Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

SIMULADOR DE PROCESADOR RISC

RICHARD HERNAN ROCA DEL AGUILA

Director: José Bosch Estrada
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 28 de enero del 2016

Resumen

El proyecto consiste en el desarrollo de un simulador de procesador de arquitectura RISC . Las principales funciones del simulador será la carga de instrucciones en la memoria de programa del simulador , que tiene un límite de 45 instrucciones , grabar el programa y la ejecución del programa instrucción a instrucción.
La interfaz del simulador muestra cómo van cambiando de estado todos sus componentes y se van ejecutando las instrucciones del programa pasando por las 4 etapas (IF,OF,EX,WB) que tendrá el simulador, además también contara con un conjunto de registros internos CR (R1 a R8),una memoria de datos de 16 posiciones que aceptara enteros con signo de 32 bits ,solo 2 modos de direccionamiento inmediato y relativo , 4 tipos de instrucciones los cuales serán de registro, memoria, control y miscelánea.

Agradecimientos

La realización de este proyecto no hubiese sido posible sin la colaboración de diversas personas que me brindaron apoyo durante el transcurso de la carrera.

Quiero agradecer de manera especial al profesor José Bosch Estrada, quien ha sido mi tutor del proyecto y la persona que me ha apoyado y aconsejado en todo momento durante el desarrollo del proyecto y quien me brindó la posibilidad de participar en este proyecto.

Por último y no menos importante a mi familia que siempre me brindaron su apoyo.

Índice general

Índice de figuras

1. Introducción

Características de los RISC

- El procesador consta de unas pocas instrucciones muy simples y altamente optimizadas (realizan tareas básicas).
- La mayoría de las instrucciones se ejecutan en un ciclo de reloj.
- Todas las instrucciones tienen la misma longitud (esto es lo que hace posible la existencia del pipeline).
- Solamente las instrucciones de transferencia de datos usan memoria.
- Tienen un gran número de registros para disminuir el acceso a memoria.
- Separación de las instrucciones de acceso a memoria de las operaciones aritméticas.
- Solo las instrucciones de carga y almacenamiento acceden a la memoria de datos y de programa.

- Registros temporales que guardan los resultados después de cada etapa.

Simulador RISC:

Es un simulador de procesador con arquitectura RISC ejecuta programas sencillos máximo de 45 instrucciones , estas instrucciones pertenecen a un conjunto de instrucciones que se dividen en 4 bloques.

- Instrucciones de registro.
- Instrucciones de memoria.
- Instrucciones de control.
- Instrucciones de miscelánea.

El simulador tiene un pipeline que se divide en 4 etapas:

- IF (Instruction Fecht).
- OF (Operand Fecht).
- EX (Execute).
- WB (Write Back).

1.1. Motivación

Los simuladores de microprocesadores son bastante comunes en la docencia de arquitectura de computadores: la experiencia demuestra que es una buena herramienta y que su uso proporciona la práctica que requieren los conocimientos teóricos. Con la finalidad de tener un software con la que los alumnos puedan simular pequeños programas, surge la propuesta del proyecto Simulador de procesador con arquitectura RISC.

1.2. Objetivos

Desarrollar un software que simule la ejecución de programas sencillos, cuyos programas estarán escritos con un conjunto de instrucciones reducido, similar al que se da en clase de teoría, la cual se dicta en el segundo año de la carrera Estructura de Computadores.

Se decidió que el simulador ha de cumplir:

- El simulador ha de tener una interfaz gráfica donde se pueda visualizar las etapas del pipeline.
- Poder instalarse en cualquier sistema operativo de forma sencilla.
- Que el simulador se capaz de visualizar por medio de colores las 4 etapas del pipeline (IF, OF, EX, WB), controlada por el alumno en cada ciclo de reloj.
- Que al ingresar las instrucciones por el alumno, dependiendo del tipo de instrucción, el simulador sea capaz de limitar el formato de la instrucción.
- Poder introducir pequeños programas en código ensamblador y visualizar su ejecución.
- Guardar los programas introducidos en un fichero de texto.
- Abrir un fichero de texto que se haya guardado previamente con el simulador , y que se pueda ejecutar.
- La posibilidad de ejecutar el programa nuevamente una vez se haya ejecutado(reset del programa).

2. Planificación

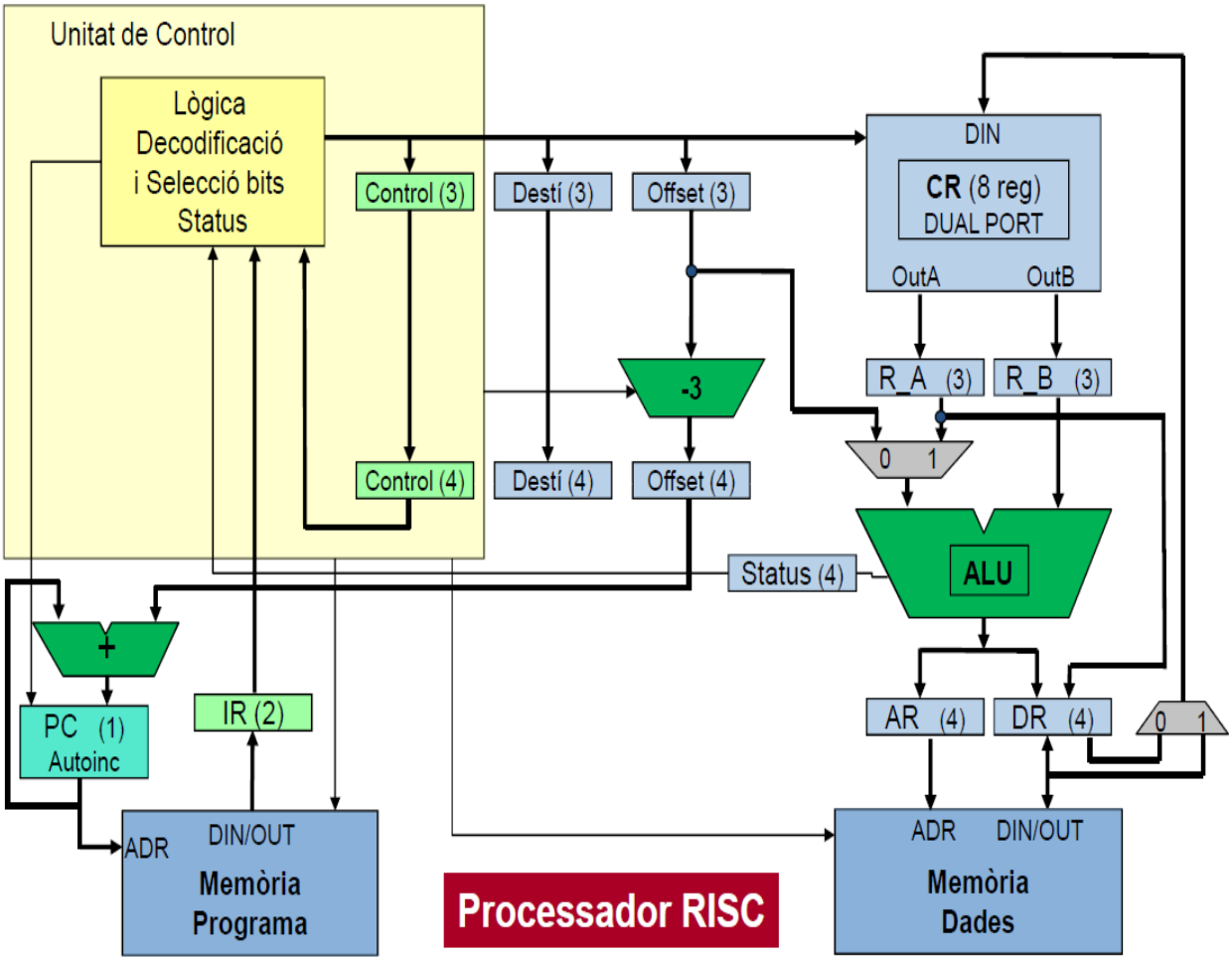
Inicial

	Setiembre				Octubre				Noviembre				Diciembre				Enero			
Nº semanas	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
Investigación																				
Desarrollo																				
Memoria																				
Presentación																				

