todo lo que puedes desear, no se puede comparar a ella.
Largura de días está en su mano derecha;
En su izquierda, riquezas y honra.
Sus caminos son caminos deleitosos,
Y todas sus veredas paz.
Ella es árbol de vida a los que de ella echan mano,
Y bienaventurados son los que la retienen.

Proverbios, capitulo 3

AGRADECIMIENTOS

A Dios, por crearle al hombre la necesidad, y la creatividad para sobrellevarla.

A Mamá, Tía y Abuela, por vivir pura y sabiamente, y cuidar del hogar. Por formar mi carácter y mi perseverancia. A Tía, tu apoyo maternal e incondicional, al brindarme las herramientas para desarrollar este trabajo.

A Papá, por mostrarme el camino a seguir. Porque pobreza y vergüenza tendrá el que menosprecia el consejo; mas el que guarda la corrección recibirá honra (Proverbios 13:18).

A María Elena, Por su comprensión, paciencia y gran amor para afrontar juntos el presente y el futuro.

RESUMEN

El Trabajo presentado a continuación trata acerca de las mejoras realizadas al simulador utilizado en la materia “Diseño de Sistemas Digitales”.

El desarrollo del simulador de computación UDO 2008 se constituye a partir del uso de un paradigma de programación utilizado a nivel comercial por grandes empresas. Conocido como Programación Orientada a Objetos, está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Se utilizó *Lenguaje Unificado de Modelado* para definir el sistema, para detallar los artefactos en el sistema, para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se realizó el programa bajo herramientas de software libre, desde la implementación de UML con BOUML y DIA, el compilador para C++ de GNU, GLADE 3 para crear la interfaz gráfica y en general el sistema GNU/Linux como base para el desarrollo.

Además, se realizaron pruebas al simulador, con el propósito de corroborar las nuevas funcionalidades del sistema.

TABLA DE CONTENIDOS

CAPÍTULO 1: INTRODUCCIÓN

1.1 ANTECEDENTES

Se tomaron en cuenta diversas investigaciones realizadas anteriormente, las cuales están relacionadas con el tema de estudio que servirán como guía para el desarrollo del trabajo de grado vinculado. El antecedente tomado en consideración es el siguiente:

Parraguez, Luis (2000) [1]. Trabajo presentado como requisito parcial para ratificar la categoría de profesor asistente en la Universidad de Oriente Núcleo de Anzoátegui.

El autor documentó la información para cubrir las necesidades del curso. El trabajo ofrece los principios y la conceptualización del diseño de computadores. En primer lugar se describen bloques funcionales que permiten realizar tareas específicas y puestas en conjunto dan lugar a resolver tareas cada vez más complejas. El estudio de estos aspectos básicos de lógica combinacional, secuencial, diseño de unidades de memoria y máquinas de estados son analizados mediante simulaciones con programas como Proteus, Circuit Maker, Verilog.

También se estudia el diseño de un computador propuesto, se describe cada uno de los componentes internos del computador y se desarrolla el conjunto de instrucciones que puede manejar. Finalmente, el documento explica los aspectos funcionales y como fue desarrollado el simulador. Todos los sistemas estudiados previamente pueden ser analizados con el simulador de computadores UDO-98 realizado por el autor.

Hernández, Vinicio (2007) [2]. Trabajo presentado como requisito parcial para optar al titulo de Ingeniero Electricista en la Universidad de Oriente Núcleo de Anzoátegui.

El autor estudió el estándar Hart de la HFC (Fundación de Comunicación Hart) con el cual pudo determinar la estructura de los distintos comandos, los formatos con los que se presentan los datos, las características de transmisión, entre otros.

Desarrolló una aplicación con interfaz gráfica utilizando Glade para un manejador de activos para instrumentación digital, con el cual el usuario puede visualizar la información de los equipos y tener acceso a la documentación de los mismos.

1.2 PLANTEAMIENTO DEL PROBLEMA

Los sistemas lógicos son la base de los computadores y en general de las máquinas de procesos de información, sean de propósito general, específico o no convencional. Es importante crear e implementar herramientas didácticas que permitan a los estudiantes interactuar con lo que se presenta en la parte teórica. Está claro que, en algunos casos esta aproximación es posible mediante prácticas de laboratorio, pero en otros casos no es así. Por lo tanto, se puede apreciar la necesidad de contar con alternativas, como los simuladores, que permiten cubrir las necesidades académicas.

El conocimiento sobre el diseño de circuitos digitales abre toda la línea conceptual de las arquitecturas orientadas a aplicaciones específicas que actualmente constituyen un área de fundamental importancia en los sistemas electrónicos e informáticos. En consecuencia, el principal objetivo es adquirir una noción de los

conceptos y técnicas relacionadas con la estructura y el diseño de los circuitos y sistemas lógicos, incluyendo el análisis de circuitos digitales.

El desarrollo de software en el área de sistemas digitales, con fines didácticos, desempeña un papel importante en la enseñanza de materias como: Diseño de Sistemas Digitales, Arquitectura de Computadores, entre otras. Debido a que los estudiantes obtienen la posibilidad de acercarse más al funcionamiento y operación interna de un procesador. El simulador utilizado actualmente en la cátedra “Diseño de Sistemas Digitales” cubre estos requisitos, pero presenta algunas limitaciones en su funcionalidad, las cuales deben ser solventadas aprovechando los recursos disponibles en la actualidad. Varios de los problemas de la versión actual son:

- Algunas instrucciones no funcionan correctamente.
- Las rutinas de interrupciones presentan algunos problemas de ejecución.
- No cuenta con un compilador que traduzca las instrucciones de lenguaje ensamblador a código objeto utilizado por el procesador.
- No existe la posibilidad de seleccionar el programa a ejecutar. Solo se puede simular un programa sustituyendo el archivo PROGRAMA.UDO.

En base a lo expuesto anteriormente, es necesario realizar nuevas funciones para el programa, depurar código existente y cambiar la plataforma de software, es decir, desarrollar nuevas aplicaciones para el simulador y adecuarlo al sistema GNU/Linux. Esto se realizará usando lenguajes como C++ o Ada. El entorno gráfico será actualizado por medio de Glade 3 para el sistema operativo GNU/Linux.

1.3 OBJETIVOS

1.3.1 Objetivo General

Desarrollar el simulador de computadores UDO-2008 basado en herramientas computacionales de software libre.

1.3.2 Objetivos Específicos

- Describir detalladamente la arquitectura del computador UDO-98.
- Cambiar la plataforma de software del sistema a GNU/Linux y otras herramientas computacionales de software libre.
- Desarrollar un compilador para el lenguaje ensamblador utilizado por el procesador.
- Incorporar el código objeto generado por el compilador al simulador para que ejecute un programa.
- Solucionar el problema del manejo de las interrupciones.
- Comprobar la ejecución de todas las instrucciones desarrolladas para el simulador a través de la realización de programas con el compilador.

1.4 ALCANCE

Se logró crear una herramienta de fácil uso para los estudiantes, lo que permite enfocarse aun más en la implementación del sistema estudiado en clases. Además, ahora el sistema puede ser multiplataforma gracias a GTK+. Ajustándose así al sistema operativo preferido de cada estudiante.

El análisis y diseño de software mediante técnicas de programación orientada a objeto es una de las formas más sofisticadas en actividades relacionadas con estas

fases del proyecto. La utilización del lenguaje de modelado UML puede mejorar la comunicación con los clientes del sistema, mejorar la documentación y permitir explorar funcionalidades y aspectos concretos de las clases del sistema durante el análisis y diseño. Las fases de desarrollo e implantación del sistema fueron realizadas con técnicas convencionales de programación, a pesar de que parte del código puede ser generado automáticamente con UML no se aplicó de esta forma para incluir código heredado de la versión anterior del simulador.

El nivel de implementación obtenido con la versión anterior, donde los archivos datos de la UAL y el CONTROL pueden ser empleados para programar memorias de un prototipo físico del sistema, todavía es válido para esta versión. Porque las clases que representan esos subsistemas conservan su estructura.

1.5 JUSTIFICACIÓN

Los simuladores de procesadores son usados primordialmente como una forma de demostrar las utilidades y características de una arquitectura, aprendizaje de microinstrucciones, la relación entre el procesamiento y el ensamblador; su intención no es de ayudar al diseño de un sistema verdadero. Como resultado, los simuladores dentro del ambiente académico rara vez son usados para validaciones de funcionalidad o desempeño real, pero son prueba de los conceptos, exploración de los aspectos de diseño y análisis cuantitativo.

Otro punto importante es el cambio del sistema operativo bajo el cual funciona el simulador vigente. Debe señalarse el decreto N° 3,390 a través del cual el gobierno de la República Bolivariana de Venezuela dispone el empleo de software libre en la administración pública. La vigencia del decreto impulsa la utilización de este tipo de software, así como también, su desarrollo y creación por parte de los Venezolanos.

Por consiguiente, es necesario enfocar las iniciativas de desarrollar software bajo estas condiciones y obtener el beneficio de sus libertades.

1.6 METODOLOGÍA

Para completar los objetivos formulados, fue necesario dividir el proyecto en varias etapas realizadas, en algunos casos, de forma solapada. La primera etapa fue una revisión de los textos disponibles en la biblioteca de la universidad, diferentes páginas Web y consulta a los profesores relacionados con el área de estudio. En segundo lugar, el cambio de la plataforma de software para el desarrollo del simulador: abarcó el estudio del funcionamiento de las herramientas a utilizar (Sistema Operativo, Generador de Entorno Gráfico), la revisión, modificación y actualización de los códigos existentes adaptarlos al nuevo programa. La tercera fase fue la realización del compilador: donde se analizó cual debía ser el funcionamiento del ensamblador de acuerdo a los requerimientos del simulador y los diferentes métodos para realizar la aplicación. Por último, la comprobación de todos los aspectos del nuevo simulador: se revisaron todas las funciones del nuevo programa, para garantizar que opere de la manera esperada y sin los fallos de la versión anterior, mediante varios ejemplos de programación.

CAPÍTULO 2: BASES TEÓRICAS

2.1 LA COMPUTADORA DIGITAL

Una computadora digital es una colección de elementos lógicos que pueden realizar algoritmos para llevar a cabo cálculo de datos y manipulación de funciones. Una computadora se compone de un microprocesador, la memoria, y algunos puertos de entrada/salida (I/O) como se muestra en la **Figura 1**. El microprocesador, a menudo llamado *Unidad Microprocesadora* (MPU) o *Unidad Central de Procesamiento* (CPU), contiene lógica para pasar a través de un algoritmo, llamado programa, que se ha almacenado previamente en la memoria. Los datos utilizados y manipulados por el programa que se lleva a cabo en el computador se almacenan en la memoria de datos. La memoria es un repositorio de datos que suele ser organizado como una serie lineal de lugares accesibles individualmente. El microprocesador puede acceder a un lugar determinado en la memoria mediante la presentación de una dirección de memoria (el índice de la ubicación deseada) para el elemento de memoria. Los elementos de Entrada / Salida (I/O) permiten que el microprocesador pueda comunicarse con el mundo exterior para adquirir nuevos datos y presentar los resultados de sus cálculos programados. Tales elementos pueden incluir un teclado o una pantalla.

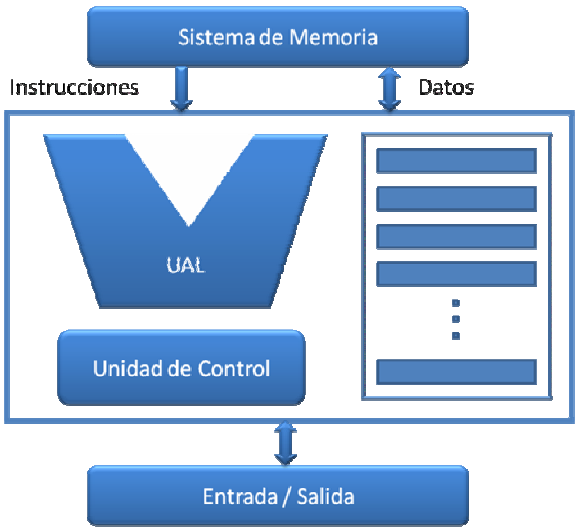

Figura 1: Componentes principales de un computador e interacción con la memoria y los periféricos

Los programas se componen de muchas operaciones individuales muy simples, llamadas instrucciones, que especifican de manera muy exacta la forma en que el microprocesador debe llevar a cabo un algoritmo. Un programa simple puede tener decenas de instrucciones, mientras que un programa complejo puede tener decenas de miles de instrucciones. En conjunto, los programas que se ejecutan en microprocesadores se llaman *software*, en contraste con el *hardware*, que se corresponde a la parte electromecánica en donde los anteriores se ejecutan. Cada tipo de microprocesador tiene su propio conjunto de instrucciones que define el conjunto de operaciones singulares y discretas que es capaz de ejecutar. Estas instrucciones realizan una tarea que, por su propia cuenta, puede parecer insignificante. Sin embargo, cuando miles o millones de estas diminutas instrucciones se conectan, pueden crear un juego de vídeo o un procesador de textos. Cada microprocesador se construye con un conjunto de instrucciones que pueden ser invocadas en secuencias arbitrarias. Por lo tanto, un microprocesador tiene el potencial para desempeñar un trabajo útil a través de las instrucciones, pero no puede hacer nada por su propia cuenta. Para que el microprocesador pueda realizar un trabajo, se requiere orientación

en forma de programación de software. Una tarea de cualquier complejidad debe desglosarse en muchos pasos diminutos que deberán realizarse en un microprocesador. Estos pasos incluyen aritmética básica, operaciones booleanas, la carga de datos desde la memoria o un elemento de entrada como un teclado, y almacenamiento de datos a la memoria o hacia un elemento de salida como una impresora.

La memoria es una estructura de un computador de características claves, porque el microprocesador accede a ella constantemente para obtener una nueva instrucción, cargar nuevos datos para operar, o almacenar un resultado. Si bien el programa y los datos son clasificaciones distintas, pueden compartir el mismo recurso de memoria física.

2.2 CONCEPTOS BÁSICOS DEL CPU

Típicamente, un CPU contiene tres componentes: conjunto de registros, unidad aritmética-lógico (UAL), y una unidad de control (UC). El conjunto de registros varía de una arquitectura a otra. Generalmente, es una combinación de registros de propósitos generales y propósitos especiales [3]. Los registros de propósitos generales tienen funciones asignadas por el programador, a diferencia de los registros especiales que tienen funciones específicas dentro del CPU. Por ejemplo, el contador de programa (PC), y el registro de instrucciones (IR). La UAL proporciona la capacidad de realizar las operaciones aritméticas, lógicas y de desplazamiento requeridas por el conjunto de instrucciones. La unidad de control es la responsable de buscar la instrucción a realizar desde la memoria, decodificarla y luego ejecutarla.

2.2.1 Unidad Aritmética - Lógica

La Unidad Aritmético Lógica es la parte del computador que realiza las operaciones aritméticas y lógicas sobre los datos. Todos los elementos del sistema - unidad de control, registros, memoria, I/O- están allí para llevar datos a la UAL de manera que procese y luego presente el resultado. La **Figura 2** indica en términos generales, como la UAL está interconectada con el resto del procesador. Los datos se presentan a la UAL mediante registros, y los resultados de las operaciones son almacenados en registros. Estos registros son ubicaciones internas temporales de almacenamiento dentro del procesador que están conectadas a la UAL. Los valores de banderas también están almacenados dentro de registros internos del procesador. La unidad de control provee señales que controlan las operaciones de la UAL así como también los movimientos de su entrada y salida [4].

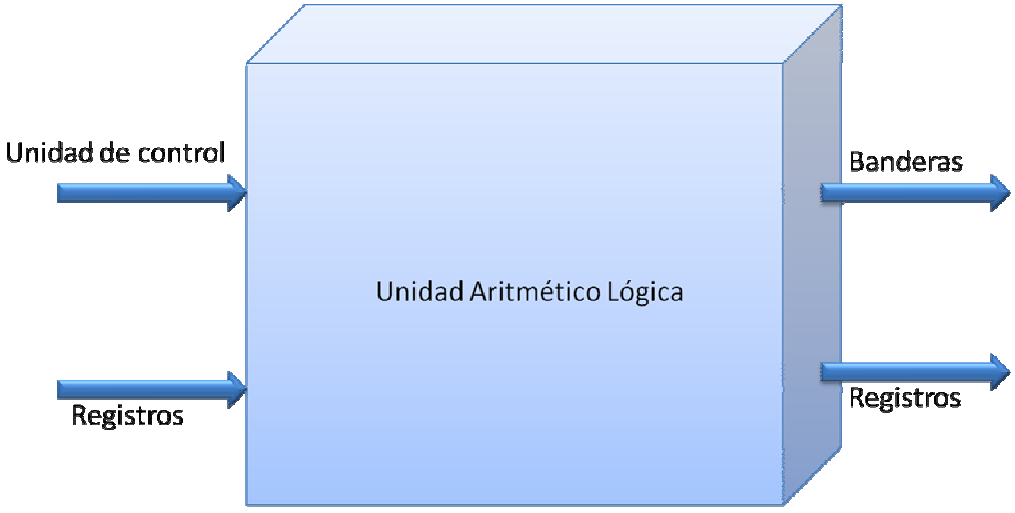

Figura 2: Comunicación de la UAL con el resto de los componentes

2.2.2 Los Registros

Los registros son usados en los sistemas de computadoras como sitios para almacenar una amplia variedad de datos. Como por ejemplo, direcciones, contadores de programa, o datos necesarios para la ejecución de un programa. Es decir, un registro es un dispositivo de hardware que almacena datos binarios. Están localizados dentro del procesador de manera que se pueda acceder a los datos con extrema rapidez [5].

El procesamiento de datos en un computador es realizado usualmente con palabras con un ancho determinado que son almacenados en registros. Por lo tanto, la mayoría de los computadores tienen registros de un ancho determinado. Usualmente, 16, 32 y 64 bits. El número de registros en una máquina varía de una arquitectura a otra. Pero es típicamente una potencia de 2, siendo entre 16, 32 y 64 las cantidades más comunes. Los registros contienen datos, direcciones, o información de control [5]. Los registros de propósitos especiales contendrán solamente alguno de los anteriores mientras que los de propósito general podrán contener información variada a distintos momentos.

2.2.3 La Unidad de Control

Es la encargada de iniciar y mantener la operación de todo el sistema. Debe tomar desde la memoria el código de la operación ha realizar y generar secuencialmente las señales de control requeridas para llevarla acabo [1]. Existen dos necesidades que deben ser cubiertas por la unidad de control: la presencia de un sistema de reloj que sincronice la transferencia entre los registros y un sistema temporizador que administre en el tiempo el uso de los recursos. Internamente, debe coordinar la información del código de operación, el tiempo de ejecución y las condiciones de la UAL para generar la secuencia de señales de control.

2.2.3.1 Unidad de Control Microprogramada

La idea de una unidad de control microprogramada fue introducida por M. V. Wilkes a principio de los años 50. Fue motivado por el deseo de reducir la complejidad relacionada con el control cableado. Las microinstrucciones especifican todas las señales de control del camino de datos, más la posibilidad de decidir condicionalmente qué microinstrucción se debe ejecutar a continuación. Como sugiere el nombre de «microprogramación», una vez que se diseñan el camino de datos y la memoria para las microinstrucciones, el control se convierte básicamente en una tarea de programación; es decir, la tarea de escribir un intérprete para el repertorio de instrucciones. La invención de la microprogramación posibilitó que el repertorio de instrucciones pudiera cambiarse alterando el contenido de la memoria de control sin tocar el hardware [6].

La manera de interactuar con el procesador para que funcione de acuerdo a lo requerido es a través del conjunto de instrucciones (lenguaje ensamblador), este representa la conexión entre las necesidades del usuario y la operación del procesador.

2.3 ENSAMBLADORES

Son programas que generan instrucciones en código máquina a partir de un programa fuente escrito en lenguaje ensamblador. Se encarga de reemplazar las direcciones simbólicas por direcciones numéricas, cambiar códigos de operación simbólicos por códigos de operación de máquina, reserva espacio para instrucciones y

datos, y traduce las constantes en representación numérica utilizada por el computador. En atención a lo expuesto, estas funciones pueden ser realizadas al escanear el programa en ensamblador y realizar la equivalencia de sus instrucciones con su código hexadecimal. Debido a que los símbolos pueden ser usados antes de que sea definido más adelante, en algunos ensambladores un solo escaneado no es suficiente para realizar toda la equivalencia. Un ensamblador simple escanea completamente un programa dos veces. Durante la primera pasada, se genera una tabla que incluye todos los símbolos y sus valores numéricos. Esta tabla se llama la tabla de símbolos, en la **Figura 3** se detalla la manera como se genera la tabla. Durante la segunda pasada, el ensamblador usará la tabla de símbolos junto con otras tablas para generar el programa objeto [5]. En la **Figura 4** se detalla el segundo paso.

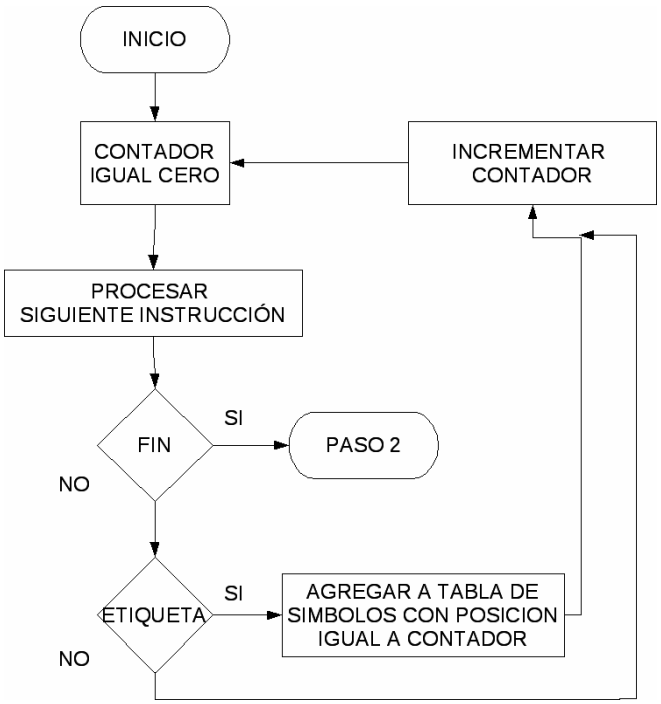

Figura 3: Primer paso simplificado de un ensamblador

Al escribir programas en lenguaje ensamblador para cualquier arquitectura en general, un número de asuntos se deben considerar:

- Directivas de ensamblador.
- Uso de símbolos.
- Uso de operaciones sintéticas.
- Sintaxis del ensamblador.
- Interacción con el sistema operativo.

