ÍNDICE GENERAL

INTRODUCCION

En la actualidad el proceso de enseñanza aprendizaje ha ido evolucionando a medida que la tecnología ha ingresado a este campo y le ha proporcionado herramientas para que la educación sea más eficiente y logre mejores resultados.

Hoy en día las herramientas interactivas orientadas a apoyar el proceso de enseñanza-aprendizaje han tomado mucha fuerza, esto debido a la posibilidad que proveen estas herramientas de aplicar los conocimientos teóricos avanzados en clases en ejercicios prácticos, facilitando la visualización del problema.

Una ventaja que otorgan las herramientas interactivas en el área de software educativo es que estas herramientas suelen ser fácilmente adaptables a las necesidades del usuario final, debido a que pueden tomar funcionalidades y características de otros tipos de software educativos como ser simuladores o sistemas con bases de datos.

A lo largo de los años, en la materia de Arquitectura de Computadoras perteneciente a la rama de la informática, el estudio del Ciclo de Instrucción del computador ha traído consigo serios problemas en el proceso de enseñanza aprendizaje y se han probado varios métodos de enseñanza y optado por distintas herramientas para llevarlo a cabo, pero los problemas persisten.

Antecedentes

Antecedentes organizacionales

En la actualidad, muchas universidades cuentan con la materia de Arquitectura de Computadoras dentro del pensum establecido para la carrera de Ingeniería de Sistemas. La Universidad Católica Boliviana ñSan Pabloò no es la excepción,

Dentro del contenido de esta materia, uno de los temas en los que se hace mayor énfasis es el Ciclo de Instrucción del Computador.

El ***objetivo principal*** de este tema es que el estudiante comprenda la secuencia exacta de pasos que debe realizar el CPU, dependiendo de su arquitectura y formato de instrucción (número de direcciones, CO, tipo de direccionamiento), y cómo este influye en la velocidad de ejecución de un programa.

Un ***ciclo de instrucción*** es el período que tarda la unidad central de proceso (CPU) en ejecutar una instrucción en lenguaje máquina. Este comprende una secuencia de acciones que se deben llevar a cabo para ejecutar cada instrucción en un programa.

Para que un CPU realice una tarea, primero debe buscar cada instrucción en la memoria principal y luego ejecutarla. A estos pasos se los conoce respectivamente como ciclo de captación y ciclo de ejecución, y el último depende de varios factores descritos a continuación.

En el ciclo de captación se realiza la búsqueda de la instrucción en la memoria principal y su respectiva decodificación, posteriormente en el ciclo de ejecución se ejecuta la instrucción y finalmente se almacena el resultado.

El número de pasos en el ciclo de captación generalmente es similar, sin importar la arquitectura del CPU, ni la instrucción que se haya dado. En cambio, el número de pasos que hay en el ciclo de ejecución sí depende de la arquitectura del CPU y del formato de instrucción, es decir, del código de operación (CO), del número de direcciones de la instrucción, del tipo de de direccionamiento y de los registros que contenga (AR y DR o banco de registros).

Los ejercicios de ciclo de instrucción pueden ser resueltos, ya sea en binario o en hexadecimal, así como también en lenguaje ensamblador.

En el desarrollo de la enseñanza del tema, se dedica una cierta cantidad de clases para explicar este proceso detalladamente al alumno, primeramente mostrándole la interacción de los componentes del CPU durante el ciclo de instrucción en binario. Posteriormente, para facilitar y acortar la tarea, cuando ya se ha captado el concepto en

binario, se procede a utilizar la notación hexadecimal y finalmente se empieza a aplicar el lenguaje ensamblador en todo el proceso.

En binario se enseña la distribución de los bits en los registros (bits utilizados para CO y para direccionamiento), y dependiendo de estos se calcula el rango de valores que se podrán utilizar y el espacio de memoria que podrá accederse. Esto da una idea clara respecto a que los recursos son limitados.

En hexadecimal empieza a mostrarse a más profundidad la interacción entre CPU y memoria principal. En esta parte se ve como se utilizan los recursos del procesador destinados a la interacción con memoria, así como también con los dispositivos de E/S, como por ejemplo los registros MAR (registro de direcciones de memoria) y MBR (registro de datos de memoria), buses internos y externos del CPU (buses con memoria y con los dispositivos de E/S).

En ensamblador se aborda el objetivo principal del tema: que el estudiante comprenda la secuencia exacta de pasos que debe realizar el CPU, dependiendo de su arquitectura y formato de instrucción (número de direcciones, CO, tipo de direccionamiento), y como este influye en la velocidad de ejecución de un programa.

En el desarrollo de las clases para entender el proceso que implica el ciclo de instrucción del computador se elaboran ejercicios que ponen en práctica los conceptos mencionados anteriormente, lo cual toma bastante tiempo dependiendo de la complejidad de ejercicio (aplicar la lógica de funcionamiento, además de la cantidad de instrucciones requeridas en el proceso a seguir) y la cantidad de diagramas a realizar. Por ejemplo, en el inicio del tema, para realizar el primer ejercicio en binario, la docente necesita dedicarle dos clases enteras buscando que los alumnos entiendan el proceso, lo cual implica que se gasta para ello un tercio del tiempo dedicado a esta parte del tema.

Al desarrollar ejercicios se emplea tiempo considerable, alrededor de 5 minutos por cada etapa, en la diagramación de las estructuras necesarias: registros, buses, posiciones de memoria, etc.

Existen otros factores que llevan a que al estudiante le tome más tiempo resolver un ejercicio, como haber cometido errores en el desarrollo del problema, como ser el confundir registros, errores de transcripción de un diagrama al siguiente o en el manejo de direcciones de memoria.

Un dato importante a considerar es que a las dos semanas de iniciado el tema, la docente toma una prueba periódica, en la cual todos los semestres aproximadamente el 50% de los estudiantes reprueban.

Cuando la docente asigna ejercicios para que el estudiante los resuelva en casa, el estudiante los resuelve por su cuenta, por lo que si desea cerciorarse de que realizó un trabajo correcto solo puede revisarlo el mismo hasta la siguiente clase.

En algunas ocasiones la resolución de ejercicios se torna algo mecánica, por lo que cuando se cambia la arquitectura del CPU o el formato de instrucción en los ejercicios en clase los estudiantes no saben cómo resolver la situación planteada.

Antecedentes tecnológicos

Actualmente existen herramientas de ayuda para la comprensión del ciclo de instrucción del procesador, como por ejemplo:

Microprocessor Tutorial: es un tutorial en línea en el cual se muestra paso a paso el desarrollo del ciclo de instrucción, utilizando lenguaje ensamblador. Permite observar, de forma general, los pasos que sigue el procesador para ejecutar una instrucción, con ejemplos ya definidos. El tutorial a su vez hace una explicación tanto de este tema, como de otros relacionados, como por ejemplo la estructura del CPU (ALU, unidad de control, registros, bus del sistema), pipelining y arquitecturas de computadora. También cuenta con pequeñas pruebas al finalizar cada tema para asegurar la comprensión de cada uno.

Pep/7 Simulator: es un applet en Java que simula el ciclo de instrucción, manejando lógica de instrucciones en ensamblador. Este programa es utilizado para explicar el funcionamiento del procesador, contando con siete registros y cuatro bits de estado

(flags: negative, zero, overflow y carry). Entre sus opciones permite ver los registros en binario, decimal y hexadecimal. Se enfoca principalmente en el manejo de registros del procesador y de la memoria principal. Así mismo, cabe recalcar que cuenta con un tutorial en línea disponible para los usuarios.

Dan! 71 CPU Simulator: este simulador muestra lo que sucede dentro una CPU durante el ciclo de instrucción. Permite crear, guardar y cargar programas que puede ejecutar. Maneja los datos en lenguaje maquina y está diseñado para ser manejado por usuarios que conocen a profundidad el tema. Muestra el contenido de todos sus registros en binario y permite ejecutar un programa paso por paso, sin mostrar el procedimiento a seguir en cada uno de ellos, o mostrar directamente el resultado. Así mismo, cabe recalcar que el simulador cuenta con un manual en línea disponible para que el usuario aprenda a utilizarlo.

El uso de simuladores aplicados al proceso de enseñanza-aprendizaje del ciclo de instrucción trae consigo una gran cantidad de ventajas, especialmente visuales, pero pone a los estudiantes en un estado pasivo al realizar todo el trabajo por ellos.

Problema

Situación Problemática

- Al realizar los esquemas de los registros de procesador y de memoria principal de manera entendible y ordenada se incrementa el tiempo para resolución del problema en sí, así mismo desvía la atención del estudiante, que en vez de enfocarse en la dependencia entre cambios arquitectónicos del procesador y la velocidad de ejecución de un programa, se dedica más a un trabajo mecánico de dibujo de componentes del CPU, de memoria y de sus formas de interconexión.
- Al realizar el estudiante los ejercicios de ciclo de instrucción realizando varios diagramas por separado no puede ver claramente lo que está pasando en el proceso de interacción entre procesador y memoria principal.

- La elaboración manual del proceso conlleva a que el estudiante cometa errores en la transcripción del contenido de los registros entre un diagrama y otro.
- Durante la elaboración de un ejercicio si se comete un error, el resto del flujo de datos e instrucciones es incorrecto y uno se da cuenta de eso solamente en la siguiente clase cuando se hace la revisión del mismo con el docente, lo cual es inadecuado para el proceso de auto-aprendizaje.
- Los simuladores existentes relacionadas al tema están dedicados a una sola arquitectura del CPU y a la vez no son fáciles de utilizar, lo cual exige tiempo extra para su aprendizaje, al mismo tiempo realizan todo el trabajo por el estudiante y disminuye el aprovechamiento deseado. Tal es el caso del Pep/7 Simulator y del Dan! 71 CPU Simulator.
- El Microprocessor Tutorial, es muy simple, y a pesar de ser un tutorial didáctico maneja datos pre-definidos para la explicación. A su vez, no se muestra de manera clara la instrucción que se está ejecutando en el procesador ya que utiliza una notación abreviada para ello, en lugar de utilizar el CO.

Formulación del Problema

La elaboración manual de los diagramas y las limitaciones conceptuales y técnicas de las herramientas aplicadas al proceso de enseñanza-aprendizaje del ciclo de instrucción del procesador, dificultan[1] el proceso de autoaprendizaje[2] respecto a la dependencia entre las prestaciones del procesador y su diseño arquitectónico.

Objetivo

Objetivo General

Desarrollar una herramienta interactiva para el apoyo al proceso de enseñanza-aprendizaje del ciclo de instrucción del procesador.

[1] Hacer difícil una cosa, introduciendo obstáculos o inconvenientes que antes no tenía.
[2] es la forma de aprender por uno mismo. Se trata de un proceso de adquisición de conocimientos, habilidades, valores y actitudes, que la persona realiza por su cuenta ya sea mediante el estudio o la experiencia.

Objetivos Específicos y Acciones

Tabla 1. Objetivos específicos y acciones.

Objetivos Específicos	Acciones
Diseñar componentes de interfaz que permitan visualizar arquitectura del CPU.	Elegir metodología de desarrollo de software.
	Elegir arquitectura de desarrollo.
	Elegir lenguaje y herramientas de programación.
	Diseñar la estructura de la arquitectura del CPU basada en registros específico.
	Diseñar la estructura de la arquitectura del CPU basada en registros de uso general.
	Diseñar estructura de memoria principal y de buses externos del CPU.
Desarrollar módulo interactivo para trabajar con lenguaje binario.	Definir datos de entrada.
	Implementar componente para la definición de formato de instrucción y códigos de operación.
	Implementar componente para preparación de la RAM para el ejercicio por parte del estudiante.
	Desarrollar componente de interacción con el estudiante, para desplazar el contenido de registros.
	Desarrollar componente interactivo de verificación de errores.
Desarrollar módulo interactivo para trabajar con lenguaje hexadecimal.	Definir datos de entrada.
	Implementar selección de arquitectura del CPU basada en registros.
	Implementar componente para la selección de formato de instrucción y códigos de operación.
	Implementar componente para preparación de la RAM para el ejercicio por parte del estudiante.

Objetivos Específicos	Acciones
Desarrollar módulo interactivo para trabajar con lenguaje hexadecimal.	Desarrollar componente de interacción con el estudiante, para el desplazamiento de contenido de registros.
	Desarrollar componente interactivo de verificación de errores.
Desarrollar módulo interactivo para trabajar con lenguaje ensamblador.	Definir datos de entrada.
	Implementar selección de arquitectura del CPU basada en registros.
	Implementar componente para la selección de formato de instrucción y códigos de operación.
	Implementar componente para preparación de la RAM para el ejercicio por parte del estudiante.
	Desarrollar componente de interacción con el estudiante, para el desplazamiento de contenido de registros.
	Desarrollar componente interactivo de verificación de errores.

Fuente: Elaboración propia 2014.

Alcances

Alcances del proyecto:

- La herramienta será un sistema de escritorio.
- La herramienta se desarrollará para el sistema operativo Windows.
- La realización de las pruebas de la herramienta se llevará a cabo con colaboración de los estudiantes de la UCB que cursaron la materia de Arquitectura de Computadoras como máximo hace dos años y con los que cursaran la misma en el semestre I-2015.

- La herramienta no será diseñada para ningún procesador real, ya que se trabajará solamente sobre los componentes básicos de una CPU para la comprensión del ciclo de instrucción.

Alcances del producto:

- El sistema trabajará con direccionamiento en lenguaje binario, hexadecimal y ensamblador.
- Será capaz de manejar arquitectura basada en registros específicos y en registros de uso general.
- Al realizar un ejercicio la herramienta controlará si los datos introducidos y secuencia de micro-operaciones son correctos, generando advertencias de error para hacer posible la detección y corrección de errores por parte del estudiante.
- El estudiante será capaz de definir el estado inicial del CPU y de la memoria principal.
- Visualizar la interacción entre CPU y Memoria Principal.
- Generar un registro de seguimiento de acciones realizadas satisfactoriamente por el estudiante (*log* de operaciones).

Limitaciones

- La resolución de ejercicios de ciclo de instrucción en lenguaje binario solo utilizaran formato de instrucción con una dirección en una arquitectura del CPU basada en registros específicos.
- La resolución de ejercicios ciclo de instrucción en lenguaje hexadecimal y ensamblador utilizaran formato de instrucción con dos direcciones como máximo en una arquitectura del CPU basada en registros específicos y en registros de uso general.
- La longitud máxima de una instrucción en hexadecimal será de 64 bits y en binario será de 16 bits.

- La longitud del código de operación y de las direcciones será medida en bits y tendrán un valor que sea múltiplo de 4, esto debido a que de esta forma se podrá comparar un ejercicio en binario con uno en hexadecimal.

Justificación

Justificación Social

Al utilizar esta herramienta el estudiante no perderá más tiempo al realizar los esquemas y en la transcripción de sus datos lo cual le permitirá concentrarse directamente en la resolución de los ejercicios, al mismo tiempo el estudiante podrá visualizar claramente lo que pasa en el proceso de interacción entre CPU y memoria principal.

La lógica de detección de errores permitirá al estudiante estar consciente de cuando está cometiendo un error o cuando está haciendo un ejercicio de la manera correcta, enriqueciendo el proceso de enseñanza-aprendizaje. Sumado al punto anterior se verá reducido el tiempo de resolución de los ejercicios con lo que el estudiante podrá enfocarse en el objetivo principal del tema.

Con la aplicación de la herramienta en el avance en clases de la materia de Arquitectura de Computadoras, se espera que el estudiante haga un seguimiento con esta herramienta de enseñanza, facilitándose esta labor al encontrarse desarrollado en español y manejando diversas arquitecturas, además de permitir al estudiante no solo ver ejemplos definidos con animaciones a una velocidad adecuada, sino también resolver sus propios ejercicios con verificación en caso de que haya cometido algún error.

Justificación Técnica

Uno de los principales objetivos de éste proyecto es que el sistema sea intuitivo, mostrando una interfaz de usuario amigable y fácil de entender.

Será una herramienta que le permitirá al estudiante elegir su propio formato de instrucción, así como elegir entre una arquitectura basada en registros específicos y en

registros de uso general, por lo que podrá desarrollar sus propios ejercicios, al mismo tiempo de asegurarse que el proceso que está siguiendo es el correcto.

El estudiante no solo definirá la estructura del procesador, sino que también definirá el estado inicial de la memoria principal donde estará guardado el programa que desea ejecutarse.

CAPÍTULO 1. MARCO TEÓRICO

1.1. ARQUITECTURA DE COMPUTADORAS

1.1.1. Ciclo de instrucción

El **ciclo de instrucción** es el periodo de tiempo en el que el procesador lleva a cabo una instrucción. Este tiempo varía de acuerdo de acuerdo a la arquitectura utilizada en el procesador.

Existen diversas arquitecturas que fueron utilizadas para diseñar procesadores, y cada arquitectura tiene definidos diferentes registros en su interior, también diferentes formatos de registros y de instrucciones, al igual que diferentes códigos de operación, por lo cual el tiempo en la ejecución de un mismo proceso en 2 procesadores diferentes puede variar.

Un ciclo de instrucción se divide en dos partes marcadas, el ciclo de captación o de búsqueda y el ciclo de ejecución, y a pesar de que en una misma arquitectura el ciclo de captación toma siempre l mismo tiempo, en el de ejecución es variable.

El **ciclo de captación** consiste en el intervalo en el cual el procesador toma la dirección almacenada en el registro puntero, va a la dirección de memoria almacenada en el puntero y posteriormente captura la instrucción almacenada en esta dirección de memoria llevándola a otro registro para su posterior ejecución.

El **ciclo de ejecución** consiste en la ejecución de la instrucción almacenada en uno de sus registros (IR), el cual está designado específicamente para almacenar instrucciones, y se ejecuta de la siguiente manera: Son separados de este registro el comando de instrucción y la dirección de memoria a la que deberá accederse posteriormente, estos datos se extraerán de acuerdo al formato de instrucción que utilice la arquitectura de cada procesador, y se ejecutará el comando de instrucción recuperado.

1.1.2. Arquitectura del Procesador

La **arquitectura del procesador** consiste en la estructura que toma el CPU, tamaño de registros (bits), formato de instrucción y Códigos de Operación, y debido a todos estos factores la velocidad de ejecución de una instrucción es variable.

Cuando se habla acerca de la **estructura del CPU** nos referimos a los recursos con los que el CPU cuenta y su organización para desempeñar su función. Entre las estructuras de que se utilizaran dentro del proyecto, basados en lo avanzado en la materia de Arquitectura de Computadoras en la Universidad Católica con Mgr. Galaburda como docente se encuentran las siguientes:

- En la primera estructura utilizada, la de arquitectura basada en registros específicos, el CPU contiene los registros PC, IR, AC, DR, MAR, MBR, I/O AR, I/O BR, al igual que contiene a la ALU y la Unidad de Control, además de contar con buses internos y externos, los cuales constan a su vez de, un bus de control, bus de datos y bus de direcciones.
- En la segunda estructura, la de arquitectura basada en registros de uso general, el CPU utiliza un Banco de Registros en lugar de los registros AC y DR para almacenar datos, es decir, registros enumerados desde R1 hasta RN.

En ambos casos para interactuar con la memoria, el procesador utiliza los registros MAR y MBR, que se conectan con los buses internos del CPU, y estos a su vez con los buses externos para finalmente conectarse con Memoria.

Cuando el procesador debe interactuar con dispositivos de E/S utiliza exactamente la misma lógica con la diferencia que utiliza los registros I/O AR e I/O BR en lugar de los registros MAR y MBR respectivamente.

El **formato de instrucción** se refiere a la estructura en la cual se dividirá el registro de instrucciones la su posterior ejecución, es decir, la cantidad de bits asignados para el código de operación, la cantidad de direcciones de memoria a las que se accederá, el

modo de direccionamiento de asignado para cada dirección en la instrucción y la cantidad de bits asignados para cada dirección.

Los **modos de direccionamiento** es la forma en la cual el CPU recopila los datos guardados en RAM y dispositivos de E/S, los modos de direccionamiento utilizados serán son los siguientes:

- El **directo**, en el cual el procesador accede a la dirección de memoria que se encuentra dentro de su registro, obtiene el dato de esa posición y lo procesa.
- El **indirecto**, en el cual el procesador accede a la dirección de memoria que se encuentra dentro de su registro, obtiene una nueva dirección de memoria, obtiene el dato de esa posición y lo procesa.
- El **inmediato**, en el cual la instrucción tiene dentro de sí el dato que necesita para ser ejecutada, sin necesidad de recuperarlo de memoria.

Los **códigos de operación (CO)** son los comandos definidos dentro del procesador utilizados para ejecutar una instrucción, estos trabajan con modos de direccionamiento definidos.

Entre los recursos del procesador nombrados anteriormente y que serán utilizados en la resolución de este proyecto tenemos los siguientes:

- **Program Conuter (PC):** es el contador de programa, es donde se encuentra la dirección de memoria siguiente a la q se accederá para obtener una nueva instrucción. Cada vez que un ciclo de captación culmina el PC se incrementa en uno.
- **Instruction Register (IR):** este registro es el encargado de decodificar una instrucción, es decir de separar el CO y las direcciones.
- **Acumulator (AC):** este registro es utilizado para almacenar datos y posteriormente ser utilizados para operaciones con ALU, cuando interactúa con esta, además de ser el registro q provee d los datos también es el registro que recibe el resultado.

- **Data Register (DR):** como su nombre indica es un registro utilizado para almacenar datos, posteriormente es utilizado como proveedor de datos para operaciones de ALU.
- **Memory Address Register (MAR):** Es el registro utilizado para almacenar la siguiente dirección de memoria a la cual se accederá e interactúa directamente con el bus interno del CPU.
- **Memory Buffer Register (MBR):** Es el registro utilizado para almacenar los datos obtenidos desde memoria o para almacenar el dato que será guardado en la dirección de memoria a la cual se accederá e interactúa directamente con el bus interno del CPU.
- **Banco de registros:** como su nombre da a entender, es una cantidad definida de registros en los cuales se almacenaran datos obtenidos desde memoria o que serán destinados a la misma, y pueden interactuar con la ALU ya sea como entrada o como salida.
- **Arithmetical Logical Unit (ALU):** es el recurso del CPU encargado de las operaciones de aritméticas y lógicas, recibiendo datos de los registros del CPU y devolviendo el resultado a uno de ellos.

1.1.3. Interacción entre Procesador y Memoria Principal

La interacción entre memoria y procesador es llevada a cabo mediante buses, los cuales transportan el contenido de registros y señales de control tanto dentro como fuera del procesador (hacia memoria o dispositivos E/S), los buses utilizados por un procesador se clasifican:

- **Por su posición en relación al CPU:**
    - **Internos:** son los buses que transportan la información dentro del procesador, es decir, entre registros, ALU y Unidad de Control.
    - **Externos**: son los buses que transportan la información hacia la memoria principal u otros dispositivos (como los dispositivos de E/S).

- **Por el tipo de datos que transportan:**
    - **Bus de direcciones:** este bus es el encargado de transportar las direcciones de memoria o de dispositivos E/S.
    - **Bus de datos:** este se encarga de transportar los datos almacenados en los registros de memoria, dispositivos E/S y de los recursos del CPU.
    - **Bus de control:** Se encarga de transportar los códigos de control generados por la unidad de control.

1.1.4. Ejercicios

Para tener un buen aprendizaje del ciclo de instrucción del procesador no basta con la enseñanza de conceptos básicos e importantes, sino también es necesario conocer el proceso que lleva a cabo un procesador al ejecutar una instrucción, por lo que la resolución de ejercicios es indispensable.