Real

	Setiembre				Octubre				Noviembre				Diciembre				Enero			
Nº semanas	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
Investigación																				
Desarrollo																				
Memoria																				
Presentación																				

3. Descripción general Simulador RISC



Modelo de interfaz gráfica para el simulador (Figura 1)

El simulador RISC consta de una interfaz gráfica con 6 botones:

LOAD INSTRUCTION: muestra una ventana para la carga de instrucciones.

WRITE INSTRUCTION: guarda las instrucciones introducidas en la memoria de Programa de la interfaz.

LOAD DADES: carga valores en las direcciones de memoria.

WRITE DADES: guarda los valores introducidos en la memoria de datos de la interfaz.

TC: ejecuta las instrucciones de la memoria de programa.

RESET: Reinicia todo el programa eliminando los resultados anteriores.

3.1. Aplicaciones existentes en el mercado

WEBMIPS:

WebMIPS es un simulador, que facilita el proceso de aprendizaje de código ensamblador, segmentación, control, y diseño de la ruta de datos. Sin embargo su mayor ventaja es la accesibilidad inmediata para el estudiante, sin ningún tipo de instalación previa, y la posibilidad de monitorizar su actividad a través de la web. Creado por Irina Branovic, Roberto Giorgi, EnricoMartinelli (Universidad de Siena, Italia).

SPIM32:

SPIM32 es un simulador auto contenido que puede ejecutar programas escritos en código ensamblador de MIPS32, aunque no ejecuta código binario. También proporciona un depurador sencillo y un conjunto mínimo de servicios del sistema operativo. Su autor es James Larus (antiguamente Profesor del departamento de informática en la universidad de Wisconsinn-Madison).

MIPSIM

Se trata de un simulador segmentado para el microprocesador MIPS. Este microprocesador es modelado a nivel de organización, y posee unidades funcionales visibles, tales como archivos de registros, registros de segmentación, ALU, multiplexores, flujo de control y de datos, etc.

3.2. Descripción del conjunto de instrucciones

El simulador consta de un conjunto de instrucciones similar al que se da en clase de teoría.

Conjunto de instrucciones de divide en 4 bloques:

- Instrucciones de registro
- Instrucciones de memoria
- Instrucciones de control
- Instrucciones de miscelánea

3.2.1. Instrucciones de Registro

Son las que realizan operaciones aritméticas o lógicas así como las de movimiento de datos entre registros y desplazamientos de bits.

Tipo de instrucción	INSTRUCCION	Función
Registros Lógicos	**AND Desti,Font1,Font2**	AND de 2 registros.
	ANDI Desti,Font1,Offset	ANDI de un registro y una constante.
	OR Desti,Font1,Font2	OR de 2 registros.
	ORI Desti,Font1,Offset	OR de un registro y una constante.
	XOR Desti,Font1,Font2	XOR de 2 registros.
	XORI Desti,Font1,Offset	XORI de un registro y una constante.
	NOT Desti,Font1	Complementa un registro.
Registros Aritméticos	**ADD Desti,Font1,Font2**	Suma 2 registros.
	ADDI Desti,Font1,Offset	CR[Desti]←CR[Font1]+Offset.
	SUB Desti,Font1,Font2	CR[Desti]←CR[Font1]-CR[Font2].
	SUBI Desti,Font1,Offset	CR[Desti]←CR[Font1]-Offset.
	MUL Desti,Font1,Font2	CR[Desti]←CR[Font1]*CR[Font2].
	MULI Desti,Font1,Offset	CR[Desti]←CR[Font1]*Offset.
	DIV Desti,Font1,Font2	CR[Desti]←CR[Font1]/CR[Font2].
	DIVI Desti,Font1,Offset	CR[Desti]←CR[Font1]/Offset.
	SQR Desti,Font1	CR[Desti]←(CR[Font1])^2
	SQRT	CR[Desti]←Round((CR[Font1])^1/2)

	Desti,Font1	
Registros Movimiento y Desplazamiento	**MOV Desti,Font1**	CR[Desti]←CR[font1]
	RTR Font1,Offset	Desplaza Offset posiciones a la derecha los bits de CR[Font1].
	RTL Font1,Offset	Desplaza Offset posiciones a la izquierda los bits de CR[Font1].

3.2.2. Instrucciones de Memoria

Carga valores de Memoria a CR; guarda valores de CR a Memoria.

Tipo de instrucción	INSTRUCCION	Función
Memoria	**LImmH Desti,Offset**	CR[Desti[31..16]]←Offset
	LImmL Desti,Offset	CR[Desti[15..0]]←Offset
	LOAD Desti,Font2,Offset	CR[Desti]←Mem[CR[Font2]+Offset]
	STORE Font1,Font2,Offset	Mem[CR[Font2]+Offset]←CR[font1]

3.2.3. Instrucciones de Control:

Instrucciones Jump: (Salto incondicional) pueden transferir el control de la posición relativa al PC.

Instrucciones Branch: (Salto condicional) 6 posibles instrucciones Branch:

Beq,Bneq,Bge,Bgoeq,Ble, pueden transferir el control de la posición relativa si se cumple la condición.

Tipo de	INSTRUCCION	Función

instrucción		
Control	**JUMP Offset**	PC←Offset
	BEQ **Font1,Font2,Offset**	Si Font1 = Font2 PC←Offset Sino PC←PC+1
	BNEQ **Font1,Font2,Offset**	Si Font1 ≠ Font2 PC←Offset Sino PC←PC+1
	BGE **Font1,Font2,Offset**	Si Font1 > Font2 PC←Offset Sino PC←PC+1
	BGOEQ **Font1,Font2,Offset**	Si Font1 >= Font2 PC←Offset Sino PC←PC+1
	BLE **Font1,Font2,Offset**	Si Font1 < Font2 PC←Offset Sino PC←PC+1

3.2.4. Instrucciones de Miscelánea

En este caso solo veremos la Noop, no hace nada pero es muy importante para controlar las dependencias entre registros, cuando se quiere usar el mismo

registro en la siguiente instrucción hace falta que pasen 3 instrucciones donde no se use este mismo registro para que no hayan dependencias, para esto se podría hacer bien usando 3 Noops después de este registro o también operar con otras 3 instrucciones donde no se esté usando este registro.