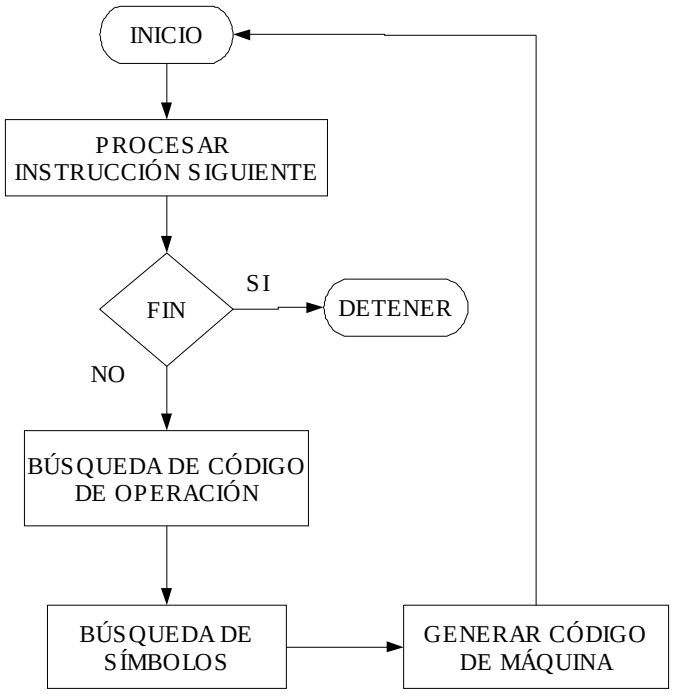

Figura 4: Segundo paso simplificado de un ensamblador

El uso de directivas de ensamblador, también llamadas pseudo-operaciones, es un asunto de mucha importancia al escribir programas en lenguaje ensamblador. Las directivas del ensamblador son comandos entendidos solamente por el ensamblador y no corresponden a instrucciones de la máquina. Afectan el modo en que el ensamblador realiza la conversión de código ensamblador y código máquina. Los símbolos son usados por los programas en lenguaje ensamblador para representar

números. Esto es con la finalidad de hacer que el código sea más fácil de leer, entender y depurar. Los símbolos son traducidos a su correspondiente valor numérico por el ensamblador.

2.3.1 Subrutinas y la pila

La mayoría de los programas están organizados en varios bloques de instrucciones llamadas subrutinas en lugar de una sola gran secuencia de instrucciones. Las subrutinas están separadas del segmento principal del programa y son solicitadas por una llamada a subrutina. La llamada es un tipo de salto que cambia temporalmente el contador de programa (PC) del microprocesador a la subrutina, lo que le permite ser ejecutado. Cuando la subrutina se ha completado, el control regresa al programa que lo llamó a través de una instrucción de retorno de subrutina. Las subrutinas proporcionan varios beneficios a un programa, incluyendo la modularidad y la facilidad de reutilización. Una subrutina modular puede ser reubicada en diferentes partes de un mismo programa al mismo tiempo que realiza la misma función básica. La reutilización está relacionado con la modularidad y lleva el concepto un paso más lejos al permitir que la subrutina pueda ser trasplantada de un programa a otro sin modificación. Este concepto acelera enormemente el proceso de desarrollo de software [7].

2.3.2 Interrupciones

Cuando se requiere que el microprocesador no siga la secuencia normal de instrucciones al ocurrir un acontecimiento, y el programador desea que el microprocesador detenga lo que está haciendo y maneje el evento con una rutina especial, entonces se llama a una interrupción. Una solicitud de una interrupción es la puesta en marcha de una operación periódica, como el control de la temperatura de una habitación. Debido a que la temperatura ambiente no cambia con frecuencia, el

microprocesador puede manejar otras tareas durante el funcionamiento normal. Se puede establecer un temporizador que expire cada pocos segundos, causando un evento de interrupción. Cuando se desencadena la interrupción, el microprocesador puede leer la temperatura ambiente, tomar las medidas apropiadas (por ejemplo, encender la ventilación), y luego reanudar su funcionamiento normal.

Una interrupción puede ser provocada por la habilitación de una señal de interrupción del microprocesador. También puede ser activado por software a través de instrucciones especiales. Cuando una interrupción se produce, el microprocesador guarda su estado empujando el PC y otros registros a la pila y, a continuación, el PC se carga con un vector de interrupción que apunta a una rutina de servicio de interrupción (ISR - interrupt service routine) en la memoria. De esta manera, interrumpir el proceso es similar a un salto a subrutina. Sin embargo, la interrupción puede ser desencadenada por un evento de hardware en lugar de un programa. Cada pin de interrupción en el microprocesador tiene un vector de interrupción asociado a él. El programador sabe que una ISR se encuentra en una ubicación de memoria para un determinado servicio de interrupción. Cuando el ISR ha concluido, un *retorno de interrupción* se ejecuta y restablece el estado del microprocesador desde la pila. El control se regresa a la rutina que se interrumpió y continúa la ejecución normal.

Mientras se ejecuta el mecanismo de interrupción, el programa que se interrumpió, puede que no tenga ningún conocimiento del evento. Debido a que el estado del microprocesador se guarda y, a continuación, es restaurado durante el *regreso de interrupción*, la rutina principal no tiene idea de que en alguna parte a lo largo de su camino su ejecución fue detenida por un período arbitrario. El programador puede optar por hacer tales conocimientos disponibles mediante el intercambio de información entre el ISR y otras rutinas, pero esto se deja a cada una de las implementaciones de software.

2.4 PROGRAMACIÓN ORIENTADA A OBJETOS

La orientación a objetos es un paradigma más de programación en el que un sistema se expresa como un conjunto de objetos que interactúan entre ellos. Un paradigma de programación nos proporciona una abstracción del sistema real a algo que podemos programar y ejecutar, y puede decirse que el tipo de abstracción está directamente relacionada con los problemas que puede resolver o al menos con la facilidad con que podremos resolverlos. Mientras que el lenguaje ensamblador es una abstracción del procesador, podríamos decir que otros lenguajes de programación como BASIC o C son abstracciones del propio lenguaje ensamblador. Aunque han supuesto un importante avance sobre éste, aún obligan a los programadores a pensar en términos de la estructura del ordenador en lugar de la estructura del problema que están intentando solucionar.

La orientación a objetos proporciona las herramientas para representar elementos en el espacio del problema concreto. No impone ninguna restricción a priori, de forma que el programador no está limitado a cierta clase de problemas. Los elementos en el espacio del problema y su representación es lo que se llaman “objetos”. Se entiendo por un lenguaje orientado a objeto el que nos proporciona las siguientes prestaciones:

- Objetos: empaquetan los datos y la funcionalidad conjuntamente. Son la base para la estructura y la modularidad en un software orientado a objetos. Un objeto consta de:

- Tiempo de vida: La duración de un objeto en un programa siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado instanciación, y cuando dejan de existir se dice que son destruidos.

- Estado: Todo objeto posee un estado, definido por sus atributos. Con él se definen las propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia.

- Comportamiento: Todo objeto ha de presentar una interfaz, definida por sus métodos, para que el resto de objetos que componen los programas puedan interactuar con él.

- Abstracción: la habilidad de un programa para ignorar ciertos aspectos de la información que está manipulando. Cada objeto del sistema es un modelo de un "actor" que puede trabajar en el sistema, informar o cambiar su estado y comunicarse con otros objetos, sin revelar cómo están implementadas estas prestaciones.

- Clases: son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común. Una clase no es más que una plantilla para la creación de objetos. Cuando se crea un objeto (instanciación) se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto. Las clases presentan el estado de los objetos a los que representan mediante variables denominadas atributos. Cuando se instancia un objeto el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece

el objeto. Los métodos son las funciones mediante las que las clases representan el comportamiento de los objetos. En dichos métodos se modifican los valores de los atributos del objeto, y representan las capacidades del objeto. Desde el punto de vista de la programación estructurada, una clase se asemejaría a un módulo, los atributos a las variables globales de dicho módulo, y los métodos a las funciones del módulo.

- Encapsular: asegura que los usuarios de un objeto no pueden cambiar su información o estado de formas permitidas. Sólo los propios métodos que ofrece el objeto deben poder cambiar su información. Cada objeto ofrece una interfaz que especifica cómo el resto de objetos deben trabajar con él. El objetivo de encapsular es mantener la integridad del objeto.

- Polimorfismo: diferentes objetos pueden tener la misma interfaz para responder al mismo tipo de mensaje, y hacerlo apropiadamente según su naturaleza.

- Herencia: organiza y facilita el polimorfismo permitiendo a los objetos definirse como especializaciones de otros, que pueden compartir y extender su funcionalidad sin tener que implementarlo de nuevo. Esto suele hacerse agrupando los objetos en clases, y definiendo otras clases como extensiones de éstas, creando árboles de clases.

2.4.1 La orientación a objetos y la notación UML

En paralelo a los avances en paradigmas de programación, también han evolucionado los métodos y notaciones para modelar y diseñar los sistemas antes de

implementarlos. La orientación a objetos no es una excepción y la forma de presentar la solución al problema planteado en forma de objetos que representan las entidades involucradas es muy susceptible de ser modelada de forma visual, es decir, mediante diagramas muy fácilmente entendibles.

El Dr. James Rumbaugh fue uno de los pioneros en técnicas de modelado de clases, jerarquías y modelos funcionales, que según él “capturaban las partes esenciales de un sistema” [8].

El modelado visual de un sistema permitirá lo siguiente:

1. Identificar y capturar los procesos del problema.
2. Disponer de una herramienta de comunicación entre los analistas de la aplicación y los conocedores de las reglas de problema.
3. Expresar la complejidad de un sistema de forma entendible.
4. Definir la arquitectura del software, sus componentes implicados (interfaz de usuario, servidor de bases de datos, lógica del problema) independientemente del lenguaje de implementación que usemos.
5. Promover la reutilización, al identificar más fácilmente los sistemas implicados y los componentes.

Así pues, a partir de técnicas desarrolladas por James Rumbaugh en modelado de objetos, Ivar Jacobson en casos de uso, Grady Booch en su metodología de descripción de objetos y con la participación de empresas como HP, IBM, Oracle y Microsoft entre otras, se creó la notación UML bajo el patrocinio de la empresa Rational (adquirida por IBM) y fue aprobada por la OMG (Object Management Group) en 1997.

Las siglas UML son la abreviatura de Unified Modeling Language (Lenguaje Unificado de Modelado) y combina en una sola notación los siguientes modelos:

- Modelado de datos (similar a un diagrama entidad relación).
- Modelado de reglas de negocio y flujos de trabajo.
- Modelado de objetos.
- Modelado de componentes de un sistema.

Se ha convertido en el estándar para visualizar, especificar, construir y documentar los elementos que intervienen en un sistema software de cualquier tamaño. Puede usarse en cualquier proceso, durante todo el ciclo de vida del proyecto e independientemente de la implementación.

Hay que tener en cuenta que UML no es una metodología, es simplemente una notación para modelar nuestro sistema. Por ello, tampoco está pensado para describir la documentación de usuario, ni siquiera su interfaz gráfica. Así pues, al empezar un proyecto software, deberemos primero escoger la metodología bajo la que se va a trabajar con él, para después usar UML a lo largo del ciclo de vida que marque la metodología escogida. UML tampoco recomienda ninguna en concreto, aunque quizá el método iterativo (reseñado en la sección **¡Error! No se encuentra el origen de la referencia.**) es el más usado y el que mejor se adapta a los proyectos diseñados con el paradigma de orientación a objetos y modelados en UML.

UML ha sido y seguirá siendo el estándar de modelado orientado a objetos de los próximos años, tanto por la aprobación de los expertos en metodologías e

ingeniería del software, como por la participación de las grandes empresas de software, la aceptación por parte del OMG[1] como notación estándar y la cantidad de herramientas de modelado que lo soportan.

2.4.2 Introducción a UML

Un modelo es una abstracción de un sistema o de un problema que hay que resolver, considerando un cierto propósito o un punto de vista determinado. El modelo debe describir completamente los aspectos del sistema que son relevantes a su propósito y bajo un determinado nivel de detalle.

El código fuente es también una expresión del modelo, la más detallada y la que además implementa la funcionalidad del mismo, pero no es cómodo como herramienta de comunicación. Además, para llegar a él es conveniente desarrollar antes otras representaciones.

Un diagrama nos permitirá representar gráficamente un conjunto de elementos del modelo, a veces como un grafo con vértices conectados, y otras veces como secuencias de figuras conectadas que representen un flujo de trabajo.

Cada punto de vista del sistema (y cada nivel de detalle) podrá modelarse y ese modelo podrá representarse gráficamente. Lo que UML propone es una notación y un conjunto de diagramas que abarcan las perspectivas más relevantes del sistema.

- Diagrama de casos de uso
- Diagrama de clases

[1] ***Grupo de Gestión de Objetos*** (**OMG** de sus siglas en inglés **Object Management Group**) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

- Diagramas de comportamiento
    - Diagrama de estados
    - Diagrama de actividad
    - Diagramas de interacción
        - Diagrama de secuencia
        - Diagrama de colaboración
- Diagramas de implementación
    - Diagrama de componentes
    - Diagrama de despliegue

Estos diagramas responden a las vistas de un sistema software. Desde la definición del problema (casos de uso), la vista lógica (clases, objetos), la vista de procesos (comportamiento) y la vista de implementación y distribución.

2.5 CICLO DE VIDA DEL SOFTWARE

Se llama ciclo de vida del software a las fases por las que pasa un proyecto software desde que es concebido, hasta que está listo para usarse. Típicamente, incluye las siguientes actividades: toma de requisitos, análisis, diseño, desarrollo, pruebas (validación, aseguramiento de la calidad), instalación (implantación), uso, mantenimiento y obsolescencia.

El proyecto tiende a pasar iterativamente por estas fases, en lugar de hacerlo de forma lineal. Así pues, se han propuesto varios modelos (en cascada, incremental, evolutivo, en espiral, o concurrente, por citar algunos) para describir el progreso real del proyecto [8].

2.5.1 Modelo en Cascada

El modelo en cascada es el más simple de todos ellos y sirve de base para el resto. Simplemente asigna unas actividades a cada fase, que servirán para completarla y para proporcionar los requisitos de la siguiente. Así, el proyecto no se diseña hasta que ha sido analizado, o se desarrolla hasta que ha sido diseñado, o se prueba hasta que ha sido desarrollado, etc. Los modelos incremental y evolutivo son una variación del modelo en cascada en la que éste se aplica a subconjuntos del proyecto. Dependiendo de si los subconjuntos son partes del total (modelo incremental) o bien versiones completas pero con menos prestaciones (modelo evolutivo) se estará aplicando uno u otro.

2.5.2 Modelo en espiral

Se basa en la creación de prototipos del proyecto, que pasan por las fases anteriores, y que van acercándose sucesivamente a los objetivos finales. Así pues, nos permite examinar y validar repetidamente los requisitos y diseños del proyecto antes de acometer nuevas fases de desarrollo.

2.5.3 Modelo Incremental

Finalmente, el modelo iterativo o incremental es el que permite que las fases de análisis, diseño, desarrollo y pruebas se retro-alimenten continuamente, y que empiecen lo antes posible. Permitirá atender a posibles cambios en las necesidades del usuario o a nuevas herramientas o componentes que los desarrolladores descubran y que faciliten el diseño o proporcionen nuevas funcionalidades.

2.6 SOFTWARE LIBRE / CÓDIGO ABIERTO

2.6.1 El Software Libre

El Software Libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Para entender el concepto, debe pensarse en «libre» como en «libertad de expresión». De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- La libertad de usar el programa, con cualquier propósito (libertad 0).

- La libertad de estudiar el funcionamiento del programa, y adaptarlo a las necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.

- La libertad de distribuir copias, con lo que puede ayudar a otros (libertad 2).

- La libertad de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie (libertad 3). De igual forma que la libertad 1 el acceso al código fuente es un requisito previo.

2.6.2 La definición de Código Abierto (Open Source)

El código abierto no significa sólo acceso al código fuente. La distribución de software de código abierto debe cumplir con los siguientes criterios:

1. Libre Redistribución: la licencia no restringirá a nadie de vender o regalar el software como un componente de una distribución de software en conjunto

que contiene programas de diversas fuentes. La licencia no tendrán necesidad de un regalías u otra tasa por dicha venta.

2. Código fuente: el programa debe incluir el código fuente, y debe permitir distribución en código fuente, así como compilado. En caso de que alguna forma de un producto no se distribuya con su código fuente, debe haber una buena publicidad de como obtener el código fuente sin más que un costo razonable de reproducción, descargándolo a través de Internet sin cargo alguno. El código fuente debe ser la forma en que un programador pueda modificar el programa. Código fuente ofuscado deliberadamente no está permitido. Formas intermedias como la salida de un preprocesador o traductor no están permitidas.

3. Obras derivadas: la licencia debe permitir modificaciones y trabajos derivados, y debe permitir que sean distribuidos en los mismos términos que la licencia del software original.

4. Integridad del código fuente del autor: la licencia puede restringir la distribución del código fuente en forma modificada solamente si la licencia permite la distribución de "archivos parche" con el código fuente con el fin de modificar el programa en tiempo de compilación. La licencia debe permitir explícitamente la distribución de software construido de código fuente modificado. La licencia puede requerir a obras derivadas llevar un nombre diferente o número de versión del software original.

5. No discriminación contra las personas o grupos: la licencia no debe discriminar a ninguna persona o grupo de personas.

6. No discriminación contra áreas de trabajo: la licencia no debe restringir a nadie de hacer uso del programa en un ámbito de trabajo. Por ejemplo, no podrán restringir el programa pueda ser utilizado en un negocio, o se utilicen para la investigación genética.

7. Distribución de Licencia: los derechos adjuntos al programa deben aplicarse a quienes se redistribuye el programa sin la necesidad de ejecución de una licencia adicional de esas partes.

8. La licencia no debe ser específica para un producto: los derechos adjuntos al programa no deben depender de que el programa sea parte de una particular distribución de software. Si el programa se extrae de esa distribución y es usado o distribuido dentro de los términos de la licencia del programa, todas las partes a las que el programa se redistribuya deben tener los mismos derechos que los que se conceden en relación con la distribución de software original.

9. La licencia no debe restringir otro software: la licencia no debe poner restricciones sobre otro software que se distribuye junto con software con licencia. Por ejemplo, la licencia no debe insistir en que todos los demás programas distribuidos sobre el mismo medio deben ser software de código abierto.

10. La licencia debe ser tecnológicamente neutral: ninguna disposición de la licencia puede ser basado en la tecnología de cualquier individuo o estilo de interfaz.

2.7 GLADE 3

Glade 3 es una herramienta para el desarrollo visual de aplicaciones. Entre sus paletas de componentes destacan las de GTK y GNOME. Dentro de GNOME se ha creado un repositorio de “Widgets”. El cual se caracteriza por tener varios componentes.

Un widget es un componente gráfico con el cual el usuario interactúa, como por ejemplo, una ventana o una caja de texto. Los programadores los utilizan para construir interfaces gráficas de usuario.

Glade puede crear la interfaz de usuario de las aplicaciones, de dos formas diferentes, bien sea generando código fuente, o, cargando dinámicamente un fichero XML de descripción de la misma en tiempo de ejecución. Cualquiera de las dos alternativas está disponible para una gran cantidad de lenguajes de programación [9].

Glade 2 es capaz de generar código para implementar la interfaz visual, que se describe en XML, en C, C++, Ada 95, Perl y Eiffel. El código que genera se compila e instala con: configure, make, make install, entre otros [2]. Una de las principales diferencias de Glade 3 es que la generación de código se ha eliminado: esto se ha hecho a propósito, porque ahora generar código es obsoleto; se prefiere utilizar los archivos glade con libglade[2]. Otra diferencia es que la nueva versión fue diseñada para hacer el máximo uso de GObject, facilitando así la integración de herramientas externas y “widgets” de manipulación [10].

---

[2] Si la generación de código es necesaria, se puede hacer con otras herramientas o plugins. Solo que no es parte de Glade 3

2.7.1 Libglademm

Aunque se puede utilizar código C++ para colocar y arreglar los widgets, esto puede convertirse en un proceso tedioso y repetitivo. Y requiere una recompilación para mostrar cambios. Glade permite diseñar la disposición de los widgets visualmente y, a continuación, guardar una descripción XML del arreglo. La aplicación puede utilizar el API libglademm que carga un archivo XML en tiempo de ejecución y obtener un puntero a widget mencionados expresamente. Esto tiene las siguientes ventajas:

- Se requiere menor cantidad de código C++

- Los cambios hechos a la Interfaz de Usuario (UI) se pueden apreciar rápidamente. Por lo tanto, la interfaz se puede mejorar.
- Los diseñadores sin conocimientos de programación pueden crear y editar UIs.

Aún se necesita código de C++ para hacer frente a los cambios provocados a la interfaz de usuario por acciones del usuario, pero utilizando libglademm para el diseño básico permite centrarse en la implementación.

2.7.2 Gtkmm

Gtkmm es la interfaz oficial de C++ para la librería GUI GTK+. Cabe citar los tipesafe callbacks, y un amplio conjunto de widgets que son fácilmente extensibles a través de la herencia. Puede crear interfaces de usuario, ya sea en código o Glade con el diseñador de interfaz de usuario, utilizando libglademm.

Características:

- Utilizar la herencia para obtener Widgets u objetos personalizados.
- Controladores de señales type-safe (Prevención de errores), estándar C++.
- Polimorfismo.
- Uso de la librería estándar de C++, incluyendo cadenas, contenedores, e iteradores.
- Internacionalización con UTF-8. Es el valor predeterminado para el formato XML, está implantado como el estándar en todo intercambio de información en Internet y es compatible con diversos sistemas operativos.
- Completa gestión de memoria C++
    - Composición de Objetos
    - Eliminación automática de widgets asignados dinámicamente.
- Uso de *Espacio de nombre*s de C++ (namespaces).
- Multi-plataforma: Linux (gcc), FreeBSD (gcc), NetBSD (gcc), Solaris (gcc, Forte), Win32 (gcc, MSVC++. Net 2003), MacOS X (gcc), otros.
- Es software libre distribuido bajo la GNU Library General Public License (LGPL). Y libre de costo para desarrollo tanto de código abierto como propietario.
- Examinado, diseñado y ejecutado en público.

Gtkmm sigue el calendario oficial de publicación de GNOME Platform Bindings. Esto garantiza la estabilidad de API/ABI y novedades en forma previsible, la entrega de C++ API[3] para las API GTK+ y GNOME lo antes posible.

[3] el inglés Application Programming Interface - Interfaz de Programación de Aplicaciones

CAPÍTULO 3: DESCRIPCIÓN DEL COMPUTADOR UDO

El Computador es un sistema constituido por sub-secciones que realizan tareas específicas y que en conjunto proporcionan una implementación completa. A continuación se explica cada una de estas partes y su funcionamiento para detallar los elementos que deben estar presentes en el simulador y agregar nuevas funciones si en necesario.

3.1 EL PROCESADOR DE DATOS

El Procesador de Datos o Unidad Aritmético Lógica, visto de una manera global, debe cumplir con las siguientes funciones:

1. Tomar los datos a ser operados, desde el bus del sistema.
2. De acuerdo a los comandos recibidos desde el Procesador de Control, realizar un conjunto de operaciones aritméticas y lógicas.
3. Colocar, sobre el bus, el resultado de la operación realizada, y retener las condiciones de la operación para ser eventualmente utilizadas por la Unidad de Control

Debido a la existencia sólo de un bus común, tanto para tomar los datos como para colocar los resultados, se requiere de la presencia de registros que sean capaces de tomar la información de manera secuencial desde el bus, activados por señales de la etapa de control, así como de registros para almacenar los resultados y las

condiciones de la operación. De esta forma, el Procesador de Datos tendrá la estructura mostrada en la **Figura 5**.