El estudiante debe llevar el contenido de un registro a otro, de acuerdo a los pasos del ciclo de captación y del ciclo de ejecución de cada instrucción, tal como lo haría un procesador, así como también cambiar el contenido de un registro o llevar el contenido de un registro de CPU o de la RAM a uno de los buses.

La resolución de ejercicios dependerá tanto del formato de instrucción como del lenguaje y de la arquitectura del procesador.

Los lenguajes utilizados para la resolución de ejercicios son los siguientes:

- **Binario:** El procesador ejecuta cada una de las instrucciones en lenguaje binario, por lo que para el procesador un código de operación o una dirección de memoria no es nada más que ceros y unos, cuando nosotros vemos las mismas instrucciones en lenguaje en alto nivel.

  Por tanto, en los ejercicios en binario el estudiante debe hacer el papel de un procesador, por lo que debe definir los códigos de operación para en binario en base al formato de instrucción especificado, así como también debe definir el estado inicial del registro PC y de la RAM, todo en binario.

- **Hexadecimal:** Si bien el procesador ejecuta los ejercicios en binario, el contenido de los registros para el ojo humano llega a ser demasiado cargado cuando se sobrepasa los 16 bits, por lo que por fines académicos se utiliza la notación en hexadecimal, reemplazando 4 bits por un dígito hexadecimal, haciendo que todo lo que se debía hacer en los ejercicios en binario, se haga en hexadecimal.
- **Ensamblador:** cuando ya se ha captado como ejecuta una instrucción un procesador en lenguaje binario, se empiezan a resolver ejercicios en lenguaje ensamblador, el cual es un lenguaje de programación de bajo nivel, el más cercano al lenguaje máquina.

  En este lenguaje se resuelven los ejercicios con el formato de instrucción siguiente: código de operación seguido de las direcciones en decimal separadas por comas.

Por tanto, en los ejercicios en binario el estudiante debe hacer el papel de un procesador, por lo que debe definir los códigos de operación para en binario en base al formato de instrucción especificado, así como también debe definir el estado inicial del registro PC y de la RAM, todo en binario.

Posteriormente el estudiante debe llevar el contenido de un registro a otro, de acuerdo a los pasos del ciclo de captación y del ciclo de ejecución de cada instrucción, tal como lo haría un procesador, así como también cambiar el contenido de un registro o llevar el contenido de un registro de CPU o de la RAM a uno de los buses.

1.2. SOFTWARE EDUCATIVO

1.2.1. Definición

Recibe la denominación de **software educativo** al software destinado a apoyar al proceso de enseñanza ï aprendizaje, promoviendo el desarrollo cognitivo del usuario en ciertas áreas específicas, englobando todos los sistemas elaborados con un fin didáctico.

Todo programa educativo puede tratar diferentes áreas del conocimiento humano, con enfoques y herramientas diversas, pero siempre buscando principalmente reflejar un entorno lo más sensible posible a las circunstancias del usuario final (normalmente docentes y estudiantes), y buscando también ser lo más intuitivo e interactivo posible. De esta forma un programa educativo debe ser enfocado para un trabajo individualizado de cada estudiante.

1.2.2. Tipos de software didáctico

1.2.2.1. Programas Tutoriales

Son programas que son orientados a la enseñanza desde un enfoque de apoyo al maestro, mostrándole un enfoque práctico de lo que avanzan teóricamente en clases.

Con estos programas, los estudiantes ven dinámicamente la resolución de ejercicios, es decir, visualizan un ejercicio planteado, y ven la forma en la cual deben resolverse, ya sea mostrando el estado final directamente, o inclusive mostrando el procedimiento de resolución paso a paso. La forma en la cual se realiza la transición entre un paso y otro dentro de la resolución del ejercicio puede ser diferente, desde el enfoque de animación, ya que algunas transiciones pueden contener animaciones, o bien todas o ninguna.

Un programa tutorial se puede dedicar solamente a darle un enfoque práctico y no contempla ningún tipo de explicación conceptual, así como también puede dar dicha explicación ya sea previamente o durante el desarrollo del ejercicio.

Un programa tutorial también puede interactuar constantemente con el usuario, a lo que se llama tutorial interactivo, en este caso el programa responde a ciertas acciones que el estudiante realiza, o a ciertas opciones que este elige, pudiendo cambiar la reacción del programa en base a las acciones del usuario (ramificados) o no (lineales).

Por último están los Sistemas Tutoriales Expertos, los cuales utilizan bases de Inteligencia Artificial para definir la metodología de enseñanza de un caso de estudio

acorde con las necesidades de cada estudiante, basando su elección en un diálogo con el mismo y realizando un proceso de retroalimentación para el programa y el estudiante.

1.2.2.2. Bases de datos

Son sistemas orientados a la enseñanza mediante el almacenamiento de datos estáticos, para facilitar su acceso, exploración y selección, en base a determinados criterios.

Su empleo básicamente se maneja de la siguiente manera: el usuario realiza una consulta (la forma es dependiente de la presentación de la interfaz de usuario), el sistema realiza la consulta a base de datos, la interfaz de usuario despliega la información.

Su uso principalmente es en el campo de ciencias exactas para el manejo de datos y sus características, en la enseñanza de estructuras jerárquicas, entre otros menos comunes.

Así como estos sistemas en su gran mayoría son sistemas basados en bases de datos convencionales, los *sistemas expertos* también pertenecen a ésta categoría, con la diferencia que un sistema experto es capaz de asesorar al usuario en su proceso de búsqueda de información.

1.2.2.3. Simuladores

Representan un entorno en el cuál se desarrolla un proceso en específico, mediante la visualización de una representación del dicho proceso.

Tiene como principal objeto que el estudiante se familiarice con el proceso que se desea estudiar y, si es posible, con su entorno. De esta forma el estudiante puede familiarizarse también con los cambios que traerían al proceso de estudio un cambio de datos de entrada estructural, o inclusive un cambio estructural.

Su orientación está dirigida al estudio de un entorno real, o al menos uno cercano a dicha realidad, y siendo primordialmente necesaria una interacción con el usuario, la cuál llegará a depender de las necesidades del usuario, así como también las del mismo entorno simulado.

Por tanto existen 3 tipos de simuladores basados en la naturaleza de su interacción con el usuario:

- Inmersivos: simulan un entorno real con componentes reales de la simulación, reaccionando ante las acciones del usuario y del entorno en caso de que sea necesario.
- Semi - inmersivos: simulan un entorno real o cercano a la realidad, pero sin utilizar componentes relacionados, siendo comúnmente sistemas de computadora que interactúan con el usuario de forma activa.
- No Inmersivos: simulan un entorno real o cercano a la realidad, pero sin utilizar componentes relacionados, siendo sistemas de computadora que interactúan con el usuario de forma pasiva, es decir, que el usuario es el que realiza todo el trabajo.

1.2.2.4. Constructores

Son programas que tienen un entorno programable. Facilitan a los usuarios unos elementos simples con los cuales pueden construir elementos más complejos o entornos. De esta manera potencian el aprendizaje heurístico y, de acuerdo con las **teorías cognitivistas**, facilitan a los alumnos la construcción de sus propios aprendizajes, que surgirán a través de la reflexión que realizarán al diseñar programas y comprobar inmediatamente, cuando los ejecuten, la relevancia de sus ideas. El proceso de creación que realiza el alumno genera preguntas del tipo: **¿Qué sucede si añado o elimino el elemento X?** (PEREZ MARQUÉS).

1.2.2.5. Herramientas interactivas

Son programas que proporcionan un entorno instrumental con el cual se facilita la **realización de ciertos trabajos generales** de tratamiento de la información: escribir, organizar, calcular, dibujar, transmitir, captar datos, etc. (PEREZ MARQUÉS).

Las herramientas interactivas pueden ser aplicables a la resolución de problemas de distintas áreas del saber humano, no solamente a la del proceso de enseñanza-aprendizaje. Tal es el caso de sistemas que necesitan una interacción con el usuario para realizar ciertas acciones, de tal forma que a medida que el usuario responda, la respuesta del sistema también puede variar.

Las herramientas interactivas necesitan del usuario para cumplir con su finalidad, ya que basan su utilidad en el concepto de la *retroalimentación* entre la herramienta y el usuario, siendo necesario el usuario para que el programa se ejecute.

Un ejemplo claro de herramientas interactivas son los cajeros automáticos, los cuales interactúan constantemente con el usuario, llevando a un proceso de retroalimentación constante, siendo que si el usuario no realiza una acción el sistema no genera una reacción sino hasta la respuesta del usuario.

Estas herramientas interactivas pueden ser orientadas al aprendizaje de un área o tema en específico tomando características básicas de los otros tipos de software anteriormente mencionados. De esta forma esta categoría puede ser una combinación entre un programa tutorial, un simulador y un sistema interactivo común y corriente.

La facilidad de adaptar este tipo de sistemas a las necesidades del usuario para su aplicación en el área de enseñanza-aprendizaje hace este tipo de herramientas una de las más fáciles de aplicar.

Una herramienta interactiva puede ser orientada no solamente a que el estudiante vaya viendo el progreso o cambio que se produce con ciertas acciones que realice, sino que también el estudiante pueda ir modificando valores de la interfaz de tal forma que se vea mas involucrado dentro de la ejecución del programa. De esta forma estas herramientas pueden ser utilizadas dentro del proceso de enseñanza aprendizaje de distintas maneras.

La resolución de ejercicios por parte de un estudiante por medio de una herramienta interactiva es una de las aplicaciones más comunes de este tipo de herramientas dentro del software educativo.

1.3. METODOLOGÍA DE DESARROLLO DE SOFTWARE

1.3.1. Generalidades

> Un proceso de desarrollo de software detallado y completo suele recibir el nombre de ñMetodologíaò. Una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas. Habitualmente se utiliza el término ñmétodoò para referirse a técnicas, notaciones y guías asociadas, que son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis o diseño (ANÓNIMO, 2014).
>
> La comparación y/o clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas (ANÓNIMO, 2014).
>
> ñUna determinada metodología no es necesariamente aplicable a cualquier tipo de proyecto, es más, para cada tipo de proyecto existe una metodología que se adapta mejor a sus necesidadesò **(**HERMES ROMERO, 2012**).**

La ingeniería de software requiere llevar a cabo numerosas tareas, dentro de etapas como las siguientes:

- **Captura de Requerimientos:** primeramente para el desarrollo de todo tipo de software, se debe analizar y definir los requerimientos funcionales, es decir, requisitos que el sistema debe satisfacer para con el campo de trabajo al que será aplicado.
- **Análisis:** posteriormente se analizan los requerimientos y se van definiendo requerimientos no funcionales y funcionalidades principales.
- **Diseño:** el siguiente paso es diseñar el sistema en sí, modelarlo para su implementación posterior, se diseñan las clases y la forma en que se llegaran a satisfacer mediante la interacción de estas dentro del sistema.

- **Implementación:** en base al diseño realizado anteriormente se empieza a implementar el modelo y se lo vuelve un sistema.
- **Experimentación:** finalmente se prueba que el sistema implementado cumpla con todos los requerimientos funcionales y no funcionales, se buscan errores y se los corrige, además de afinar los últimos detalles.

Cada una de estas tareas dentro del proceso de desarrollo varía en estructura, secuencialidad e importancia, dependientemente de la metodología utilizada.

Es imperante definir claramente las características de un proyecto antes de la selección de una metodología para su desarrollo, puesto que si se elige una metodología sin tomarlas en cuenta, en lugar de ayudar al proceso de desarrollo solo lo entorpecería.

1.3.2. Selección de metodología.

Tabla 2. Tabla comparativa de metodologías de desarrollo.

CRITERIO	CASCADA	INCREMENTAL	PROTOTIPADO	ESPIRAL
Disponibilidad De recursos	Todos	Algunos	Algunos	Algunos
Complejidad del proyecto	Baja	Media	Media	Alta
Entendimiento de requeriminetos	Especifico	Vago	Vago	Vago
Tecnologia del producto	Vieja	Nueva	Vago	Vago
Manejo de la perspectiva del riesgo	No	Si	Si	Si
Conocimiento del dominio del problema	Alto	Regular	Regular	Pobre

Fuente: CHIPIA LOBO, 2010

Características principales del proyecto:

- Requerimientos funcionales aun ambiguos.
- Necesidad de retroalimentación.
- Riesgo alto de insatisfacción de los usuarios finales.
- Disponibilidad de recursos reducidos.
- Necesidad de contar con versiones del sistema funcionales.

El utilizar el prototipado como metodología de desarrollo claramente satisface las características principales del proyecto, dando también un mayor nivel de flexibilidad en el desarrollo, siendo una de las metodologías más utilizadas en el software educativo.

1.3.3. Prototipado

Esta metodología de desarrollo está comprendida entre los modelos de desarrollo evolutivo. Como tal, cada prototipo debe utilizar pocos recursos y debe ser construido en un tiempo razonablemente corto, orientado a la retroalimentación con el usuario final.

Existen dos maneras de usar el Prototipado, de una manera incremental reutilizando prototipos anteriores, a esto se lo denomina evolutivo. La otra manera es con prototipos ñdesechablesò, es decir, que se rescata todo lo importante y se rediseña el sistema con las correcciones y se incrementa en cuanto a cobertura de requerimientos.

En el momento en que se hace la captura de requerimientos para el posterior desarrollo de cualquier tipo de software, es muy frecuente que el cliente tenga definido el objetivo y objetivos principales del software a implementar, pero que no tenga claro o simplemente no especifique requerimientos más allá de estos. De la misma forma en algunas ocasiones el cliente no tiene claros no solamente requerimientos funcionales, sino que tampoco tiene claros algunos requerimientos no funcionales.

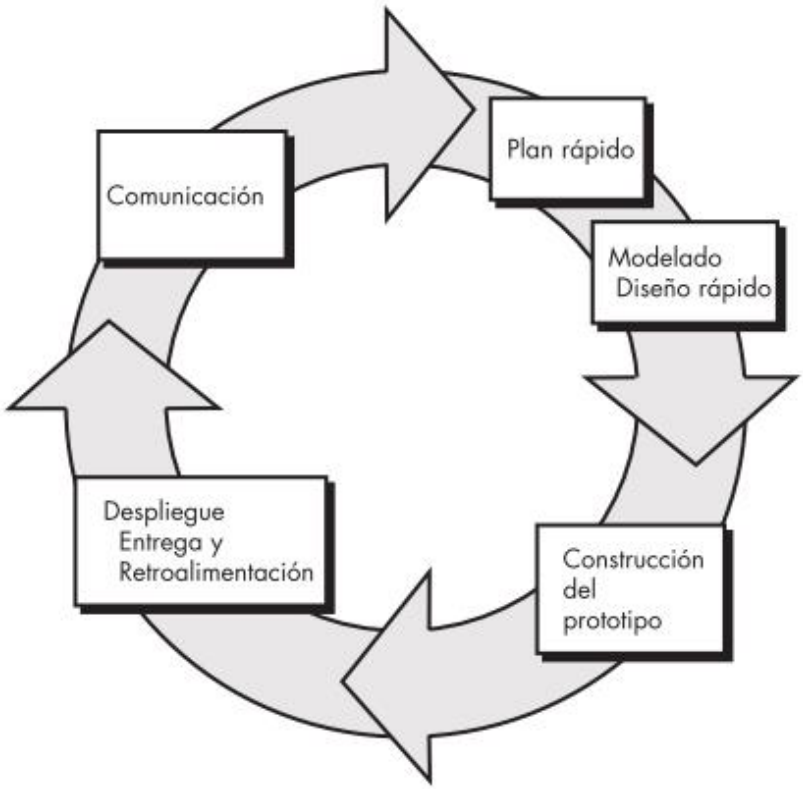
Muchas otras veces se ve que durante el desarrollo del sistema los desarrolladores tienen ciertas dudas sobre el uso de algunas herramientas o algoritmos, de la adaptabilidad del sistema a distintos entornos, o de la misma interacción con el usuario.

En todos estos casos el paradigma de hacer prototipos es, posiblemente, la mejor opción para el desarrollo.

Así cómo es posible el desarrollo por prototipos como el proceso de desarrollo principal, es comúnmente utilizado en complementación con algún otro proceso de desarrollo. Ya sea de una u otra forma, este paradigma es una ayuda idónea para la comprensión y definición de requerimientos.

El paradigma de hacer prototipos comienza con comunicación. Usted se reúne con otros participantes para definir los objetivos generales del software, identifica cualesquiera requerimientos que conozca y detecta las áreas en las que es imprescindible una mayor definición. Se planea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado (en forma de un ñdiseño rápidoò). Éste se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales (por ejemplo, disposición de la interfaz humana o formatos de la pantalla de salida). El diseño rápido lleva a la construcción de un prototipo. Éste se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida de que el prototipo es afinado para satisfacer las necesidades de distintos participantes, y al mismo tiempo le permite a usted entender mejor lo que se necesita hacer. (PRESSMAN, 2010: 37)

Figura 2. Paradigma de hacer prototipos.

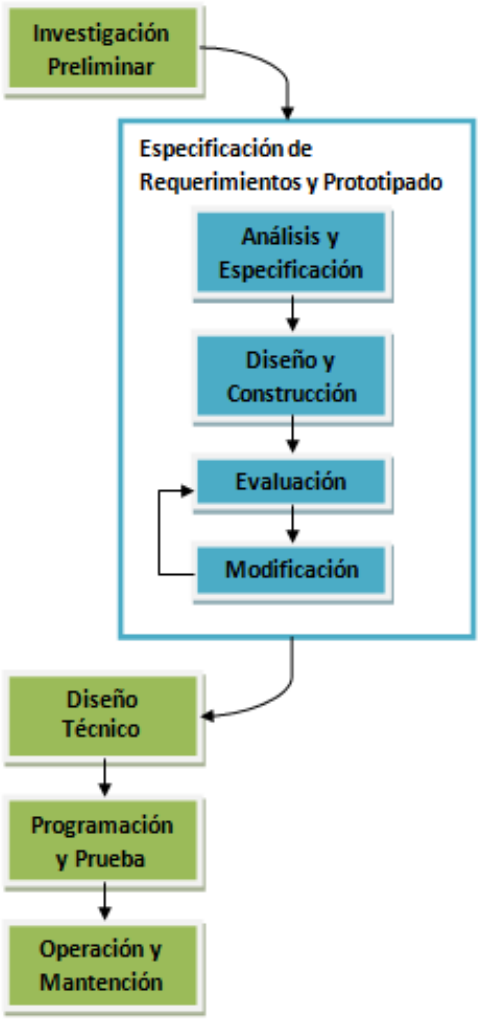


Fuente: PRESSMAN 2010.

La idea principal del prototipado es que se halle en el proceso de desarrollo un mecanismo que ayude a tener bien definidos los requerimientos del sistema. De esta forma, para cada prototipo pueden utilizarse fragmentos de programas existentes, aplicar herramientas o reutilizar código de prototipos anteriores, de tal forma que se puedan generar prototipos funcionales de la forma más rápida posible.

1.3.4. Etapas del prototipado

Figura 3. Etapas del Modelo de Prototipos



Fuente: Elaboración propia en base a YANEZ CABRERA 2014

1.4. TECNOLOGÍAS DE DESARROLLO

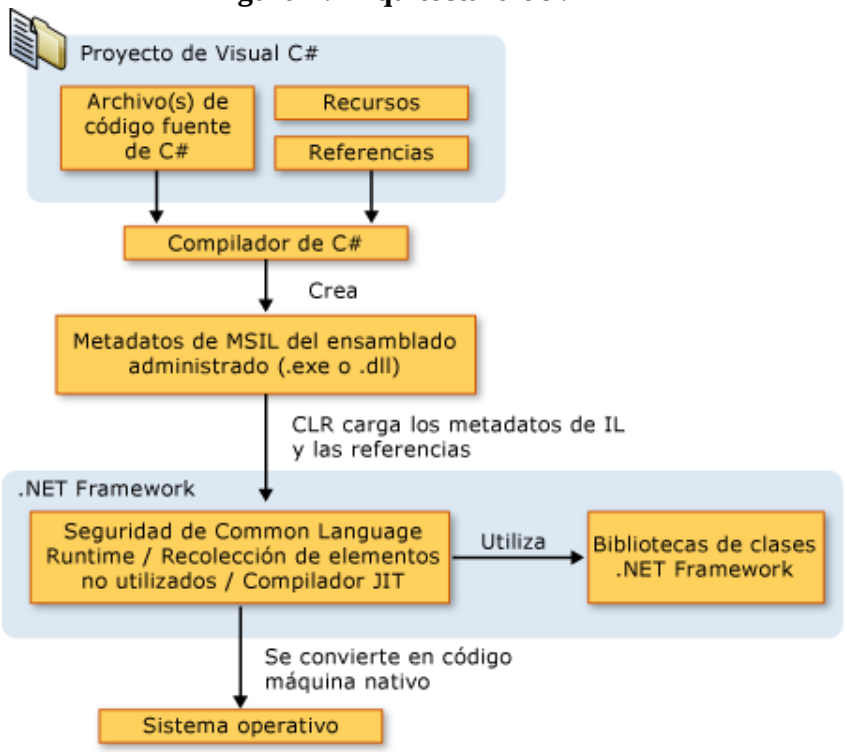
1.4.1 Visual Studio

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C# y Visual C++ utilizan todos el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y hace más sencilla la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes utilizan las funciones de .NET Framework, las cuales ofrecen acceso a tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y Servicios Web XML. (MSDN, 2007)

1.4.2. C#

C# es un lenguaje orientado a objetos elegante y con seguridad de tipos que permite a los desarrolladores compilar diversas aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Puede utilizar C# para crear aplicaciones cliente de Windows, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos, y mucho, mucho más. Visual C# proporciona un editor de código avanzado, cómodos diseñadores de interfaz de usuario, depurador integrado y numerosas herramientas más para facilitar el desarrollo de aplicaciones basadas el lenguaje C# y .NET Framework. (MSDN, 2014)

Figura 4. Arquitectura de .NET



Fuente: MSDN 2014.

1.4.3. Librerías Visual Studio

System: Es la librería principal del programa, para el manejo de datos y sentencias, al igual que el engranaje de compilación.

System.Collections.Generic: contiene interfaces y clases que definen colecciones genéricas, permitiendo que los usuarios creen colecciones con establecimiento inflexible de tipos para proporcionar una mayor seguridad de tipos y un rendimiento mejor que los de las colecciones no genéricas con establecimiento inflexible de tipos.

System.ComponentModel: proporciona clases que se utilizan para implementar el comportamiento en tiempo de ejecución y tiempo de diseño de los componentes y controles. Este espacio de nombres incluye las interfaces y clases base para implementar atributos y convertidores de tipos, enlazar a orígenes de

datos y dotar de licencia a componentes. Las clases en este espacio de nombres se dividen en las siguientes categorías:

- Clases de componentes básicos.
- Licencia de componentes.
- Atributos.
- Descriptores y persistencia.
- Convertidores de tipos.

System.Drawing: proporciona acceso a funcionalidad de gráficos básica de GDI+. Se ofrece una funcionalidad más avanzada en los espacios de nombres System.Drawing.Drawing2D, System.Drawing.Imaging y System.Drawing.Text.

La clase Graphics proporciona métodos para dibujar en el dispositivo de pantalla. Clases como Rectangle y Point encapsulan primitivos de GDI+. La clase Pen se utiliza para dibujar líneas y curvas, mientras que las clases derivadas de la clase abstracta Brush se utilizan para rellenar el interior de las formas.

System.Linq: Librería manejada principalmente para conexión y manejo de datos con SQL.

System.Text: El espacio de nombres **System.Text** contiene clases que representan codificaciones de caracteres ASCII, Unicode, UTF-7, y UTF-8; clases base abstractas para la conversión de bloques de caracteres en bloques de bytes y viceversa; y una clase auxiliar que manipula y da formato a objetos String sin necesidad de crear instancias intermedias de **String**.