Tipo de instrucción	INSTRUCCION	Función
Miscelánea	**NOOP**	No hace nada

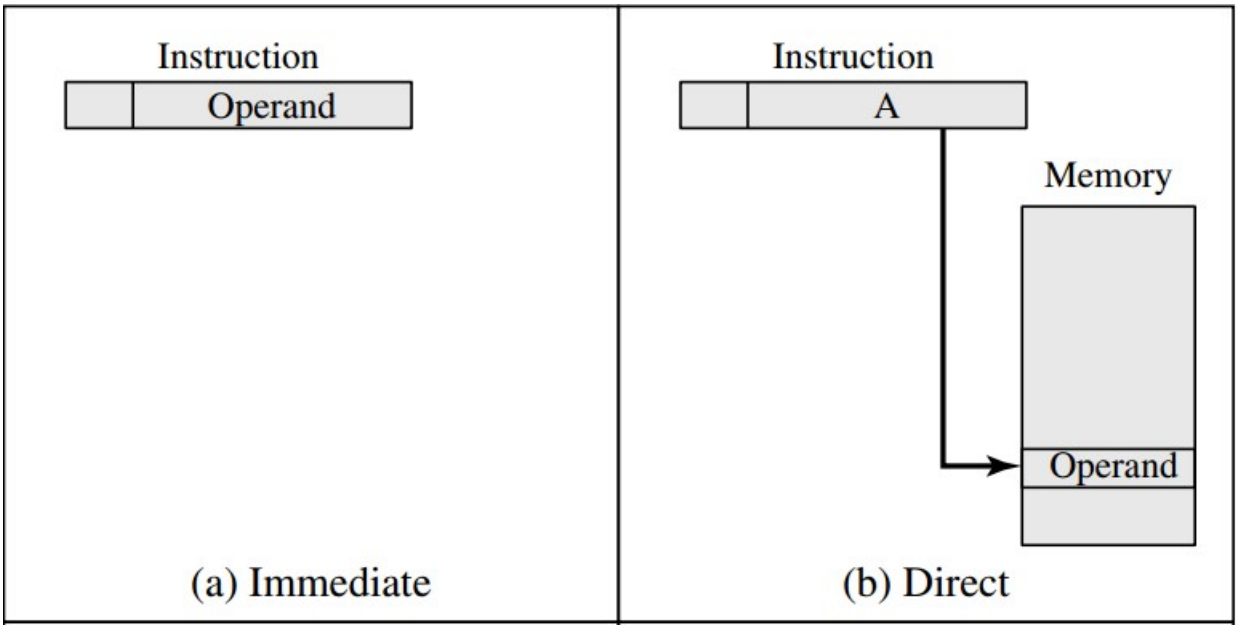
3.3. Modos de direccionamiento

Son las diferentes maneras de especificar un operando dentro de una instrucción. Un modo de direccionamiento especifica la forma de calcular la dirección de memoria efectiva de un operando mediante el uso de la información contenida en registros y/o constantes.

Existen diferentes modos de direccionamiento, estos variaran dependiendo de la arquitectura hardware. Eliminar los modos de direccionamiento más complejos podría presentar una serie de beneficios, aunque podría requerir de una serie de instrucciones adicionales, e incluso de otro registro.

La mayoría de las maquinas RISC disponen de apenas cinco modos de direccionamiento simple, mientras que otras máquinas CISC tienen más de una docena de modos de direccionamiento.

Este simulador utilizara para el conjunto de instrucciones los siguientes modos de direccionamiento:



Modos de direccionamiento (Figura 2)

3.3.1. Inmediato

En la instrucción está incluido el operando. En este modo el operando es especifico en la instrucción misma. Una instrucción de modo inmediato tiene un campo de operando en vez de un campo de dirección. El campo del operando contiene el operando actual que se debe utilizar en conjunto con la operación especificada en la instrucción. Las instrucciones de este modo son útiles para inicializar los registros en un valor constante. Cuando el campo de dirección especifica un registro del procesador, la instrucción se dice que está en modo de registro.

Ejemplo: MOV R1 R2

Modo de direccionamiento Inmediato (Figura 3)

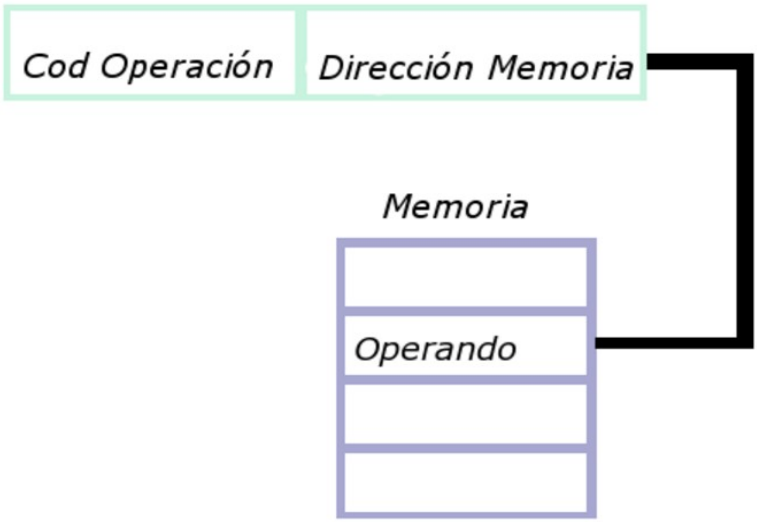
3.3.2 Directo

El campo de operando en la instrucción contiene la dirección en memoria donde se encuentra el operando.

En este modo la dirección efectiva es igual a la parte de dirección de la instrucción. El operando reside en la memoria y su dirección es dada directamente por el campo de dirección de la instrucción.

Si hace referencia a un registro de la máquina, el dato estará almacenado en este registro se le llama direccionamiento directo a registro (este modo no se aplica para este simulador); si hace referencia a una posición de memoria, el dato estará almacenado en esta dirección de memoria (dirección efectiva) y hablaremos de direccionamiento directo a memoria.

Ejemplo: LOAD R1 R8 5



Modo de direccionamiento Directo (Figura 4)

3.4. Descripción del entorno de trabajo

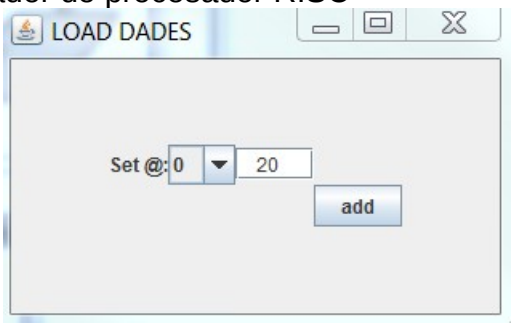
3.4.1. Carga de memoria

Botón LOAD DADES carga en memoria valores.

Botón LOAD DADES (Figura 5)

Ejemplo carga de datos:

Pulsar en LOAD DADES muestra una ventana en la cual podemos seleccionar mediante un desplegable de 0 a 15(@0-@15) posiciones de memoria. Para guardar los datos pulsamos al botón de ADD se guardara el valor introducido en dicha posición.



Ventana LOAD DADES (Figura 6)

Por defecto todas las posiciones de memoria y el conjunto de registros están con el valor 0 al arrancar el programa.

Memoria de dades	
@0 | 0	@8 | 0
@1 | 0	@9 | 0
@2 | 0	@10| 0
@3 | 0	@11| 0
@4 | 0	@12| 0
@5 | 0	@13| 0
@6 | 0	@14| 0
@7 | 0	@15| 0

Memoria de datos (Figura 6)

CR (8 reg)	
R1|0	R2|0
R3|0	R4|0
R5|0	R6|0
R7|0	R8|0

Conjunto de registros (Figura 7)

3.4.2. Grabar en memoria

Botón WRITE DADES que se encargara de escribir los datos introducidos en los registros de memoria.