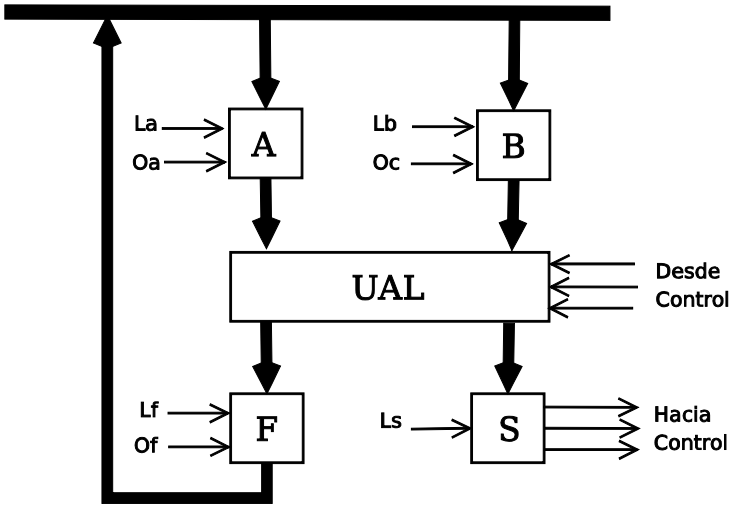

Figura 5: Procesador de Datos

El diseño del computador es de 4 bits. El bus debe ser de mayor ancho que los datos para contener direcciones y códigos de instrucción.

Los registros A y B reciben y retienen la información tomada desde el bus y constituyen la entrada de información a la Unidad Aritmético Lógica (UAL). De igual forma, dos registros de 4 bits conservan el resultado (F) y las condiciones de la operación efectuada (S). Las señales $\mathbf{L_A}$, $\mathbf{L_B}$ y $\mathbf{L_F}$ cargan la información en los registros, en el momento apropiado, la señal O_F permite que los resultados sean puestos sobre el bus, evitando la contención de información.

Las condiciones de la operación realizada, posee el significado mostrado en la **Tabla 1**.

Tabla 1: Condiciones de la UAL

Bits	**Condiciones**

3	(S) Signo. 1 negativo
2	(P)Paridad. 1 par
1	(D) Desbordamiento. 1 capacidad excedida
0	(Z) Cero. 1 el resultado es cero

La codificación de 4 bits para los tipos de operación de la UAL permite desarrollar 16 operaciones diferentes, clasificadas en la **Tabla 2**.

3.2 LA UNIDAD DE MEMORIA

La Unidad de Memoria debe almacenar el conjunto de instrucciones del programa que tiene que ejecutar la máquina, y los Datos que se procesan. De esta manera, está constituida por un conjunto ordenado de registros, entendidos como unidad fundamental de memorización.

El tipo de operación a realizar (escritura o lectura) y el momento en que deba producirse está gobernado por el sistema de control mediante señales apropiadas. De esta manera, el modelo es el mostrado en la **Figura 6**.

Tabla 2: Códigos y operaciones de la UAL

Código	**Descripción**	**RTN**
0	Suma	$F \leftarrow A + B$
1	Resta	$F \leftarrow A - B$
2	Multiplicación	$F \leftarrow A \times B$
3	División	$F \leftarrow A / B$
4	Incremento	$F \leftarrow A + 1$
5	Decremento	$F \leftarrow A - 1$

6	Transferencia	$F \leftarrow A$
7	Reset	$F \leftarrow 0$
8	Set	$F \leftarrow 1's$
9	Complemento	$F \leftarrow \overline{A}$
A	Conjunción	$F \leftarrow A \wedge B$
B	Disyunción	$F \leftarrow A \vee B$
C	Disyunción Exclusiva	$F \leftarrow A \oplus B$
D	Desplazamiento a la Izquierda	$F \leftarrow A / 2$
E	Desplazamiento a la Derecha	$F \leftarrow A \times 2$
F	Transferencia Especial	$F \leftarrow B$

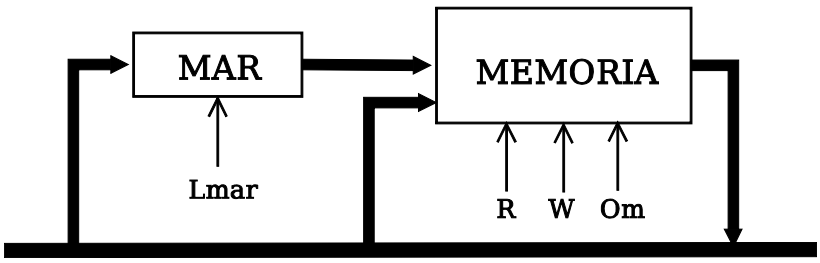

Figura 6: Unidad de Memoria

El Registro de Direccionamiento de Memoria (MAR) toma la dirección desde el bus y la retiene al momento de activarse la señal de carga $\mathbf{L_{MAR}}$, desde el sistema de control. Las señales **W**, **R** y $\mathbf{O_M}$ respectivamente permiten: escribir en la memoria (pasar un datos desde el bus hacia un registro interno), leer (pasar un dato desde un registro interno hasta el tampón de salida de la memoria) y colocar el dato del tampón de salida de la memoria sobre el bus.

3.3 ENTRADA Y SALIDA DEL COMPUTADOR

El subsistema de entrada y salida consta de sólo un registro de entrada y uno de salida, de manera similar a como están organizadas pequeñas máquinas de propósitos específicos. De esta manera, la **Figura 7** muestra la arquitectura usada.

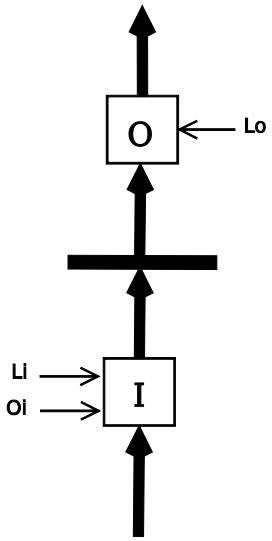

Figura 7: Sub-sistema de Entrada y Salida

El registro I está destinado al ingreso de datos externos. El registro O está destinado a la salida de datos hacia el mundo exterior. Las señales L_I y L_O activan la carga de datos desde la unidad de control. La señal O_I habilita la entrada de datos hacia el bus para evitar la contención lógica.

3.4 LA UNIDAD DE CONTROL

Internamente, de acuerdo a lo analizado en las bases teóricas, la Unidad de Control debe coordinar la información del código de operación, el tiempo de ejecución y las condiciones del procesador de datos para generar la secuencia de

señales de control. La **Figura 8** muestra el diagrama de bloques de los elementos necesarios.

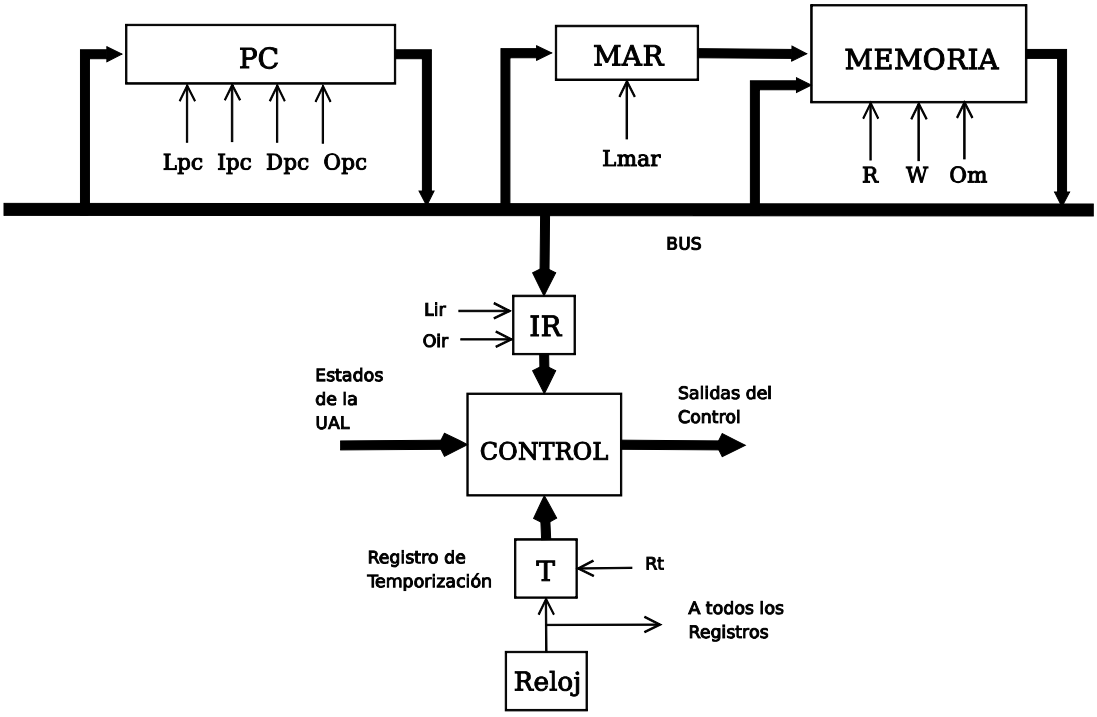

Figura 8: Diagrama básico de la Unidad de Control

La señal L_{IR} carga el registro de instrucciones desde el bus. La señal R_T repone al temporizador cada vez que una instrucción ha finalizado. El reloj es un oscilador libre que genera un tren de pulsos periódicos y marca la transferencia de estados de la máquina.

Como se ha explicado, la memoria almacena las instrucciones y datos; por lo tanto, la Unidad de Control debe saber diferenciar un tipo de información de otro. Para realizar esto, existe un registro que indica constantemente la dirección en memoria en donde se encuentra el código de la próxima instrucción a ejecutarse, llamado Contador de Programa (PC). Debido a que este registro afecta el

direccionamiento de memoria, su salida de datos debe cargarse en el Registro de Direccionamiento de Memoria (MAR).

Las señales I_{PC}, D_{PC}, L_{PC} Y O_{PC} incrementan, decrementan, cargan y habilitan al registro PC, respectivamente. Con esto se completa el diseño del computador. La **Figura 9** muestra la arquitectura definitiva.

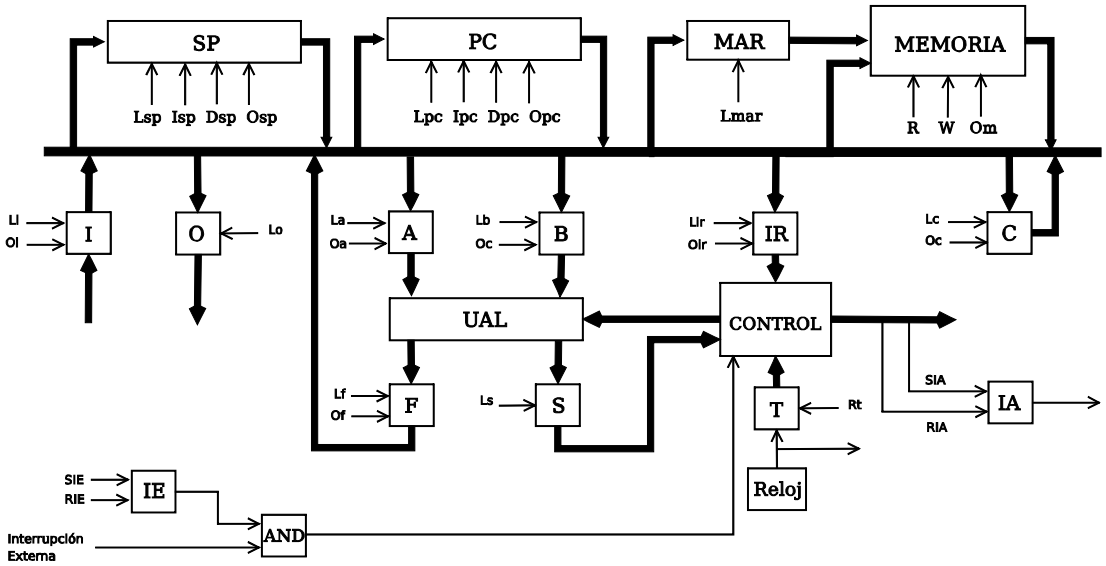

Figura 9: Arquitectura del Computador UDO

3.5 CICLO DE BÚSQUEDA PARA EL MANEJO DE INTERRUPCIONES

Normalmente la petición de interrupción se activa mediante una línea física conectada al sistema de control, que debe decidir si el servicio será concedido o ignorado.

La implementación de esta estructura de decisión se realiza con un registro de un bit, denominado Registro de habilitación de Interrupción y una compuerta AND (**Figura 9**). Es necesario un par de instrucciones para el manejo del registro IE, una

para habilitar una interrupción (SIE) y otra para impedirla (RIE). Una vez que la interrupción ha sido aceptada, el computador debe notificar esta situación al dispositivo solicitante mediante el registro denominado Aceptación de Interrupción (IA).

La solicitud de interrupción, por su naturaleza, no está sincronizada con el reloj de la máquina y se puede producir a cualquier nivel durante la ejecución de cualquier instrucción. Es necesario que termine la operación que está en proceso para poder atender una solicitud de interrupción antes de comenzar una nueva instrucción, es decir, en tiempo cero.

Si se dan todas las condiciones, el computador en lugar de cargar el registro de instrucciones con el código de la siguiente instrucción, deberá hacer las siguientes operaciones:

- Inhibir la posibilidad de nuevas interrupciones (el computador UDO solo procesa una).
- Notificar la aceptación de la interrupción.
- Salvar el PC en la pila, para garantizar la dirección de retorno correcta.
- Usar el contenido de una dirección de memoria determinada como la nueva dirección de ejecución, donde se supone debe estar la rutina de servicio de la interrupción[4].

La dirección 03 contiene el inicio del vector del Servicio de Interrupción. Este servicio esta programado para leer el estado del puerto de entrada y copiar el

[4] El procedimiento correcto es almacenar los contenidos de los registros y el estado de las banderas

contenido al puerto de salida. Consta de tres direcciones de memoria (0xb0, 0xb1 y 0xb2) donde cada celda contiene una instrucción como se muestra en la **Tabla 3**.

Tabla 3 Instrucciones del Servicio de Interrupción

Posición de Memoria	Instrucción
0xb0	IN
0xb1	MOV O,I
0xb2	IRET

La posición del vector y del conjunto de instrucciones impone una restricción de memoria para un programa que implemente el servicio de interrupción. Para cualquier otro programa que no necesite la interrupción; las celdas de almacenamiento pueden ser sustituidas con partes del programa escrito por el programador.

3.6 CONJUNTO DE INSTRUCCIONES

Como se ha mencionado, la primera tarea del Procesador de Control es llevar el código de la instrucción a ejecutar desde la memoria y depositarlo en el Registro de Instrucciones. Tarea que es común a todas las instrucciones, y recibe el nombre de Ciclo de Búsqueda.

El Ciclo de Búsqueda debe iniciarse en tiempo cero, sin conocer el código específico de cada operación, y está formado por las siguientes actividades:

1. Copiar el contenido del PC a MAR, preparando la memoria con la dirección donde se encuentra el código de la instrucción.
2. Copiar el contenido de la memoria, direccionada por MAR, hacia el IR. Momento a partir del cual el Control conoce la instrucción que se ejecutará.

3. Preparar el PC para que apunte al código de la siguiente instrucción.

Estos pasos deben realizarse en orden debido a restricciones en el uso del bus. La **Tabla 4** resume los pasos empleando Notación de Transferencia entre Registros (RTN), señalando los controles que deben estar activos.

Tabla 4: Ciclo de Búsqueda de las Instrucciones

T	Operaciones	Controles
0	$MAR \leftarrow PC$	L_{MAR},O_{PC}
1	$IR \leftarrow M; PC \leftarrow PC + 1$	L_{IR},R,O_{M},I_{PC}

La instrucción no se ha ejecutado, sólo se ha copiado su código dentro del Control y se preparó el PC para que apunte a lo que, se asume, será la siguiente instrucción.

Las instrucciones tienen asignados un código que reconoce el computador, y un grupo de caracteres que la identifican con facilidad en relación al objetivo de la instrucción, conocido como Mnemónico. Los Códigos de Operación son de ocho bits y están expresados en Hexadecimal. Con esto, el sistema es capaz de reconocer hasta 256 instrucciones y obliga a que el bus tenga un ancho de 8 bits. La **Tabla 5** muestra el conjunto de instrucciones del computador y su código.

Tabla 5: Conjunto de Instrucciones y Micro-Operaciones

CO	Mnemónico	P	C	SPDZ	T	Micro Operaciones
0	NOP	1	3		3	$T \leftarrow 0$
1	MOV A,F	1	3		3	$A \leftarrow F; T \leftarrow 0$
2	MOV B,F	1	3		3	$B \leftarrow F; T \leftarrow 0$
3	MOV F,A	1	3		3	$F \leftarrow A; T \leftarrow 0$

4	MOV F,B	1	3		3	F ← B; T ← 0
5	MOV O,F	1	3		3	O ← F; T ← 0
6	MOV A,I	1	3		3	A ← I; T ← 0
7	MOV B,I	1	3		3	B ← I; T ← 0
8	MOV O,I	1	3		3	O ← I; T ← 0
10	LDI A,Dato	2	4		2 3	MAR ← PC A ← M; PC ← PC + 1; T ← 0
11	LDI B,Dato	2	4		2 3	MAR ← PC B ← M; PC ← PC + 1; T ← 0
12	LDI O,Dato	2	4		2 3	MAR ← PC O ← M; PC ← PC + 1; T ← 0
13	LDI SP,Dir	2	4		2	MAR ← PC SP ← M; PC ← PC + 1; T ←
14	MOV C,I	1	3		3	C ← 1; T ← 0
15	MOV C,F	1	3		3	C ← F; T ← 0
16	LDI C, Dato	2	4		2	MAR ← PC C ← M; PC ← PC + 1; T ←
20	SUM	1	3	XXXX	2	F ← A + B; T ← 0
21	SUMA Dat1,Dat2	3	7	XXXX	2 3 4 5 6	MAR ← PC A ← M; PC ← PC + 1 MAR ← PC B ← M; PC ← PC + 1 F ← A + B; T ← 0
22	RES	1	3	XXXX	2	F ← A − B; T ← 0
23	RESTA Dat1,Dat2	3	7	XXXX	2 3 4 5 6	MAR ← PC A ← M; PC ← PC + 1 MAR ← PC B ← M; PC ← PC + 1 F ← A − B; T ← 0
24	MUL	1	3	XXXX	2	F ← A × B; T ←
26	DIV	1	3	XXXX	2	F ← A / B; T ←
28	INC	1	3	XXXX	2	F ← A + 1; T ←

29	DEC	1	3	XXXX	2	$F \leftarrow A - 1;\ T \leftarrow$
2A	TRAA	1	3	XXXX	2	$F \leftarrow A;\ T \leftarrow 0$
2B	RESET	1	3	XXXX	2	$F \leftarrow 0;\ T \leftarrow 0$
2C	SET	1	3	XXXX	2	$F \leftarrow 1;\ T \leftarrow 0$
2D	COMP	1	3	XXXX	2	$F \leftarrow \overline{A};\ T \leftarrow 0$
2E	TRAB	1	3	XXXX	2	$F \leftarrow B;\ S \leftarrow B;\ T \leftarrow 0$
30	IN	1	4		2	$I \leftarrow Dato;\ T \leftarrow 0$
31	INA	1	4		2	$I \leftarrow Dato$
					3	$A \leftarrow I;\ T \leftarrow 0$
32	INB	1	4		2	$I \leftarrow Dato$
					3	$B \leftarrow I;\ T \leftarrow 0$
33	OUT	1	3		2	$O \leftarrow F;\ T \leftarrow 0$
40	JMP Dir	2	4		2	$MAR \leftarrow PC$
						$PC \leftarrow M;\ T \leftarrow 0$
41	JMPS Dir	2	4/3	1XXX	2	$MAR \leftarrow PC$
					3	$PC \leftarrow M;\ T \leftarrow 0$
				0XXX	2	$PC \leftarrow PC + 1;\ T \leftarrow 0$
42	JMPP Dir	2	4/3	X1XX	2	$MAR \leftarrow PC$
					3	$PC \leftarrow M;\ T \leftarrow 0$
				X0XX	2	$PC \leftarrow PC + 1;\ T \leftarrow 0$
43	JMPD Dir	2	4/3	XX1X	2	$MAR \leftarrow PC$
					3	$PC \leftarrow M;\ T \leftarrow 0$
				XX0X	2	$PC \leftarrow PC + 1;\ T \leftarrow 0$
44	JMPZ Dir	2	4/3	XXX1	2	$MAR \leftarrow PC$
					3	$PC \leftarrow M;\ T \leftarrow 0$
				XXX0	2	$PC \leftarrow PC + 1;\ T \leftarrow 0$
47	CALL Dir	2	7		2	$MAR \leftarrow SP;\ PC \leftarrow PC + 1$
					3	$M \leftarrow PC;\ SP \leftarrow SP - 1$
					4	$PC \leftarrow PC - 1$
					5	$MAR \leftarrow PC$
					6	$PC \leftarrow M;\ T \leftarrow 0$
48	RET	1	5		2	$SP \leftarrow SP + 1$

					3	MAR ← SP
					4	PC ← M; T ← 0
49	CALLS Dir	2	7/3	1XXX	2	MAR ← SP; PC ← PC + 1
				1XXX	3	M ← F; SP ← SP - 1
				1XXX	4	PC ← PC - 1
				1XXX	5	MAR ← PC
				1XXX	6	PC ← M; T ← 0
				0XXX	2	PC ← PC + 1; T ← 0
50	PUSH F	1	4		2	MAR ← SP
						M ← F; SP ← SP - 1; T ←
51	POP A	1	5		2	SP ← SP + 1
					3	MAR ← SP
					4	A ← M; T ← 0
60	STI F,Dir	2	5		2	MAR ← PC
					3	MAR ← M
					4	M ← F; PC ← PC + 1; T ←
61	STI I,Dir	2	4		2	MAR ← PC
					3	MAR ← M
					4	M ← I; PC ← PC + 1; T ← 0
70	SIE	1	3		2	IE ← 1; T ← 0
71	RIE	1	3		2	IE ← 0; T ← 0
72	IRET	1	5		2	SP ← SP + 1
					3	MAR ← SP
					4	PC ← M; IA ← 0; T ← 0
80	AND	1	3	XXXX	2	F ← A ∧ 2; T ← 0
81	OR	1	3	XXXX	2	F ← A ∨ 2; T ← 0
82	XOR	1	3	XXXX	2	F ← A ⊕ 2; T ← 0
83	SHL	1	3	XXXX	2	F ← A / 2; T ← 0
84	SHR	1	3	XXXX	2	F ← A × 2; T ← 0
FF	HALT	1	3		2	PC ← PC - 1; T 0

El conjunto de instrucciones o ISA (del inglés Instruction Set Architecture) es una especificación que detalla las instrucciones que el ordenador puede entender y ejecutar, el conjunto de todos los comandos implementados por este diseño particular. Se describen los aspectos del procesador generalmente visibles a un programador, incluyendo los tipos de datos nativos, las instrucciones, los registros, la arquitectura de memoria y las interrupciones, entre otros aspectos.

La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la micro arquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Entre estos elementos se encuentran las microinstrucciones mostradas en la **Tabla 5**.

3.7 SIMULADOR CPU-UDO98

Es la versión actual utilizada en clases para mostrar la arquitectura de un computador. Se destaca la flexibilidad y “realismo” porque las clases del programa se corresponden con la arquitectura de registros de la máquina, se logran diferentes niveles funcionales con solo modificar los archivos de datos de la ALU y el CONTROL, los que a su vez pueden se empleados de manera directa para programar las memorias de solo lectura que forman el prototipo físico real.