System.Windows.Forms: El espacio de nombres **System.Windows.Forms** contiene clases para crear aplicaciones para Windows que aprovechan todas las ventajas de las características de la interfaz de usuario disponibles en el sistema operativo Microsoft Windows. (MSDN, 2014)

1.4.4. Ruby

Ruby es un lenguaje de programación interpretado, creado por el japonés Yukihiro ñMatzò Matsumoto, el cual trabajó en el lenguaje desde 1993 hasta publicarlo en 1995. Pero el lenguaje no alcanzó su reconocimiento masivo sino hasta 2006. Este lenguaje busca incorporar la programación funcional con la imperativa.

Ruby es un lenguaje de código abierto enfocado principalmente a productivo, dinámico y orientado a objetos. Su sintaxis es inspirada en una combinación principalmente de *Python* y *Perl*, entre otros, tal como dice su creador Matz, ñquería un lenguaje que fuera más poderoso que Perl, y más orientado a objetos que Pythonò. (Matz, 2001)

Matz también dejo claro que buscaba con este lenguaje un lenguaje natural, no sencillo: ñRuby es simple en apariencia, pero complejo por dentro, como el cuerpo humanoò. (Matz, 2000)

Ruby es considerado el lenguaje más dinámico y orientado a objetos inventado hasta ahora, y en este lenguaje absolutamente todo es un objeto, el cual a su vez puede ser sobreescrito para para mejorar la funcionalidad dentro de un programa en específico.

1.4.5. Ruby on Rails

Rails es un framework para el lenguaje de programación Ruby, orientado al desarrollo de aplicaciones web. Este framework es de código abierto, y utiliza las ñ*gemas***ò** (bibliotecas) que son distribuidas a travez de *RubyGems,* las cuales son desarrolladas por distintas personas en todo el mundo, para facilitar labores repetitivas dentro del desarrollo de aplicaciones web.

Rails utiliza la arquitectura *Modelo Vista Controlador (MVC)* y trata de combinar, al igual que *Ruby*, la simplicidad de desarrollar código con la facilidad de uso y robustez en el desarrollo.

Uno de los objetivos principales de este framework es la productividad sostenible de aplicaciones desarrolladas con el mismo, facilitando tareas y evitando la repetición de código para que el desarrollador se concentre principalmente en desarrollar lo que hace única a su aplicación como tal.

En la conexión con bases de datos utiliza la herramienta *Active Record*, para enlazar los modelos de negocio con la base de datos, convirtiendo las clases en tablas, los atributos en campos y los objetos relacionados en claves foráneas. Aún así, en Rails pueden crearse modelos q no necesariamente estén enlazados con bases de datos.

1.4.6. JavaScript

JavaScript es un lenguaje de scripting de código abierto cuya sintaxis es muy parecida a la de *Java*. Es un lenguaje orientado a objetos, imperativo, multiplataforma, débilmente tipado e interpretado, utilizado para el desarrollo de aplicaciones web. En este sentido, el uso de JavaScript puede ser de dos formas: del lado del cliente y del lado del servidor.

Del lado del cliente es utilizado para manipular tanto la funcionalidad de un navegador como su modelo de objetos, esto para mejorar la interfaz de usuario y hacerla mas dinámica e interactiva.

Del lado del cliente puede trabajar como un lenguaje de programación cualquiera, normalmente mediante el uso de un framework, proporcionando una lógica de negocio y manejo de base de datos.

Existen muchas extensiones de JavaScript que hoy en día son utilizados constantemente principalmente para el desarrollo de aplicaciones web, tales como *JQuery, AJAX, Angula.js*, entre otros.

CAPÍTULO 2. MARCO PRÁCTICO

2.1. DISEÑO DE COMPONENTES DE INTERFAZ QUE PERMITAN VISUALIZAR ARQUITECTURA DEL CPU.

2.1.1. Prototipo 1

Requerimientos:

- Diseñar la estructura de la arquitectura del CPU basada en registros específico.
- Diseñar estructura de memoria principal y de buses externos del CPU.

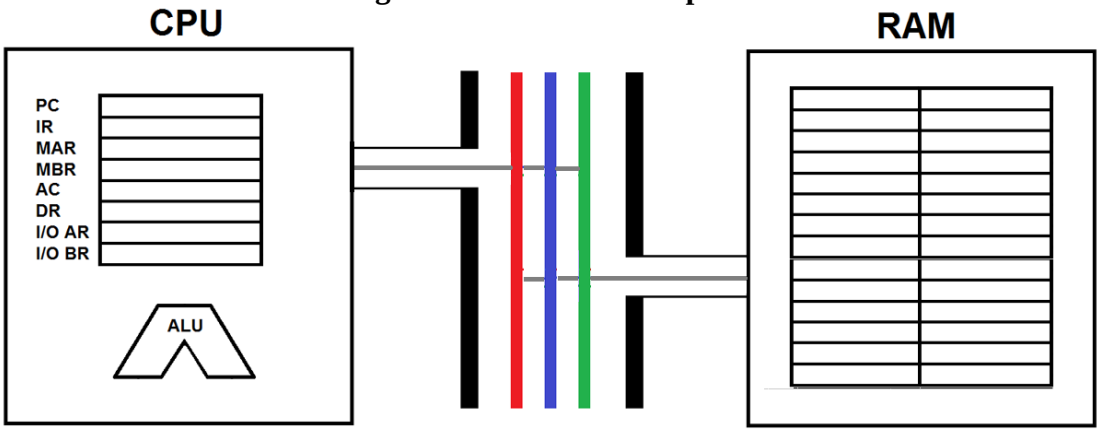
Para este objetivo se opto por diseñar primeramente una imagen fija como fondo de la interfaz de usuario.

Se diseñó en principio una imagen la cual serviría como plantilla para luego instalar los componentes de programación necesarios para que el estudiante pueda ingresar los contenidos de los registros.

Primeramente se plasmó el diseñó del CPU con arquitectura basada en registros específicos y sus componentes en la imagen ñplantillaò, y posteriormente se procedió al diseño de los buses externos del CPU. Por último se diseño la estructura de la memoria principal.

Cuando la imagen ya se encontraba definida, se prosiguió a la instalación de los componentes de programación, instalando un ñTextBoxò para cada registro de CPU y para cada registro y dirección de la memoria principal, además de asignar a cada ñTextBoxò un nombre significativo con su papel en la interacción con el estudiante.

Figura 5. Interfaz Prototipo 1.



Fuente: Elaboración propia 2014

2.1.2. Prototipo 2

Requerimientos:

- Diseñar la estructura de la arquitectura del CPU basada en registros específico.
- Diseñar estructura de memoria principal y de buses externos del CPU.

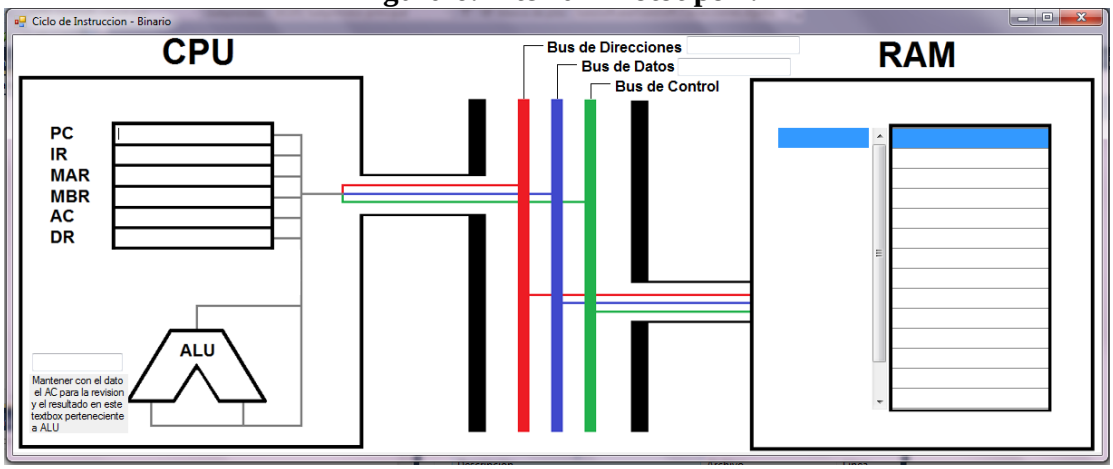
Al mostrar el prototipo a estudiantes que ya cursaron la materia y analizar con este detalladamente las falencias de esta interfaz se pudieron conseguir las siguientes observaciones:

- El uso de los registros I/O AR e I/O BR era innecesario, puesto que la herramienta se encargaría proveer un ambiente para la resolución de ejercicios que reflejan la interacción de CPU con RAM, no con dispositivos de E/S.
- Otra observación importante es que no se encontraba un enlace gráfico entre los componentes de CPU, por lo que se instalo un bus interno de CPU en la interfaz.
- Al salir la instrucción del CPU hacia la RAM, se debería diferenciar hacia donde la información contenida, es decir, hacia cual de los buses.
- En la imagen no queda claro cuál es el *bus de direcciones*, *bus de datos* y *bus de control*.

- Los registros y direcciones de la RAM no deberían estar contenidos en un ñTextBoxò, puesto que esto dificultaría la verificación de pasos del ciclo de instrucción.

Para la instalación de registros y direcciones de la RAM en la interfaz de usuario se decide utilizar dos ñDataGridViewò, en el cual se generarían automáticamente las celdas al cargar el componente de interfaz de usuario.

Figura 6. Interfaz Prototipo 2.



Fuente: Elaboración propia 2014

Para que el ñScrollBarò funcione para ambos ñDataGridViewò de forma simultánea se utilizó el evento ñScrollò de este componente.

También se utilizaron los eventos ñMouseClickò y ñSelectionChangedò para que cuando se seleccione una dirección de memoria, automáticamente se seleccione su correspondiente registro y viceversa.

2.2. DESARROLLO DEL MÓDULO INTERACTIVO PARA TRABAJAR CON LENGUAJE BINARIO.

2.2.1. Prototipo 3

Requerimientos:

- Desarrollar componente de interacción con el estudiante, para desplazar el contenido de registros.

Después de haber conseguido el diseño de una interfaz de usuario definida, se procedió al desarrollo del componente de interacción para este módulo.

El componente se desarrolló para que el estudiante pueda realizar el llenado y manipulación del contenido de registros tanto del CPU como de la RAM, ya sea mediante la escritura de datos desde teclado, como la copia de contenidos de un registro a otro.

Se programaron restricciones en la escritura, solo se permitió el ingreso de datos con los caracteres ñ0ò y ñ1ò, mediante el evento ñKeyPressò, así como también se limitó la longitud máxima de caracteres a ocho, con la propiedad ñMaxLengthò. Si el valor era menor se autocompletaban los registros con ceros.

En la programación de la copia del contenido de registros hubo más complicaciones, por lo que los sucesos importantes se resumen en los siguientes puntos:

- En principio se intento utilizar los eventos ñMouseUpò y ñMouseDownò, pero se presentaron inconvenientes. Si se hacía click en el registro ñAò y se soltaba el botón del mouse en el registro ñBò, de todas formas se ejecutaba el código del evento del registro ñAò y no el del ñBò, como si se hubiese hecho click en ñAò y soltado el botón también en ñAò, por lo que no había ningún cambio.
- Se procedió a la búsqueda de posibles soluciones para cumplir con esta función, analizando las ventajas y desventajas de otros eventos o comandos como ñctrl+cò y ñctrl+vò.

- Finalmente se optó por elegir el evento ñMouseDoubleClickò, siendo que se haría doble click en el registro origen y posteriormente doble click en el registro destino para poder copiar el contenido.
- Se optó por este evento debido a que al ser utilizado el programa con un touchpad en computadoras portátiles, muchas veces el usuario puede cometer un error al hacer un solo click en el lugar incorrecto debido a la sensibilidad del touchpad.

En este prototipo también se decidió generar automáticamente el contenido de la RAM para darle mayor realismo a la interfaz de usuario, generando números consecutivos desde el 0 hasta el 2^8 en binario.

Figura 7. Pseudocódigo MouseDoubleClick.

```
origen <- vacio
destion <- vacio
contenido <- vacio

procedimiento DobleClickRegistro(registro)
    Si (vacio(origen) ^ !vacio(registro(contenido)))
        origen <- registro
        contenido <- registro(contenido)
    c/c
        Si (!vacio(contenido))
            destino <- registro
            Si (destino != origen)
                registro(contenido) = contenido
            fin si
            origen <- vacio
            destion <- vacio
            contenido <- vacio
        fin si
    fin si
fin procedimiento
```

Fuente: Elaboración propia 2014

Figura 8. Pseudocódigo KeyPress.

```
procedimiento KeyPress(caracter)
    Si ((caracter es tecla de navegacion) v caracter = 0 v caracter = 1)
        continuar()
    c/c
        CancelarCaracter()
    fin si
fin procedimiento
```

Fuente: Elaboración propia 2014

Figura 9. Pseudocódigo Llenar DataGriedView.

```
procedimiento CrearTabla(limite, direcciones)
    tabla <- Nuevatablala()
    col <- NuevaColumna()
    AgragarColumna(tabla) <- col
    tam <- potenciar(2,direcciones)
    cont <- 0
    Mientras (cont < tam)
        fila <- CrearFila(tabla)
        fila[col] <- DecimalABinario(cont)
        AgregarFila(tabla(filas) <- fila
        incrementar(cont)
    fin mientras
fin procedimiento, terminar devolviendo tabla
```

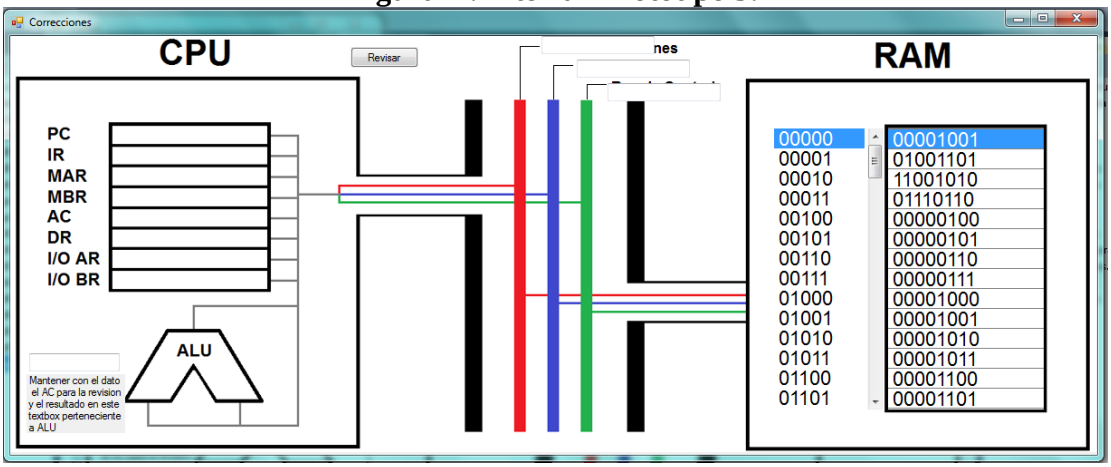
Fuente: Elaboración propia 2014

Figura10. Pseudocódigo Decimal a Binario.

```
procedimiento DecimalABinario(numero)
    binario <- ""
    cosciente <- num
    Mientras (cosciente > 1)
        binario <- Concatenar(ConvertirACadena(cosciente % 2), binario)
        cosciente = cosciente / 2;
    fin mientras
    binario = Concatenar(ConvertirACadena(cosciente), binario)
    cont <- 0
    Mientras (cont < tam)
        binario = Concatenar("0", bin)
        incrementar(cont)
    fin mientras
fin procedimiento, terminar devolviendo binario
```

Fuente: Elaboración propia 2014

Figura11. Interfaz Prototipo 3.



Fuente: Elaboración propia 2014

2.2.2. Prototipo 4

Requerimientos:

- Desarrollar componente de interacción con el estudiante, para desplazar el contenido de registros.
- Desarrollar componente interactivo de verificación de errores, en binario.

Al tener ya el componente de interacción con el usuario definido, se realizaron ajustes después de las siguientes observaciones:

- La RAM llena confunde a los estudiantes, deben generarse solo las celdas y mantener el contenido de los registros y direcciones vacios.
- Se debe controlar la longitud de los campos, pero no se debe restringir al estudiante a escribir mayor contenido en un registro, pues este es uno de los errores comunes en los estudiantes, el cual el mismo debe controlar.

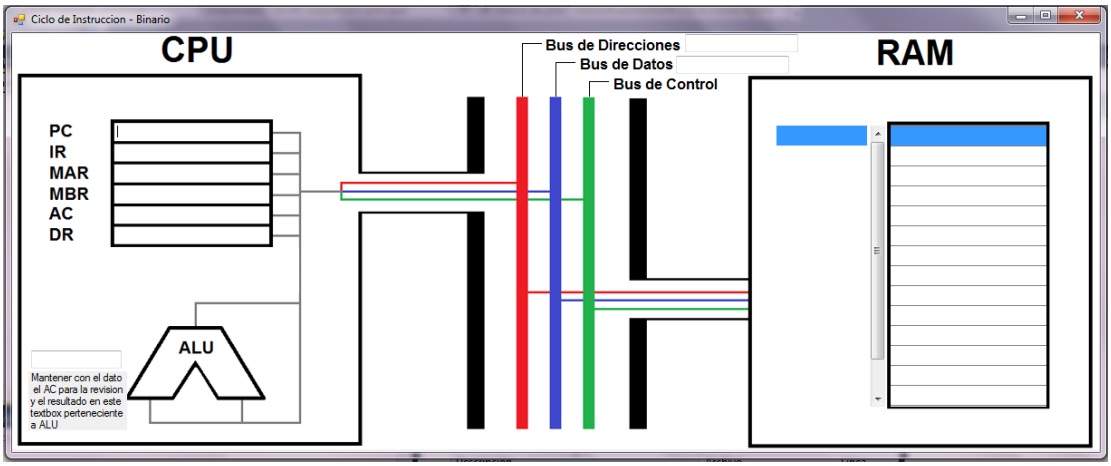
Además de esas modificaciones a ese componente, se procedió con el desarrollo del componente interactivo de verificación de errores, para el ciclo de captación e el de ejecución de la operación de lectura de un dato desde memoria. Se usaron los eventos:

- ñTextChangedò para el control de la longitud del registro, para que la longitud sea exactamente la misma a la definida como el tamaño de un registro de CPU (en este caso aun era predefinido como ocho).
- ñMouseDobleClickò para la copia del contenido de registros, definiendo una variable origen con el nombre del registro de origen, y una variable destino como el nombre del registro destino, para verificar si era correcta la secuencia elegida por el usuario.

La verificación en síntesis consiste en verificar si el registro el cual cambió de contenido es el que debía cambiar por la correlación del proceso que se está ejecutando. Si es que es el correcto, es decir, si es que el usuario lleva el contenido del registro origen al

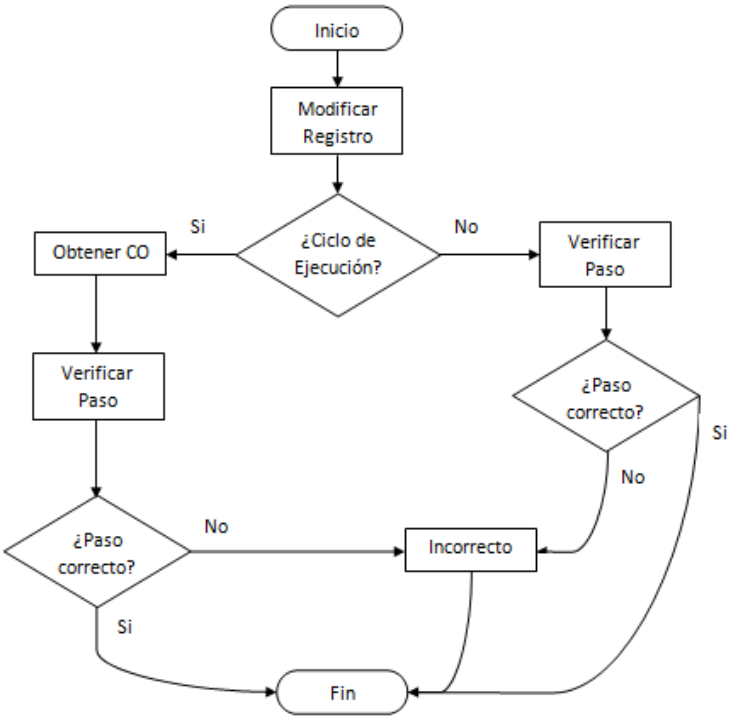
registro destino correspondiente para ese paso del proceso, el usuario puede seguir modificando, caso contrario se muestra un mensaje de ñIncorrectoò.

Figura12. Interfaz Prototipo 4.



Fuente: Elaboración propia 2014

Figura13. Diagrama de Flujo, Verificación de errores.



Fuente: Elaboración propia 2014

Figura14. Pseudocódigo Verificar Paso en Captación.

```
ejec <- falso

procedimiento captacion(paso)
    resp <- falso
    Segun (paso) hacer
        caso 1:
            Si (pc es origen) ^ (mar es destino)
                resp <- verdadero
                incrementar(paso)
            fin si
        caso 2:
            si (mar es origen) ^ (busDirs es destino)
                resp <- verdadero
                incrementar(paso)
            fin si
        caso 3:
            si (busDirs es origen) ^ (ram es destino)
                si (direccion seleccionada = busDirs(contenido))
                    resp <- verdadero
                    incrementar(paso)
                fin si
            fin si
        caso 4:
            si (ram es origen) ^ (busDatos es destino)
                si (direccion seleccionada = busDirs(contenido))
                    resp <- verdadero
                    incrementar(paso)
                fin si
            fin si
        caso 5:
            si (busDatos es origen) ^ (mbr ^ destino)
                resp <- verdadero
                incrementar(paso)
            fin si
        caso 6:
            si (mbr es origen) ^ (ir es destino)
                resp <- verdadero
                paso <- 1
                ejec <- verdadero
            fin si
    fin segun
fin procedimiento, terminar devolviendo resp
```

Fuente: Elaboración propia 2014

Figura15. Pseudocódigo Verificar Paso en ejecución con LOAD.

```
ejec <- falso

procedimiento captacion(paso)
    resp <- falso
    Segun (paso) hacer
        caso 1:
            Si (pc es origen) ^ (mar es destino)
                resp <- verdadero
                incrementar(paso)
            fin si
        caso 2:
            si (mar es origen) ^ (busDirs es destino)
                resp <- verdadero
                incrementar(paso)
            fin si
        caso 3:
            si (busDirs es origen) ^ (ram es destino)
                si (direccion seleccionada = busDirs(contenido))
                    resp <- verdadero
                    incrementar(paso)
                fin si
            fin si
        caso 4:
            si (ram es origen) ^ (busDatos es destino)
                si (direccion seleccionada = busDirs(contenido))
                    resp <- verdadero
                    incrementar(paso)
                fin si
            fin si
        caso 5:
            si (busDatos es origen) ^ (mbr ^ destino)
                resp <- verdadero
                incrementar(paso)
            fin si

        caso 6:
            si (mbr es origen) ^ (dr es destino)
                resp <- verdadero
                paso <- 1
                ejec <- falso
            fin si
    fin segun
fin procedimiento, terminar devolviendo resp
```

Fuente: Elaboración propia 2014

Figura16. Pseudocódigo Obtener CO.

```
co <- vacio

procedimiento obtenerCO()
    co <- vacio
    cont <- 0
    mientras (cont < tamanho de CO)
        co <- Concatenar(co, IR[cont])
        incrementar(cont)
    fin mientras
fin procedimiento
```

Fuente: Elaboración propia 2014

2.2.3. Prototipo 5

Requerimientos:

- Implementar componente para la definición de formato de instrucción y códigos de operación, en binario.
- Desarrollar componente interactivo de verificación de errores, en binario.