Botón WRITE DADES (Figura 8)

Ejemplo:

Guardamos en la posición @0←20 presionando él botón ADD. Y así sucesivamente si se quiere guardar más valores en la memoria de datos.

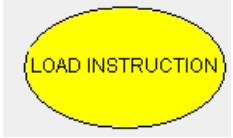
Escribir los valores introducidos (WRITE DADES):

Memoria de dades	
@0 | 20	@8 | 0
@1 | 0	@9 | 0
@2 | 0	@10| 0
@3 | 0	@11| 0
@4 | 0	@12| 0
@5 | 0	@13| 0
@6 | 0	@14| 0
@7 | 0	@15| 0

Memoria de datos (Figura 9)

3.4.3. Cargar instrucción

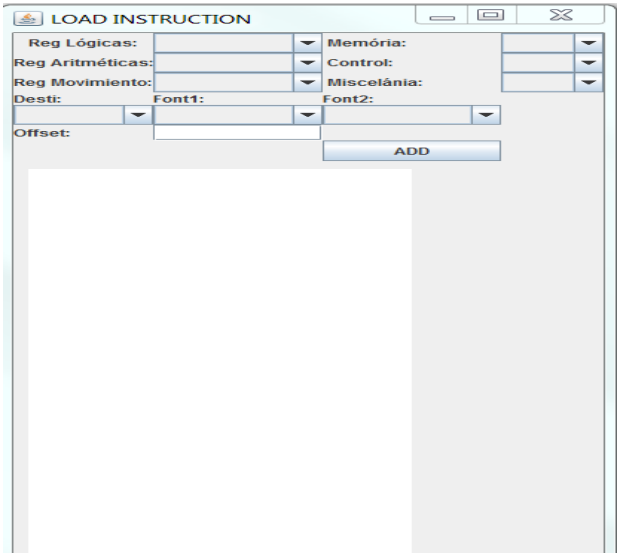
Botón LOAD INSTRUCTION muestra una ventana para poder insertar las instrucciones en la memoria de programa.



Botón LOAD INSTRUCTION (Figura 10)

Ejemplo carga de instrucciones:

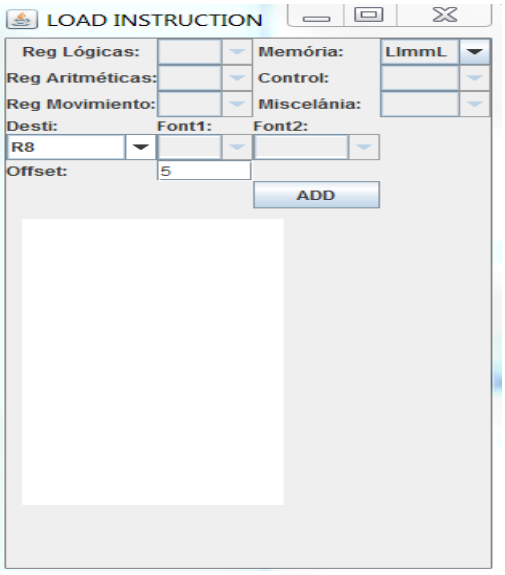
Al pulsar LOAD INSTRUCTION muestra una ventana en la cual seleccionaremos el tipo de instrucción seguido de los posibles parámetros dependiendo del código de instrucción elegido. Al finalizar la instrucción pulsamos al botón ADD para mostrarlo en la ventana LOAD INSTRUCTION.



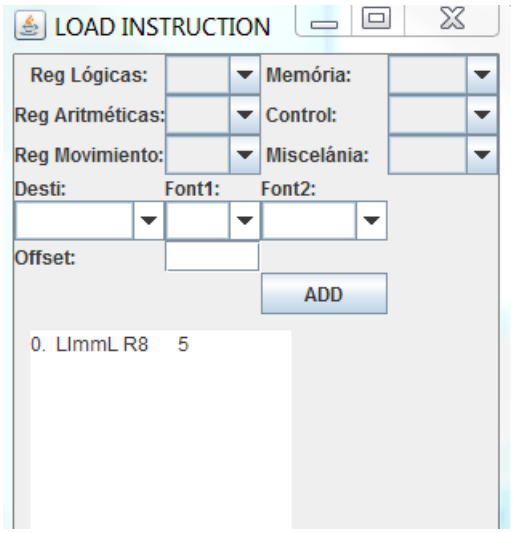
Ventana LOAD INSTRUCTION (Figura 11)

Ejemplo:

Se desea cargar la parte baja del R8 con 5



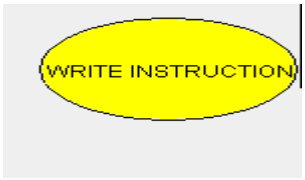
Ventana LOAD INSTRUCTION(Fig 12)



Ventana LOAD INSTRUCTION(Fig 13)

3.4.4. Grabar instrucción

Botón WRITE INSTRUCTION se encarga de escribir todas las instrucciones del programa.



Boton write instruction (Figura 14)

Guardamos en la posición @0←20 presionando él botón ADD. Y así sucesivamente si se quiere guardar más valores en la memoria de datos.

Ejemplo:

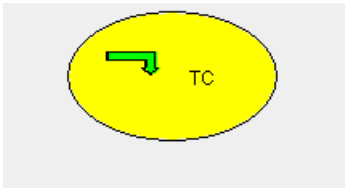
LImmL R8 5

LImmL R7 10

Memoria de Programa		
0. LImmL R8	16.	32.
1. LImmL R7	17.	33.
2. NOOP	18.	34.
3. NOOP	19.	35.
4. NOOP	20.	36.
5.	21.	37.
6.	22.	38.
7.	23.	39.
8.	24.	40.
9.	25.	41.
10.	26.	42.
11.	27.	43.
12.	28.	44.
13.	29.	45.
14.	30.	46.
15.	31.	47.

Memoria de Programa ejemplo (Figura 15)

Botón Tc: Ejecuta Instrucciones una a una después de haber cargado todas las instrucciones del programa en cada ciclo de reloj.