El sistema actual cubre las metas pedagógicas pero presentas algunas faltas que impiden aprovechar todas sus capacidades, entre las cuales se destacan: El programa no cuenta con controles ActiveX o llamadas a API de Windows para abrir y cargar programas de máquina sin necesidad de reescribir constantemente el archivo PROGRAMA.UDO, cada vez que se requiere simular un nuevo programa [1]. No cuenta con un compilador que permita escribir programas en lenguaje ensamblador, en lugar de hacerlo en código de máquina. Existe un registro “C” de propósito general en la interfaz del programa que no cuenta con una instrucción para su uso. La

instrucción de interrupción no realiza el almacenamiento del contador de programa ni el salto a la rutina de interrupción.

CAPÍTULO 4: DESARROLLO DEL SIMULADOR

4.1 Modelos basados en UML

Después de analizar los modelos, sus características y cotejarlos con las necesidades el proyecto se estableció como modelo de desarrollo el iterativo porque permite las fase de análisis, diseño, desarrollo y prueba se realimenten continuamente, y que empiecen lo antes posible. Permite atender a posibles cambios en las necesidades del usuario o a nuevas herramientas o componentes que los desarrolladores descubran.

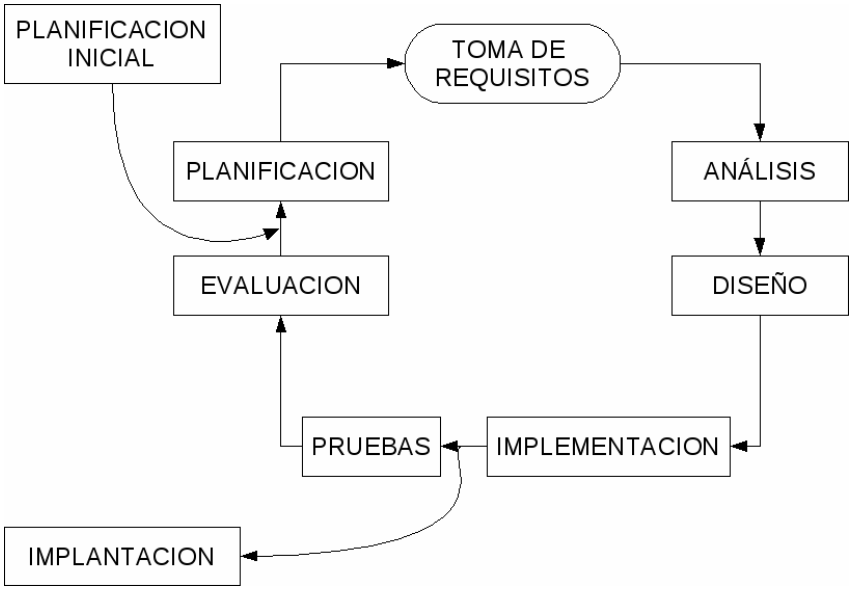

Figura 10: Modelo iterativo del ciclo de vida del proyecto

Con la selección de este modelo (**Figura 10**) se trató de obtener rápidamente una versión funcional del software, y añadirle prestaciones a partir de lo que se ha aprendido en la versión anterior. El aprendizaje proviene tanto del desarrollo anterior, como del uso del software, si es posible. En este tipo de desarrollo es imprescindible

establecer una lista de control del proyecto, donde se va registrando las funcionalidades que faltan por implementar. Y que proporciona las bases para cada nueva iteración.

Este último método es el más usado (aunque sea involuntariamente) en proyectos de software libre, por la propia naturaleza cambiante de los requisitos o la aportación constante de nuevos colaboradores.

4.1.1 Diagramas

Los diagramas fueron realizados con herramientas especializadas en la creación de proyectos con UML, específicamente BOUML y UMBRELLO.

Para iniciar el proceso de descripción de los requisitos del programa se implementó el diagrama de casos de uso. El valor verdadero de un caso de uso reposa en dos áreas [11]:

- La descripción escrita del comportamiento del sistema al afrontar una tarea o un requisito de negocio. Esta descripción se enfoca en el valor suministrado por el sistema a entidades externas tales como usuarios humanos u otros sistemas.

- La posición o contexto del caso de uso entre otros casos de uso. Dado que es un mecanismo de organización, un conjunto de casos de usos coherentes y consistentes, promueve una imagen fácil del comportamiento del sistema, un entendimiento común entre el cliente/propietario/usuario y el equipo de desarrollo.

El diagrama de la **Figura 11** describe la funcionalidad del sistema. Los casos de uso están representados por elipses y los actores están representados por las figuras humanas. El actor Estudiante necesita crear código y utilizar el programa. El sistema puede ser modificado por los estudiantes, representados como desarrolladores, y por el profesor. Al utilizar el programa, se incluyen tres casos de usos propios del programa en funcionamiento.

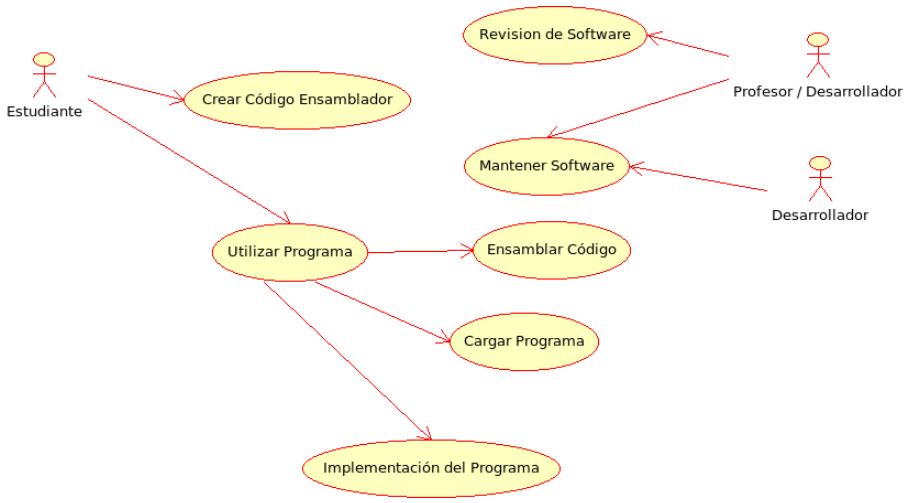

Figura 11: Diagrama de casos del sistema

El diagrama de clases esta definido por cada clase utilizada en el programa. Junto con la definición de los atributos y los métodos utilizados por cada una de ellas como se observa en la **Figura 12**. Algunos de los métodos presentes en el gráfico, agregados al momento de establecer el diagrama para el prototipo, corresponden a funciones de la versión anterior del simulador. Otros son añadidos como parte del proceso de diseño e implementación del ciclo de vida del software.

En este caso el diagrama está conectado por relaciones de dependencia entre las tres clases utilizadas por VentanaPrototipo; porque algunos de sus métodos y

propiedades utilizan las funciones de Ensamblador, UnidadControl y UAL para realizar los pasos del programa.

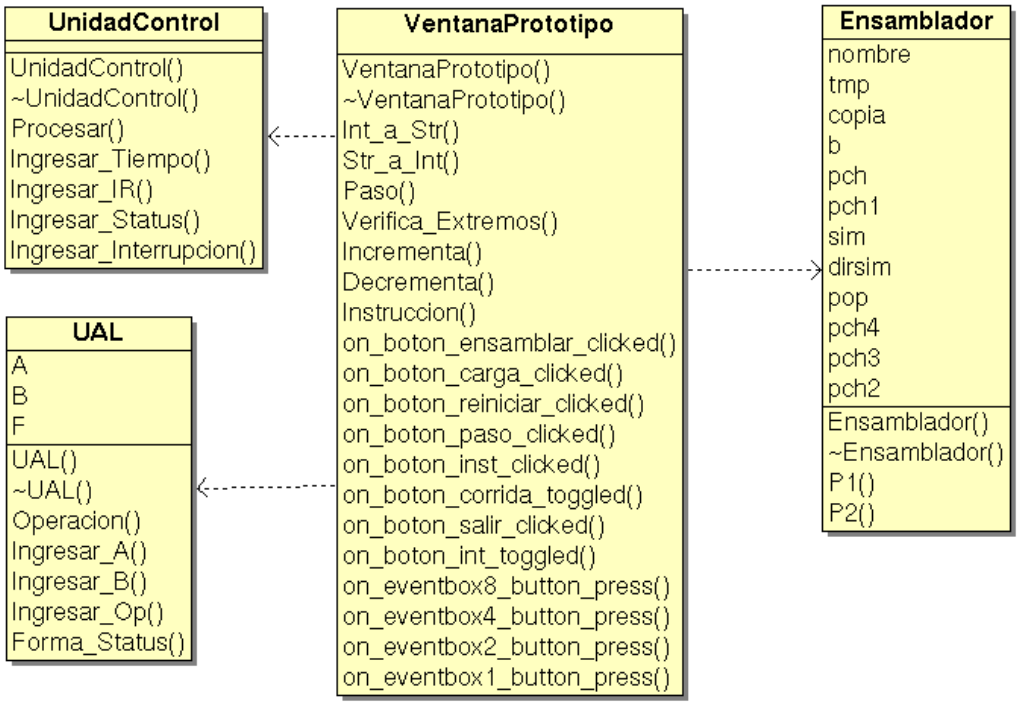

Figura 12: Diagrama de Clases del Prototipo del simulador

Basado en el diagrama de clases y el código del simulador se creó el diagrama de actividades de cada una de los objetos del programa. En muchos aspectos, los diagramas de actividades son el equivalente de la orientación a objetos de los diagramas de flujo y los DFD del desarrollo estructurado[5]. Su uso es principalmente la exploración y representación de la lógica comprendida en operaciones complejas, reglas de negocio, casos de uso o procesos software. De forma similar a los tradicionales diagramas de flujo, todo diagrama de actividades tiene un punto de

[5] Diagrama de flujo de datos (DFD por sus siglas en español e ingles) es una representación gráfica del "flujo" de datos a través de un sistema de información. También se puede utilizar para la visualización de procesamiento de datos -diseño estructurado-

partida y un final. Las actividades representarán cada paso importante que se produce en el proceso que se está modelando (puede representar un caso de uso o bien un conjunto de ellos).

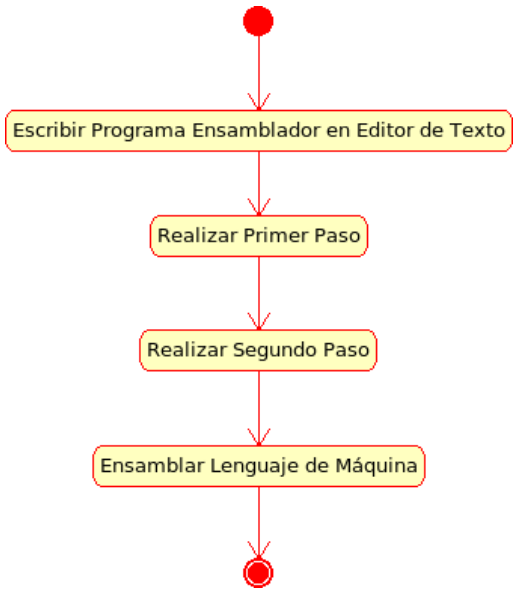

Figura 13: Diagrama de Actividades del objeto Ensamblador

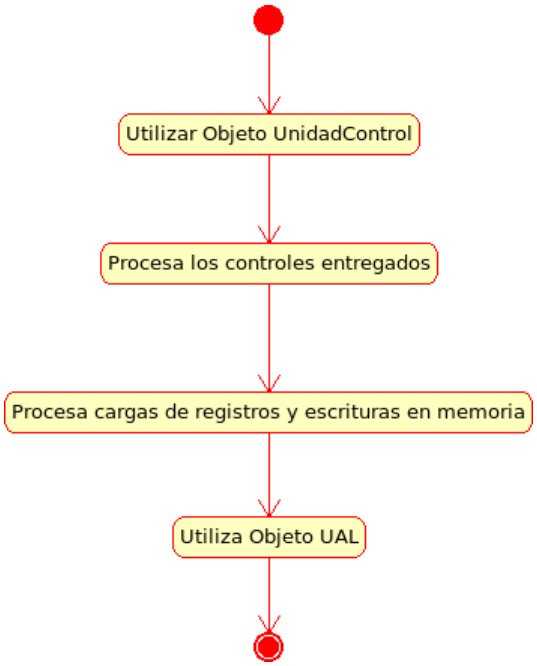

Figura 14: Diagrama de Actividades de un paso del simulador

Figura 15: Diagrama de Actividades de la UAL

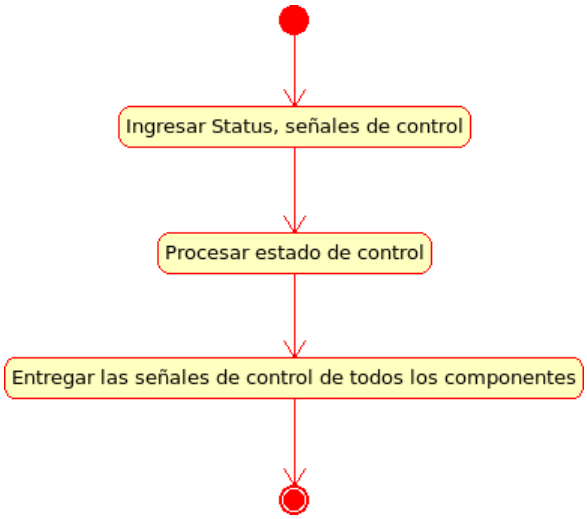

Figura 16: Diagrama de Actividades de la Unidad de Control

4.2 Elaboración de la estructura del ensamblador

Basado en los modelos de dos pasos de los ensambladores mostrados en las bases teóricas y los requisitos de nuevas funciones para el simulador. Se creó un pseudo-código para explicar el funcionamiento del sistema de ensamblaje a utilizar.

Las funciones principales del ensamblador son:

- Traducir las instrucciones mnemónicas a sus equivalentes en lenguaje de máquina.
- Asignar direcciones de memoria a las etiquetas simbólicas utilizadas por el programador.

```
PASO 1
Abrir archivo *.UDO
Crear archivo TABSIMB.DAT
Crear archivo INTERM.DAT
Leer la primera línea
```

```
Tomar la dirección de inicio
Mientras (línea ≠ final de archivo)
{
 Si (línea es comentario)
      Ignorar
      Si (hay comentario en línea)
           Ignorar
      Dividir en palabras la línea
      Mientras (palabra ≠ final de línea)
      {
           Si (palabra es una etiqueta)
          Si (etiqueta existe)
                Error etiqueta duplicada
          Sino
                Almacenar etiqueta
                Almacenar posición de memoria
           Si (palabra es instrucción)
           Comparar con lista de instrucciones
           Almacenar longitud de instrucción
    }
}
Cerrar todos los archivos.
```

Al inicio del primer paso se abre el archivo con extensión .UDO escrito en cualquier editor de texto disponible. Se crean dos archivos, uno donde se almacena una tabla con las etiquetas y las direcciones a donde estas apuntan; y otro archivo intermedio con el código modificado para la lectura en el segundo paso. Toma la primera línea del archivo *.UDO que contiene una directiva de ensamblador que indica la dirección de inicio, a partir de la cual se escribirá la lista de instrucciones en

la memoria. A continuación, lee cada una de las líneas del programa e ignora las líneas que son comentarios o cualquier comentario dentro de una línea. En cada línea realiza lo siguiente:

- Asigna direcciones a todas las declaraciones.
- Guarda los valores (direcciones) que se asignan a todas las etiquetas para su uso en el paso 2 (Maneja referencias hacia adelante).
- Realiza el tratamiento de la directiva de ensamblador.

Este paso funciona como un analizador léxico (escáner), que es un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de *tokens* [12]. El analizador contiene reglas para indicar símbolos (específicamente "$") que indican el comienzo de un nuevo *token*. Estos *tokens* sirven para una posterior etapa del proceso de traducción, siendo la entrada para el *parser* o analizador sintáctico (Segundo paso).

El siguiente estado es el análisis sintáctico lo que significa comprobar que los *tokens* forman una expresión válida, esto se hace usualmente usando una gramática libre de contexto que define recursivamente componentes que pueden aparecer en una expresión y el orden en que estos deben aparecer. Las reglas que definen un lenguaje de programación no siempre se pueden expresar usando únicamente una gramática libre de contexto, por ejemplo la validación de tipos y la declaración correcta de identificadores. Estas reglas pueden expresarse formalmente usando gramáticas de atributos [13].

La fase final es el análisis semántico, que trabaja en las implicaciones de la expresión ya validada y realiza las actuaciones pertinentes. En el caso de cualquier

compilador: generar código. Las gramáticas de atributos pueden ser usadas también para definir estas acciones.

```
PASO 2
Abrir archivo TABSIMB.DAT
Abrir archivo INTERM.DAT
Crear archivo SALIDA.DAT
Leer primera línea de archivo intermedio
Crear lista temporal de etiquetas y sus direcciones
Mientras (línea ≠ final de archivo)
{
 Limpiar variables temporales de línea

      Dividir en palabras la línea
      Mientras (palabra ≠ final de línea)
      {
      Ignorar etiquetas y posiciones de memoria
           Si (palabra es instrucción)
                Tomar instrucción
                Buscar operando
      }
      Ensamblar código objeto de instrucción
}
Cerrar todos los archivos.
```

El paso 2 realiza lo siguiente:

- Ensambla las instrucciones.
- Asigna la dirección de memoria definida por la directiva INICIO.

- Escribe el código objeto del programa y la lista de ensamblaje.

El código es una adaptación de los diagramas de flujo explicados en las bases teóricas para ser escritos en C++ con las funciones de las librerías estándar del lenguaje. La **Figura** detalla el manejo de las tablas creadas y los pasos del proceso.

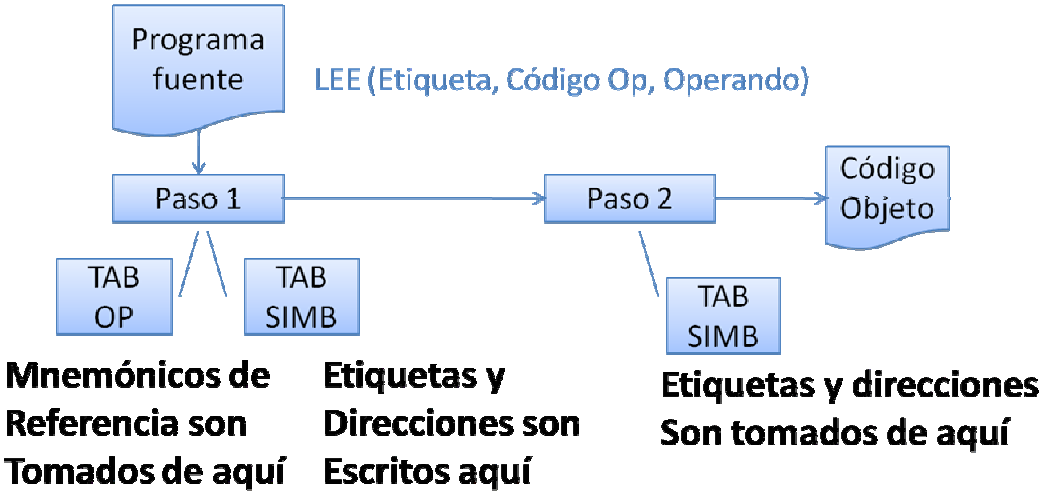

Figura 16: Implementación de un Ensamblador de dos pasos

4.3 Descripción de la Unidad Aritmético - lógica

El modelo de UAL utilizado en la versión anterior del software está basado en un modelo VLSI combinatorio. La información del comportamiento de este subsistema está contenido en un archivo *.DAT. Ahora es sustituida por una clase llamada UAL. Esta clase permite interactuar a través de los métodos del objeto “ual” con el resto del programa.

El diagrama de actividades detallado en la sección 0 contiene el paso “Ingresar A,B,Op”, estos valores de registros son funciones públicas de la clase para que se realice la operación (que también es una función pública).

```
int Operacion(int&,int&);  //Funciones de acceso a la
                           //clase
void Ingresar_A(int);
void Ingresar_B(int);
int Ingresar_Op(int);

De acuerdo con el código de operación hacer
{
    Resultado F = AθB;
    Status S = Banderas(F);
}
```

La función de entregar estatus es privada dentro de la clase UAL, al igual que los valores utilizados por esta función.

```
Forma_Status(Valor resultado F)
{
    int resultado;
    Si (Valor==0) Activa bandera Z;
    Sino          desactiva bandera Z;

    Si (Valor<0)  Activa bandera S;
    Sino          desactiva bandera S;

    Si (abs(Valor)>15) Activa bandera D;
    Sino              desactiva bandera D;

    Si (Numero de bits de Valor es par)
         Activa bandera P;
```

```
    Sino desactiva bandera P;

    Resultado=8*S+4*P+2*D+1*Z;
}
```

4.4 Descripción de la Unidad de Control

La UC es un circuito combinatorio que genera un conjunto de salidas como respuesta al a combinación de entradas (IR, T y banderas). El complejo circuito es descrito por una extensa serie de condiciones que determinan en que estado se encuentra el sistema y que líneas deben activarse.

Como muestra el diagrama de actividades, se debe procesar el estado del control. Que puede ser especificado a través de pseudo-código como abstracción del diagrama.

```
¿Existe interrupción?
{
    Controles para el ciclo de búsqueda
    Con interrupción
}
Sino
Ciclo de búsqueda normal
{
    switch(tiempo)
    {
        Tiempo 0
        Tiempo 1
        Tiempo 2
            switch (instrucción)
```

```
                NOP
                MOV
                .
                .
                .
                HALT
        }
}
```

La ejecución de la instrucción siempre ocurre en el tiempo 2, luego de realizado el ciclo de búsqueda. Los pasos deben completarse en ese orden debido a las restricciones del bus.

4.5 Interfaz del simulador

La disposición de los elementos en pantalla se realizó completamente con Glade 3. Que ofreció la ventaja de poder cambiar y agregar elementos al diseño sin necesidad de compilar de nuevo el programa. El árbol de *widgets* permitió visualizar de forma jerárquica los *Widgets*, tanto padres como hijos, entre otros, y la posibilidad de copiar, mover o pegar los elementos. En la **Figura** se muestra el árbol con parte de la estructura del simulador.

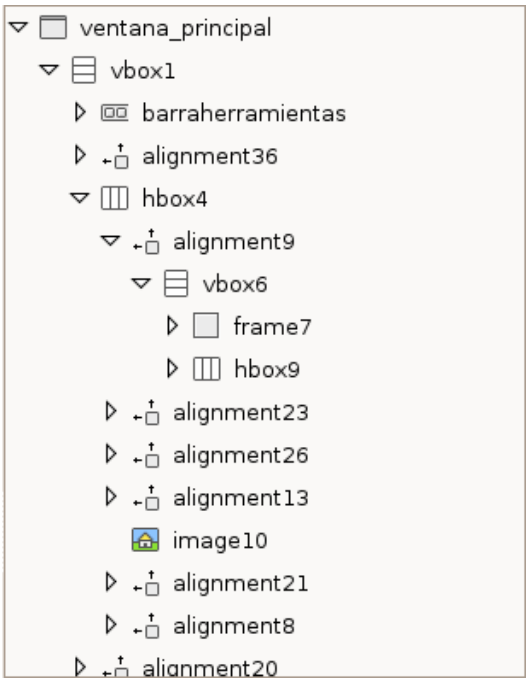

Figura 17: Elementos principales de la estructura del simulador

La colocación de los distintos widgets se organizó mediante contenedores que son elementos de interfaz que permiten agrupar y empaquetar los widgets. Los contenedores son necesarios porque todos los elementos que forman parte de la interfaz deben encontrarse empaquetados. Esta técnica permite que la adaptación del tamaño de los widgets sea transparente al momento de hacer el diseño. Utilizando diferentes combinaciones entre contenedores se pudo completar una compleja y completa interfaz de usuario. La figura muestra la estructura del GUI al momento de diseñarla mediante Glade 3.