Para que un estudiante resuelva un ejercicio de ciclo de instrucción, antes debe estar correctamente planteado y estructurado. Por tanto cuando el docente plante el ejercicio a los estudiantes, estos abstraen el problema y obtienen los datos necesarios para plasmar el estado inicial del CPU y la RAM en los diagramas. De la misma manera, el estudiante debería plasmar esa abstracción en el simulador para plantear el estado inicial del CPU, definiendo el formato de instrucción y el valor de los códigos de operación a utilizarse en el programa.

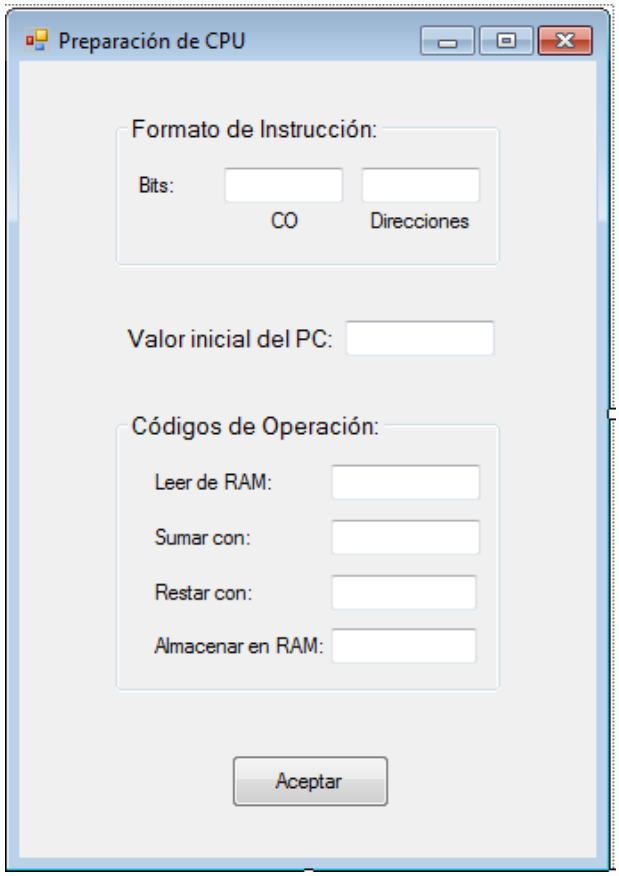
En este prototipo se permitía al usuario elegir una cantidad de bits para el campo de CO y de direcciones, siendo el límite máximo 32 bits en total, por lo que si a uno de los campos se le asignaba un valor igual o mayor a 30 bits, al otro se le asignaba dos automáticamente y se ponía el valor de este campo en 30.

De la misma manera el estudiante debía ingresar el valor inicial en binario del registro PC y de cada CO, pero si el tamaño era menor se rellenaban los bits restantes con ceros.

Al realizar los controles principales por el estudiante de manera automática, lo único que se debía controlar para avanzar era si todos los campos necesarios estaban llenos, caso contrario se mostraba un mensaje para avisar que campo debía ser llenado.

También se modificó el valor esperado de los registros en el componente principal, en el que el estudiante resuelve el ejercicio como tal, pasando los valores definidos en la interfaz de preparación del CPU como parámetros al constructor del componente principal. De esta forma cuando el usuario resolvía ejercicios se cargaba automáticamente el valor inicial de la PC y se modificaron todos los algoritmos relacionados con la longitud del campo de CO y direcciones del CPU, así como del tamaño total de los registros.

Figura 17. Interfaz prototipo 5, preparación de CPU.



Fuente: Elaboración propia 2014

2.2.4. Prototipo 6

Requerimientos:

- Implementar componente para la definición de formato de instrucción y códigos de operación, en binario.
- Implementar componente para preparación de la RAM para el ejercicio por parte del estudiante (llenado de registros iniciales), en binario.
- Desarrollar componente interactivo de verificación de errores, en binario.

Después de hacer un análisis del tercer prototipo, se llego a la conclusión que los controles de ahí en adelante para el estudiante solo se limitarían a mostrarle donde está su error cuando quisiera seguir pero no a corregir ninguno de ellos por él.

Por lo tanto, se reprogramó todo el componente siguiendo esa lógica, bloqueando los campos inferiores hasta que los superiores cumplan con los requerimientos inferiores, mostrando a medida que avanza el usuario mensajes para que pueda saber que error está cometiendo. El control se hizo mediante el evento ñTextChangedò, siendo que para verificar cada ñTextBoxò al ser modificado se aplica un algoritmo el cual verifica que el valor sea correcto y si se cumplen las condiciones de desbloqueo del siguiente bloque de datos que debe llenar el usuario, bloques que se dividen en:

- Paso1: El usuario debe llenar correctamente el campo de direcciones y de CO.
- Paso2: El estudiante debe dar un valor inicial de PC válido.
- Paso3: El estudiante le asigna un valor a cada uno de los COs que necesita.

Al finalizar el llenado de datos y hacer click en el botón ñAceptarò, el componente verifica que todos los datos estén en orden, en caso que no sea así aparece un mensaje con el error.

Si es que el usuario deja en blanco los códigos de operación para sumar y restar se muestra un mensaje notificando sobre esto y preguntando si desea continuar. Como un ejercicio como mínimo debe tener una lectura de un número desde una dirección de

memoria y su escritura en otra dirección, se le permite continuar en caso de que así lo desee.

Una modificación importante de este prototipo es que la longitud total máxima de un registro del CPU, la cual solo llega a 16 bits, esto se debe a que un tamaño mayor sería innecesario para fines didácticos.

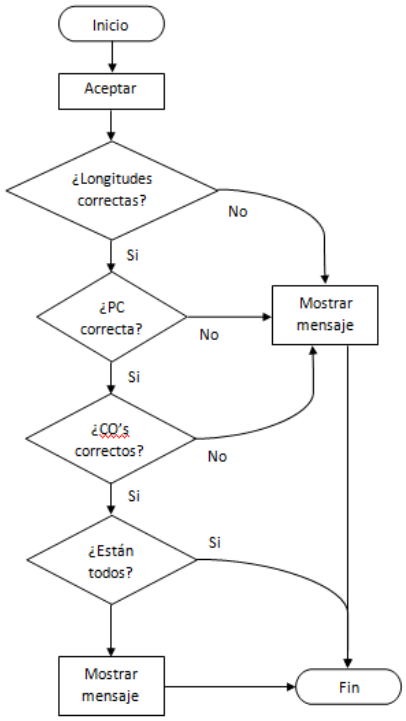
Siguiendo también fines didácticos, se define que la longitud del campo de CO y de direcciones sea un múltiplo de 4, para poder hacer una comparación de los ejercicios a realizarse en clases en binario con los que se realizaran en hexadecimal.

También se implemento una nueva interfaz de usuario en la cual el usuario define el estado inicial de los registros de la RAM, en la cual se verifica solamente la longitud de los campos y que solamente ingresen los caracteres ñ0ò y ñ1ò.

Al tener previamente el estado inicial de la RAM definido por el usuario, la interfaz del componente principal no solo aparece automáticamente con el contenido del registro PC en el valor inicial definido por el usuario, sino también la memoria RAM aparece con los datos definidos por el mismo.

Figura 18.

Diagrama de flujo verificación final, interfaz de Preparación de CPU.



Fuente: Elaboración propia 2014

Figura 19. Pseudocódigo verificación de campo CO.

```
procedimiento verificarCO(contenido)
    resp <- true
    Si (!vacio(contenido))
        Si (contenido % 4 != 0) ^ (contenido != 0)
            mensaje <- "El campo de CO debe ser multiplo de 4"
            resp <- false
        fin si
        c/c
            Si (contenido > 8)
                mensaje <- "El campo de CO no puede ser mayor a 8 bits";
                resp <- false
            c/c
                mensaje <- vacio;
                resp <- false
            fin si
        fin si
    c/c
        mensaje <- "El campo de CO no puede estar vacío";
        resp <- false
    fin si
fin procedimiento, devolver mensaje, resp
```

Fuente: Elaboración propia 2014

Figura 20. Pseudocódigo verificación de campo de Dir.

```
procedimiento verificarDir(contenido)
    resp <- true
    Si (!vacio(contenido))
        Si (contenido % 4 != 0) ^ (contenido != 0)
            mensaje <- "El campo direcciones debe ser multiplo de 4"
            resp <- false
        fin si
        c/c
            Si (contenido > 12)
                mensaje <- "El campo direcciones no puede ser mayor a 12 bits";
                resp <- false
            c/c
                mensaje <- vacio;
                resp <- false
            fin si
        fin si
    c/c
        mensaje <- "El campo direcciones no puede estar vacío";
        resp <- false
    fin si
fin procedimiento, devolver mensaje, resp
```

Fuente: Elaboración propia 2014

Figura 21. Pseudocódigo verificación de PC inicial.

```
procedimiento verificarPC(contenido)
    Si (!vacio(contenido))
        Si (tamanho(contenido) != (dir + co))
            mensaje <- concatenar("La dimensión del PC debe ser igual a ", (dir + co))
            resp <- false;
        c/c
            si (!verificarAccesoAMemoria(contenido))
                mensaje <- "El valor del PC debe estar dentro del rango";
                resp <- false;
            c/c
                mensaje <- vacio;
                resp <- true;
            fin si
        fin si
    c/c
        mensaje <- "El PC no debe estar vacío";
        resp <- false;
    fin si
fin procedimiento, devolver mensaje, resp
```

Fuente: Elaboración propia 2014

Figura 23. Pseudocódigo verificación de acceso a memoria.

```
procedimiento verificarAccesoAMemoria(contenido)
    resp <- false
    cont <- 0
    mientras (cont < co)
        si (contenido[cont] != '0')
            resp <- false
            cont <- co
        fin si
        incrementar(cont)
    fin mientras
fin procedimiento, devolver resp
```

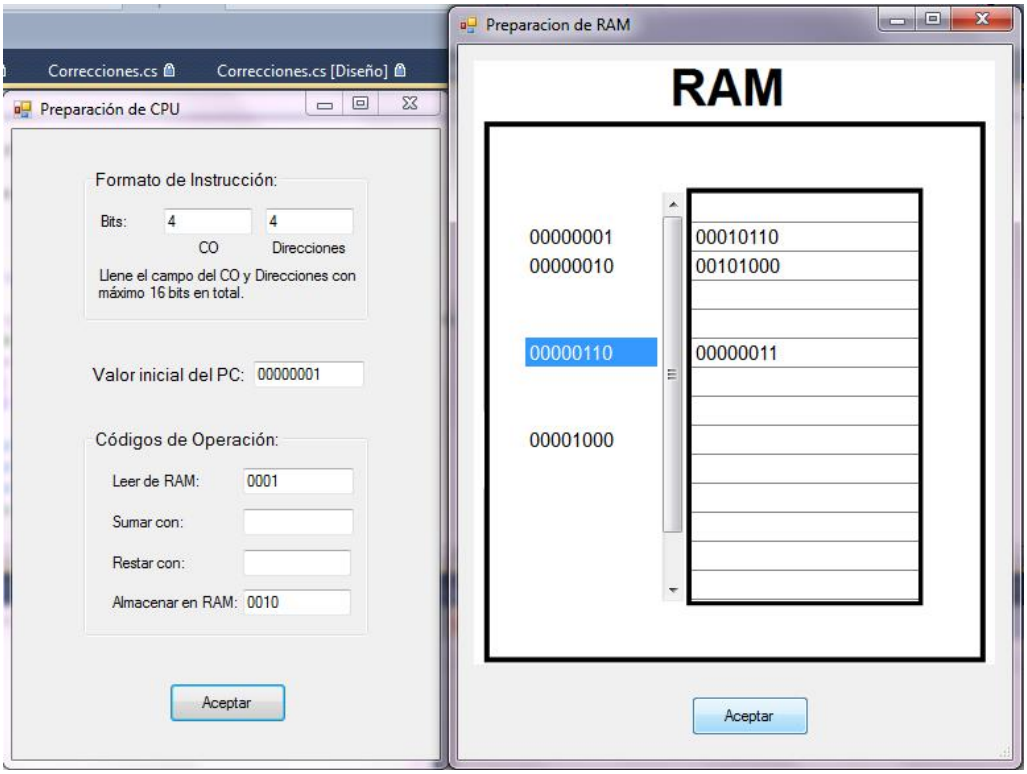
Fuente: Elaboración propia 2014

Figura 23. Pseudocódigo para llenar tabla enviada en interfaz principal.

```
procedimiento CrearTabla(listaDatos)
    tabla <- Nuevatablala()
    col <- NuevaColumna()
    AgragarColumna(tabla) <- col
    cont <- 0
    Mientras (cont < tamanho(listaDatos))
        fila <- CrearFila(tabla)
        fila[col] <- listaDatos[cont]
        AgregarFila(tabla(filas) <- fila
        incrementar(cont)
    fin mientras
fin procedimiento, terminar devolviendo tabla
```

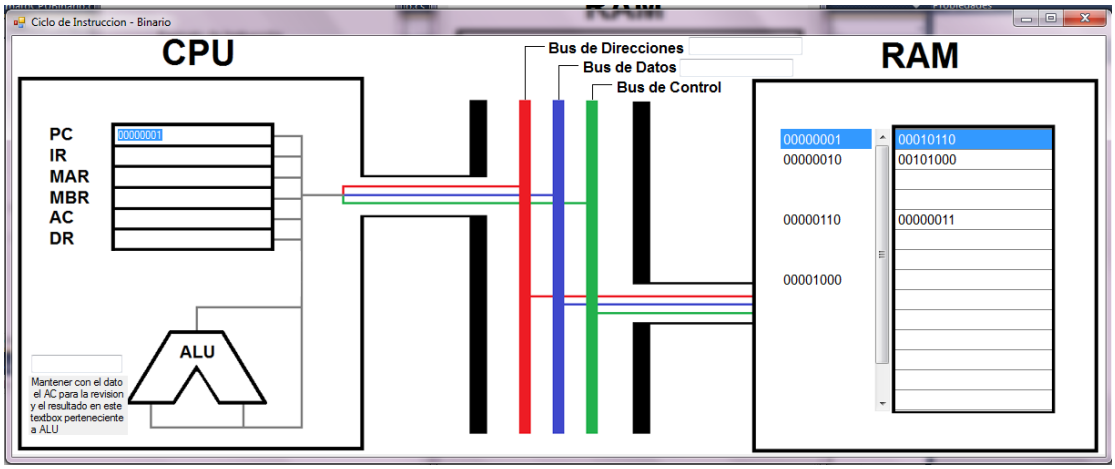
Fuente: Elaboración propia 2014

Figura 24. Interfaces de preparación de CPU y RAM.



Fuente: Elaboración propia 2014

Figura 25. Interfaz principal con parámetros enviados.



Fuente: Elaboración propia 2014

2.2.5. Prototipo 7

Requerimientos:

- Desarrollar componente de interacción con el estudiante, para desplazar el contenido de registros.
- Desarrollar componente interactivo de verificación de errores, en binario.
- Refactorización de código.

En este prototipo se busco realizar una afinación del código ya funcional conseguido en el desarrollo de los 4 prototipos iniciales.

En el componente principal verifica si el estudiante realiza el incremento del PC correctamente.

Primeramente se procedió a crear clases necesarias para generar un código más mantenible y menos acoplado.

Creando objetos tales como la clase Binario, la cual es una clase estática la cual contiene todos los métodos necesarios para manipular datos en binario, o algunos otros controladores, los cuales principalmente se encargan de la lógica de control que puede ser separada de los componentes de interfaz de usuario.

Figura 26. Pseudocódigo para conversión de Binario a Decimal

```
procedimiento BinarioADecimal(binario)
    decimal <- 0
    cont <- 0
    mientras (cont < tamanho(binario))
        si (binario[i] = '1')
            decimal <- decimal + pontencia(2, tamanho(binario) - 1 - cont))
        fin si
    fin mientras
fin procedimiento, devolver decimal
```

Fuente: Elaboración propia 2014

Figura 27. Pseudocódigo para sumar dos números binarios

```
procedimiento sumar(num1, num2, tam)
    suma <- BinarioADecimal(num1) + BinarioADecimal(num2)
    resp <- DecimalABinario(suma, tam)
fin procedimiento, devolver resp
```

Fuente: Elaboración propia 2014

Figura 28. Pseudocódigo para verificar suma

```
procedimiento SumaCorrecta(ac, dr, alu)
    resp <- false
    si (sumar(ac, dr, tamanho(ac)), alu))
        resp <- true
    fin si
fin procedimiento, devolver resp
```

Fuente: Elaboración propia 2014

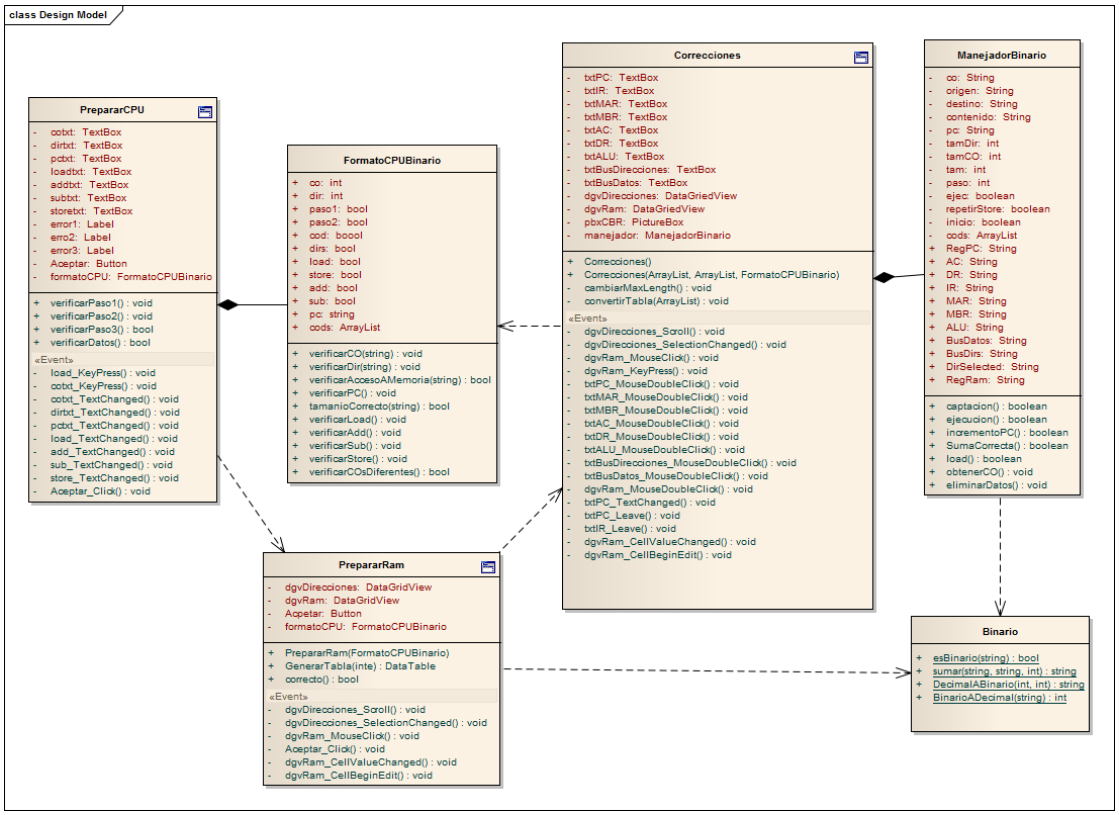
Figura 29. Pseudocódigo para incremento de PC

```
pc <- pc_inicial

procedimiento incrementoPC(contenido, tam)
    resp <- false
    si (sumar(pc, DecimalABinario(1, tam), tam) = contenido)
        pc <- contenido
        resp <- true
    fin si
fin procedimiento, devolver resp
```

Fuente: Elaboración propia 2014

Figura 30. Diagrama de clases.



Fuente: Elaboración propia 2014

2.2.6. Prototipo 8

Requerimientos:

- Desarrollar componente de interacción con el estudiante, para desplazar el contenido de registros.
- Desarrollar componente interactivo de verificación de errores, en binario.

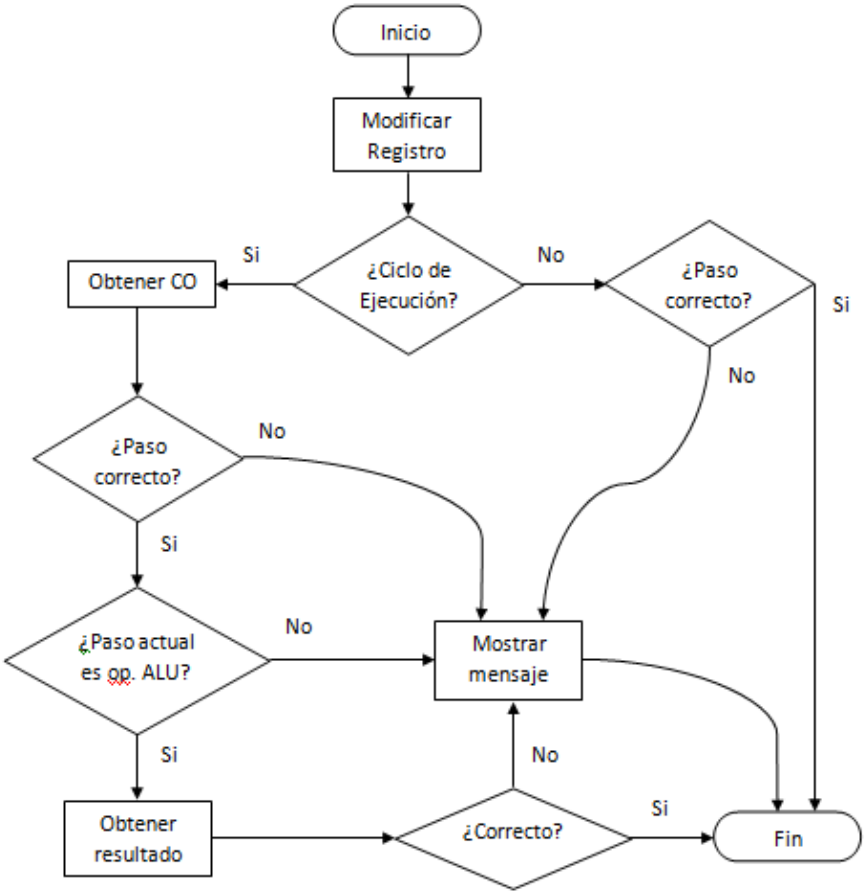
En el componente principal se implementó la verificación del ciclo de ejecución del CO para almacenar el contenido de AC en la RAM.

También se implementó la verificación del ciclo de ejecución para las dos operaciones de ALU, es decir, suma y resta de dos números. Para esta parte se modificó el

componente de interacción con el estudiante para manejar el contenido del registro de ALU, el cual no fue implementado anteriormente.