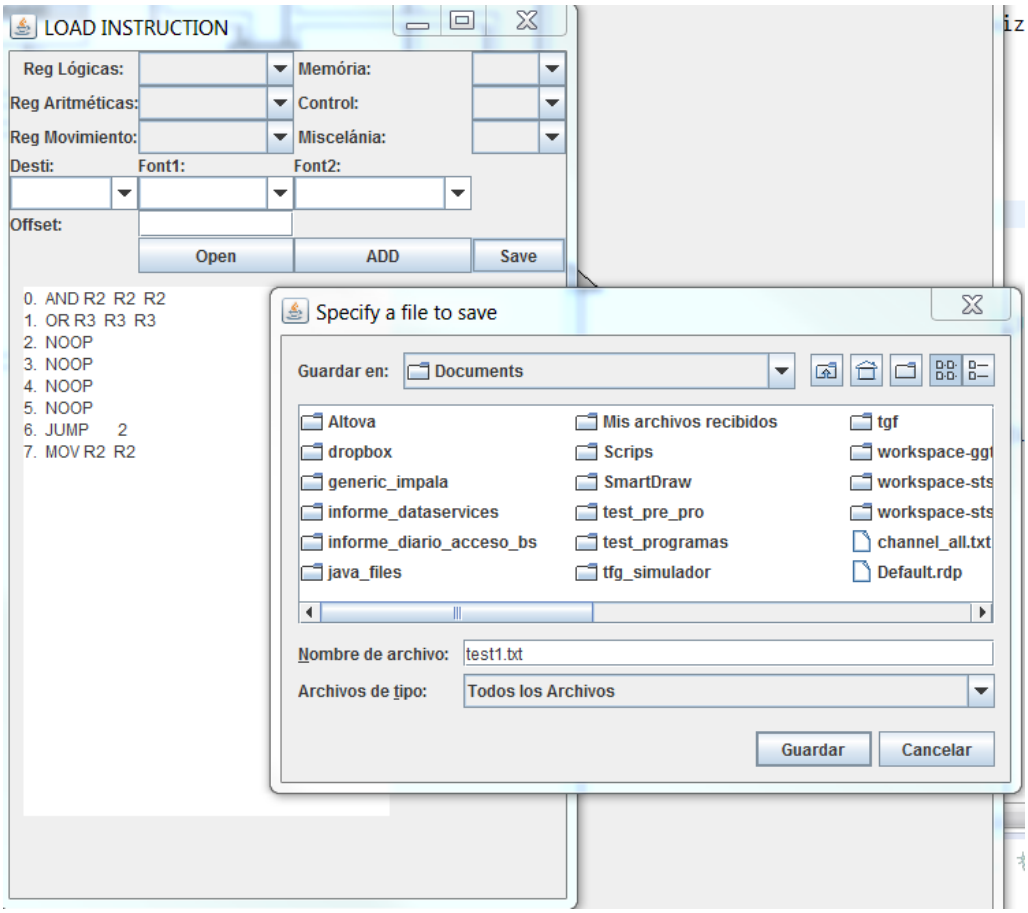
Boton TC (ciclo de reloj) Figura 16

3.4.5. Grabar programa

Botón LOAD INSTRUCTION muestra una ventana para poder guardar instrucciones en un fichero *.txt .

Ejemplo:

Se agregaran unas cuantas instrucciones y guardara en un fichero de texto.



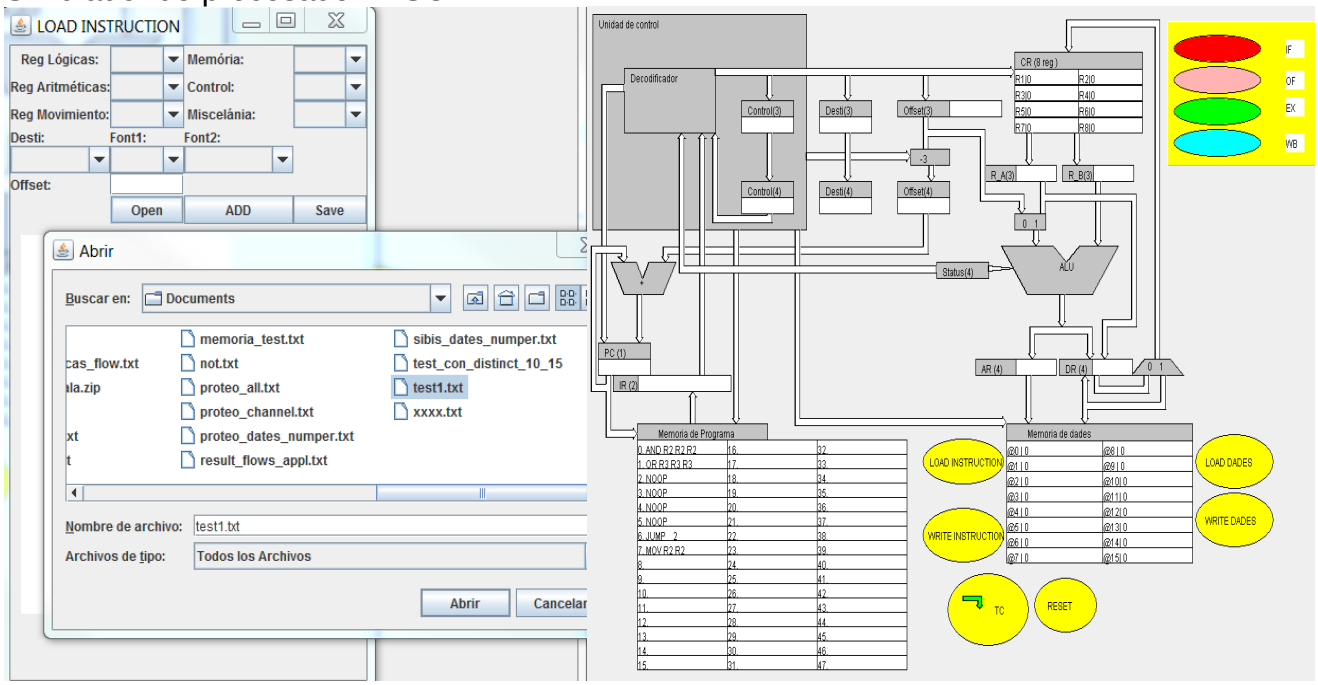
Grabar programa(Figura 17)

3.4.6. Abrir programa desde fichero

Botón LOAD INSTRUCTION muestra una ventana para poder abrir fichero que se haya guardado previamente con el simulador , después de haber dado a abrir fichero para poder visualizarlo en la memoria de programa del simulador se ha de presionar al botón WRITE INSTRUCTION.

Ejemplo:

Se abre un fichero de texto el cual tendrá las instrucciones previamente guardadas.



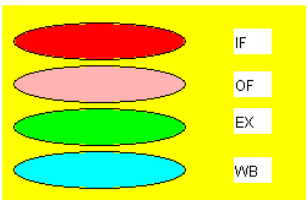
Cargar programa (Figura 18)

3.5. Etapas del simulador RISC

Las etapas del simulador se marcan con los siguientes colores para observar el flujo y como van cambiando la interfaz en la ejecución de cada instrucción.

Los acrónimos son del inglés:

- IF: Instruction Fetch
- OF: Operand Fetch
- EX: Execute
- WB: Write Back



Representación en colores de las etapas (Figura 19)

3.5.1. IF (Instruction Fetch)

Busca la instrucción en la memoria principal (memoria de programa). Entonces la CPU pasa la instrucción de la memoria principal a través del bus de datos al IR (registro de instrucciones), guardando las instrucciones temporalmente de manera que pueda ser codificada y ejecutada por las demás etapas.

3.5.2 OP (Operand Fetch)

Recoge los datos desde la memoria principal (solo si es necesario), se accede al banco de registros por los operandos ,se calcula el valor del operando inmediato (solo si es necesario), también lee la dirección efectiva de la memoria principal si la instrucción tiene una dirección indirecta, y se recogen los datos requeridos de la memoria principal para ser procesados y colocados en los registros de datos.

3.5.3. EX (Execute)

A partir del registro de instrucción de control, los datos que forman la instrucción son decodificados por la unidad de control. Está interpreta la información como una secuencia de señales de control que son enviadas a las unidades funcionales relevantes de la CPU para realizar la operación requerida por la instrucción.

3.5.4. WB(Write Back)

El resultado generado por la operación es almacenado en la memoria principal o enviado a un dispositivo de salida dependiendo de la instruccion.Basandose en los resultados de la operación, el contador de programa se incrementa para apuntar a la siguiente instrucción.