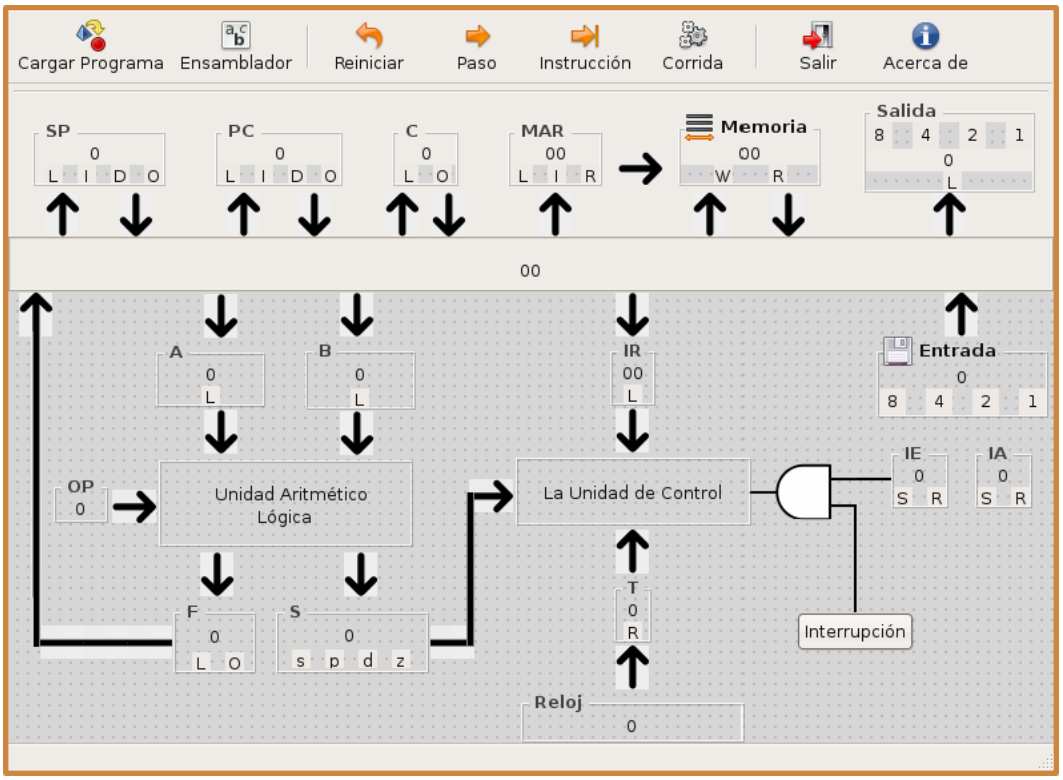

Figura 17: Estructura del prototipo visualizado durante la ejecución del Diseñador de Interfaces

A cada elemento descrito en el archivo XML (.Glade) se le asignó un puntero y en algunos casos un controlador de señales, específicamente los eventos de presionar botones, en el header (gui.h) del archivo gui.cpp:

```
    virtual        void        on_boton_ensamblar_clicked();
//Controlador
                                                        //de señal
    Gtk::ToolButton*      pboton_ensa;              //Puntero
correspondiente
                                          //al boton ensamblar

    //Otro ejemplo:
```

```
    Gtk::Label* preg_sp;            //Puntero a registros
SP
```

Se accede a cada elemento del archivo glade a través de la función get_widget, este proceso es el mismo para cada uno de los demás componentes de la interfaz.

```
    glade_xml->get_widget("boton_ensamblar",pboton_ensa);
```

Luego se conectó a cada componente con la función correspondiente para que al presionar un botón haga lo que se solicite. Otros elementos utilizan métodos propios para realizar acciones específicas:

```
    pboton_ensa->signal_clicked().connect(
    sigc::mem_fun(*this,&VentanaPrototipo::on_boton_ensam
blar_clicked) );
    //
    //Ejemplo de elementos que utilizan métodos propios:
    pbus->set_text(preg_sp->get_text());
    event_osp->set_state(Gtk::STATE_ACTIVE);
```

La utilización de la interfaz permite aprovechar las etiquetas propias de la API para almacenar los valores actuales de los registros, actuando como verdaderos asientos para retener los valores. Después de conectar las funciones están disponibles para agregar capacidades al sistema:

```
    void VentanaPrototipo::on_boton_ensamblar_clicked()
    {
       ...
    }
```

CAPÍTULO 5: USO Y PRUEBAS DEL SOFTWARE

La compilación del software es a través de g++ (GNU C++ Compiler) utilizando las librerías libgtkmm y libglademm con el siguiente comando:

```
g++ $(pkg-config --libs --cflags gtkmm-2.4) $(pkg-config --libs --cflags libglademm-2.4) gui.cpp gui_principal.cpp ual.cpp uc.cpp ensamblador.cpp -o CPU_UDO2008
```

Con esto se creó el ejecutable que inicia la interfaz gráfica de usuario. Como se muestra en la **Figura** .

El proceso de ensamblaje consiste en editar un archivo *.UDO en cualquier editor de texto. Dicho archivo debe conservar la siguiente estructura:

- Debe comenzar con la directiva de ensamblador INICIO *Dir* que indica la posición de memoria de inicio del programa a ensamblar. *Dir* es un número entre cero y doscientos cincuenta y cinco en decimal.
- Cada línea del programa debe contener solo una instrucción, en letras minúsculas:
    - Operador [Operando]. Ejemplo “mov a,i”
- Todas las líneas deben empezar con un espacio (tab o simple). La excepción a esta regla es iniciar la línea con una etiqueta:
    - Etiq: Operador [Operando]. Ejemplo “mov a,i”

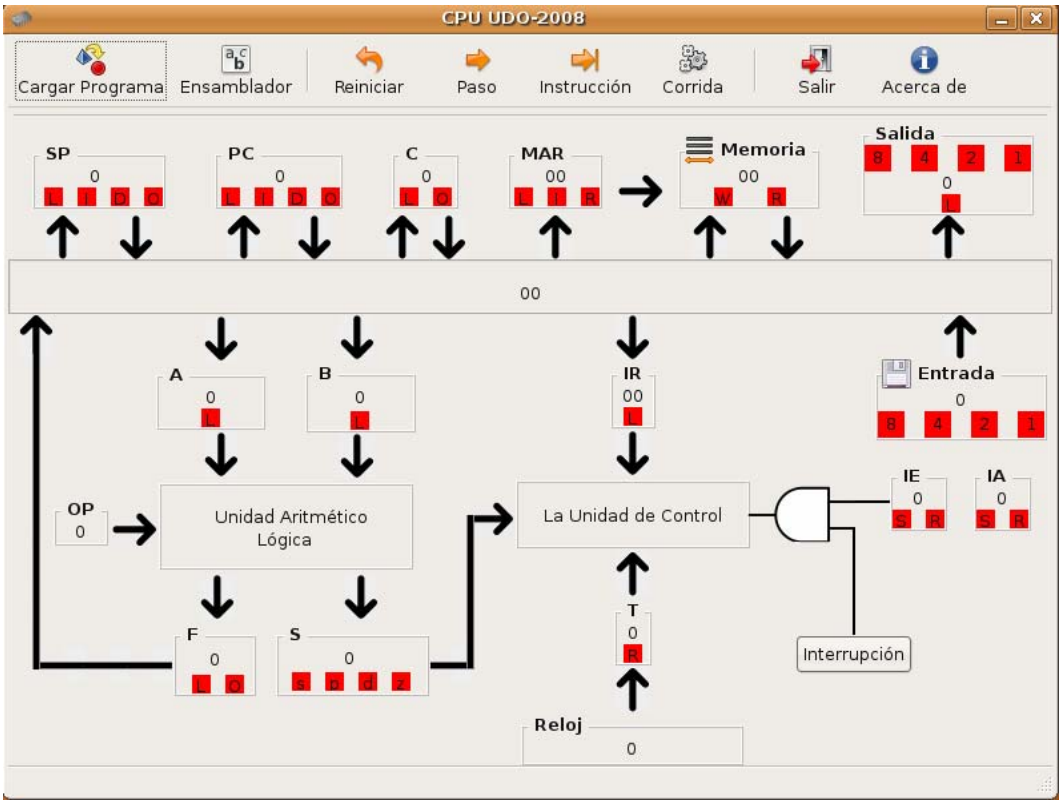

Figura 19: Interfaz de Usuario de Simulador CPU UDO2008

El propósito de estas reglas es evitar errores al compilar debido a que el ensamblador no cuenta con detección de errores.

Luego de editar un archivo se debe presionar el botón de ensamblar, donde se permite seleccionar cualquier archivo con la extensión .UDO. Con lo cual se crea el archivo de salida del ensamblador SALIDA.DAT junto con otros dos archivo: Tabla de símbolos (TABSIMB.DAT) y archivo intermedio (INTERM.DAT) que es utilizado por el paso 2 durante el proceso.

El resto de los botones tiene las siguientes funciones:

- Cargar Programa: carga el código objeto del archivo SALIDA.DAT a las posiciones de memoria correspondientes en el simulador.
- Reiniciar: vacía todos los registros, la memoria.
- Paso: realiza un ciclo de reloj del simulador.
- Instrucción: corre un ciclo de máquina del programa, donde cada instrucción realiza los ciclos de reloj necesarios.
- Corrida: ejecuta continuamente el contenido de la memoria.

Por otra parte, el usuario también puede generar interrupciones externas mediante la activación del botón de interrupción asociado a la señal de interrupción con el registro habilitador de interrupciones (IE), a través de la compuerta AND.

5.1 Prueba del software

Se realizaron una serie de programas para garantizar que todas las funciones e instrucciones del simulador funcionen correctamente:

5.1.1 PROGRAMA EJEMPLO N°1:

```
INICIO  10
inic: ina ;esta es una prueba de comentario
      ldi b,0h
      xor
      jmpz inic
      ina
      ldi
      b,1h
      res
      jmpz cab
      halt
;
```

```
cab:  ldi a,0h    ;otra línea de comentario
      traa
sgt0: mov o,f
      inc
      mov b,f
      ina
      traa
      jmpz inic
      mov f,b
      mov a,f
      jmp sgt0
; una línea completa de comentario
```

Este programa generó los archivos descritos en la sección anterior que permiten asegurar que el ensamblaje fue correcto:

5.1.1.1 INTERM.DAT:

```
INICIO  10
0xa  inic:$ina$$$
0xb  $$ldi$b,0h$
0xd  $$xor$
0xe  $$jmpz$inic$
0x10 $$ina$
0x11 $$ldi$$b,1h$
0x13 $$res$$
0x14 $$jmpz$cab$
0x16 $$halt$
0x17 cab:$ldi$$a,0h$
0x19 $$traa$
```

```
0x1a sgt0:$mov$$o,f$
0x1b $$inc$
0x1c $$mov$$b,f$
0x1d $$ina$$
0x1e $$traa$
0x1f $$jmpz$inic$
0x21 $$mov$$f,b$
0x22 $$mov$$a,f$$
0x23 $$jmp$$sgt0$
```

El archivo intermedio se encarga de arreglar los elementos presentados en *.UDO para que el paso 2 detecte el operador y los operandos. Se puede observar que los *tokens* ($) están presentes antes y después de cada símbolo que puede ser reconocido por el ensamblador. Al principio de cada línea indica la posición de memoria de inicio de cada instrucción.

5.1.1.2 TABSIMB.DAT

```
inic:  0xa
cab:   0x17
sgt0:  0x1a
```

Se muestra que el programa ha detectado correctamente las etiquetas presentes en lenguaje ensamblador. El paso 1 detecta si existen etiquetas duplicadas, error que puede ser mostrado en consola.

5.1.1.3 SALIDA.DAT

```
10
0x31
```

```
0x11
0x0
0x82
0x44
0xa
0x31
0x11
0x1
0x22
0x44
0x17
0xff
0x10
0x0
0x2a
0x5
0x28
0x2
0x31
0x2a
0x44
0xa
0x4
0x1
0x40
0x1a
```

Se muestra el contenido del archivo final luego de los dos pasos. La primera línea contiene un número decimal que representa la dirección de inicio de

almacenamiento de las instrucciones y después cada línea representa un código de instrucción u operando(s).

Otro conjunto de programas que contienen el resto de las instrucciones del simulador fueron analizados de la misma manera y generaron los mismos archivos con la información correcta de acuerdo a lo descrito por el lenguaje ensamblador.

5.1.2 Ejecución del código de prueba

Al ejecutar el código de prueba de los diferentes programas se verificó el correcto funcionamiento de cada uno de los elementos de señalización de los registros. Así como también el almacenamiento de las instrucciones en memoria, los cómputos realizados por la unidad aritmético lógica y el comportamiento de la señal de interrupción.

El botón de reinicio restablece correctamente todos los valores de los registros a cero, y lleva todas las señales a un estado apagado.

CAPÍTULO 6: CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- Se desarrolló la aplicación bajo herramientas de software libre (GPL y afines). Esto permitió tener acceso a código de los sistemas utilizados para modificar funciones de algunos objetos -específicamente, con GTK+ y Glade- y adaptarlo a la forma necesaria para el simulador.

- La descripción de la arquitectura del computador se realizó a través del modelo existente con la extensión de algunas características y las mejoras de otras.

Se cambió la plataforma de software del sistema a GNU/Linux y otras herramientas computacionales de software libre. Ahora, se puede acceder al diseño implantado para aprender de él, modificarlo, sin infringir licencias o copiar ilícitamente el software; lo que constituye un beneficio social y tecnológico que está acorde con las necesidades pedagógicas de la Universidad de Oriente.

El desarrollo de la aplicación se refleja en mejoras a las funcionalidades existentes y la inclusión de nuevas características. Específicamente, la creación del compilador para el lenguaje ensamblador utilizado por el procesador. La incorporación del código objeto generado por el compilador al simulador para que ejecute un programa. Y la solución al problema del manejo de las interrupciones.

- La aplicación de pruebas en la etapa final del proyecto permitió evaluar la calidad del software, garantizando la eficiencia del mismo y la depuración del código.

6.2 Recomendaciones

- Realizar seguimiento al simulador CPU UDO-2008 a fin de implementar mejoras para extender la vida del software. Actualizándolo al constante avance tecnológico. Así como también, facilitar la instalación del sistema de simulación a diferentes distribuciones del Sistema Operativo GNU/Linux a través de los sistemas de empaquetado (deb, rpm, tarballs) y a los Sistemas Operativos de Windows.

- Extender las capacidades del ensamblador. Como la detección y notificación de errores al compilar; implementar un editor propio con detección de sintaxis e incorporarlo al simulador; extender la cantidad de directivas de ensamblador para facilitar la creación de programas.

- Se recomienda el desarrollo de aplicaciones (convencionales o web) tomando como marco de trabajo el modelado UML así como también software libre o código abierto.

BIBLIOGRAFÍA

[1]. Parraguez, Luis. *Herramientas Computacionales para la Asignatura Diseñoo de Sistemas Digitales.* Puerto La Cruz : Universidad De Oriente, 2000.

[2]. Hernández, Vinicio. *Diseño de un manejador de activos para instrumentación digital, con protocolo Hart, basado en herramientas computacionales de software libre.* Puerto La Cruz : Universidad De Oriente, 2007.

[3]. Mostafa, Abd y Hesham, El. *Fundamentals of Computer Organization and Architecture.* Hoboken, New Jersey : Jhon Wiley & Sons, Inc, 2005.

[4]. Stallings, William. *Computer Organization and Architecture.* Sexta edición. New Jersey : Prentice Hall, 2003.

[5]. Null, Linda y Lobur, Julia. *The Essentials of Computer Organization and Architecture.* 40 Tall Pine Drive Sudbury, MA 01776 : Jones and Bartlett Publishers, Inc, 2003.

[6]. Hennessy, John y Patterson, David. *Arquitectura de Computadores un enfoque cuantitativo.* 4. 500 Sansome Street, Suite 400, San Francisco, CA 94111 : Morgan Kaufmann, 2007.

[7]. Balch, Mark. *Complete Digital Design. A comprehensive guide to digital electronics and computer system arquitecture* . Chicago : McGraw-Hill, 2003.

[8]. Gibert Ginestà, Marc y González Peña, Álvaro. *Ingeniería del software en entornos de SL.* Primera edición. Av. Tibidabo, 39-43, 08035 Barcelona, España : Fundació per a la Universitat Oberta de Catalunya, 2005.

[9]. Araujo, Yorkier. *Manual GLADE Y ANJUTA.* Caracas, Venezuela : Fundación Bolivariana de Informática y Telemática (Fundabit)", 2006.

[10]. Glade. Glade - a User Interface Designer for GTK+ and GNOME. [En línea] 2008. [Citado el: 24 de Julio de 2008.] http://glade.gnome.org/.

[11]. Hurtado, Daisy y Ramirez, Luis. *Desarrollo de un software bajo plataforma web para la automatización de la sala de lectura del Departamento de Computación y Sistema de la Universidad de Oriente Núcleo Anzoátegui.* Puerto La Cruz : Universidad de Oriente, 2007.

[12]. Wkipedia, la enciclopedia libre. Analizador léxico. [En línea] 2008. [Citado el: 18 de Agosto de 2008.] http://es.wikipedia.org/wiki/Analizador_l\'exico.

[13]. Wikipedia, la enciclopedia libre. Analizador sintáctico. [En línea] 2008. [Citado el: 15 de Agosto de 2008.] http://es.wikipedia.org/wiki/Analizador_sint\'actico.

ANEXO 1: CÓDIGO COMPLETO DEL PROGRAMA

A.1 FUNCIÓN PRINCIPAL DEL PROGRAMA

```
#include <gtkmm.h>
#include <libglademm/xml.h>
#include <iostream>
#include "uc.h"
#include "ual.h"
#include "ensamblador.h"
#include "gui.h"

int main(int argc, char* argv[])
{
 //ventana_principal via Glib::RefPtr para cargar la interfaz
 Glib::RefPtr<Gnome::Glade::Xml> ventana_principal;
 Gtk::Main kit(argc, argv);

 // carga la interfaz a travez de create()
 ventana_principal = Gnome::Glade::Xml::create("gui.glade");

 // obtener la ventana principal, los widgets hijos se cargan
automaticamente
 VentanaPrototipo* ventana;
 ventana_principal->get_widget_derived("ventana_principal",
ventana);

 // inicia el evento loop
 Gtk::Main::run( *ventana );

    return 0;
}
```