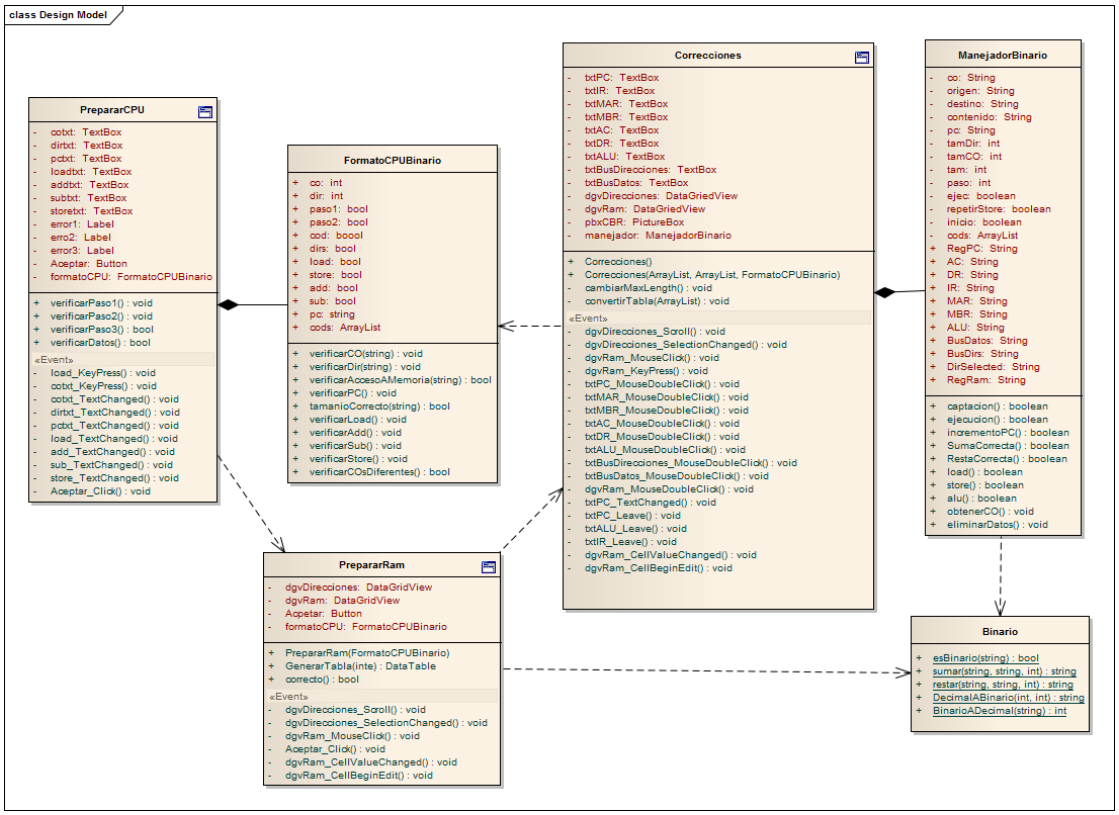
Para verificar si las operaciones de ALU son o no correctas, se crearon varias funciones, las cuales verifican primeramente el valor de la suma o resta de los contenidos de los registros AC y DR, y la comparan con el valor ingresado por el usuario.

Figura 31. Diagrama de flujo de verificación de pasos.



Fuente: Elaboración propia 2014

Figura 32. Diagrama de clases.



Fuente: Elaboración propia 2014

2.2.7. Prototipo 9

Requerimientos:

- Implementar componente para preparación de la RAM para el ejercicio por parte del estudiante (llenado de registros iniciales), en binario.

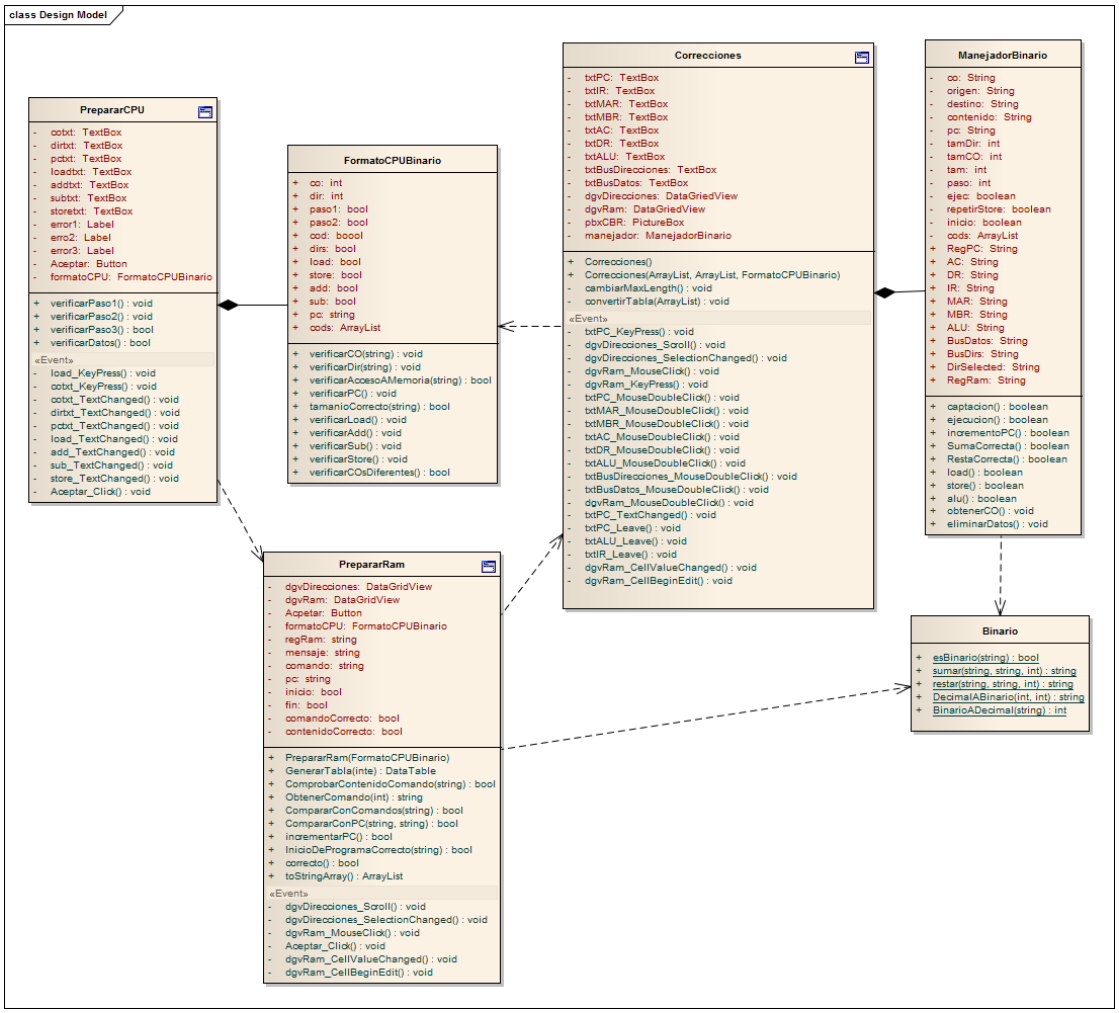
En este prototipo se mejora la funcionalidad del componente de preparación de la RAM, buscando dejar un contenido de RAM más consistente. Se realizan principalmente las siguientes acciones de verificación:

- La dirección a la cual apunta el valor inicial del PC debe existir.
- Debe existir al menos una instrucción además de la inicial.
- Se verifica que las instrucciones sean consecutivas.

- El comando de lectura desde RAM debe ser la instrucción de la dirección a la que apunta el valor inicial del PC.
- Comprueba las instrucciones consecutivas en RAM hasta encontrar un registro que no sea una instrucción, es decir, que no contenga un código de operación.
- La dirección a la que apunta el campo de direcciones de cada instrucción debe tener contenido, a menos que la instrucción sea de escritura a la RAM.

Todas las verificaciones anteriormente mencionadas se llevan a cabo al momento que el usuario hace click en el botón ñAceptarò.

Figura 33. Diagrama de clases.



Fuente: Elaboración propia 2014

2.2.8. Prototipo 10

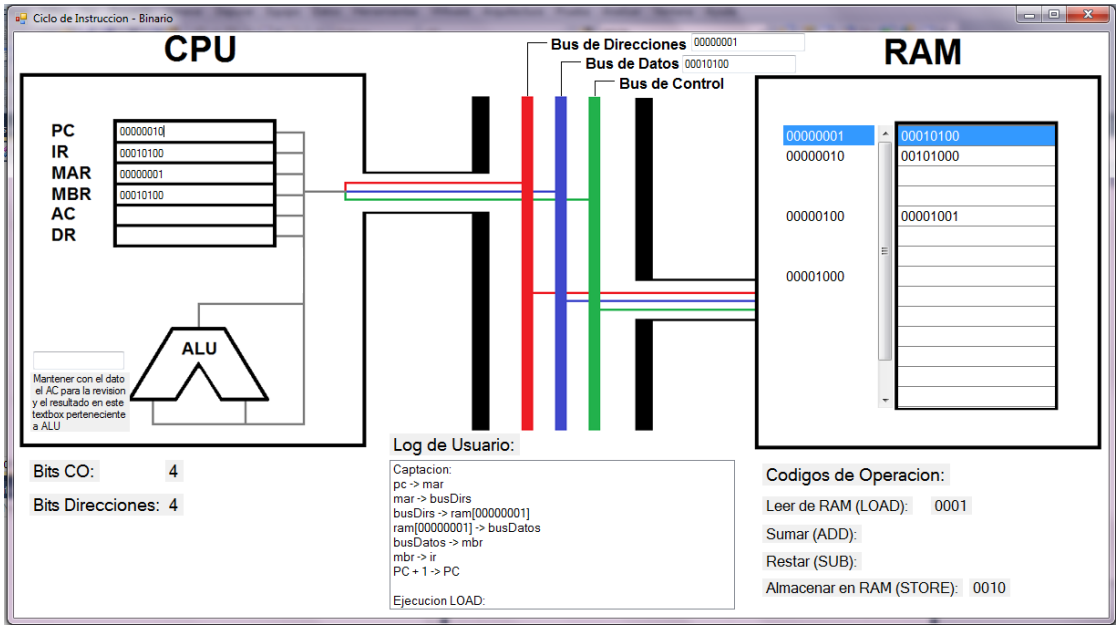
Requerimientos:

- Rediseño de interfaz de usuario.

Se procedió a incrementar un *log* de operaciones de usuario, para que el estudiante pueda visualizar los pasos correctos que realizó durante la resolución del ejercicio.

También se agregan a la interfaz de usuario los datos llenados por el mismo en la interfaz de preparación del CPU.

Figura 34. Interfaz principal con *log* de operaciones.



Fuente: Elaboración propia 2015

2.2.9. Prototipo 11

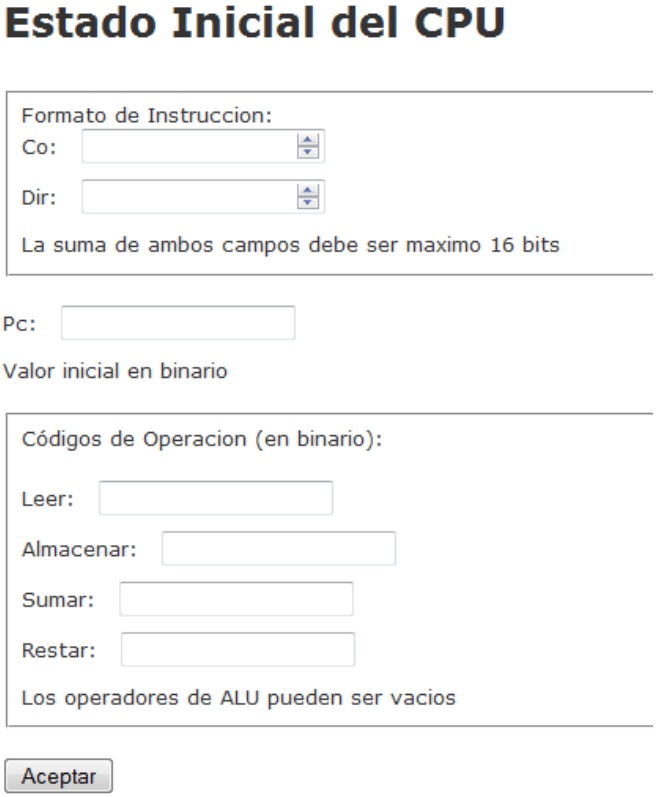
Requerimientos:

- Implementar componente para la definición de formato de instrucción y códigos de operación, en binario.
- Implementar componente para preparación de la RAM, en binario.

- Herramienta web.

Al definirse que la herramienta debía estar disponible para cualquier persona independientemente de su locación geográfica o preferencia de sistema operativo, se definió que era necesario que la herramienta sea migrada a un entorno web para que así sea accesible para cualquier persona que desee utilizarla, teniendo como única restricción el acceso a internet.

Figura 35. Interfaz de preparación del CPU en binario.

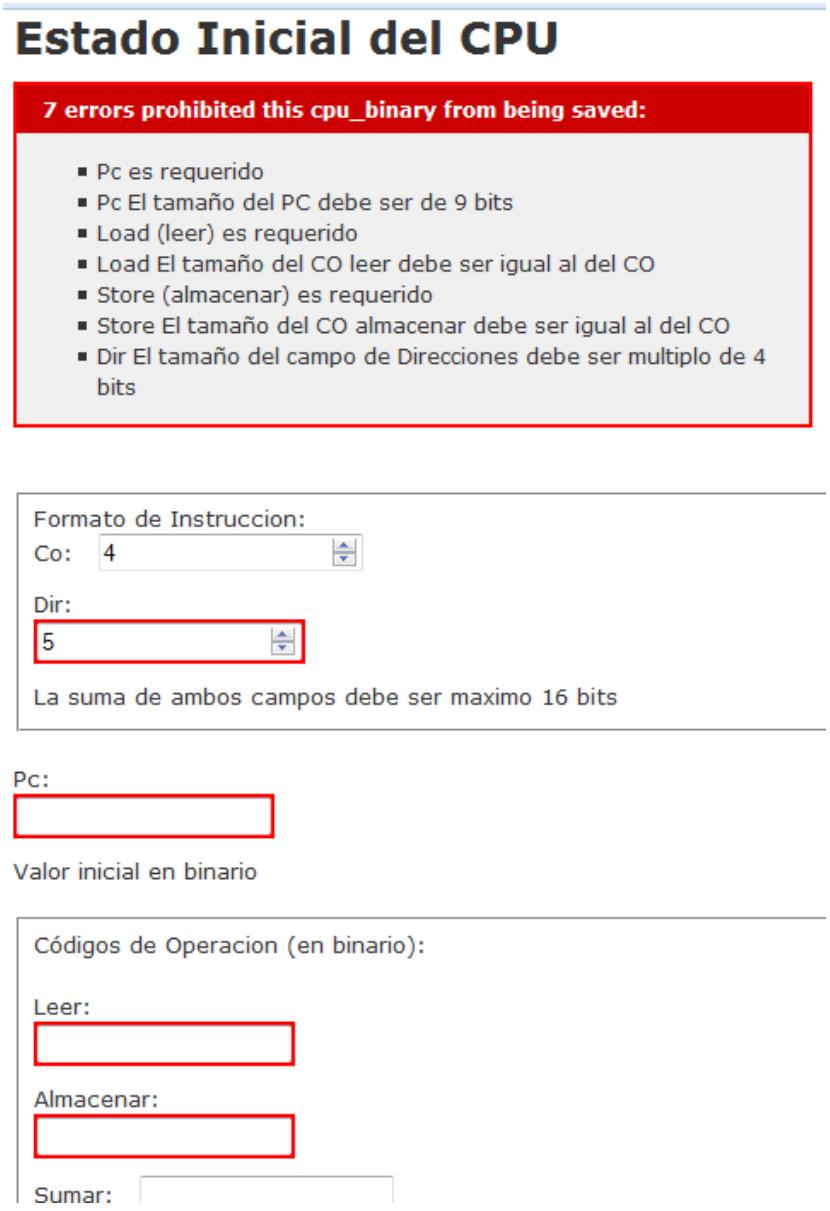


Fuente: Elaboración propia 2015

Para realizar esta validación de datos de entrada ingresados por parte del usuario se utilizaron los métodos *valídate*, propios de los modelos en *Ruby on Rails*. En estas validaciones se verifica que el contenido de los campos son los correctos en cuanto a longitud y contenido, es decir, los datos q deben ser binarios sean binarios y los que sean decimales sean decimales, así como también que la longitud del registro *PC* y de los

códigos de operación coincidan con los datos ingresados en el formato de instrucción definido por el usuario.

Figura 36. Control en preparación del CPU en binario, con errores.

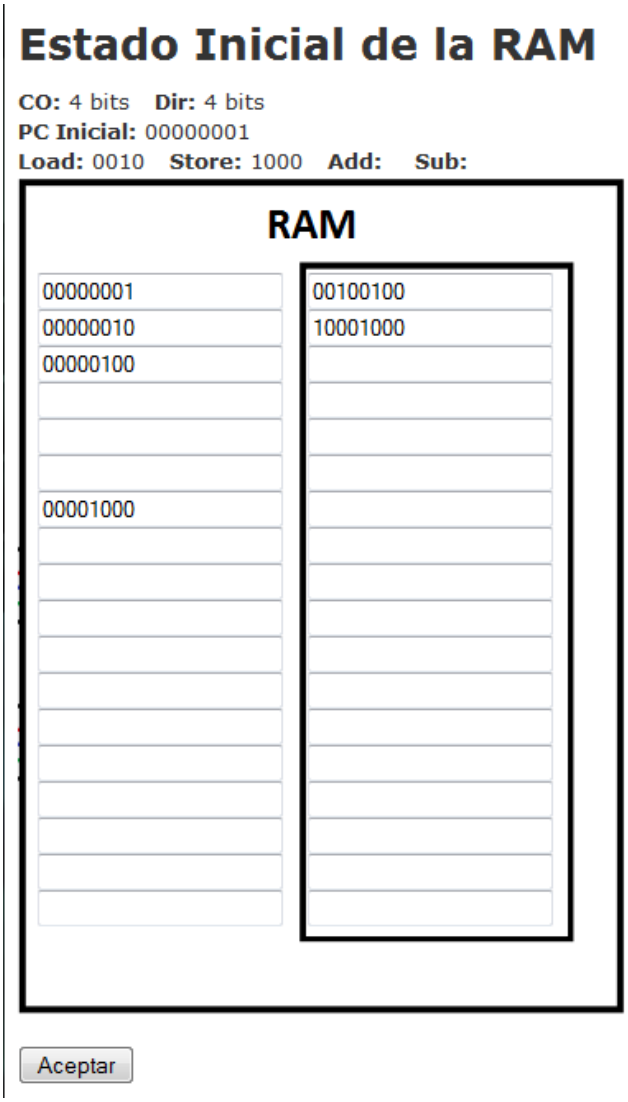


Fuente: Elaboración propia 2015

En cuanto a la preparación del estado inicial de la RAM y su verificación, la interfaz de usuario fue llenada creando una tabla llene de *text*_fields.

Figura 37. Interfaz de preparación de la RAM en binario.

Prototipo 11.



Fuente: Elaboración propia 2015

Al cambiar de *.NET* a *Ruby* se cambia también el modelo de objetos y se agrega una base de datos, puesto que la información de cada objeto solo puede ser recuperada si fue previamente guardada en la base de datos al cambiar de vista.

Aún no se utilizan JavaScripts en este prototipo para el control del ingreso de datos por parte del usuario desde el lado del cliente.

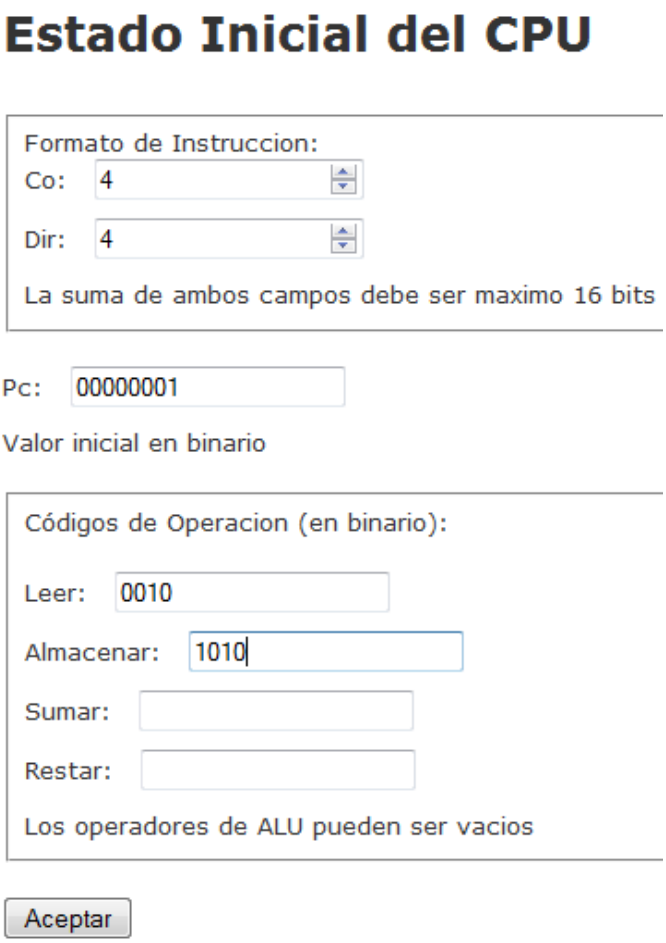
2.2.10. Prototipo 12

Requerimientos:

- Implementar componente para preparación de la RAM, en binario.
- Desarrollar componente interactivo de verificación de errores, en binario.
- Herramienta web.

Posteriormente se aplicó *JavaScripts* para la validación de datos en la interfaz de preparación del CPU desde el lado del cliente.

Figura 38. Interfaz de preparación del CPU en binario con regex-mask. Prototipo 12.



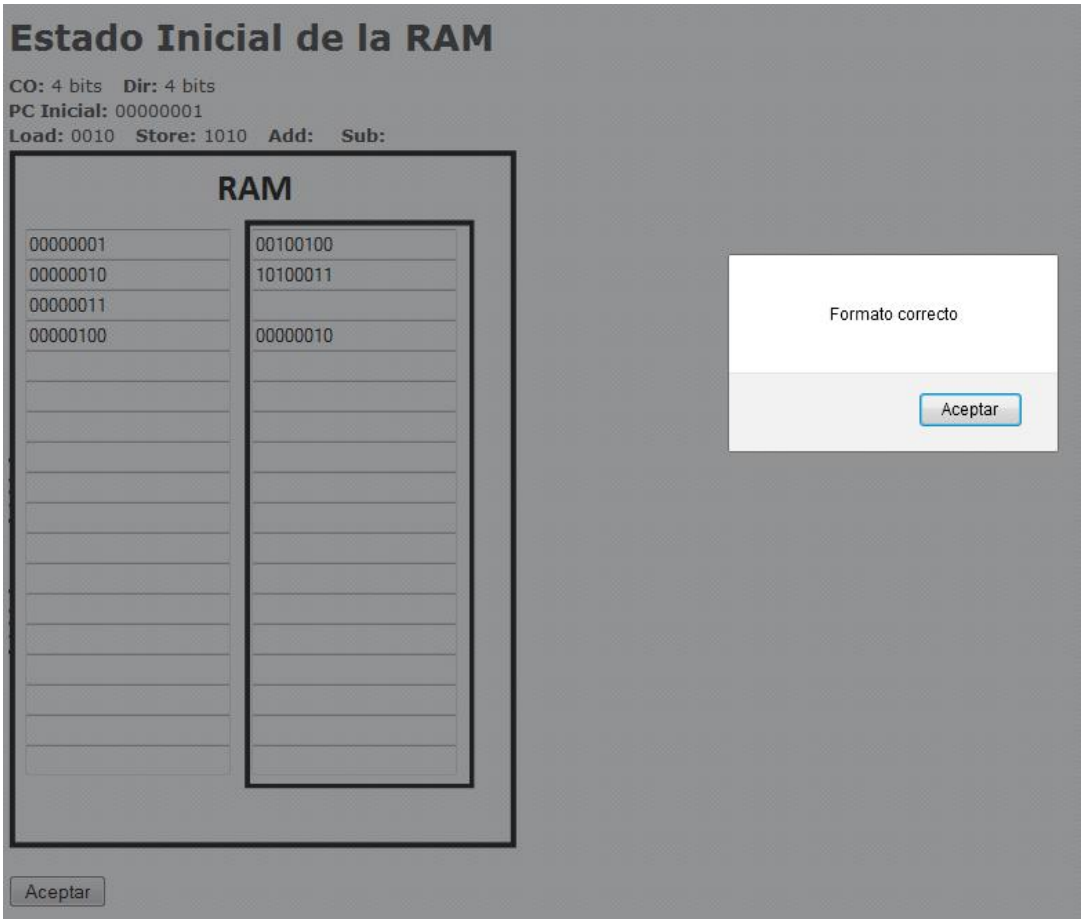
Fuente: Elaboración propia 2015

Se utilizo para la validación anteriormente nombrada un *plugin* de *JQuery* denominado *regex-mask-plugin* el cual se encarga de restringir el ingreso de valores a un *input text* mediante al función *keypress*, permitiendo solamente el ingreso de los caracteres que estén dentro de una expresión regular.