Al finalizar todas las etapas se carga él PC←PC +1 para cargar así la siguiente instrucción o se actualiza con una dirección diferente donde la próxima instrucción será recogida.

4. Desarrollo

Este proyecto no tiene antecedentes, no es la continuación de otros proyectos, principalmente consiste en desarrollar un simulador con fines educativos , capaz de simular un procesador de arquitectura RISC que tenga una interfaz gráfica similar a la figura 3.0 después de cargar las instrucciones del conjunto de instrucciones se podrá visualizar las 4 etapas para cada instrucción.

Estos son los requerimientos exigidos para el Simulador:

-Conjunto de instrucciones , en total son 31 Instrucciones , dividida en 4 bloques. En las tablas figura 1.1 (Instrucciones de Registro), figura 1.2 (Instrucciones de Memoria),figura 1.3 (Instrucciones de control) y figura 1.4 (Instrucciones de miscelánea).

-Memoria de datos consta de 16 registros que pueden guardar enteros de 32 bits, inicialmente tendrán el valor 0 figura 2.0.

-Conjunto de registros consta de 8 registros que pueden guardar enteros de 32 bits , inicialmente tendrán el valor 0 figura 2.1.

-Memoria de Programa , el límite de la memoria de programa es de 48 instrucciones figura 2.6 de las cuales las 3 últimas instrucciones se completaran con 3 NOOPS para que pueda terminar el programa.

-IR este registro se activara para la microinstrucción IF , carga la instrucción que se está ejecutando actualmente.

-PC este registro se activara para la microinstrucción IF muestra el índice de la siguiente instrucción a ejecutar.

-Control(3) este registro se activara para la microinstrucción OF muestra el código de operación del índice de la instrucción actual - 1.

-Desti(3) este registro se activara para la microinstrucción OF muestra el parámetro destino (si existe) para este tipo de operación ,del índice de la instrucción actual - 1.

-Offset(3) este registro se activara para la microinstrucción OF muestra el parámetro offset (si existe) de este tipo de operación ,del índice de la instrucción actual - 1.

-R_A(3) este registro se activara para la microinstrucción OF muestra el parámetro font2 (si existe) de este tipo de operación ,del índice de la instrucción actual - 1.

-R_B(3) este registro se activara para la microinstrucción OF muestra el parámetro font1 (si existe) de este tipo de operación ,del índice de la instrucción actual - 1.

-AR(4) este registro se activara con la microinstrucción EX muestra algún valor si el destino es alguna dirección de memoria.

-DR(4) este registro se activara con la microinstrucción EX muestra algún valor si el destino es algún registro del conjunto de registros.

4.1. Casos de usos

A continuación se describirá los casos de usos los más relevantes en la interfaz del simulador para comprender mejor el desarrollo de la misma.

UC1:Carga de instrucciones en el simulador.

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario carga las instrucciones en el simulador , al presionar el botón LOAD INSTRUCTION de la interfaz gráfica se abrirá una ventana donde seleccionara la instrucción.
Precondiciones	Ninguna
Escenario principal	1.El usuario activa el botón de la interfaz del simulador LOAD INSTRUCTION que llamara a la función de cargar instrucciones. 2.El simulador mostrara una ventana donde se podrá seleccionar entre 4 tipos de instrucción de registro,movimiento,memoria y miscelánea . 3. En función del tipo de instrucción y el código de operación que se seleccione se habilitaran o deshabilitaran los parámetros Destino,fuente1,fuente2 y offset. 4.Despues de haber seleccionado la instrucción y sus parámetros , al

	presionar el botón ADD se comprobara que el campo offset introducido no esté vacío y que sea un entero con signo de 16 bits. 5.Si las comprobaciones de la entrada de instrucción son correctas la instrucción se agregara a la lista de instrucciones del simulador, presionando el botón ADD. 6.La instrucción agregada se mostrara el área de texto con la finalidad que el usuario pueda tener ver la instrucción añadida y el índice que le corresponde a la instruccion.
Postcondiciones	La lista de instrucciones del simulador ha añadido una instrucción.

UC2: Escritura de instrucciones en la memoria de programa

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario carga las instrucciones en la memoria de programa, al presionar el botón WRITE INSTRUCTION.
Precondiciones	**UC1** ha sido finalizado o se ha pasado por el caso de uso de cargar desde fichero.
Escenario principal	1.El usuario presiona el botón WRITE INSTRUCTION que llamara a la función de carga de instrucciones. 2.El simulador agregara 3 instrucciones NOOP al final de las instrucciones cargadas (con la finalidad de que la última instrucción termine correctamente).
Postcondiciones	Se mostrara en la memoria de programa del simulador todas las instrucciones cargadas en el UC1 o

	carga desde fichero.

UC3: Ejecución de las instrucciones

Actor principal	Usuario del simulador.
Personal involucrado e intereses	El usuario ejecuta las instrucciones una a una , al presionar el botón TC de la interfaz gráfica.
Precondiciones	El usuario ha cargado el programa en el simulador(UC1 y UC2) o carga desde fichero.
Escenario principal	1.El usuario activa el botón de la interfaz del simulador TC que llamara a la función de ejecutar instrucciones que se muestran la interfaz en memoria de programa. 2.La casilla donde se encuentra la instrucción que se está ejecutando cambiara de color en nuestro caso de blanco a amarillo. 3.Cuando la instrucción que se esté ejecutando llegue al final de programa es decir al índice más alto del programa se mostrara un dialogo con el texto “Fin de programa”.
Postcondiciones	Los componentes del simulador como registros temporales ,memoria ,conjunto de registros y buses han cambiado de color o contienen algún valor que se mostrara.

UC4: Reset del simulador

Actor principal	Usuario del simulador.
Personal involucrado e intereses	El usuario presiona el botón RESET el simulador reinicia el programa.

Precondiciones	La memoria de programa del simulador no está vacía.
Escenario principal	1. Se borraran todos los registros y buses menos el programa cargado anteriormente.
Postcondiciones	Se mostrara solo el programa ejecutado anteriormente.

UC5: Carga de memoria

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario carga los valores en el simulador , al presionar el botón LOAD DADES de la interfaz gráfica.
Precondiciones	Todas las direcciones están inicializadas con 0.
Escenario principal	1.El usuario activa el botón de la interfaz del simulador LOAD DADES que llamara a la función de cargar memoria. 2.El simulador mostrara una ventana donde se podrá seleccionar entre las 16 posiciones de memoria. 4.Despues de haber seleccionado la dirección de memoria , al presionar el botón ADD se comprobara que el campo introducido no esté vacío y que sea un entero con signo de 32 bits. 5.Si las comprobaciones de la entrada son correctas se agregara el valor a memoria, sino se mostrara un dialogo con el mensaje “El número introdujo no es válido”.
Postcondiciones	El mapa de memoria ha agregado un nuevo valor.

UC6: Escritura de valores en la memoria de datos.

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario carga los valores en la memoria de datos, al presionar el botón WRITE DADES.
Precondiciones	**UC4** ha sido finalizado.
Escenario principal	1.El usuario presiona el botón WRITE DADES que llamara a la función de escritura de datos en memoria. 2.Muestra todos los valores cargados en UC5 en la dirección de memoria correspondiente.
Postcondiciones	Se mostrara en la memoria de datos del simulador todas los valores cargados en el UC5.