A.2 FUNCIÓN PRINCIPAL DEL *GUI*

```
#include <gtkmm.h>
#include <libglademm/xml.h>
#include <fstream>
#include <iostream>
#include <ios>
#include <sstream>
#include <cstring>
#include <cstdlib>       //atoi
#include <ctime>
#include "gui.h"
#include "uc.h"
#include "ual.h"
#include "ensamblador.h"

#define PRESIONADO 1     //Boton Presionado
#define LIBERADO   0           //Boton Liberado

using namespace std;

//declaración de objetos utilizados e inicialización variables
globales
 UnidadControl uc;
 UAL ual;
 Ensamblador ensamblador;
 int temp=0; long   int   memoria[255];   int   i;int   entrada=0,
estado8, estado4, estado2, estado1;
 int z,d,p,s,output;int h;

VentanaPrototipo::VentanaPrototipo(BaseObjectType* base_object,
                  const         Glib::RefPtr<Gnome::Glade::Xml>&
glade_xml)
        : Gtk::Window(base_object)
{
 //se accede a los widgets hijos con get_widget
```

```
glade_xml->get_widget("reg_sp", preg_sp);
glade_xml->get_widget("reg_op", preg_op);
glade_xml->get_widget("reg_a", preg_a);
glade_xml->get_widget("reg_b", preg_b);
glade_xml->get_widget("reg_s", preg_s);
glade_xml->get_widget("reg_f", preg_f);
glade_xml->get_widget("reg_t", preg_t);
glade_xml->get_widget("reg_ir", preg_ir);
glade_xml->get_widget("reg_mar", preg_mar);
glade_xml->get_widget("reg_pc", preg_pc);
glade_xml->get_widget("reg_c", preg_c);
glade_xml->get_widget("reg_o", preg_o);
glade_xml->get_widget("memoria", pmem);
glade_xml->get_widget("bus", pbus);
glade_xml->get_widget("reg_i", preg_i);
glade_xml->get_widget("reg_ie", preg_ie);
glade_xml->get_widget("reg_ia", preg_ia);
glade_xml->get_widget("reg_reloj", preg_reloj);

glade_xml->get_widget("boton_ensamblar",pboton_ensa);
glade_xml->get_widget("boton_carga", pboton_carga);
glade_xml->get_widget("boton_reiniciar", pboton_reinic);
glade_xml->get_widget("boton_paso", pboton_paso);
glade_xml->get_widget("boton_inst", pboton_inst);
glade_xml->get_widget("boton_corrida", pboton_corrida);
glade_xml->get_widget("boton_salir", pboton_salir);
glade_xml->get_widget("boton_acerca", pboton_acerca);
glade_xml->get_widget("boton_int", pboton_int);

glade_xml->get_widget("eventbox8", event_8);
glade_xml->get_widget("eventbox4", event_4);
glade_xml->get_widget("eventbox2", event_2);
glade_xml->get_widget("eventbox1", event_1);
glade_xml->get_widget("eventbox_s8", event_s8);
glade_xml->get_widget("eventbox_s4", event_s4);
```

```
glade_xml->get_widget("eventbox_s2", event_s2);
glade_xml->get_widget("eventbox_s1", event_s1);

glade_xml->get_widget("eventbox_z", event_z);
glade_xml->get_widget("eventbox_d", event_d);
glade_xml->get_widget("eventbox_p", event_p);
glade_xml->get_widget("eventbox_s", event_s);

glade_xml->get_widget("eventbox_lsp", event_lsp);
glade_xml->get_widget("eventbox_isp", event_isp);
glade_xml->get_widget("eventbox_dsp", event_dsp);
glade_xml->get_widget("eventbox_osp", event_osp);

glade_xml->get_widget("eventbox_lpc", event_lpc);
glade_xml->get_widget("eventbox_ipc", event_ipc);
glade_xml->get_widget("eventbox_dpc", event_dpc);
glade_xml->get_widget("eventbox_opc", event_opc);

glade_xml->get_widget("eventbox_lc", event_lc);
glade_xml->get_widget("eventbox_oc", event_oc);

glade_xml->get_widget("eventbox_lo", event_lo);
glade_xml->get_widget("eventbox_la", event_la);
glade_xml->get_widget("eventbox_lb", event_lb);
glade_xml->get_widget("eventbox_lf", event_lf);
glade_xml->get_widget("eventbox_of", event_of);
glade_xml->get_widget("eventbox_lir", event_lir);

glade_xml->get_widget("eventbox_lmar", event_lmar);
glade_xml->get_widget("eventbox_imar", event_imar);
glade_xml->get_widget("eventbox_rmar", event_rmar);

glade_xml->get_widget("eventbox_rmem", event_rmem);
glade_xml->get_widget("eventbox_wmem", event_wmem);
```

```
      glade_xml->get_widget("eventbox_sie", event_sie);
      glade_xml->get_widget("eventbox_rie", event_rie);
      glade_xml->get_widget("eventbox_sia", event_sia);
      glade_xml->get_widget("eventbox_ria", event_ria);
      glade_xml->get_widget("eventbox_rt", event_rt);

         event_z->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_d->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_p->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_s->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_z->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
         event_d->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
         event_p->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
         event_s->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));

         event_lsp->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_isp->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_dsp->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_osp->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
         event_lsp->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
         event_isp->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
```

```
        event_dsp->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
        event_osp->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));

        event_lpc->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
        event_ipc->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
        event_dpc->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
        event_opc->modify_bg(Gtk::STATE_NORMAL,
Gdk::Color("#FF0000"));
        event_lpc->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
        event_ipc->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
        event_dpc->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));
        event_opc->modify_bg(Gtk::STATE_ACTIVE,
Gdk::Color("#1FFF00"));

    event_lc->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_oc->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lc->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_oc->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_lo->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_la->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lb->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lf->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_of->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lir->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lo->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_la->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_lb->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
```

```
    event_lf->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_of->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_lir->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

    event_lmar->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_imar->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_rmar->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_lmar->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_imar->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_rmar->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

    event_rmem->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_wmem->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_rmem->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_wmem->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

    event_sie->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_rie->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_sia->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_ria->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_rt->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_sie->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_rie->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_sia->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_ria->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
    event_rt->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

    //añade la función de boton a Gtk EventBox y otras funciones
    event_8->add_events(Gdk::BUTTON_PRESS_MASK);
    event_8->signal_button_press_event().connect
     sigc::mem_fun(*this,&VentanaPrototipo::on_eventbox8_button_pres
s) );
    event_8->set_tooltip_text("Bit 3");
    event_8->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
    event_8->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
```

```
     event_4->add_events(Gdk::BUTTON_PRESS_MASK);
     event_4->signal_button_press_event().connect(
      sigc::mem_fun(*this,&VentanaPrototipo::on_eventbox4_button_pres
s) );
     event_4->set_tooltip_text("Bit 2");
     event_4->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_4->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

     event_2->add_events(Gdk::BUTTON_PRESS_MASK);
     event_2->signal_button_press_event().connect(
      sigc::mem_fun(*this,&VentanaPrototipo::on_eventbox2_button_pres
s) );
     event_2->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_2->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

     event_1->add_events(Gdk::BUTTON_PRESS_MASK);
     event_1->signal_button_press_event().connect(
      sigc::mem_fun(*this,&VentanaPrototipo::on_eventbox1_button_pres
s) );
     event_1->set_tooltip_text("Bit 0");
     event_1->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_1->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));

     event_s8->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_s4->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_s2->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_s1->modify_bg(Gtk::STATE_NORMAL, Gdk::Color("#FF0000"));
     event_s8->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
     event_s4->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
     event_s2->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
     event_s1->modify_bg(Gtk::STATE_ACTIVE, Gdk::Color("#1FFF00"));
```

```
        pboton_ensa->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_ensamblar_clicked)
);
        pboton_carga->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_carga_clicked) );
        pboton_reinic->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_reiniciar_clicked)
);
        pboton_paso->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_paso_clicked) );
        pboton_inst->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_inst_clicked) );
        pboton_salir->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_salir_clicked) );
        pboton_acerca->signal_clicked().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_Acerca_de_clicked) );
        pboton_int->signal_toggled().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_int_toggled) );
        pboton_corrida->signal_toggled().connect(
sigc::mem_fun(*this,&VentanaPrototipo::on_boton_corrida_toggled) );

      // Limpia la memoria
      for (int i=0; i<=255; i++) {
            memoria[i] = 0;
      }
      //vector de servicio de interrupción
      memoria[3]=0xb0;
     }

     VentanaPrototipo::~VentanaPrototipo()
     {
     }

     bool
VentanaPrototipo::on_eventbox1_button_press(GdkEventButton*)
```

```
{
 if(estado1 == LIBERADO)
 {
       estado1 = PRESIONADO;
       entrada=entrada+1;
       event_1->set_state(Gtk::STATE_ACTIVE);
 }
 else
 {
       event_1->set_state(Gtk::STATE_NORMAL);
       estado1 = LIBERADO;
       entrada=entrada-1;
 }
  return true;
}

bool
VentanaPrototipo::on_eventbox2_button_press(GdkEventButton*)
{

 if(estado2 == LIBERADO)
 {
       estado2 = PRESIONADO;
       entrada=entrada+2;
       event_2->set_state(Gtk::STATE_ACTIVE);
 }
 else
 {
       event_2->set_state(Gtk::STATE_NORMAL);
       estado2 = LIBERADO;
       entrada=entrada-2;
 }
  return true;
}
```

```
     bool
VentanaPrototipo::on_eventbox4_button_press(GdkEventButton*)
     {

      if(estado4 == LIBERADO)
      {
            estado4 = PRESIONADO;
            entrada=entrada+4;
            event_4->set_state(Gtk::STATE_ACTIVE);
      }
      else
      {
            event_4->set_state(Gtk::STATE_NORMAL);
            estado4 = LIBERADO;
            entrada=entrada-4;
      }
       return true;
     }

     bool
VentanaPrototipo::on_eventbox8_button_press(GdkEventButton*)
     {

      if(estado8 == LIBERADO)
      {
            estado8 = PRESIONADO;
            entrada=entrada+8;
            event_8->set_state(Gtk::STATE_ACTIVE);
      }
      else
      {
            event_8->set_state(Gtk::STATE_NORMAL);
            estado8 = LIBERADO;
            entrada=entrada-8;
      }
```

```
  return true;
}

void VentanaPrototipo::on_boton_int_toggled()
{
}

void VentanaPrototipo::on_boton_paso_clicked()
{
 Paso();
}

Glib::ustring VentanaPrototipo::Int_a_Str(int valor) const
{
 //convierte los enteros hexadecimales a string
 ostringstream convertir;
 convertir << hex << valor;
 Glib::ustring salida(convertir.str());
 return salida;
}

int VentanaPrototipo::Str_a_Int(Glib::ustring valor) const
{
 //convierte los string a enteros (hexadecimales)
 int val;
 stringstream convertir(valor);
 convertir >> hex >> val;
 return val;
}

void VentanaPrototipo::Paso()
{

 //garantiza que todas las referencias vuelven a cero
 Op=0;DPC=0;DSP=0;IMAR=0;IPC=0;ISP=0;
```

```
	LA=0;LB=0;LC=0;LF=0;LI=0;LIR=0;LMAR=0;LO=0;LPC=0;LSP=0;
	OC=0;OF=0;OI=0;OPC=0;OSP=0;R=0;
	RIA=0;RIE=0;RMAR=0;RT=0;
	SIA=0;SIE=0;W=0;

	preg_t->set_text(Int_a_Str(temp));

	uc.Ingresar_Status(Str_a_Int(preg_s->get_text()));
	//Revisa el estado del boton de interrupcion y establece la
interrupcion
	if((pboton_int->get_active()==true)&&(Str_a_Int(preg_ie-
>get_text())==1))
		uc.Ingresar_Interrupcion(1);
	else
		uc.Ingresar_Interrupcion(0);
	uc.Ingresar_IR(Str_a_Int(preg_ir->get_text()));
	uc.Ingresar_Tiempo(Str_a_Int(preg_t->get_text()));

	uc.Procesar(Op,DPC,DSP,IMAR,IPC,ISP,
			LA,LB,LC,LF,LI,LIR,LMAR,LO,LPC,LSP,
			OC,OF,OI,OPC,OSP,R,
			RIA,RIE,RMAR,RT,
			SIA,SIE,W);

	// Primero procesa los controles O y R (memoria) para
	// actualizar el BUS
		if (OSP==1)
		{
			pbus->set_text(preg_sp->get_text());
			event_osp->set_state(Gtk::STATE_ACTIVE);
		}
		else
			event_osp->set_state(Gtk::STATE_NORMAL);

		if (OPC==1)
```

```
{
      pbus->set_text(preg_pc->get_text());
      event_opc->set_state(Gtk::STATE_ACTIVE);
}
else
      event_opc->set_state(Gtk::STATE_NORMAL);

if (R==1)
{
      i=Str_a_Int(preg_mar->get_text());

      pmem->set_text(Int_a_Str(memoria[i]));
      pbus->set_text(pmem->get_text());
      event_rmem->set_state(Gtk::STATE_ACTIVE);
}
else
      event_rmem->set_state(Gtk::STATE_NORMAL);

if (OC==1)
{
      pbus->set_text(preg_c->get_text());
      event_oc->set_state(Gtk::STATE_ACTIVE);
}
else
      event_oc->set_state(Gtk::STATE_NORMAL);

if (OF==1)
{
      pbus->set_text(preg_f->get_text());
      event_of->set_state(Gtk::STATE_ACTIVE);
}
else
      event_of->set_state(Gtk::STATE_NORMAL);

if (OI==1)  pbus->set_text(preg_i->get_text());
```

```
//Procesa Cargas y escritura de memoria
if (LSP==1)
{
      preg_sp->set_text(pbus->get_text());
      event_lsp->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lsp->set_state(Gtk::STATE_NORMAL);

if (LPC==1)
{
      preg_pc->set_text(pbus->get_text());
      event_lpc->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lpc->set_state(Gtk::STATE_NORMAL);

if (LMAR==1)
{
      preg_mar->set_text(pbus->get_text());
      event_lmar->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lmar->set_state(Gtk::STATE_NORMAL);

if (LIR==1)
{
      preg_ir->set_text(pbus->get_text());
      uc.Ingresar_IR(Str_a_Int(preg_ir->get_text()));

      event_lir->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lir->set_state(Gtk::STATE_NORMAL);
```

```
if (LC==1)
{
      preg_c->set_text(pbus->get_text());
      event_lc->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lc->set_state(Gtk::STATE_NORMAL);

if (LO==1)
{
      preg_o->set_text(pbus->get_text());
      output=Str_a_Int(preg_o->get_text());
      event_lo->set_state(Gtk::STATE_ACTIVE);

      if(((output&0x8)>>3)==1)
            event_s8->set_state(Gtk::STATE_ACTIVE);
      else
            event_s8->set_state(Gtk::STATE_NORMAL);

      if(((output&0x4)>>2)==1)
            event_s4->set_state(Gtk::STATE_ACTIVE);
      else
            event_s4->set_state(Gtk::STATE_NORMAL);

      if(((output&0x2)>>1)==1)
            event_s2->set_state(Gtk::STATE_ACTIVE);
      else
            event_s2->set_state(Gtk::STATE_NORMAL);

      if((output&0x1)==1)
            event_s1->set_state(Gtk::STATE_ACTIVE);
      else
            event_s1->set_state(Gtk::STATE_NORMAL);
}
```

```
else
      event_lo->set_state(Gtk::STATE_NORMAL);

if (LI==1)
{
      preg_i->set_text(Int_a_Str(entrada));

}

if (LA==1)
{
      preg_a->set_text(pbus->get_text());
      ual.Ingresar_A(Str_a_Int(preg_a->get_text()));
      event_la->set_state(Gtk::STATE_ACTIVE);
}
else
      event_la->set_state(Gtk::STATE_NORMAL);

if (LB==1)
{
      preg_b->set_text(pbus->get_text());
      ual.Ingresar_B(Str_a_Int(preg_b->get_text()));
      event_lb->set_state(Gtk::STATE_ACTIVE);
}
else
      event_lb->set_state(Gtk::STATE_NORMAL);

preg_op->set_text(Int_a_Str(ual.Ingresar_Op(Op)));

ual.Operacion(F,S);

if (W==1)
{
      i=Str_a_Int(preg_mar->get_text());
```

```
      pmem->set_text(pbus->get_text());
      memoria[i] = Str_a_Int(pmem->get_text());
      event_wmem->set_state(Gtk::STATE_ACTIVE);
}
else
      event_wmem->set_state(Gtk::STATE_NORMAL);

// Otros Controles que no afectan o no se sirven del bus
if (LF==1)
{
    //if(abs(F)>15)
    //      F=F&15;
preg_f->set_text(Int_a_Str(Verifica_Extremos(F,15)));
      preg_s->set_text(Int_a_Str(S));
      if(((S&0x8)>>3)==1)
        event_s->set_state(Gtk::STATE_ACTIVE);
      else
            event_s->set_state(Gtk::STATE_NORMAL);
      if(((S&0x4)>>2)==1)
            event_p->set_state(Gtk::STATE_ACTIVE);
      else
            event_p->set_state(Gtk::STATE_NORMAL);
      if(((S&0x2)>>1)==1)
            event_d->set_state(Gtk::STATE_ACTIVE);
      else
            event_d->set_state(Gtk::STATE_NORMAL);
      if((S&0x1)==1)
            event_z->set_state(Gtk::STATE_ACTIVE);
      else
            event_z->set_state(Gtk::STATE_NORMAL);

      event_lf->set_state(Gtk::STATE_ACTIVE);
}
else
{
```

```
                event_lf->set_state(Gtk::STATE_NORMAL);
          }

          //set_text  utiliza  Int_a_Str  para  incrementar  lo
convertido en Str_a_Int
          //obtenido de get_text
          if (ISP==1)
          {
                preg_sp-
>set_text(Int_a_Str(Incrementa(Str_a_Int(preg_sp->get_text()))));
                event_isp->set_state(Gtk::STATE_ACTIVE);
          }
          else
                event_isp->set_state(Gtk::STATE_NORMAL);

          if (DSP==1)
          {
                preg_sp-
>set_text(Int_a_Str(Decrementa(Str_a_Int(preg_sp->get_text()))));
                event_dsp->set_state(Gtk::STATE_ACTIVE);
          }
          else
                event_dsp->set_state(Gtk::STATE_NORMAL);

          if (IPC==1)
          {
          preg_pc->set_text(Int_a_Str(Incrementa(Str_a_Int(preg_pc-
>get_text()))));
                event_ipc->set_state(Gtk::STATE_ACTIVE);
          }
          else
                event_ipc->set_state(Gtk::STATE_NORMAL);

          if (DPC==1)
          {
```

```
            preg_pc->set_text(Int_a_Str(Decrementa(Str_a_Int(preg_pc-
>get_text()))));
                  event_dpc->set_state(Gtk::STATE_ACTIVE);
            }
            else
                  event_dpc->set_state(Gtk::STATE_NORMAL);

            if (IMAR==1)
            {
            preg_mar-
>set_text(Int_a_Str(Incrementa(Str_a_Int(preg_mar->get_text()))));
                  event_imar->set_state(Gtk::STATE_ACTIVE);
            }
            else
                  event_imar->set_state(Gtk::STATE_NORMAL);

            if (RMAR==1)
            {
                  preg_mar->set_text("0");
                  event_rmar->set_state(Gtk::STATE_ACTIVE);
            }
            else
                  event_rmar->set_state(Gtk::STATE_NORMAL);

            if (SIE==1)
            {
                  preg_ie->set_text("1");
                  event_sie->set_state(Gtk::STATE_ACTIVE);
            }
            else
                  event_sie->set_state(Gtk::STATE_NORMAL);

            if (RIE==1)
            {
                  preg_ie->set_text("0");
```

```
      event_rie->set_state(Gtk::STATE_ACTIVE);
}
else
      event_rie->set_state(Gtk::STATE_NORMAL);

if (SIA==1)
{
      preg_ia->set_text("1");
      event_sia->set_state(Gtk::STATE_ACTIVE);
}
else
      event_sia->set_state(Gtk::STATE_NORMAL);

if (RIA==1)
{
      preg_ia->set_text("0");
      event_ria->set_state(Gtk::STATE_ACTIVE);
}
else
      event_ria->set_state(Gtk::STATE_NORMAL);

if (RT==1)
{
      preg_t->set_text("0");temp=0;
      event_rt->set_state(Gtk::STATE_ACTIVE);
}
else
{
      temp=Str_a_Int(preg_t->get_text());
      ++temp;
      temp = Verifica_Extremos(temp, 15);
      event_rt->set_state(Gtk::STATE_NORMAL);
}
h++;
```

```
            preg_reloj-
>set_text(Int_a_Str(Verifica_Extremos(h,1024)));
     }

     int    VentanaPrototipo::Verifica_Extremos(int    numero,    int
extremosup)
     {
      int regreso = numero;

      if (regreso > extremosup) {
            regreso = regreso - extremosup - 1;
      }
      if (regreso < 0) {
            regreso = extremosup + 1 + regreso;
      }
      return regreso;
     }

     int VentanaPrototipo::Incrementa(int Registro)
     {
      int incremento;

      Registro++;
      Registro = Verifica_Extremos(Registro, 255);
      incremento=Registro;
      return incremento;
     }

     int VentanaPrototipo::Decrementa(int Registro)
     {
      int decremento;

      Registro--;
      Registro = Verifica_Extremos(Registro, 255);
      decremento=Registro;
```

```
 return decremento;
}

void VentanaPrototipo::on_boton_inst_clicked()
{
 int tempT;
       Paso();
 do {
       Paso();
       tempT=Str_a_Int(preg_t->get_text());
 } while (tempT != 0);
}

void VentanaPrototipo::on_boton_salir_clicked()
{
 Gtk::Main::quit();
}

void VentanaPrototipo::on_boton_carga_clicked()
{
 char op[10];
 int   i;
 long int valor;
 ifstream entrada("SALIDA.DAT");

 entrada.getline(op,10);
 i = atoi(op);

 while ( !entrada.eof() )
 {
       entrada.getline(op,10);
       valor=strtol(op,NULL,0);
       memoria[i] = valor;
       i++;
 }
```

```
      entrada.close();
     }

     void VentanaPrototipo::on_boton_ensamblar_clicked()
     {
      Gtk::FileChooserDialog dialog("Por favor seleccione un
archivo",
             Gtk::FILE_CHOOSER_ACTION_OPEN);
      dialog.set_transient_for(*this);

      //Agrega botones de respuesta a el seleccionador de archivos
      dialog.add_button(Gtk::Stock::CANCEL, Gtk::RESPONSE_CANCEL);
      dialog.add_button(Gtk::Stock::OPEN, Gtk::RESPONSE_OK);

      //Filtro para que solo los archivos .UDO sean seleccionados:
            Gtk::FileFilter filter_text;
            filter_text.set_name("Archivos UDO");
      filter_text.add_pattern("*.UDO");
            dialog.add_filter(filter_text);

      //Muestra el dialogo y espera una respuesta
            int result = dialog.run();

      //Dependiendo de la respuesta toma una acciÃ³n
            switch(result)
            {
            case(Gtk::RESPONSE_OK):
            {
            cout << "Abrir" << endl;

            //es std::string, no Glib::ustring.
            string filename = dialog.get_filename();
            cout << "Archivo seleccionado: " << filename << endl;
            //Realiza el ensamblaje del archivo *.UDO
```

```
        ensamblador.P1(filename);
        ensamblador.P2();
        break;
        }
        case(Gtk::RESPONSE_CANCEL):
        {
        cout << "Cancelado." << endl;
        break;
        }
        default:
        {
        cout << "Presionado boton inesperado" << endl;
              break;
        }
        }
}

void VentanaPrototipo::on_boton_reiniciar_clicked()
{
 h=0;
 // Limpia la memoria
 for (int i=0; i<=255; i++)
        memoria[i] = 0;

 event_z->set_state(Gtk::STATE_NORMAL);
        event_d->set_state(Gtk::STATE_NORMAL);
        event_p->set_state(Gtk::STATE_NORMAL);
        event_s->set_state(Gtk::STATE_NORMAL);
        event_lsp->set_state(Gtk::STATE_NORMAL);
        event_isp->set_state(Gtk::STATE_NORMAL);
        event_dsp->set_state(Gtk::STATE_NORMAL);
        event_osp->set_state(Gtk::STATE_NORMAL);
 event_lpc->set_state(Gtk::STATE_NORMAL);
        event_ipc->set_state(Gtk::STATE_NORMAL);
        event_dpc->set_state(Gtk::STATE_NORMAL);
```

```
      event_opc->set_state(Gtk::STATE_NORMAL);
event_lc->set_state(Gtk::STATE_NORMAL);
event_oc->set_state(Gtk::STATE_NORMAL);
event_lo->set_state(Gtk::STATE_NORMAL);
event_la->set_state(Gtk::STATE_NORMAL);
event_lb->set_state(Gtk::STATE_NORMAL);
event_lf->set_state(Gtk::STATE_NORMAL);
event_of->set_state(Gtk::STATE_NORMAL);
event_lir->set_state(Gtk::STATE_NORMAL);
event_lmar->set_state(Gtk::STATE_NORMAL);
      event_imar->set_state(Gtk::STATE_NORMAL);
      event_rmar->set_state(Gtk::STATE_NORMAL);
event_rmem->set_state(Gtk::STATE_NORMAL);
      event_wmem->set_state(Gtk::STATE_NORMAL);
event_sie->set_state(Gtk::STATE_NORMAL);
      event_rie->set_state(Gtk::STATE_NORMAL);
event_sia->set_state(Gtk::STATE_NORMAL);
      event_ria->set_state(Gtk::STATE_NORMAL);
      event_rt->set_state(Gtk::STATE_NORMAL);
event_8->set_state(Gtk::STATE_NORMAL);
event_4->set_state(Gtk::STATE_NORMAL);
event_2->set_state(Gtk::STATE_NORMAL);
event_1->set_state(Gtk::STATE_NORMAL);
event_s8->set_state(Gtk::STATE_NORMAL);
event_s4->set_state(Gtk::STATE_NORMAL);
event_s2->set_state(Gtk::STATE_NORMAL);
event_s1->set_state(Gtk::STATE_NORMAL);

preg_sp->set_text("0");
preg_op->set_text("0");
preg_a->set_text("0");
preg_b->set_text("0");
preg_s->set_text("0");
preg_f->set_text("0");
preg_t->set_text("0");
```

```
 preg_ir->set_text("0");
 preg_mar->set_text("0");
 preg_pc->set_text("0");
 preg_c->set_text("0");
 preg_o->set_text("0");
 pbus->set_text("0");
 pmem->set_text("0");
 preg_i->set_text("0");
 preg_ie->set_text("0");
 preg_ia->set_text("0");
 preg_reloj->set_text("0");
}

void VentanaPrototipo::on_boton_corrida_toggled()
{
 pboton_salir->set_sensitive(false);
 pboton_ensa->set_sensitive(false);
 pboton_carga->set_sensitive(false);
 pboton_reinic->set_sensitive(false);
 pboton_paso->set_sensitive(false);
 pboton_inst->set_sensitive(false);
 pboton_salir->set_sensitive(false);

 bool tempIR;
 do
 {
       Paso();
       //Revisa si hay algun evento pendiente
       //Actualiza el GUI para los eventos ocurridos
       while( Gtk::Main::events_pending() )
       {
                   Gtk::Main::iteration();
             tempIR = pboton_corrida->get_active();
       }
 }while(tempIR != false);
```

```
 pboton_salir->set_sensitive(true);
 pboton_ensa->set_sensitive(true);
 pboton_carga->set_sensitive(true);
 pboton_reinic->set_sensitive(true);
 pboton_paso->set_sensitive(true);
 pboton_inst->set_sensitive(true);
 pboton_salir->set_sensitive(true);
}

void VentanaPrototipo::on_Acerca_de_clicked()
{
 m_Dialog.set_transient_for(*this);

 m_Dialog.set_name("CPU UDO");
 m_Dialog.set_version("2008");
 m_Dialog.set_copyright("Miguel Barrios");
 m_Dialog.set_comments("Simulador del computador UDO");
 m_Dialog.set_license("Los Trabajos de Grado son de exclusiva
propiedad de la Universidad\n y solo  podrán ser utilizados para
otros fines con el consentimiento\n del Consejo de Núcleo
respectivo, quien lo participará\n al Consejo Universitario");

 m_Dialog.set_website("http://www.anz.udo.edu.ve");
 m_Dialog.set_website_label("Universidad De Oriente");

 std::list<Glib::ustring> list_authors;
 list_authors.push_back("Miguel Barrios");
 list_authors.push_back("Luis Parraguez");
 m_Dialog.set_authors(list_authors);
 m_Dialog.run();

 m_Dialog.present();
 m_Dialog.hide();
}
```