En cuanto a la preparación del estado inicial de la RAM y su verificación, también se utilizaron JavaScripts para la implementación de la verificación de errores en la preparación de un programa por parte del estudiante.

Figura 39. Interfaz de preparación de la RAM en binario.

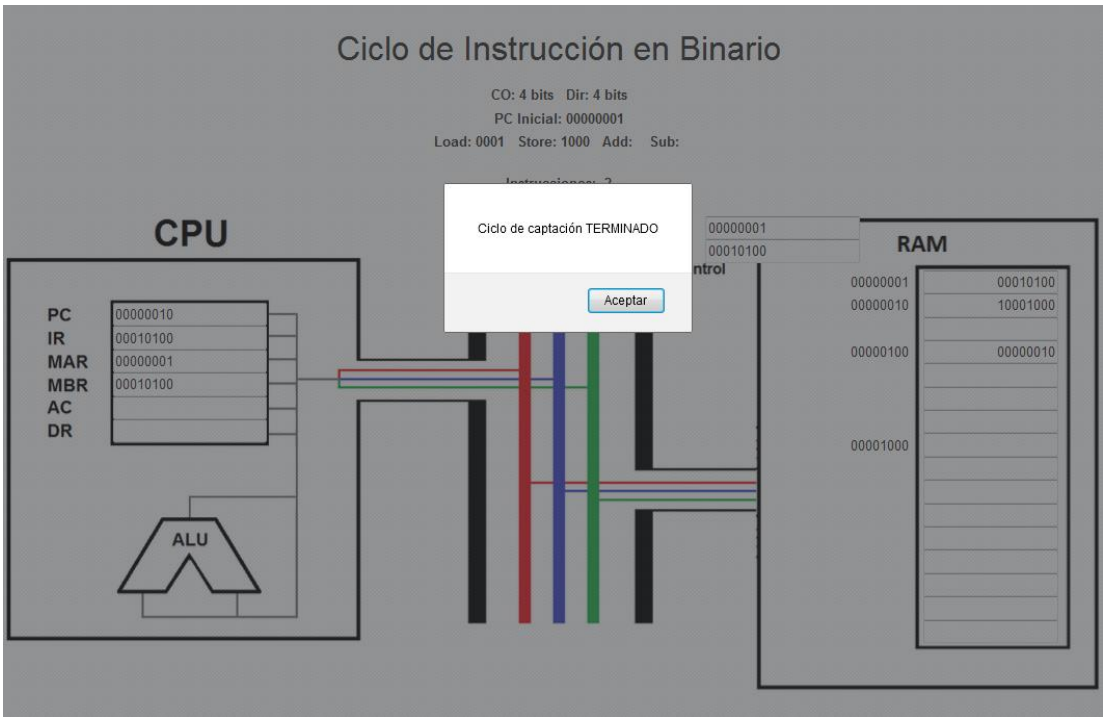
Prototipo 12.



Fuente: Elaboración propia 2015

También se implementó en este prototipo la interfaz principal de usuario para la resolución de ejercicios de ciclo de instrucción en binario, mostrando en la parte superior la información necesaria de lo establecido en las anteriores interfaces por parte del estudiante, para la resolución del ejercicio por parte del mismo.

Figura 40. Interfaz de principal para resolución de ejercicios. En binario.



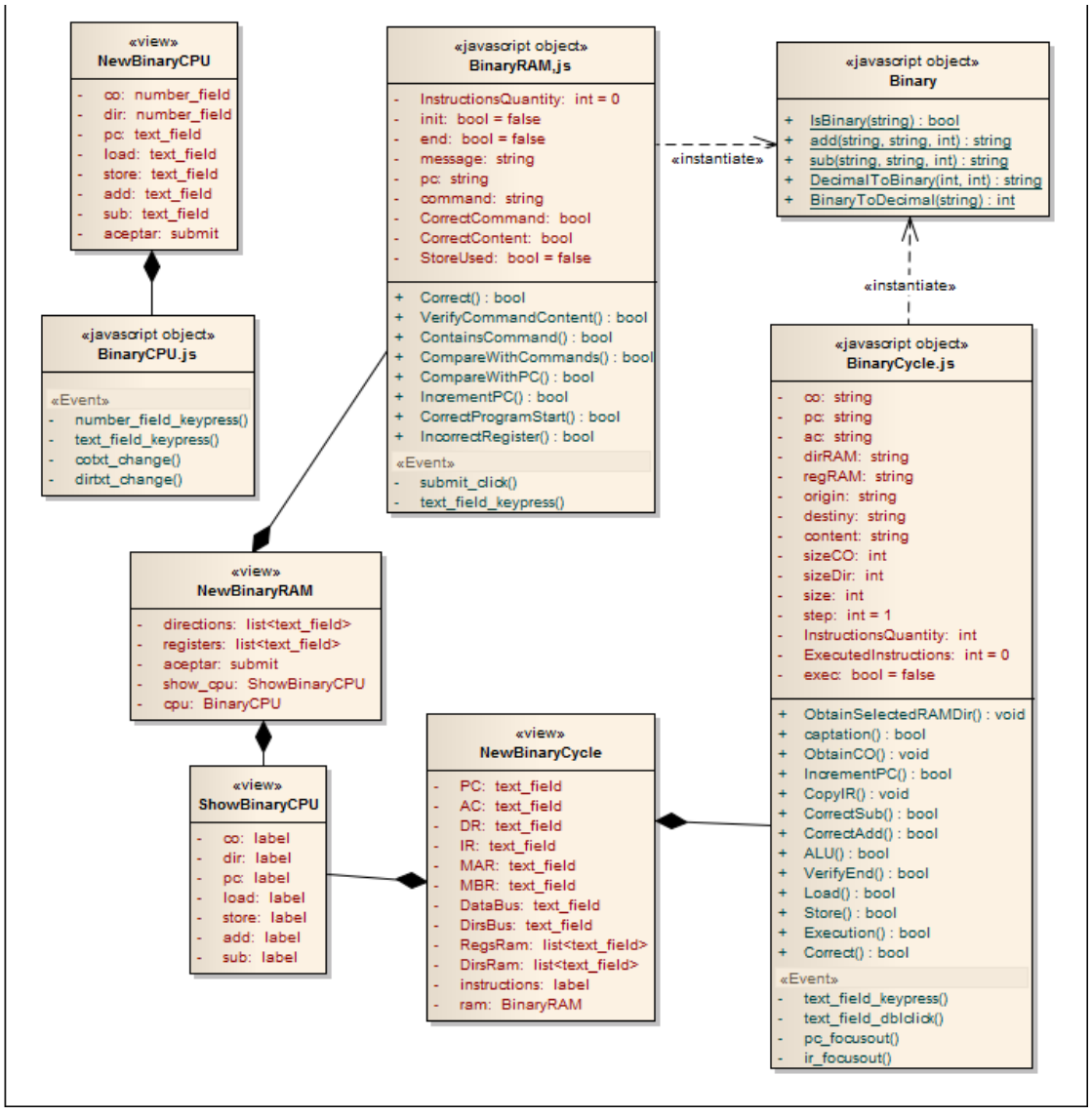
Fuente: Elaboración propia 2015

En esta interfaz se utilizan *JavaScripts* para el control de errores paso a paso en la resolución del ejercicio por parte del estudiante. Para la copia de contenido de registros se optó por mantener el uso del evento ñdoble clickò, haciendo esta operación primeramente en el registro origen y posteriormente en el registro destino.

La verificación del incremento del registro *PC* y la suma de registros en una operación de *ALU* es realizada con el evento *focusout* de *JQuery*, con esto, la verificación solamente es realizada cuando el estudiante intenta salir del registro *PC* y *AC* respectivamente.

Al cambiar de *C#* a *Ruby* se cambia también el modelo de objetos y se agrega una base de datos, puesto que la información de cada objeto solo puede ser recuperada si fue previamente guardada en la base de datos al cambiar de vista. De la misma forma al cambiar de un sistema de escritorio a una herramienta web nuevos objetos se utilizan, los controladores y los JavaScripts, enlazados a las vistas.

Figura 41. Aplicación de JavaScripts, prototipo12, binario.



Fuente: Elaboración propia 2015

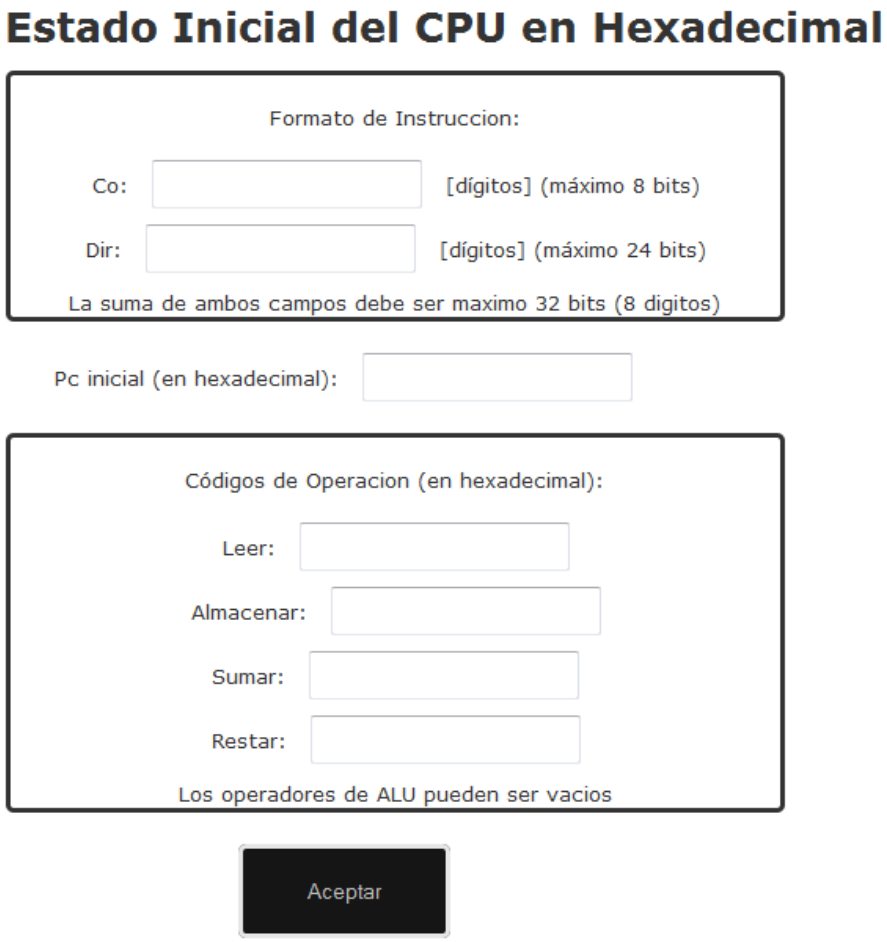
2.3. DESARROLLO DEL MÓDULO INTERACTIVO PARA TRABAJAR CON LENGUAJE HEXADECIMAL.

2.3.1. Prototipo 12

Requerimientos:

- Implementar componente para la definición de formato de instrucción y códigos de operación, en hexadecimal.

Figura 42. Interfaz de preparación de la RAM en hexadecimal. Prototipo 12.



Fuente: Elaboración propia 2015

En la interfaz destinada a la preparación del CPU en hexadecimal se empieza a implementar el uso básico de CSS para mejorar la *experiencia de usuario*. Para la validación de datos se utiliza la misma lógica que en la preparación del CPU en binario, mediante el uso de *validates*.

También se incrementa a esa lógica el uso del evento de JQuery *keyup*. Esta función es utilizada para que cuando el estudiante ingrese los números en hexadecimal, los caracteres de la ñaò a la ñfò, sin importar si las ingresa en minúsculas, sean mostradas como mayúsculas.

2.3.2. Prototipo 13

Requerimientos:

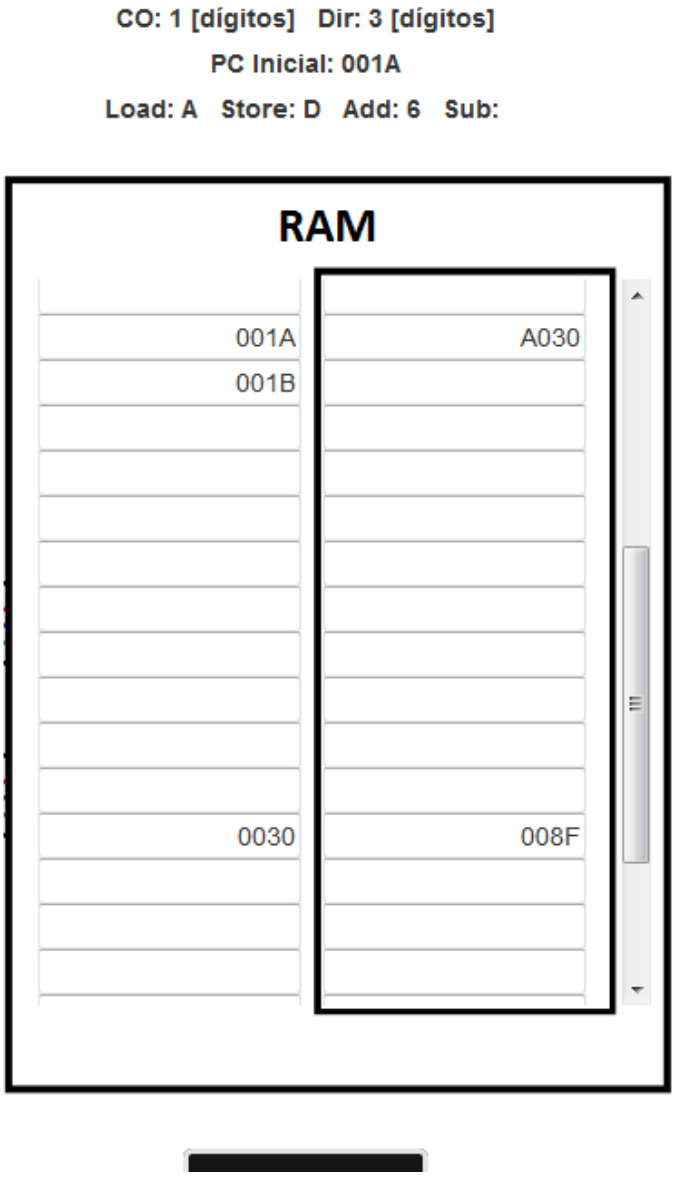
- Implementar componente para preparación de la RAM, en hexadecimal.
- Desarrollar componente interactivo de verificación de errores, en hexadecimal.
- Selección de lenguaje.

Para este prototipo ya se implementa el uso de CSS para mejorar la *experiencia de usuario*, proporcionándole al estudiante una interfaz más amigable, instalando una página principal y una barra de navegación en la parte superior.

En este prototipo se implementan las interfaces de preparación de RAM y la interfaz principal para la resolución de ejercicios de ciclo de instrucción en hexadecimal.

En este módulo, a diferencia del módulo en binario, en la preparación de la RAM por parte del estudiante se verifica si es que el último comando es un *STORE* y no solamente si es que éste fue utilizado en algún momento.

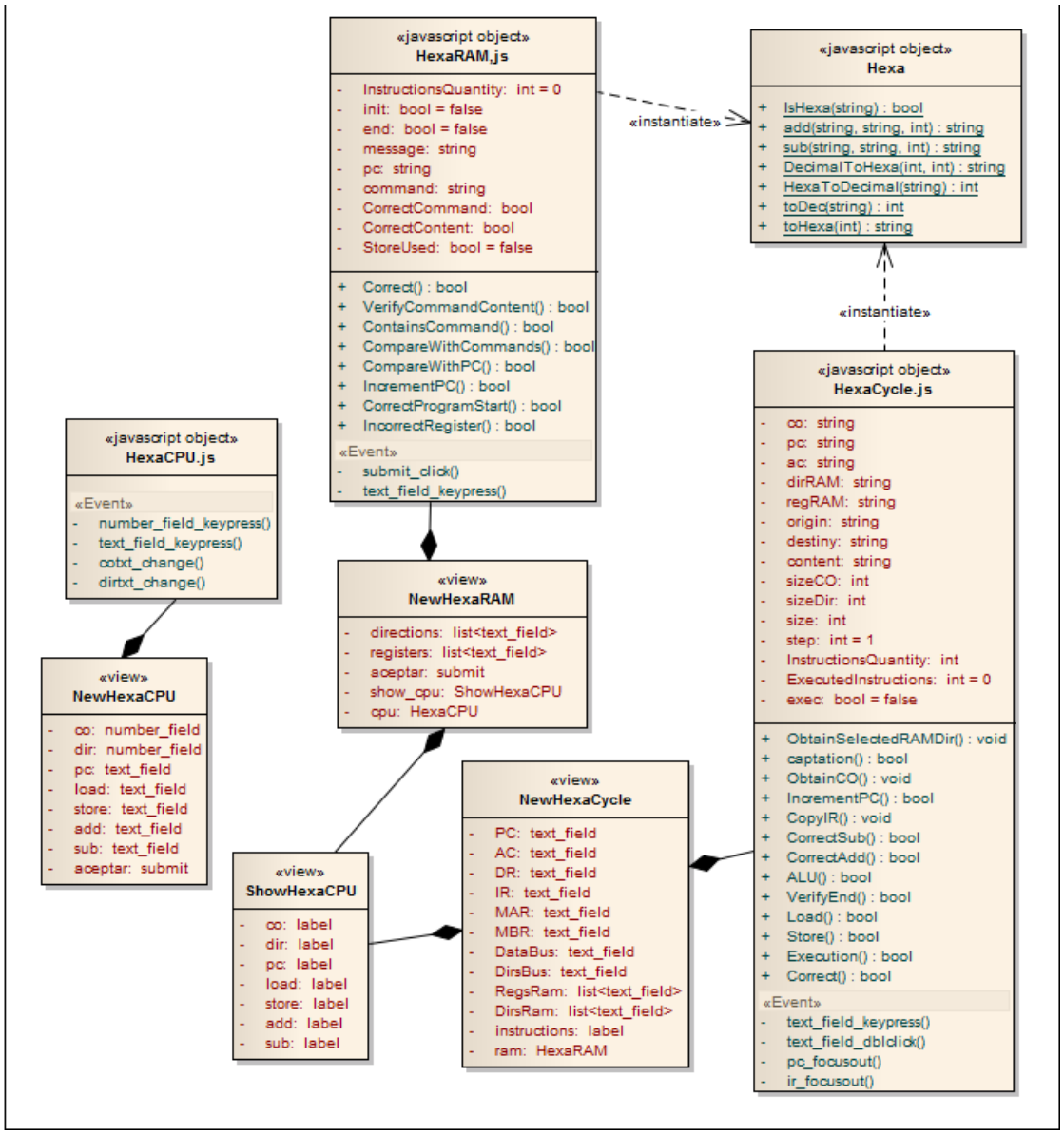
Figura 43. Preparación de RAM, en hexadecimal.



Fuente: Elaboración propia 2015

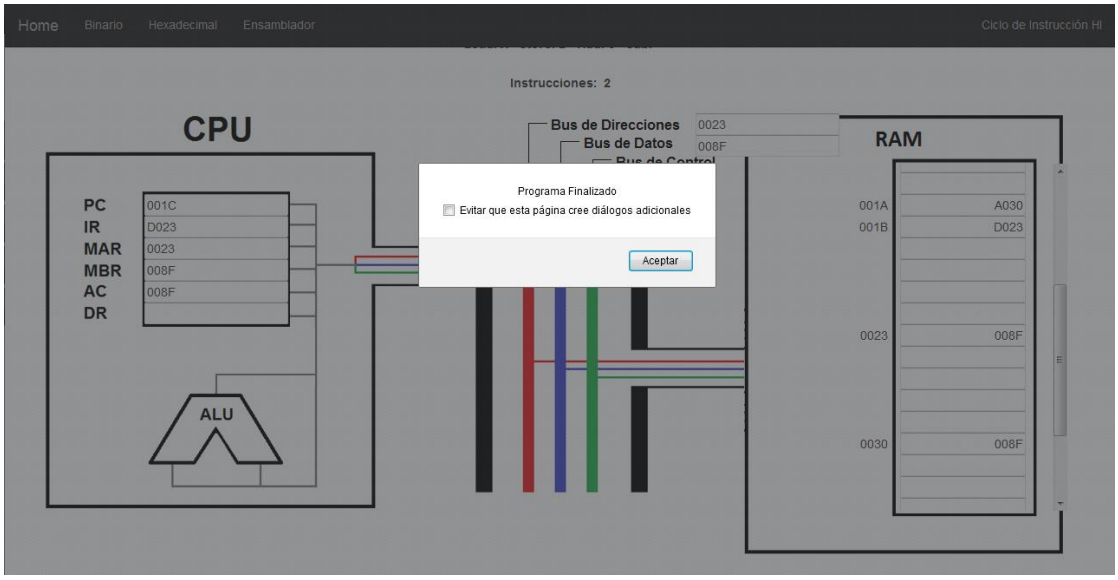
En la interfaz principal para la resolución de ejercicios en hexadecimal, la estructura utilizada en el módulo en binario es mantenida, también mostrando los datos principales ingresados por el estudiante en anteriores interfaces.

Figura 44. Aplicación de JavaScripts, prototipo 13, hexadecimal.



Fuente: Elaboración propia 2015

Figura 45. Preparación de RAM, en hexadecimal.



Fuente: Elaboración propia 2015

2.4. DESARROLLO DEL MÓDULO INTERACTIVO PARA TRABAJAR CON LENGUAJE ENSAMBLADOR.

2.4.1. Prototipo 13

Requerimientos:

- Implementar componente para la definición de formato de instrucción y códigos de operación, en ensamblador.

También se implementa la interfaz de preparación del CPU en ensamblador, realizando el control para ocultar el campo de selección de la cantidad de direcciones en caso de que la arquitectura elegida sea la basada en registros de uso general.

Esto se da debido a que la arquitectura basada en registros de uso general no puede tener instrucciones de una sola dirección, ya que una dirección debe ser de memoria, y la otra debe ser un registro del CPU.

Figura 46. Preparación del CPU, en ensamblador.