UC7: Guardar programa.

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario guarda el programa, al presionar el botón SAVE de la ventana LOAD INSTRUCTION.
Precondiciones	**UC1** ha sido finalizado.
Escenario principal	1.El usuario presiona el botón SAVE que llamara a la función de guardar programa. 2.Se abrirá un ventana donde se tendrá que dar la ruta , donde se guardara el programa.
Postcondiciones	Se ha guardado el programa en un fichero de texto.

UC8: Abrir programa.

Actor principal	Usuario del simulador
Personal involucrado e intereses	El usuario al presionar el botón OPEN de la ventana LOAD INSTRUCTION seleccionara el fichero , donde tendrá

	que estar guardado el programa.
Precondiciones	**UC7** ha sido finalizado.
Escenario principal	1.El usuario presiona el botón OPEN que llamara a la función de abrir fichero. 2.Se abrirá un ventana donde se tendrá que seleccionar el fichero , donde se guardara el programa.
Postcondiciones	Se ha guardado el programa cargado desde el fichero en la memoria de programa

UC9: Etapa instrucción fetch

Actor principal	Simulador o interfaz gráfica.
Personal involucrado e intereses	Interfaz del simulador , registro de instrucciones , PC y buses.
Precondiciones	La instrucción que se está ejecutando no está vacía.
Escenario principal	1.Carga en el registro de instrucción el número de instrucción actual. 2.Pinta de color rojo los buses que conectan al contador de programa, decodificador y a registro de instrucciones.
Postcondiciones	1.La instrucción afectada además de poder visualizarse en el registro de instrucciones , también se mostrara de color amarillo en la memoria de programa del simulador. 2.Si la instrucción cargada esta entre las 4 primeras , al finalizar esta instrucción se incrementa el contador de programa a uno , sino no lo incrementa , este se incrementara al finalizar las 4 etapas.

UC10: Etapa operand fetch

Actor principal	Simulador
Personal involucrado e intereses	Interfaz del simulador , registros de control(3) , Desti(3),Offset(3) , R_A(3) ,R_B(3) y buses.
Precondiciones	La instrucción que se está ejecutando no está vacía y además se ha pasado por el UC9.
Escenario principal	1. Carga en el código de operación de la instrucción actual – 1. 2.Carga en el registro Desti(3) si existe este valor en la instrucción y lo muestra. 3.Carga en registro Offset(3) si existe este valor en la instrucción y lo muestra. 4.Carga en registro R_A(3) si existe este valor en la instrucción como font2 y lo muestra. 5.Carga en registro R_B(3) si existe este valor en la instrucción como font1 y lo muestra. 6.Pinta de color rosado los buses que conectan estos registros y el que conecta el decodificador con el de memoria de datos.
Postcondiciones	1.Se mostraran los registros y buses afectados de color PINK. 2. Si la instrucción cargada esta entre las 4 primeras , al finalizar esta instrucción se incrementa el contador de programa a uno , sino no lo incrementa , este se incrementara al finalizar las 4 etapas.

UC11: Etapa execute

Actor principal	Simulador o interfaz gráfica.

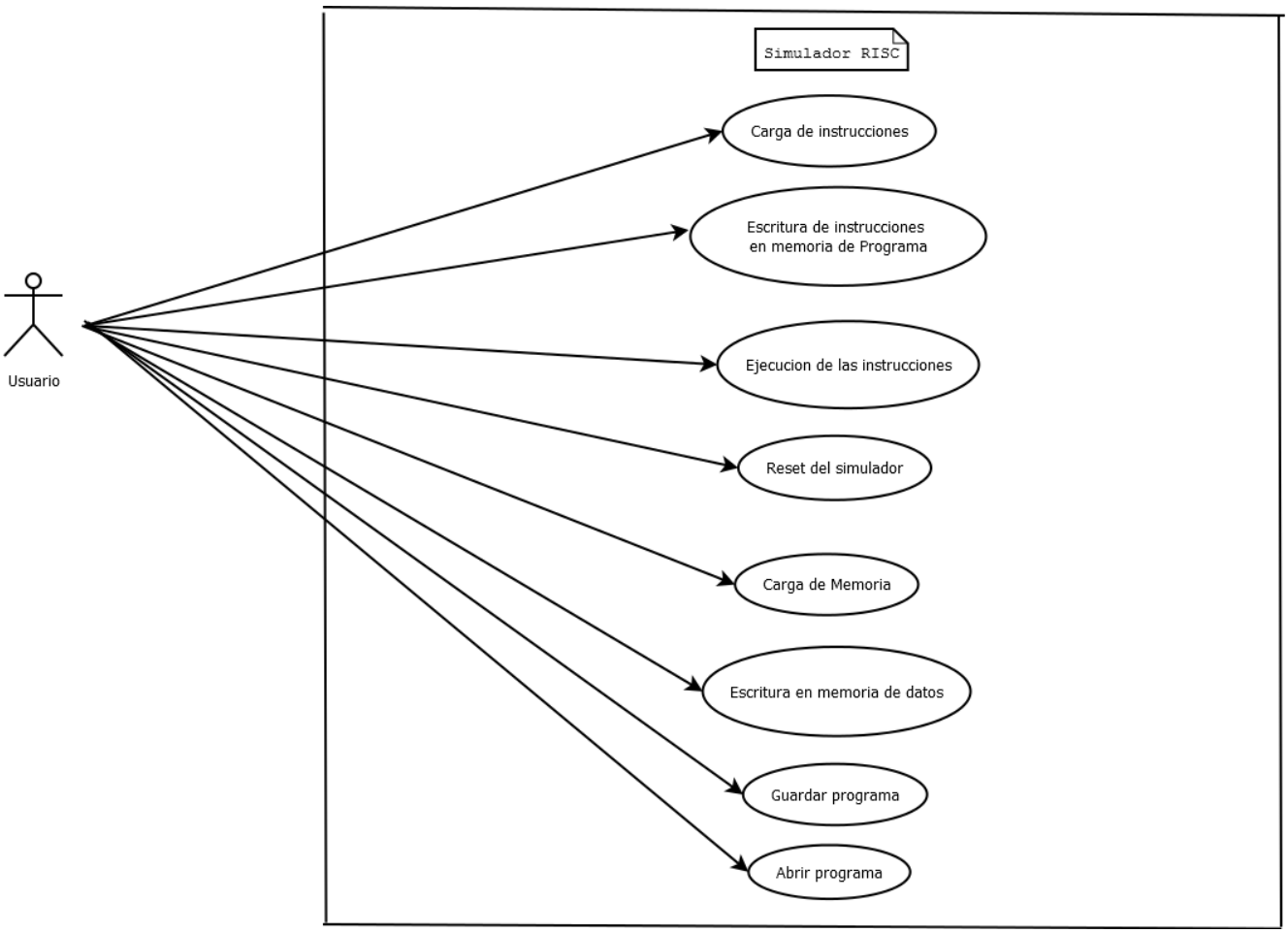
Personal involucrado e intereses	Interfaz del simulador , registros de control(4) , Desti(4),Offset(4) , R_A(4) ,R_B(4) y buses.
Precondiciones	La instrucción que se está ejecutando no está vacía y además se ha pasado por el UC10.
Escenario principal	1. Carga en el código de operación de la instrucción actual – 2. 2.Carga en el registro Desti(4) si existe este valor en la instrucción y lo muestra. 3.Carga en registro Offset(4) si existe este valor en la instrucción y lo muestra. 4.Carga en registro A_R(4) si existe este valor en la instrucción como memoria destino en la instrucción y lo muestra. 5.Carga en registro D_R(4) si existe este valor en la instrucción como registro destino y lo muestra. 6.Pinta de color verde los buses que conectan estos registros y el que conecta al status(4) con la ALU y el decodificador.
Postcondiciones	1.Se mostraran los registros y buses afectados de color verde. 2. Si la instrucción cargada esta entre las 4 primeras , al finalizar esta instrucción se incrementa el contador de programa a uno , sino no lo incrementa , este se incrementara al finalizar las 4 etapas.