A.3 ENCABEZADO DE FUNCIÓN PRINCIPAL *GUI*

```
#ifndef GUI_H
#define GUI_H

#include <libglademm.h>
#include <gtkmm.h>

class VentanaPrototipo : public Gtk::Window
{
 public:
      VentanaPrototipo(BaseObjectType* cobject,
                  const         Glib::RefPtr<Gnome::Glade::Xml>&
glade_xml);
      virtual ~VentanaPrototipo();
      Glib::ustring Int_a_Str(int valor) const;
      int Str_a_Int(Glib::ustring valact) const;
      void Paso();
      int Verifica_Extremos(int numero, int extremosup);
      int Incrementa(int Registro);
      int Decrementa(int Registro);
      void Instruccion();

      int Op,DPC,DSP,IMAR,IPC,ISP;
      int LA,LB,LC,LF,LI,LIR,LMAR,LO,LPC,LSP;
      int OC,OF,OI,OPC,OSP,R;
      int RIA,RIE,RMAR,RT;
      int SIA,SIE,W;
      int F,S;

 protected:
      //controladores de señales, algo como "signal handlers"
 virtual void on_boton_ensamblar_clicked();
      virtual void on_boton_carga_clicked();
      virtual void on_boton_reiniciar_clicked();
      virtual void on_boton_paso_clicked();
```

```
      virtual void on_boton_inst_clicked();
      virtual void on_boton_corrida_toggled();
virtual void on_boton_salir_clicked();
virtual void on_Acerca_de_clicked();
virtual void on_boton_int_toggled();
virtual bool on_eventbox8_button_press(GdkEventButton* event);
virtual bool on_eventbox4_button_press(GdkEventButton* event);
virtual bool on_eventbox2_button_press(GdkEventButton* event);
virtual bool on_eventbox1_button_press(GdkEventButton* event);

      //Punteros a los widgets
      Gtk::Label*       preg_sp;
      Gtk::Label*       preg_op;
      Gtk::Label*       preg_a;
      Gtk::Label*       preg_b;
      Gtk::Label*       preg_s;
      Gtk::Label*       preg_f;
      Gtk::Label*       preg_t;
      Gtk::Label*       preg_ir;
      Gtk::Label*       preg_mar;
      Gtk::Label*       preg_pc;
      Gtk::Label*       preg_c;
      Gtk::Label*       preg_o;
      Gtk::Label*       pbus;
      Gtk::Label*       pmem;
      Gtk::Label*       preg_i;
      Gtk::Label*       preg_ie;
      Gtk::Label*       preg_ia;
      Gtk::Label*       preg_reloj;

      Gtk::ToolButton*    pboton_ensa;
            Gtk::ToolButton*    pboton_carga;
            Gtk::ToolButton*    pboton_reinic;
            Gtk::ToolButton*    pboton_paso;
      Gtk::ToolButton*    pboton_inst;
```

```
      Gtk::ToggleToolButton*  pboton_corrida;
Gtk::ToolButton*    pboton_salir;
Gtk::ToolButton*    pboton_acerca;
      Gtk::ToggleButton*  pboton_int;

Gtk::EventBox*    event_8;
Gtk::EventBox*    event_4;
Gtk::EventBox*    event_2;
Gtk::EventBox*    event_1;

Gtk::EventBox*    event_s8;
Gtk::EventBox*    event_s4;
Gtk::EventBox*    event_s2;
Gtk::EventBox*    event_s1;

Gtk::EventBox*    event_z;
Gtk::EventBox*    event_d;
Gtk::EventBox*    event_p;
Gtk::EventBox*    event_s;

Gtk::EventBox*    event_lsp;
Gtk::EventBox*    event_isp;
Gtk::EventBox*    event_dsp;
Gtk::EventBox*    event_osp;

Gtk::EventBox*    event_lpc;
Gtk::EventBox*    event_ipc;
Gtk::EventBox*    event_dpc;
Gtk::EventBox*    event_opc;

Gtk::EventBox*    event_lc;
Gtk::EventBox*    event_oc;

Gtk::EventBox*    event_lo;
Gtk::EventBox*    event_la;
```

```
        Gtk::EventBox*    event_lb;
        Gtk::EventBox*    event_lf;
        Gtk::EventBox*    event_of;
        Gtk::EventBox*    event_lir;

        Gtk::EventBox*    event_lmar;
        Gtk::EventBox*    event_imar;
        Gtk::EventBox*    event_rmar;

        Gtk::EventBox*    event_rmem;
        Gtk::EventBox*    event_wmem;

        Gtk::EventBox*    event_sie;
        Gtk::EventBox*    event_rie;
        Gtk::EventBox*    event_sia;
        Gtk::EventBox*    event_ria;
        Gtk::EventBox*    event_rt;

        Gtk::AboutDialog m_Dialog;
};

#endif // GUI_H
```

A.4 FUNCIÓN DE UAL

```
//Programa Unidad Arimético Lógica

#include <iostream>
#include <cstdlib> //abs
#include <cmath>         //no es necesaria
#include "ual.h"

using namespace std;

UAL::UAL()
{
}
```

```
UAL::~UAL()
{
}

void UAL::Ingresar_A(int valA)
{
 A=valA;
}

void UAL::Ingresar_B(int valB)
{
 B=valB;
}

int UAL::Ingresar_Op(int valOp)
{
 Op=valOp;
 return Op;
}

int UAL::Operacion(int& rF,int& rS){

       switch(Op)
       {
             case 0 : F=A+B; break;
             case 1 : F=A-B; break;
             case 2 : F=A*B; break;
             case 3 : if(B!=0)
                                F=A/B;
                      else
                                F=31;
                      break;
             case 4 : F=A+1; break;
             case 5 : F=A-1; break;
```

```
                case 6 : F=A  ; break;
                case 7 : F=0  ; break;
                case 8 : F=15 ; break;
                case 9 : F=(~A)&15; break;
                case 10: F=(A&B)&15; break;
                case 11: F=(A|B)&15; break;
                case 12: F=(A^B)&15; break;
                case 13: F=(A<<1)&15; break;
                case 14: F=(A>>1); break;
                case 15: F=B; break;
                default: F=0; break;
          }

          if(Op==15)
                Status=B;
          else
                Status=Forma_Status(F);

          rF=F;
          rS=Status;
    }

    int UAL::Forma_Status(int Valor)
    {
     int resultado;

     if(Valor==0) Z=1; else Z=0;

     if(Valor<0) S=1; else S=0;

     if(abs(Valor)>15) D=1; else D=0;

     if((Valor==0)||(Valor==3)||(Valor==5)||(Valor==6)||
        (Valor==9)||(Valor==10)||(Valor==12)||(Valor==15)) P=1; else
P=0;
```

```
 resultado=8*S+4*P+2*D+1*Z;
 return resultado;
}
```

A.5 ENCABEZADO DE FUNCIÓN DE UAL

```
#ifndef UAL_H
#define UAL_H

class UAL
{
 public:
      UAL();
      ~UAL();
      int  Operacion(int&,int&);   //Funciones  de  acceso  a  la
clase
      void Ingresar_A(int);
      void Ingresar_B(int);
      int Ingresar_Op(int);

 private:
      int Forma_Status(int Valor);//Funcion miembro que retorna
estatus
      int A;                              //A interno
      int B;                              //B interno
      int F;                              //F interno
      int Op,Z,D,P,S,Status;
};


#endif // UAL_H
```

A.6 FUNCIÓN DE UNIDAD DE CONTROL

```
//Programa Unidad de Control
#include <iostream>
```

```
#include "uc.h"

using namespace std;

UnidadControl::UnidadControl()
{}

UnidadControl::~UnidadControl()
{}

void UnidadControl::Ingresar_Tiempo(int t)
{
 Tiempo=t;
}

void UnidadControl::Ingresar_IR(int reg)
{
 IR=reg;
}

void UnidadControl::Ingresar_Status(int s)
{
 Status=s;
 S=(s&0x8)>>3;
 P=(s&0x4)>>2;
 D=(s&0x2)>>1;
 Z=(s&0x1);
}

void UnidadControl::Ingresar_Interrupcion(int i)
{
 Interrupcion=i;
}

void UnidadControl::Procesar(
```

```
     int &rOp,int &rDPC,int &rDSP,int &rIMAR,int &rIPC,int &rISP,
     int  &rLA,int  &rLB,int  &rLC,int  &rLF,int  &rLI,int  &rLIR,int
&rLMAR,
     int  &rLO,int  &rLPC,int  &rLSP,int  &rOC,int  &rOF,int  &rOI,int
&rOPC,
     int  &rOSP,int  &rR,int  &rRIA,int  &rRIE,int  &rRMAR,int  &rRT,int
&rSIA,
     int &rSIE,int &rW
                        )
     {
      Op=0;DPC=0;DSP=0;IMAR=0;IPC=0;ISP=0;
      LA=0;LB=0;LC=0;LF=0;LI=0;LIR=0;LMAR=0;LO=0;LPC=0;LSP=0;
      OC=0;OF=0;OI=0;OPC=0;OSP=0;R=0;
      RIA=0;RIE=0;RMAR=0;RT=0;
      SIA=0;SIE=0;W=0;

      if(Interrupcion==1)
      {
            switch(Tiempo)
            {
                  case 0: LMAR=1;OSP=1;SIA=1; break;
                  case 1:     W=1;OPC=1;DSP=1; break;
                  case 2: RMAR=1; break;
                  case 3:
                  case 4:
                  case 5: IMAR=1; break;
                  case 6: LPC=1;R=1; break;
                  case 7:     RIE=1;RT=1; break;
            }
      }
      else
      {
            switch(Tiempo)
            {
                  case 0: LMAR=1;OPC=1; break;
```

```
case 1: R=1;LIR=1;IPC=1; break;
default:
      switch(IR)
      {
            case 0:     //NOP
                  switch(Tiempo)
                  {
                        case 2: RT=1; break;
                  }
                  break;
            case 1:     //MOV A,F
                  switch(Tiempo)
                  {
                              case                  2:
                        LA=1;OF=1;RT=1; break;

                  }
                  break;
            case 2:     //MOV B,F
                  switch(Tiempo)
                  {
                              case                  2:
                        LB=1;OF=1;RT=1; break;
                  }
                  break;
            case 3:     //MOV F,A
                  switch(Tiempo)
                  {
                              case                  2:
                        LF=1;Op=6;RT=1; break;

                  }
                  break;
            case 4:     //MOV F,B
                  switch(Tiempo)
```

```
      {
                case                  2:
            LF=1;Op=15;RT=1;      break;

      }
      break;
case 5:     //MOV O,F
      switch(Tiempo)
      {
                case                  2:
            LO=1;OF=1,RT=1; break;

      }
      break;
case 6:     //MOV A,I
      switch(Tiempo)
      {
                case                  2:
            LA=1;OI=1;RT=1; break;

      }
      break;
     case 7://MOV B,I
      switch(Tiempo)
      {
                case                  2:
            LB=1;OI=1;RT=1; break;

      }
      break;
case 8:     //MOV O,I
      switch(Tiempo)
      {
            case   2:   LO=1;OI=1;RT=1;
      break;
```

```
      }
      break;
case 0x10:  //LDI A,Dato
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
            case                3:
      LA=1;R=1;IPC=1;RT=1; break;
      }
      break;
case 0x11:  //LDI B,Dato
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
            case                3:
      LB=1;R=1;IPC=1;RT=1; break;
      }
      break;
case 0x12:  //LDI O,Dato
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
            case                3:
      LO=1;R=1;IPC=1;RT=1; break;
      }
      break;
case 0x13:  //LDI SP,Dir
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
```

```
                case                3:
            LSP=1;R=1;IPC=1;RT=1;
            break;
      }
      break;
case 0x14:  //MOV C,I
      switch(Tiempo)
      {
                case                2:
            LC=1;OI=1;RT=1; break;

      }
      break;
case 0x15:  //MOV C,F
      switch(Tiempo)
      {
                case                2:
            LC=1;OF=1;RT=1; break;

      }
      break;
case 0x16:  //LDI C,Dato
      switch(Tiempo)
      {
                case  2:  LMAR=1;OPC=1;
            break;
                case                3:
            LC=1;R=1;IPC=1;RT=1; break;
      }
      break;
case 0x20:  //SUM
      switch(Tiempo)
      {
                case                2:
            LF=1;Op=0;RT=1; break;
```

```
      }
      break;
case 0x21:  //SUMA Dato1,Dato2
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
            case                3:
      LA=1;R=1;IPC=1; break;
            case  4:  LMAR=1;OPC=1;
      break;
            case                5:
      LB=1;R=1;IPC=1; break;
            case                6:
      LF=1;Op=0;RT=1; break;

      }
      break;
case 0x22:  //RES
      switch(Tiempo)
      {
            case                2:
      LF=1;Op=1;RT=1; break;
      }
      break;
case 0x23:  //RESTA Dato1,Dato2
      switch(Tiempo)
      {
            case  2:  LMAR=1;OPC=1;
      break;
            case                3:
      LA=1;R=1;IPC=1; break;
            case  4:  LMAR=1;OPC=1;
      break;
```

```
                case                5:
            LB=1;R=1;IPC=1; break;
                case                6:
            LF=1;Op=1;RT=1; break;
      }
      break;
case 0x24:  //MUL
      switch(Tiempo)
      {
                case                2:
            LF=1;Op=2;RT=1; break;
      }
      break;
case 0x26:  //DIV
      switch(Tiempo)
      {
                case                2:
            LF=1;Op=3;RT=1; break;
      }
      break;
case 0x28:  //INC
      switch(Tiempo)
      {
                case                2:
            LF=1;Op=4;RT=1; break;
      }
      break;
case 0x29:  //DEC
      switch(Tiempo)
      {
                case                2:
            LF=1;Op=5;RT=1; break;
      }
      break;
case 0x2A:  //TRAA
```

```
      switch(Tiempo)
      {
            case                  2:
      LF=1;Op=6;RT=1; break;
      }
      break;
case 0x2B:  //RESET
      switch(Tiempo)
      {
            case                  2:
      LF=1;Op=7;RT=1; break;
      }
      break;
case 0x2C:  //SET
      switch(Tiempo)
      {
            case                  2:
      LF=1;Op=8;RT=1; break;
      }
      break;
case 0x2D:  //COMP
      switch(Tiempo)
      {
            case                  2:
      LF=1;Op=9;RT=1; break;
      }
      break;
case 0x2E:  //TRAB
      switch(Tiempo)
      {
      case 2: LF=1;Op=15;RT=1;
      break;
      }
      break;
case 0x30:  //IN
```

```
      switch(Tiempo)
      {
            case 2: LI=1;RT=1; break;
      }
      break;
case 0x31:  //INA
      switch(Tiempo)
      {
            case 2: LI=1; break;
            case 3: LA=1;OI=1;RT=1;
            break;
      }
      break;
case 0x32:  //INB
      switch(Tiempo)
      {
            case 2: LI=1; break;
            case 3: LB=1;OI=1;RT=1;
            break;
      }
      break;
case 0x33:  //OUT
      switch(Tiempo)
      {
                  case                  2:
            LO=1;OF=1;RT=1; break;
      }
      break;
case 0x40:  //JUMP Dir
      switch(Tiempo)
      {
                  case  2:  LMAR=1;OPC=1;
            break;
                  case                  3:
            LPC=1;R=1;RT=1; break;
```

```
      }
      break;
case 0x41:  //JUMPS Dir
      switch(S)
      {
            case 0: switch(Tiempo)
            {
                  case 2: IPC=1;RT=1;
                        break;

            }
            break;
            case 1: switch(Tiempo)
            {
                  case 2: LMAR=1;OPC=1;
                        break;
                  case                3:
            LPC=1;R=1;RT=1; break;
            }
            break;
      }
      break;
case 0x42:  //JUMPP Dir
      switch(P)
      {
            case 0: switch(Tiempo)
                  {
                  case 2: IPC=1;RT=1;
                        break;
                  }
                  break;
            case 1: switch(Tiempo)
                  {
                  case 2: LMAR=1;OPC=1;
                        break;
```

```
                        case 3:
                              LPC=1;R=1;R
                               T=1; break;
                        }
                        break;
                  }
                  break;
            case 0x43:  //JUMPD Dir
                  switch(D)
                  {
                  case 0: switch(Tiempo)
                        {
                        case  2:  IPC=1;RT=1;  break;
                        }
                        break;
                  case 1: switch(Tiempo)
                        {
                        case 2: LMAR=1;OPC=1;
                              break;
                             case                  3:
                        LPC=1;R=1;RT=1;
                              break;
                        }
                        break;
                  }
                  break;
            case 0x44:  //JUMPZ Dir
                  switch(Z)
                  {
                        case 0: switch(Tiempo)
                              {
                              case 2: IPC=1;RT=1;
                                    break;
                              }
                              break;
```

```
                case 1: switch(Tiempo)
                        {
                        case 2: LMAR=1;OPC=1;
                                break;
                        case                3:
                        LPC=1;R=1;RT=1;
                        break;
                        }
                        break;
        }
        break;
case 0x47:  //CALL Dir
        switch(Tiempo)
        {
        case 2: LMAR=1;OSP=1;IPC=1;
                break;
        case 3: OPC=1;W=1;DSP=1; break;
        case 4: DPC=1; break;
        case 5: LMAR=1; OPC=1; break;
        case  6:  LPC=1;R=1;RT=1;  break;
        }
        break;
case 0x48:  //RET
        switch(Tiempo)
        {
        case 2: ISP=1; break;
        case 3: LMAR=1;OSP=1; break;
        case 4: LPC=1;R=1;RT=1; break;
        }
        break;
case 0x49:  //CALLS Dir
        switch(S)
        {
        case 0: switch(Tiempo)
                {
```

```
                case 2: IPC=1;RT=1; break;
                }
                break;
          case 1:
                switch(Tiempo)
                {
                case 2: LMAR=1;OSP=1;IPC=1;
                          break;
                case 3: OPC=1;W=1;DSP=1;
                      break;
                case 4: DPC=1; break;
                case 5: LMAR=1;OPC=1;
                      break;
                case 6: LPC=1;R=1;RT=1;
                      break;
                }
                break;
          }
          break;
    case 0x50:  //PUSH F
          switch(Tiempo)
          {
                case 2: LMAR=1;OSP=1;
                break;
                     case                 3:
                OF=1;W=1;DSP=1;RT=1; break;
          }
          break;
    case 0x51:  //POP A
          switch(Tiempo)
          {
                case 2: ISP=1; break;
                case 3: LMAR=1; OSP=1;
                      break;
                case 4: LA=1;R=1;RT=1;
```

```
                  break;
      }
      break;
case 0x60:  //STI F,Dir
      switch(Tiempo)
      {
      case 2: LMAR=1;OPC=1; break;
      case 3: LMAR=1;R=1; break;
      case 4: OF=1;W=1;IPC=1;RT=1;
                  break;
      }
break;
case 0x61:  //STI I,Dir
      switch(Tiempo)
      {
      case 2: LMAR=1;OPC=1; break;
      case 3: LMAR=1;R=1; break;
      case 4: OI=1;W=1;IPC=1;RT=1;
                  break;
      }
      break;
case 0x70:  //SIE
      switch(Tiempo)
      {
            case 2: SIE=1; RT=1; break;
      }
      break;
case 0x71:  //RIE
      switch(Tiempo)
      {
            case 2: RIE=1;RT=1; break;
      }
      break;
case 0x72:  //IRET
      switch(Tiempo)
```

```
      {
            case 2: ISP=1; break;
            case 3: LMAR=1;OSP=1;
                  break;
                case                  4:
            LPC=1;R=1;RIA=1;RT=1;
            break;
      }
      break;
case 0x80:  //AND
      switch(Tiempo)
      {
            case 2: LF=1;Op=10;RT=1;
            break;
      }
      break;
case 0x81:  //OR
      switch(Tiempo)
      {
            case 2: LF=1;Op=11;RT=1;
            break;
            }
      break;
case 0x82:  //XOR
      switch(Tiempo)
      {
            case 2: LF=1;Op=12;RT=1;
            break;
      }
      break;
case 0x83:  //SHL
      switch(Tiempo)
      {
            case 2: LF=1;Op=13;RT=1;
            break;
```

```
                        }
                        break;
                  case 0x84:  //SHR
                        switch(Tiempo)
                        {
                              case 2: LF=1;Op=14;RT=1;
                              break;
                              }
                        break;
                  case 0xFF:  //HALT
                        switch(Tiempo)
                        {
                              case 2: DPC=1; RT=1; break;
                        }
                        break;
            }
            break;
      }     //switch
 }     //else

 rOp=Op;rDPC=DPC;rDSP=DSP;rIMAR=IMAR;rIPC=IPC;rISP=ISP;
 rLA=LA;rLB=LB;rLC=LC;rLF=LF;rLI=LI;rLIR=LIR;rLMAR=LMAR;
 rLO=LO;rLPC=LPC;rLSP=LSP;rOC=OC;rOF=OF;rOI=OI;rOPC=OPC;
 rOSP=OSP;rR=R;rRIA=RIA;rRIE=RIE;rRMAR=RMAR;rRT=RT;rSIA=SIA;
 rSIE=SIE;rW=W;

}//Procesar
```

A.7 ENCABEZADO DE FUNCIÓN DE UNIDAD DE CONTROL

```
#ifndef UC_H
#define UC_H

class UnidadControl
{
 public:
      UnidadControl();
```

```
        virtual ~UnidadControl();
        void Procesar(

 int&,int&,int&,int&,int&,int&,int&,int&,int&,int&,

 int&,int&,int&,int&,int&,int&,int&,int&,int&,int&,

 int&,int&,int&,int&,int&,int&,int&,int&,int&
                          );
        void Ingresar_Tiempo(int);
        void Ingresar_IR(int);
        void Ingresar_Status(int);
        void Ingresar_Interrupcion(int);

 private:
        int IR, En_Interrupcion, Interrupcion, Tiempo, Status;
        int Z,D,P,S;
        int Op,DPC,DSP,IMAR,IPC,ISP;
        int LA,LB,LC,LF,LI,LIR,LMAR,LO,LPC,LSP;
        int OC,OF,OI,OPC,OSP,R;
        int RIA,RIE,RMAR,RT;
        int SIA,SIE,W;
};

#endif // UC_H
```