Fuente: Elaboración propia 2015

2.4.2. Prototipo 14

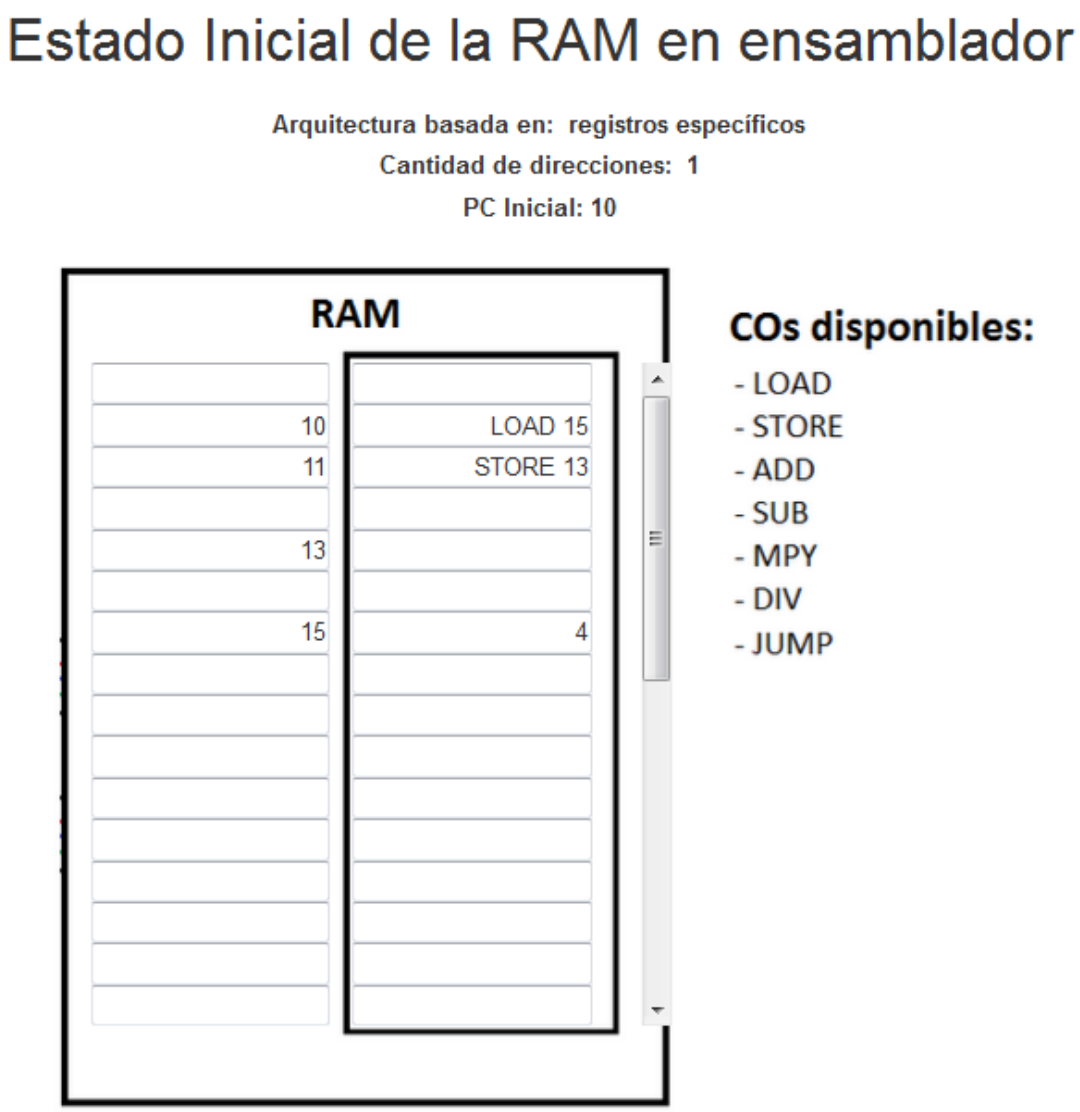
Requerimientos:

- Implementar componente para preparación de la RAM con arquitectura basada en registros específicos, con una y dos direcciones, en ensamblador.
- Implementar componente para preparación de la RAM con arquitectura basada en registros de uso general con dos direcciones, en ensamblador.

En este prototipo se implementan las interfaces de preparación de RAM con arquitectura basada en registros específicos, misma arquitectura utilizada para la resolución de ejercicios en binario y hexadecimal, esta vez para la resolución de ejercicios en ensamblador.

Primeramente se implemento la preparación para arquitectura basada en registros específicos para una dirección y luego se complemento para dos.

Figura 47. Preparación de RAM, en ensamblador. Una dirección con arquitectura basada en registros específicos.

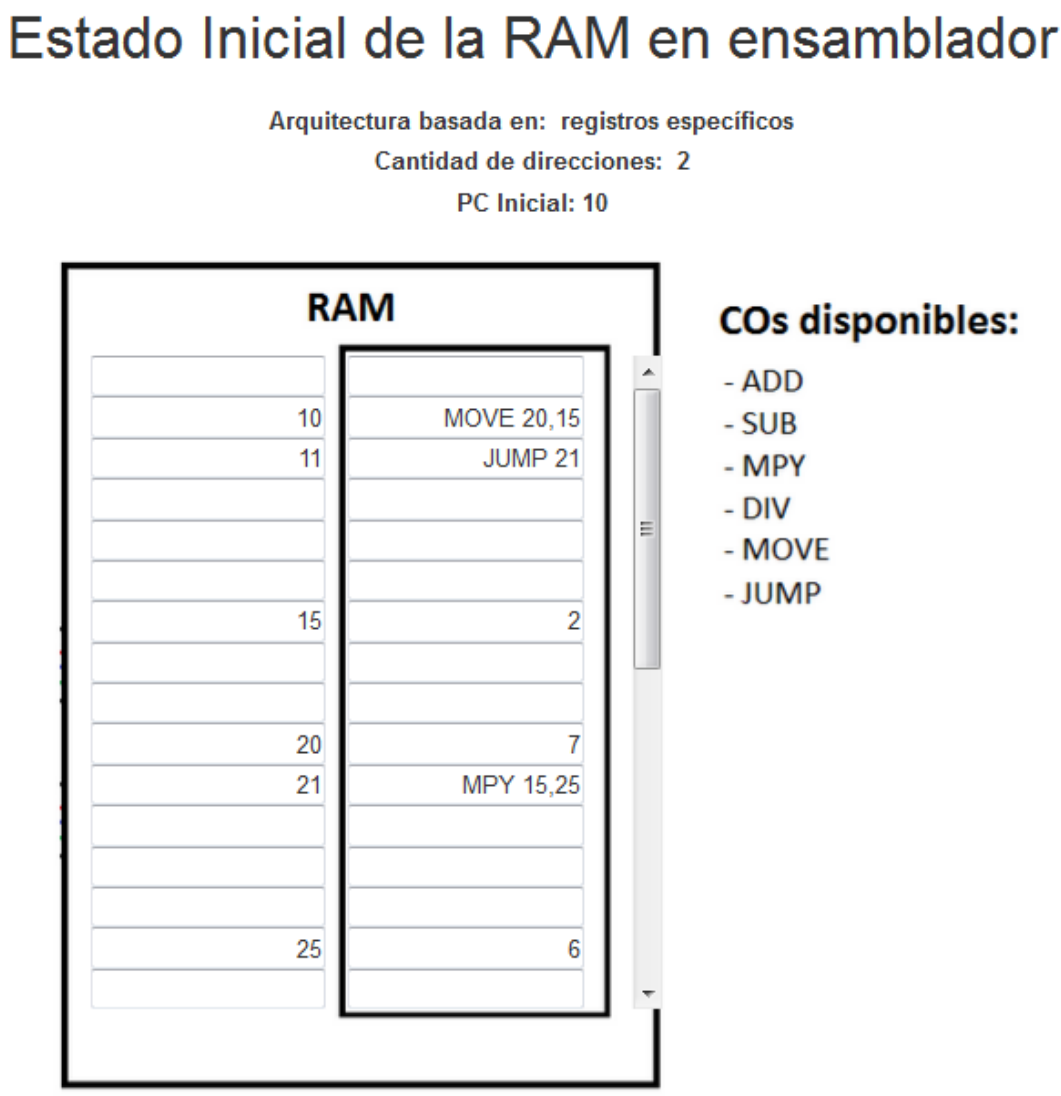


Fuente: Elaboración propia 2015

Tanto para una dirección como para dos, la preparación por parte del estudiante es hecha en la misma interfaz, utilizando el mismo *JavaScript* para la validación del estado inicial de la RAM.

Cuando se trabaja con dos direcciones se estableció que el formato de una instrucción sea el siguiente: código de operación, *espacio*, dirección uno, *coma*, dirección dos.

Figura 48. Preparación de RAM, en ensamblador. Dos direcciones con arquitectura basada en registros específicos.

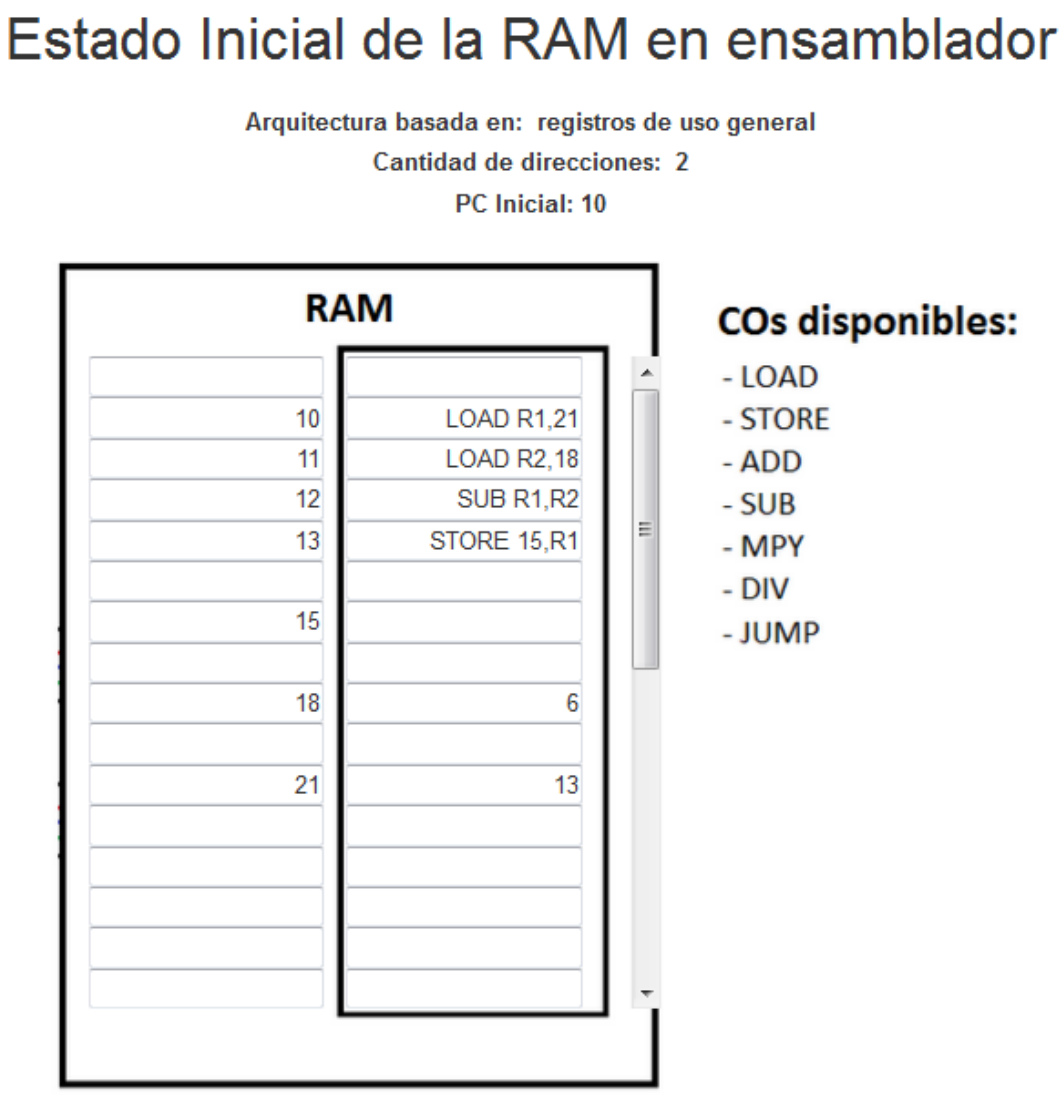


Fuente: Elaboración propia 2015

En la interfaz para la arquitectura basada en registros de uso general, como solamente se tiene la opción de usar dos direcciones, el formato establecido para la instrucción es el siguiente, dependiente de cada caso:

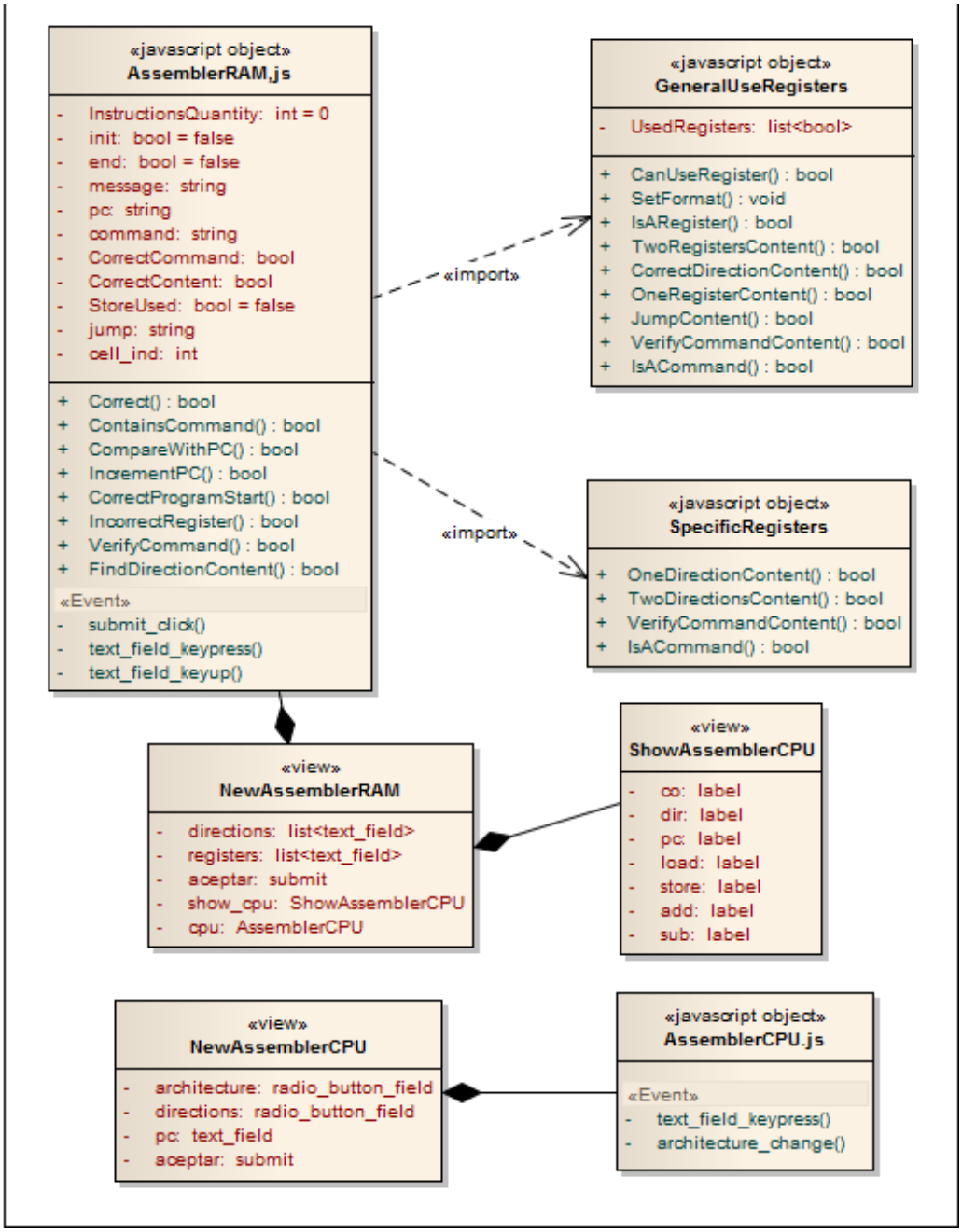
- LOAD: Código de operación, *espacio*, R#, *coma*, dirección.
- STORE: Código de operación, *espacio*, dirección, *coma*, R#.
- JUMP: Código de operación, *espacio*, dirección.
- Operación de ALU: Código de operación, *espacio*, R#, *coma*, R#.

Figura 49. Preparación de RAM, en ensamblador. Dos direcciones con arquitectura basada en registros de uso general.



Fuente: Elaboración propia 2015

Figura 50. Aplicación de JavaScripts, prototipo 14, ensamblador.



Fuente: Elaboración propia 2015

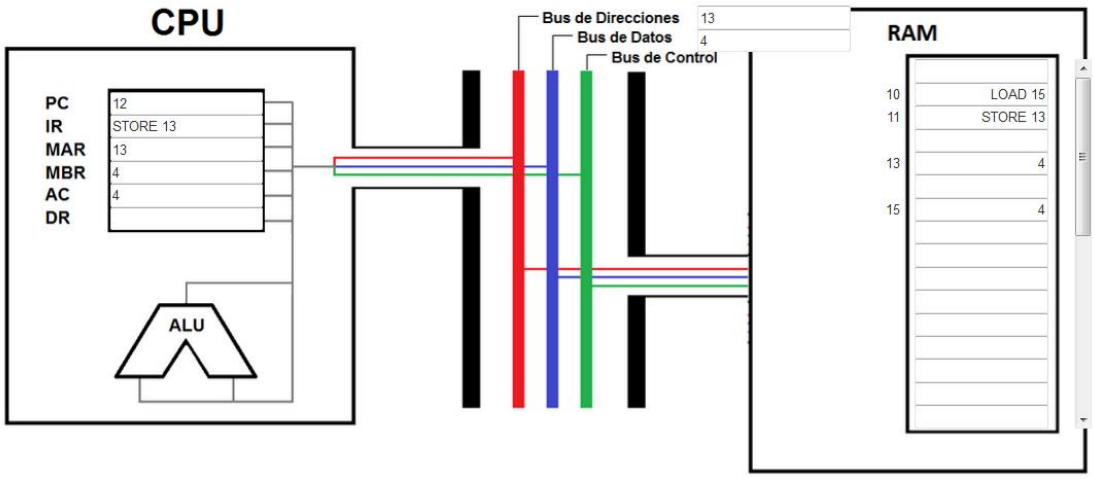
2.4.3. Prototipo 15

Requerimientos:

- Desarrollar componente interactivo de verificación de errores con arquitectura basada en registros específicos, con una y dos direcciones, en ensamblador.
- Desarrollar componente interactivo de verificación de errores con arquitectura basada en registros de uso general con dos direcciones, en ensamblador.

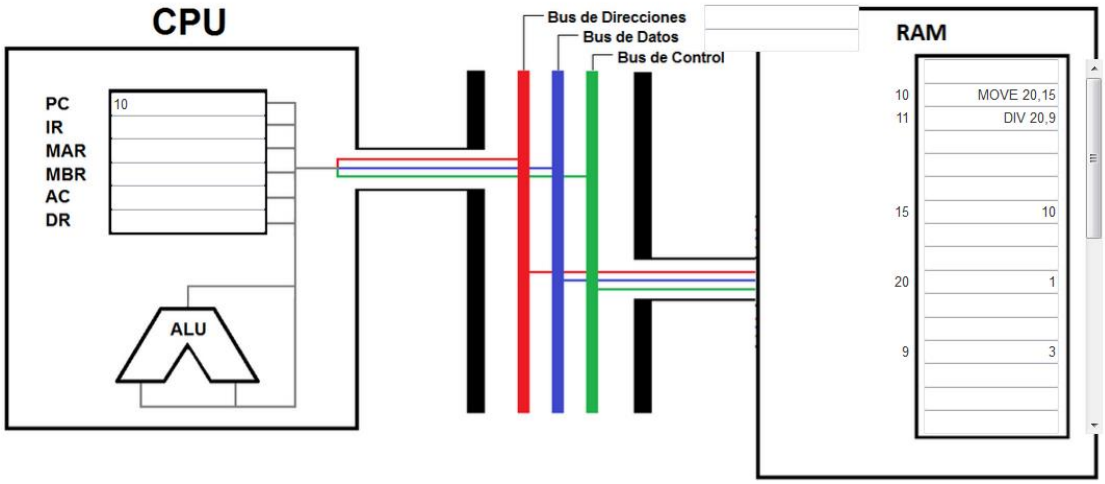
Teniendo ya la preparación de la RAM para ambas arquitecturas, solo quedaba la implementación de la interfaz principal para la resolución de ejercicios de ciclo de instrucción con detección de errores paso a paso.

Figura 51. Interfaz principal en ensamblador. Una dirección con arquitectura basada en registros específicos.



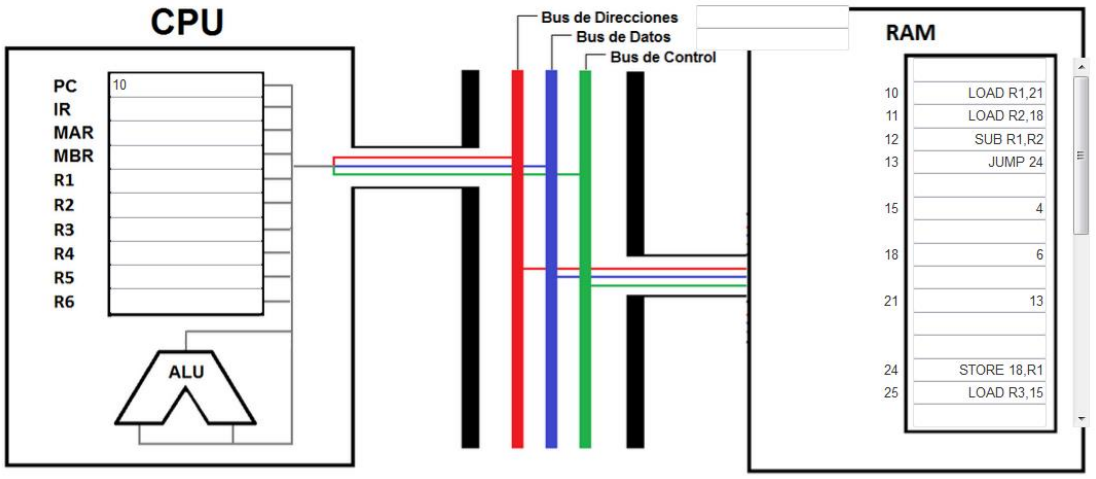
Fuente: Elaboración propia 2015

Figura 52. Interfaz principal en ensamblador. Dos direcciones con arquitectura basada en registros específicos.