A.8 FUNCIÓN DEL ENSAMBLADOR

```
#include <fstream>
#include <iostream>
#include <cstring>       //strbrk, strtok...
#include <cstdlib>       //atoi
#include "ensamblador.h"

#defineMAXPLBRA    256   //Max longitud de linea de entrada
```

```
using namespace std;

Ensamblador::Ensamblador()
{}

Ensamblador::~Ensamblador()
{}

int Ensamblador::P1(std::string filename)
{
char a[40][15]={{"nop"},{"mov"},{"sum"},{"res"},
                  {"mul"},{"div"},{"inc"},{"dec"},
                  {"traa"},{"reset"},{"set"},{"comp"},
                  {"trab"},{"in"},{"ina"},{"inb"},
                  {"out"},{"jmp"},{"jmps"},{"jmpp"},
                  {"jmpd"},{"jmpz"},{"call"},{"ret"},
                  {"calls"},{"push"},{"pop"},{"sti"},
                  {"sie"},{"rie"},{"and"},{"or"},
                  {"xor"},{"shl"},{"shr"},{"ldi"},
                  {"iret"},{"suma"},{"resta"},{"halt"}
                };

 //Limpiar el arreglo para el nombre
 for(i=0;i<80;i++)
       nombre[i]=0;

 f=0;f1=0;n=0;lc=0;i=0;s=0;l=0;
 filename.copy(nombre,80,0);
 //Abre un archivo de ENTRADA para lectura. El archivo debe
 //existir.
 ifstream entrada(nombre);
 //Crea el archivo con la tabla de simbolos de etiquetas y la
 //posición de memoria
 ofstream tabsimb("TABSIMB.DAT");
 //Archivo intermedio para el proceso del primer paso
```

```
ofstream interm("INTERM.DAT");

while (entrada.peek() == ';')
      entrada.ignore(255,'\n');

// Lee la primera linea
entrada.getline(tmp,MAXPLBRA);
// divide en palabras el string de linea
pch = strtok (tmp," .\t");
while (pch != NULL)
      {
      if(strcmp(pch,"INICIO")==0)   //Toma la dir de inicio
      {
            interm << pch << "\t";
            pch = strtok(NULL," .\t");
            lc = atoi(pch) ;
            interm << lc << "\n";
      }
      pch = strtok (NULL, " .\t");
}

s=lc;

// obtiene la entrada del usuario hasta EOF
while ( !entrada.eof() )
{
      while (entrada.peek() == ';')
            entrada.ignore(255,'\n');

      //reinicio de banderas
      f = 0;f1 = 0;
      //Toma el resto de las lineas
      entrada.getline(tmp,MAXPLBRA);
      if(entrada.eof()!=true)
            interm << "0x" << hex << lc << "\t";
```

```
i=0;
while(tmp[i]!='\0')     //Revisa y elimina comentarios
{                       //de cada linea que toma
      if(isspace(tmp[i]))
            tmp[i]='$';
      if(tmp[i]==';')
            while(tmp[i]!='\0')
            {
                  tmp[i]='\0';
                  ++i;
            }
      ++i;
}
strcpy (copia,tmp);     // almacena una copia de la linea
                              // porque strtok modifica el
                        //original
pch = strtok (tmp,"$"); //  divide  en  palabras  el  string
                        //de linea
      while (pch != NULL)
      {
   pch1 = strpbrk (pch, ":");//Compara si la palabra
                              //es una etiqueta
      if (pch1 != NULL)
            {
         for(i=0;i<n;i++)
         {
            if(strcmp(pch,b[i])==0)
            {
               f=1;
               break;
            }
            f=0;
         }
         if(f==0)
         {
```

```
                        tabsimb << pch << "\t" << "0x" << hex << lc
<< "\n";
                        strcpy(b[n++],pch);
                        }
                        }

                  for(i=0;i<40;i++)
                  {
                        if(strcmp(pch,a[i])==0)
                        {
                              f1=0;
                              break;
                        }
                        f1=1;
                  }

                  if(f1==0)
                  {
                        if(strcmp(pch,"nop")==0)
                              lc++;
                        else if(strcmp(pch,"mov")==0)
                                    lc++;
                        else if(strcmp(pch,"ldi")==0)
                                    lc=lc+2;
                        else if(strcmp(pch,"sum")==0)
                                    lc++;
                        else if(strcmp(pch,"res")==0)
                                    lc++;
                        else if(strcmp(pch,"suma")==0)
                                    lc=lc+3;
                        else if(strcmp(pch,"resta")==0)
                                    lc=lc+3;
                        else if(strcmp(pch,"mul")==0)
                                    lc++;
                        else if(strcmp(pch,"div")==0)
```

```
            lc++;
else if(strcmp(pch,"inc")==0)
            lc++;
else if(strcmp(pch,"dec")==0)
            lc++;
else if(strcmp(pch,"traa")==0)
            lc++;
else if(strcmp(pch,"reset")==0)
            lc++;
else if(strcmp(pch,"set")==0)
            lc++;
else if(strcmp(pch,"comp")==0)
            lc++;
else if(strcmp(pch,"trab")==0)
            lc++;
else if(strcmp(pch,"in")==0)
            lc++;
else if(strcmp(pch,"ina")==0)
            lc++;
else if(strcmp(pch,"inb")==0)
            lc++;
else if(strcmp(pch,"out")==0)
            lc++;
else if(strcmp(pch,"jmp")==0)
            lc=lc+2;
else if(strcmp(pch,"jmps")==0)
            lc=lc+2;
else if(strcmp(pch,"jmpp")==0)
            lc=lc+2;
else if(strcmp(pch,"jmpd")==0)
            lc=lc+2;
else if(strcmp(pch,"jmpz")==0)
            lc=lc+2;
else if(strcmp(pch,"call")==0)
            lc=lc+2;
```

```
                else if(strcmp(pch,"ret")==0)
                        lc++;
                else if(strcmp(pch,"calls")==0)
                        lc=lc+2;
                else if(strcmp(pch,"push")==0)
                        lc++;
                else if(strcmp(pch,"pop")==0)
                        lc++;
                else if(strcmp(pch,"sti")==0)
                        lc=lc+2;
                else if(strcmp(pch,"sie")==0)
                        lc++;
                else if(strcmp(pch,"rie")==0)
                        lc++;
                else if(strcmp(pch,"iret")==0)
                        lc++;
                else if(strcmp(pch,"and")==0)
                        lc++;
                else if(strcmp(pch,"or")==0)
                        lc++;
                else if(strcmp(pch,"xor")==0)
                        lc++;
                else if(strcmp(pch,"xor")==0)
                        lc++;
                else if(strcmp(pch,"shl")==0)
                        lc++;
                else if(strcmp(pch,"shr")==0)
                        lc++;
                else if(strcmp(pch,"halt")==0)
                        lc++;
        }
        pch = strtok (NULL, "$");
        }
interm << copia << "\n";
```

```
            if(f==1)
                  cout << "ERROR:ETIQUETA DUPLICADA\n";
       }

       l=lc-s-1;
            entrada.close();
            tabsimb.close( );
       interm.close( );
       return 0;
      }

      int Ensamblador::P2()
      {
      char a[40][15]={{"nop"},{"mov"},{"sum"},{"res"},
                        {"mul"},{"div"},{"inc"},{"dec"},
                        {"traa"},{"reset"},{"set"},{"comp"},
                        {"trab"},{"in"},{"ina"},{"inb"},
                        {"out"},{"jmp"},{"jmps"},{"jmpp"},
                        {"jmpd"},{"jmpz"},{"call"},{"ret"},
                        {"calls"},{"push"},{"pop"},{"sti"},
                        {"sie"},{"rie"},{"and"},{"or"},
                        {"xor"},{"shl"},{"shr"},{"ldi"},
                        {"iret"},{"suma"},{"resta"},{"halt"}
                     };
       FILE *fp1;
       i=0;f1=0;n=0;lc=0;j=0;

       //Abre el archivo de INTERM para lectura. El archivo debe
existir.
       ifstream interm("INTERM.DAT");
       //Abre el archivo con la tabla de simbolos de etiquetas y la
posicion de memoria
       fp1=fopen("TABSIMB.DAT","r");
       //Archivo de salida para el proceso del segundo paso
       ofstream salida("SALIDA.DAT");
```

```
// Lee la primera linea
interm.getline(tmp,MAXPLBRA);
// divide en palabras el string de linea
pch4 = strtok (tmp," .\t");
while (pch4 != NULL)
      {     //Toma la direccion de inico
      if(strcmp(pch4,"INICIO")==0)
      {
            pch4 = strtok(NULL," .\t");
            lc = atoi(pch4) ;
            salida << lc << "\n";
      }
      pch4 = strtok (NULL, " .\t");
}
i=0;
while(!feof(fp1))             //toma las direcciones y
                              //etiquetas del programa
{
      fscanf(fp1,"%s\t%s",etiq,dir);
      longitud=strlen(etiq);
      if(etiq[longitud-1]==':')
            etiq[longitud-1]='\0';
      strcpy(sim[i],etiq);
      strcpy(dirsim[i],dir);
      i++;
}
n=i;

// obtiene la entrada del usuario hasta EOF
while ( !interm.eof() )
{
      //Limpia los arreglos para cada linea
      for(i=0;i<10;i++)
      {
```

```
            op[i]=0;operando[i]=0;
         }
         interm.getline(tmp,MAXPLBRA); //Toma el resto de las
                                       //lineas
         pch4 = strtok (tmp,"$");      // divide en palabras el
                                       //string de linea
            while (pch4 != NULL)
            {
           pch3 = strncmp (pch4, ":",1);//Compara si la palabra
                                        //es una etiqueta
           pch2 = strncmp (pch4,"0x",2);//Compara si es una dir
           for(i=0;i<40;i++)
           {
              if(strcmp(pch4,a[i])==0)//Bandera para tomar
                                       //operadores
              {
                    f1=0;
                    strcpy(op,a[i]);  //toma   el   operador   que
concuerde
                    break;
              }
              f1=1;
           }

           if ((pch3 != 0) && (pch2 != 0) && (f1!=0))// busca los
operandos
           strcpy(operando,pch4);     //toma solo los operandos

           pch4 = strtok (NULL, "$");
         }

         if(strcmp(op,"nop")==0)
               salida << "0x0" << endl;
         else               if(strcmp(op,"mov")==0              &&
(strcmp(operando,"a,f")==0))
```

```
                        salida << "0x1" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"b,f")==0))
                        salida << "0x2" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"f,a")==0))
                        salida << "0x3" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"f,b")==0))
                        salida << "0x4" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"o,f")==0))
                        salida << "0x5" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"a,i")==0))
                        salida << "0x6" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"b,i")==0))
                        salida << "0x7" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"o,i")==0))
                        salida << "0x8" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"c,i")==0))
                        salida << "0x14" << endl;
            else                if(strcmp(op,"mov")==0              &&
(strcmp(operando,"c,f")==0))
                        salida << "0x15" << endl;
            else                if(strcmp(op,"ldi")==0              &&
(strncmp(operando,"a,",2)==0))
            {
                  salida << "0x10" << endl;
                  pop = strtok (operando,",h");
                  j=0;
                  while (pop != NULL)
```

```
                {//ignora el registro al que apunta y toma el valor
                      if(j==1)
                            salida << "0x" << pop << endl;
                      pop = strtok (NULL,",h");
                      j++;
                }
            }
            else              if((strcmp(op,"ldi")==0)              &&
(strncmp(operando,"b,",2)==0))
             {
                  salida << "0x11" << endl;
                  pop = strtok (operando,",h");
                  j=0;
                  while (pop != NULL)
                  {
                        if(j==1)
                              salida << "0x" << pop << endl;
                        pop = strtok (NULL,",h");
                        j++;
                  }
             }
            else              if((strcmp(op,"ldi")==0)              &&
(strncmp(operando,"o,",2)==0))
             {
                  salida << "0x12" << endl;
                  pop = strtok (operando,",h");
                  j=0;
                  while (pop != NULL)
                  {
                        if(j==1)
                              salida << "0x" << pop << endl;
                        pop = strtok (NULL,",h");
                        j++;
                  }
             }
```

```
            else             if((strcmp(op,"ldi")==0)             &&
(strncmp(operando,"sp,",3)==0))
             {
                  salida << "0x13" << endl;
                  pop = strtok (operando,",h");
                  j=0;
                  while (pop != NULL)
                  {
                        if(j==1)
                              salida << "0x" << pop << endl;
                        pop = strtok (NULL,",h");
                        j++;
                  }
             }
            else             if((strcmp(op,"ldi")==0)             &&
(strncmp(operando,"c,",2)==0))
             {
                  salida << "0x16" << endl;
                  pop = strtok (operando,",h");
                  j=0;
                  while (pop != NULL)
                  {
                        if(j==1)
                              salida << "0x" << pop << endl;
                        pop = strtok (NULL,",h");
                        j++;
                  }
             }
            else if(strcmp(op,"sum")==0)
                        salida << "0x20" << endl;
            else if(strcmp(op,"suma")==0)
                   {
                        salida << "0x21" << endl;
                        pop = strtok (operando,",h");
                        j=0;
```

```
            while (pop != NULL)
            {
                  if(j==0)
                        salida << "0x" << pop << endl;
                  if(j==1)
                        salida << "0x" << pop << endl;
                  pop = strtok (NULL,",h");
                  j++;
            }
      }
else if(strcmp(op,"res")==0)
            salida << "0x22" << endl;
else if(strcmp(op,"resta")==0)
      {
            salida << "0x23" << endl;
            pop = strtok (operando,",h");
            j=0;
            while (pop != NULL)
            {
                  if(j==0)
                        salida << "0x" << pop << endl;
                  if(j==1)
                        salida << "0x" << pop << endl;
                  pop = strtok (NULL,",h");
                  j++;
            }
      }
else if(strcmp(op,"mul")==0)
            salida << "0x24" << endl;
else if(strcmp(op,"div")==0)
            salida << "0x26" << endl;
else if(strcmp(op,"inc")==0)
            salida << "0x28" << endl;
else if(strcmp(op,"dec")==0)
            salida << "0x29" << endl;
```

```
else if(strcmp(op,"traa")==0)
            salida << "0x2a" << endl;
else if(strcmp(op,"reset")==0)
            salida << "0x2b" << endl;
else if(strcmp(op,"set")==0)
            salida << "0x2c" << endl;
else if(strcmp(op,"comp")==0)
            salida << "0x2d" << endl;
else if(strcmp(op,"trab")==0)
            salida << "0x2e" << endl;
else if(strcmp(op,"in")==0)
            salida << "0x30" << endl;
else if(strcmp(op,"ina")==0)
            salida << "0x31" << endl;
else if(strcmp(op,"inb")==0)
            salida << "0x32" << endl;
else if(strcmp(op,"out")==0)
            salida << "0x33" << endl;
else if(strcmp(op,"jmp")==0)
       {    salida << "0x40" << endl;
            for(i=0;i<=n;i++)
            {
                  if(strcmp(operando,sim[i])==0)
                  {
                        salida << dirsim[i] << endl;
                        break;
                  }
            }
       }
else if(strcmp(op,"jmps")==0)
       {
            salida << "0x41" << endl;
            for(i=0;i<=n;i++)
            {
                  if(strcmp(operando,sim[i])==0)
```

```
                {
                        salida << dirsim[i] << endl;
                        break;
                }
            }
        }
else if(strcmp(op,"jmpp")==0)
        {
            salida << "0x42" << endl;
            for(i=0;i<=n;i++)
            {
                if(strcmp(operando,sim[i])==0)
                {
                        salida << dirsim[i] << endl;
                        break;
                }
            }
        }
else if(strcmp(op,"jmpd")==0)
        {
            salida << "0x43" << endl;
            for(i=0;i<=n;i++)
            {
                if(strcmp(operando,sim[i])==0)
                {
                        salida << dirsim[i] << endl;
                        break;
                }
            }
        }
else if(strcmp(op,"jmpz")==0)
        {
            salida << "0x44" << endl;
            for(i=0;i<=n;i++)
            {
```

```
                    if(strcmp(operando,sim[i])==0)
                    {
                          salida << dirsim[i] << endl;
                          break;
                    }
              }
        }
else if(strcmp(op,"call")==0)
        {
              salida << "0x47" << endl;
              for(i=0;i<=n;i++)
              {
                    if(strcmp(operando,sim[i])==0)
                    {
                          salida << dirsim[i] << endl;
                          break;
                    }
              }
        }
else if(strcmp(op,"ret")==0)
              salida << "0x48" << endl;
else if(strcmp(op,"calls")==0)
        {
              salida << "0x49" << endl;
              for(i=0;i<=n;i++)
              {
                    if(strcmp(operando,sim[i])==0)
                    {
                          salida << dirsim[i] << endl;
                          break;
                    }
              }
        }
else if(strcmp(op,"push")==0)
              salida << "0x50" << endl;
```

```
else if(strcmp(op,"pop")==0)
            salida << "0x51" << endl;
else if(strcmp(op,"sti")==0)
       {
            pop = strtok (operando,",h");
            j=0;
            while (pop != NULL)
            {
                  if(strcmp(pop,"f")==0)
                        salida << "0x60" << endl;
                  if(strcmp(pop,"i")==0)
                        salida << "0x61" << endl;
                  if(j==1)
                        salida << "0x" << pop << endl;
                  pop = strtok (NULL,",h");
                  j++;
            }
       }
else if(strcmp(op,"sie")==0)
            salida << "0x70" << endl;
else if(strcmp(op,"rie")==0)
            salida << "0x71" << endl;
else if(strcmp(op,"iret")==0)
            salida << "0x72" << endl;
else if(strcmp(op,"and")==0)
            salida << "0x80" << endl;
else if(strcmp(op,"or")==0)
            salida << "0x81" << endl;
else if(strcmp(op,"xor")==0)
            salida << "0x82" << endl;
else if(strcmp(op,"shl")==0)
            salida << "0x83" << endl;
else if(strcmp(op,"shr")==0)
            salida << "0x84" << endl;
else if(strcmp(op,"halt")==0)
```

```
                    salida << "0xff" << endl;
 }
 fclose(fp1);
       salida.close( );
 interm.close( );
 return 0;
}
```

A.9 ENCABEZADO DE FUNCIÓN DEL ENSAMBLADOR

```
#ifndef ENSAMBLADOR_H
#define ENSAMBLADOR_H

#defineMAXPLBRA    256   //Max longitud de linea de entrada

class Ensamblador
{
 public:
       Ensamblador();
       virtual ~Ensamblador();
       int P1(std::string);
       int P2();

 private:
 char nombre[80];
 // para la entrada del usuario
 char tmp[MAXPLBRA];
 // respaldo de la entrada del usuario
 char copia[MAXPLBRA];
 char b[20][10];
 int  i,lc,s,l;
 int  f,f1,n;
 char * pch;
 char * pch1;

 //Pertenecientes al paso 2
```

```
 char etiq[10],dir[10],op[10],operando[10];
 char sim[80][10];
 char dirsim[80][10];
 char * pop;
 char * pch4;
 int   pch3;
 int   pch2;
 int   longitud,j;
};

#endif // ENSAMBLADOR_H
```

ANEXO 2: MANUAL PARA EL USUARIO

El Simulador utiliza elementos para el escritorio GNOME, funciona en KDE con algunas ligeras diferencias en la interfaz. A través del gestor de paquetes en tu distribución debes instalar los siguientes:

libglademm
gtkmm o gtkmm2

El programa necesita ser compilado porque carece de un sistema de empaquetamiento, se proporciona un conjunto de archivos fuentes que deben ser descritos junto con el comando `gcc` ó `g++` en un terminal de GNU/Linux. Debes dirigirte a la carpeta que contiene los archivos a compilar y escribir:

```
g++ $(pkg-config --libs --cflags gtkmm-2.4) $(pkg-config --libs --cflags libglademm-2.4) gui.cpp gui_principal.cpp ual.cpp uc.cpp ensamblador.cpp -o CPU_UDO2008
```

Para ejecutar el programa es necesario escribir en el terminal:

```
./CPU_UDO2008
```

O simplemente hacer doble click en el archivo ejecutable creado dentro de la carpeta que contiene los archivos fuentes. La ventaja de hacerlo por el terminal es que a través de este se mostrarán todos los mensajes del programa y el contenido de la memoria cuando se requiera.

El capitulo 5 del trabajo explica la forma como debe ser escrito un programa ensamblador. El propósito de estas reglas es evitar errores al compilar debido a que el ensamblador no cuenta con detección de errores. De igual forma, se detalla el funcionamiento de cada uno de los botones presentes en la interfaz. Para visualizar el contenido de la memoria en el terminal se debe presionar la palabra memoria en la pantalla.

- Cargar Programa: carga el código objeto del archivo SALIDA.DAT a las posiciones de memoria correspondientes en el simulador.
- Reiniciar: vacía todos los registros, la memoria.
- Paso: realiza un ciclo de reloj del simulador.
- Instrucción: corre un ciclo de máquina del programa, donde cada instrucción realiza los ciclos de reloj necesarios.
- Corrida: ejecuta continuamente el contenido de la memoria.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

TÍTULO	**DESARROLLO DEL SIMULADOR DE COMPUTADORES UDO-2008 BASADO EN HERRAMIENTAS COMPUTACIONALES DE SOFTWARE LIBRE**
SUBTÍTULO	

AUTOR (ES):

APELLIDOS Y NOMBRES	CÓDIGO CULAC / E MAIL
BARRIOS C. MIGUEL E.	CVLAC: V-17.222.237 E MAIL: miguelbarriosc@gmail.com

PALÁBRAS O FRASES CLAVES:

_____SIMULADOR, SISTEMAS DIGITALES, CONTROL, PROGRAMACION_____

_____

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ÀREA	SUBÀREA
INGENIERÍA ELÉCTRICA	SISTEMAS DIGITALES Y CONTROL

RESUMEN (ABSTRACT):

El Trabajo presentado a continuación trata acerca de las mejoras realizadas al simulador utilizado en la materia “Diseño de Sistemas Digitales”. El desarrollo del simulador de computación UDO 2008 se constituye a partir del uso de un paradigma de programación utilizado a nivel comercial por grandes empresas. Conocido como Programación Orientada a Objetos, está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Se utilizó *Lenguaje Unificado de Modelado* para definir el sistema, para detallar los artefactos en el sistema, para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se realizó el programa bajo herramientas de software libre, desde la implementación de UML con BOUML y DIA, el compilador para C++ de GNU, GLADE 3 para crear la interfaz gráfica y en general el sistema GNU/Linux como base para el desarrollo.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

CONTRIBUIDORES:

APELLIDOS Y NOMBRES	ROL / CÓDIGO CVLAC / E_MAIL				
Parraguez, Luis	ROL	CA	AS X	TU	JU
	CVLAC:				
	E_MAIL				
	E_MAIL				
Heraoui, Margarita	ROL	CA	AS	TU	JU: X
	CVLAC:				
	E_MAIL				
	E_MAIL				
Navarro, Danilo	ROL	CA	AS	TU	JU: X
	CVLAC:				
	E_MAIL				
	E_MAIL				

FECHA DE DISCUSIÓN Y APROBACIÓN:

2009	02	10
AÑO	MES	DÍA

LENGUAJE. SPA

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ARCHIVO (S):

NOMBRE DE ARCHIVO	TIPO MIME
TESIS.Simulador_de_computador.doc	Application/ msword

CARACTERES EN LOS NOMBRES DE LOS ARCHIVOS: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z. a b c d e f g h i j k l m n o p q r s t u v w x y z. 0 1 2 3 4 5 6 7 8 9.

ALCANCE

ESPACIAL: SIMULADOR DE SISTEMAS DIGITALES(**OPCIONAL**)

TEMPORAL: SEIS MESES _____ (**OPCIONAL**)

TÍTULO O GRADO ASOCIADO CON EL TRABAJO:

_____INGENIERO ELECTRICISTA____

NIVEL ASOCIADO CON EL TRABAJO:

_____PREGRADO _____

ÁREA DE ESTUDIO:

_____DEPARTAMENTO DE ELECTRICIDAD

INSTITUCIÓN:

UNIVERSIDAD DE ORIENTE –NUCLEO ANZOÁTEGUI_

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

DERECHOS

Articulo 44: Los Trabajos de Grado son de exclusiva propiedad de la Universidad y sólo podrán ser utilizados para otros fines con el consentimiento del Consejo de Núcleo respectivo, quién lo participará al Consejo Universitario".

AUTOR AUTOR AUTOR

TUTOR JURADO JURADO

POR LA SUBCOMISION DE TESIS