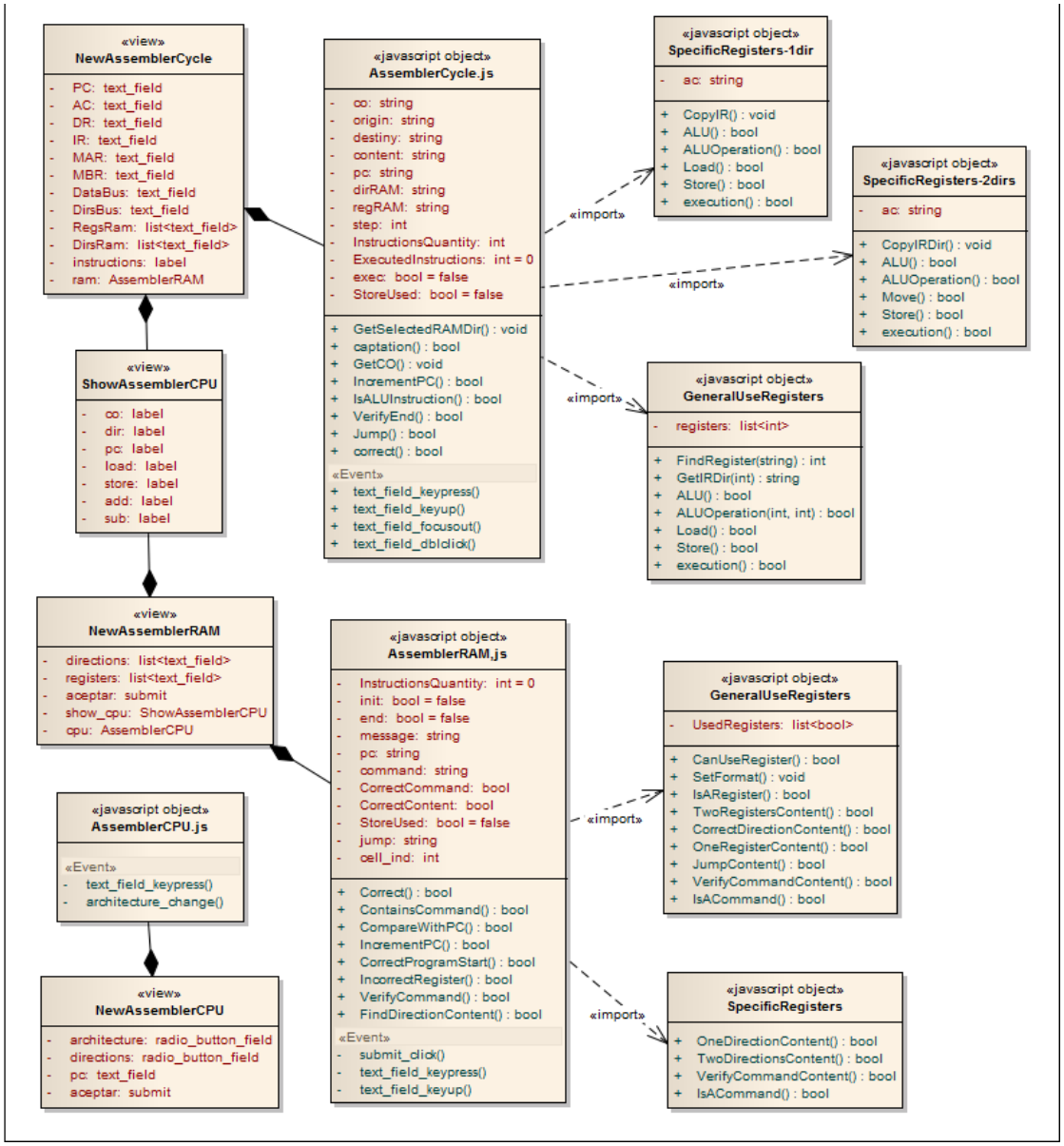
Fuente: Elaboración propia 2015

Figura 53. Interfaz principal en ensamblador. Dos direcciones con arquitectura basada en registros de uso general.



Fuente: Elaboración propia 2015

Figura 54. Aplicación de JavaScripts, prototipo 15, ensamblador.



Fuente: Elaboración propia 2015

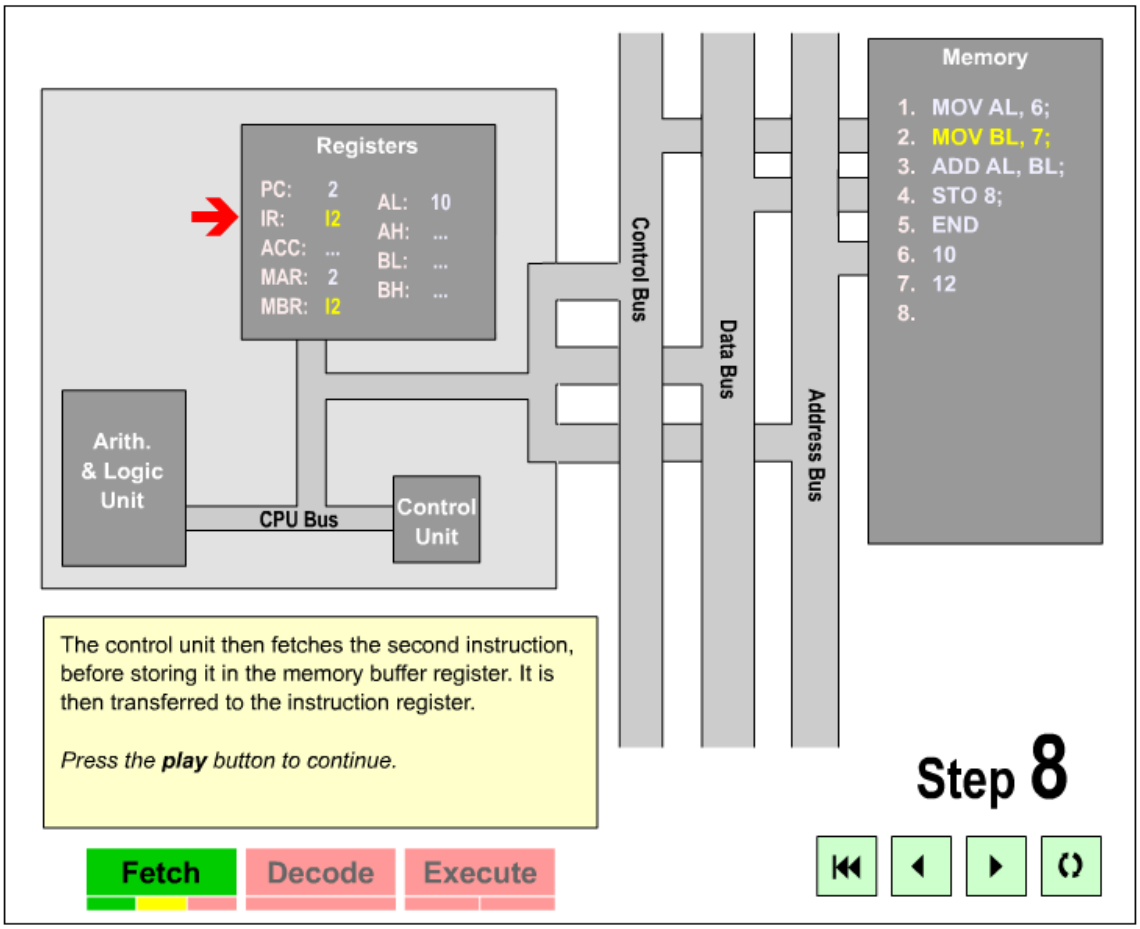
ANEXOS

ANEXOS

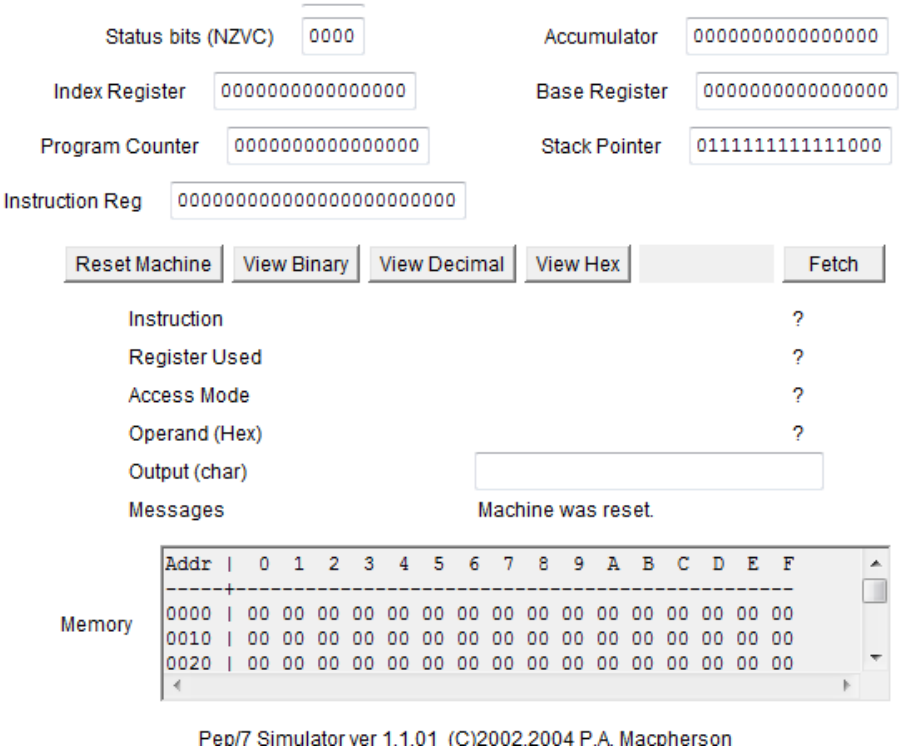
Anexo 1

Screenshots de las herramientas analizadas, encontradas en los tres enlaces de bibliografía.

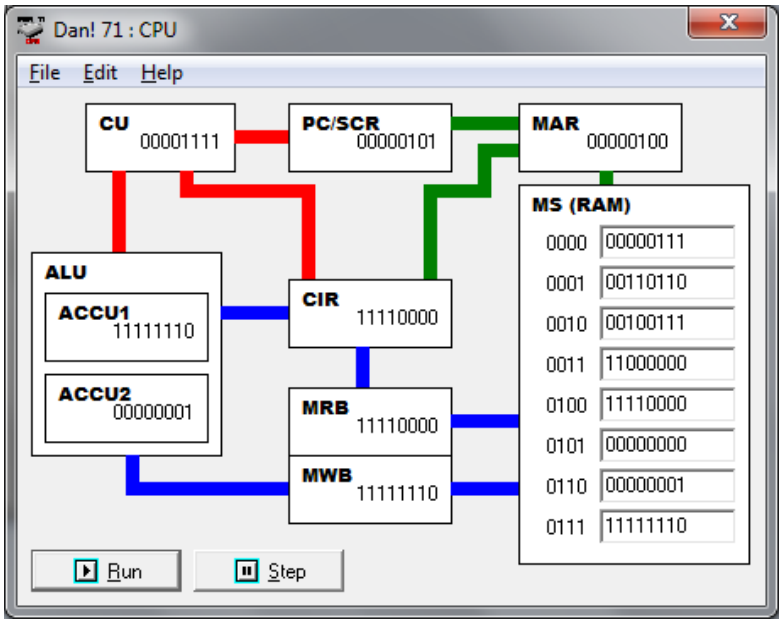
Microprocessor tutorial



pep/7 simulator



dan!71 cpu simulator



Anexo 2

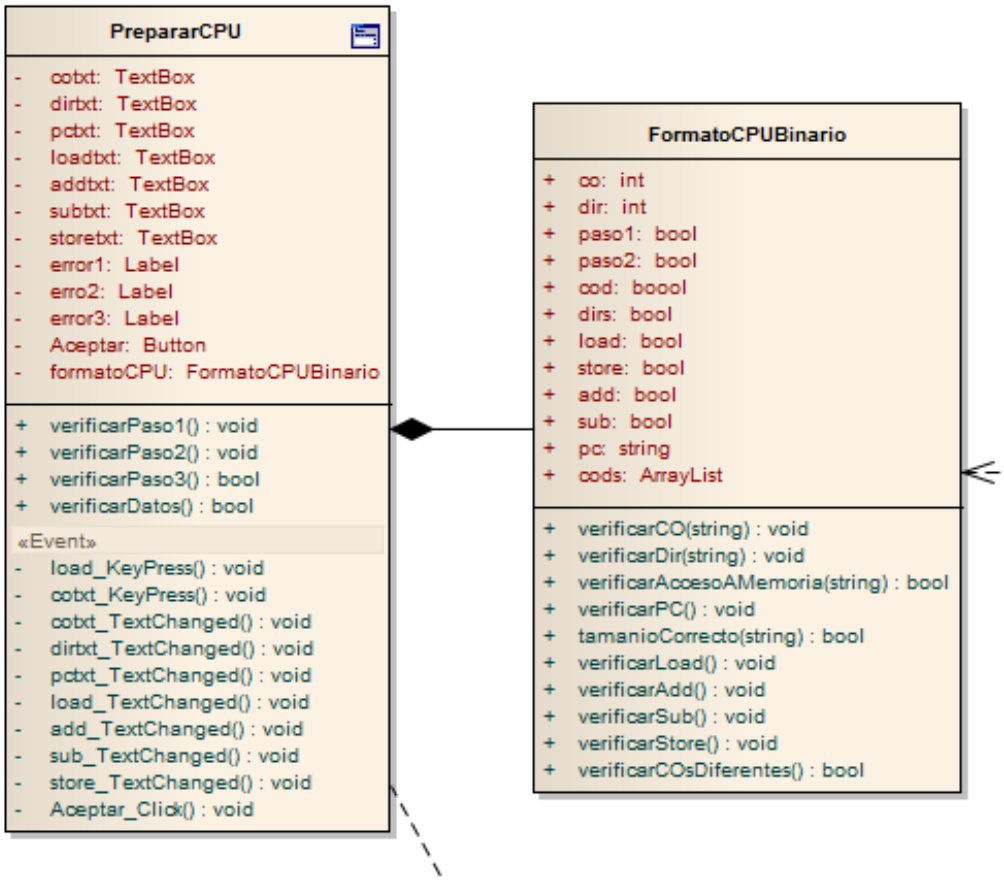
Se anexa el cuestionario utilizado para una entrevista oral con alumnos de pasados semestres de la materia de Arquitectura de Computadoras en la UCB.

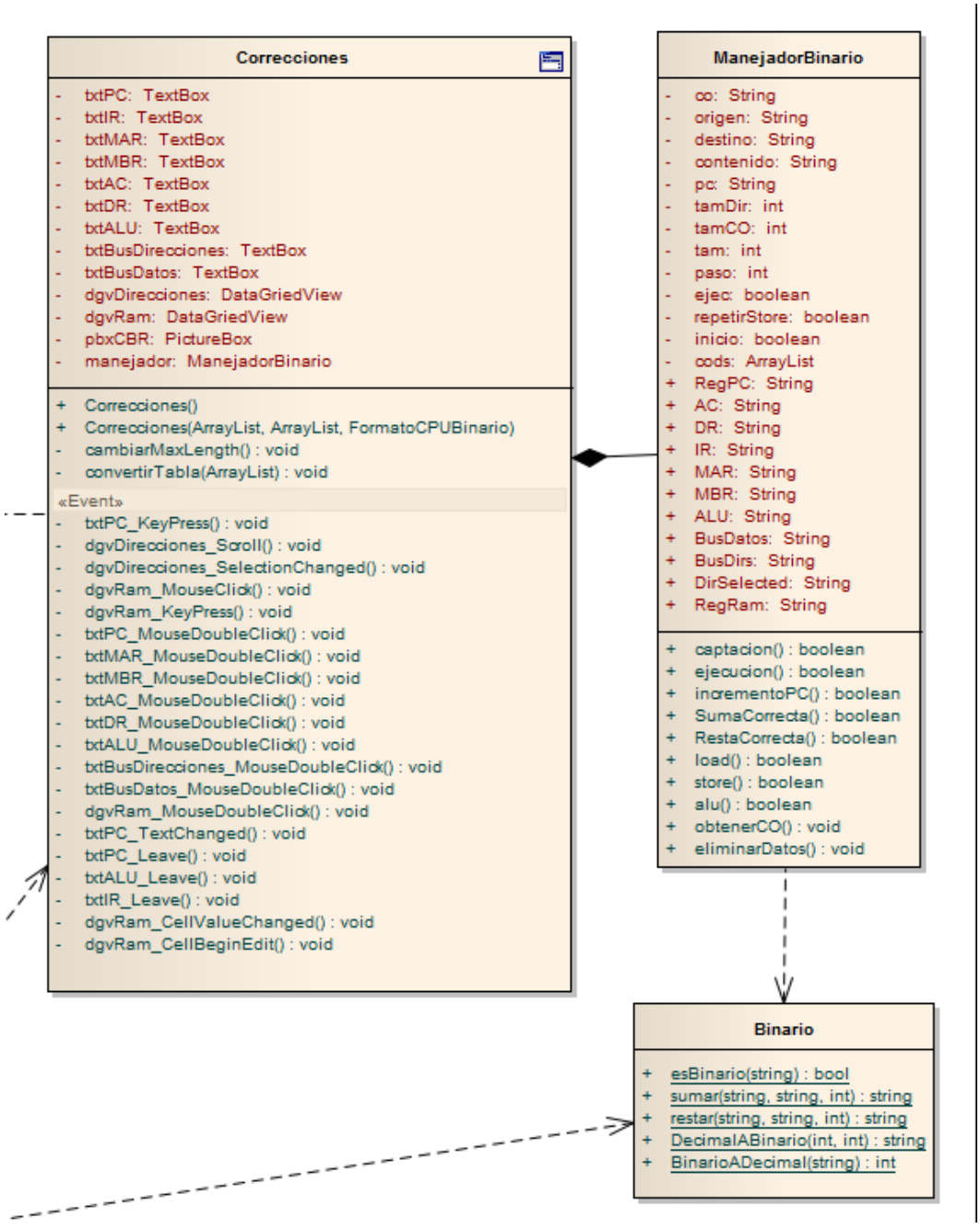
Cuestionario

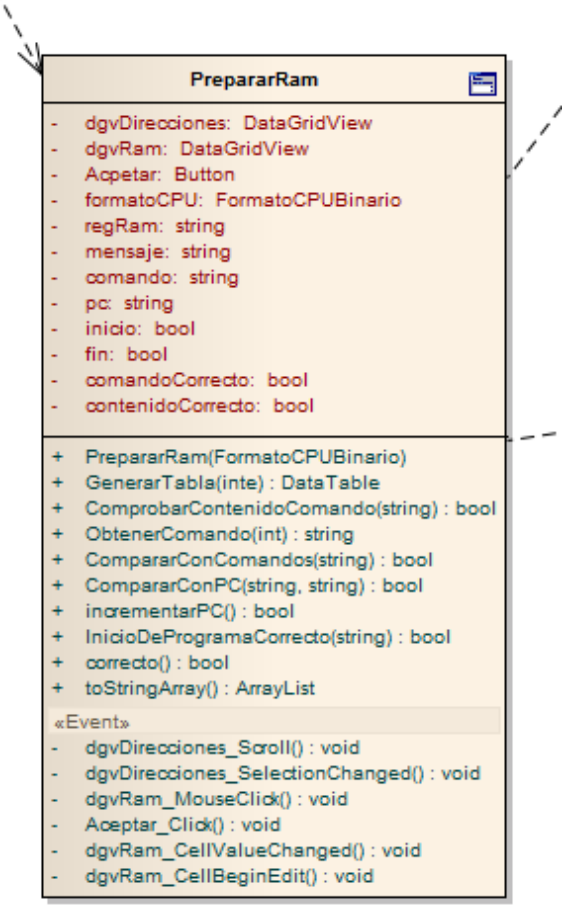
1. ¿Para usted son claros los diagramas utilizados en clases para la enseñanza del ciclo de instrucción?
   Enfatizaron que si al momento de la explicación, pero que si uno se pierde la clase no entienden bien como funciona ni con apuntes de un compañero.
2. ¿Cree usted que el tiempo empleado en realizar los diagramas es demasiado?
   La respuesta fue un si casi unánime.
3. En su opinión, ¿el tiempo empleado en la preparación de los diagramas desvía la atención del tema principal que es el ciclo de instrucción?
   No por completo, pero si recibe más atención de la que debería por el hecho de ser tantos.
4. ¿Comprende de forma rápida el ciclo de instrucción en sus distintos formatos? ¿Cuáles le causaron más problemas?
   Los que asistieron a todas las clases si, aunque tuvieron problemas al adaptarse cada que se utilizaba un formato un poco diferente.
5. ¿Cree que sería de ayuda la aplicación de una herramienta de software para el proceso de enseñanza-aprendizaje?
   Si, haría el trabajo menos monótono y más autodidacta.
6. ¿Qué sugerencias tiene respecto a la funcionalidad de una herramienta orientada a este tema?
   Desarrollar una herramienta que sea fácil de usar, simple y efectiva.

Anexo 3

El Aplicación de JavaScripts en el prototipo 9 llega a ser tan grande que es difícil distinguir a detalle los procesos y atributos de cada clase si no se hace un acercamiento a cada una de ellas, por tanto en este anexo el diagrama será fraccionado para su mejor visualización.




Correcciones
- txtPC: TextBox
- txtIR: TextBox
- txtMAR: TextBox
- txtMBR: TextBox
- txtAC: TextBox
- txtDR: TextBox
- txtALU: TextBox
- txtBusDirecciones: TextBox
- txtBusDatos: TextBox
- dgvDirecciones: DataGriedView
- dgvRam: DataGriedView
- pbxCBR: PictureBox
- manejador: ManejadorBinario
+ Correcciones()
+ Correcciones(ArrayList, ArrayList, FormatoCPUBinario)
- cambiarMaxLength() : void
- convertirTabla(ArrayList) : void
«Event»
- txtPC_KeyPress() : void
- dgvDirecciones_Scroll() : void
- dgvDirecciones_SelectionChanged() : void
- dgvRam_MouseClick() : void
- dgvRam_KeyPress() : void
- txtPC_MouseDoubleClick() : void
- txtMAR_MouseDoubleClick() : void
- txtMBR_MouseDoubleClick() : void
- txtAC_MouseDoubleClick() : void
- txtDR_MouseDoubleClick() : void
- txtALU_MouseDoubleClick() : void
- txtBusDirecciones_MouseDoubleClick() : void
- txtBusDatos_MouseDoubleClick() : void
- dgvRam_MouseDoubleClick() : void
- txtPC_TextChanged() : void
- txtPC_Leave() : void
- txtALU_Leave() : void
- txtIR_Leave() : void
- dgvRam_CellValueChanged() : void
- dgvRam_CellBeginEdit() : void
ManejadorBinario
- co: String
- origen: String
- destino: String
- contenido: String
- pc: String
- tamDir: int
- tamCO: int
- tam: int
- paso: int
- ejec: boolean
- repetirStore: boolean
- inicio: boolean
- cods: ArrayList
+ RegPC: String
+ AC: String
+ DR: String
+ IR: String
+ MAR: String
+ MBR: String
+ ALU: String
+ BusDatos: String
+ BusDirs: String
+ DirSelected: String
+ RegRam: String
+ captacion() : boolean
+ ejecucion() : boolean
+ incrementoPC() : boolean
+ SumaCorrecta() : boolean
+ RestaCorrecta() : boolean
+ load() : boolean
+ store() : boolean
+ alu() : boolean
+ obtenerCO() : void
+ eliminarDatos() : void
Binario
+ esBinario(string) : bool
+ sumar(string, string, int) : string
+ restar(string, string, int) : string
+ DecimalABinario(int, int) : string
+ BinarioADecimal(string) : int


PrepararRam
- dgvDirecciones: DataGridView
- dgvRam: DataGridView
- Acpetar: Button
- formatoCPU: FormatoCPUBinario
- regRam: string
- mensaje: string
- comando: string
- pc: string
- inicio: bool
- fin: bool
- comandoCorrecto: bool
- contenidoCorrecto: bool
+ PrepararRam(FormatoCPUBinario)
+ GenerarTabla(inte) : DataTable
+ ComprobarContenidoComando(string) : bool
+ ObtenerComando(int) : string
+ CompararConComandos(string) : bool
+ CompararConPC(string, string) : bool
+ incrementarPC() : bool
+ InicioDeProgramaCorrecto(string) : bool
+ correcto() : bool
+ toStringArray() : ArrayList
«Event»
- dgvDirecciones_Scroll() : void
- dgvDirecciones_SelectionChanged() : void
- dgvRam_MouseClick() : void
- Aceptar_Click() : void
- dgvRam_CellValueChanged() : void
- dgvRam_CellBeginEdit() : void