UC12: Etapa write back

Actor principal	Simulador o interfaz gráfica.
Personal involucrado e intereses	Interfaz del simulador , conjunto de registros(CR),memoria de datos y

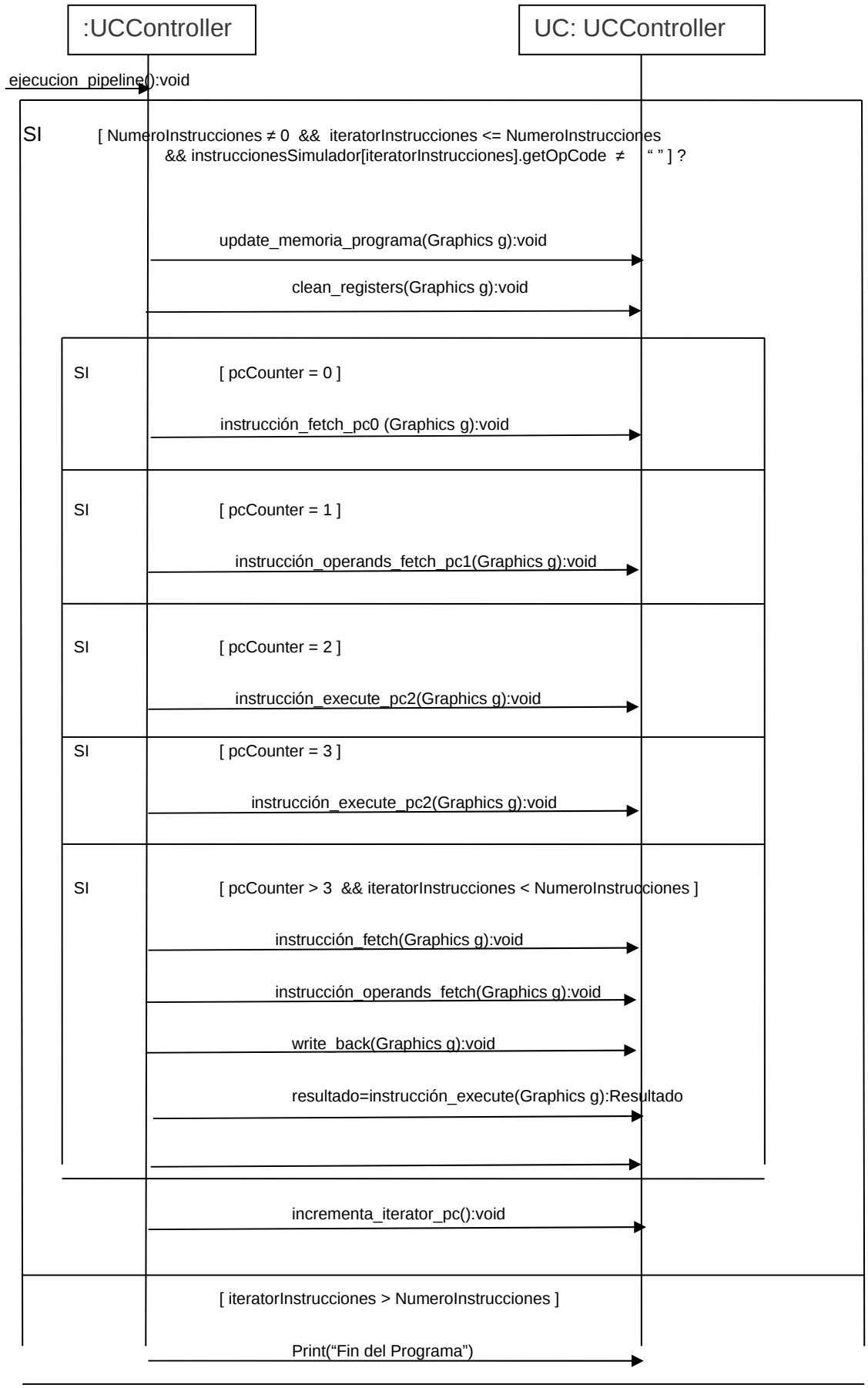
	buses.
Precondiciones	La instrucción que se está ejecutando no está vacía y además se ha pasado por el UC11.
Escenario principal	1. Escribe los resultados de la instrucción actual – 3, en el conjunto de registros o memoria de datos. 2.Pinta de color celeste los registros modificados y los buses que conectan al conjunto de registros y a la memoria de datos.
Postcondiciones	1.Se mostraran los registros y buses afectados de color celeste. 2. Si la instrucción cargada esta entre las 4 primeras , al finalizar esta instrucción se incrementa el contador de programa a uno , sino no lo incrementa , este se incrementara al finalizar las 4 etapas.

4.2. Diagrama de Casos de uso entre el usuario y la interfaz gráfica del simulador

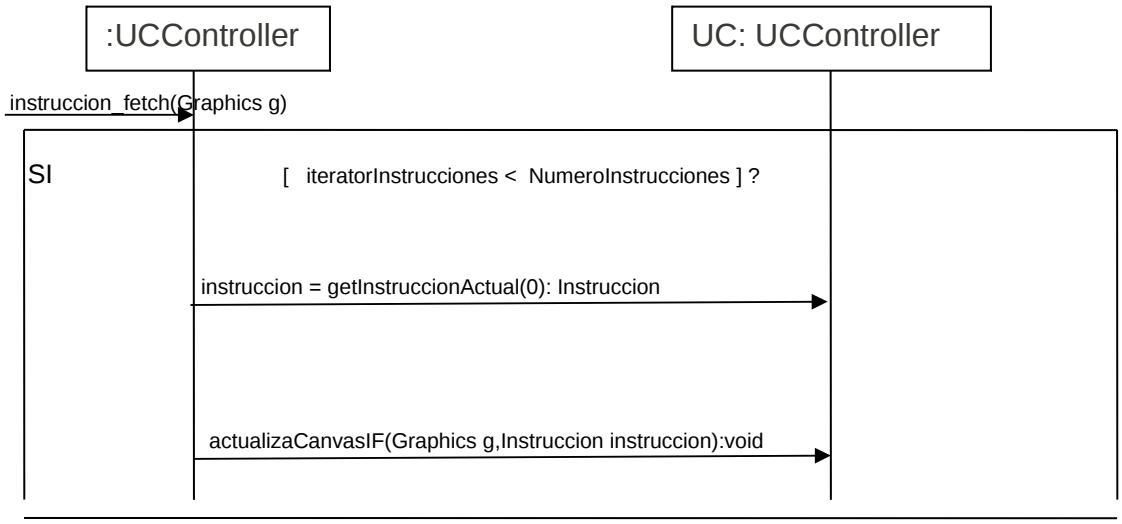


Casos de usos (Figura 20)

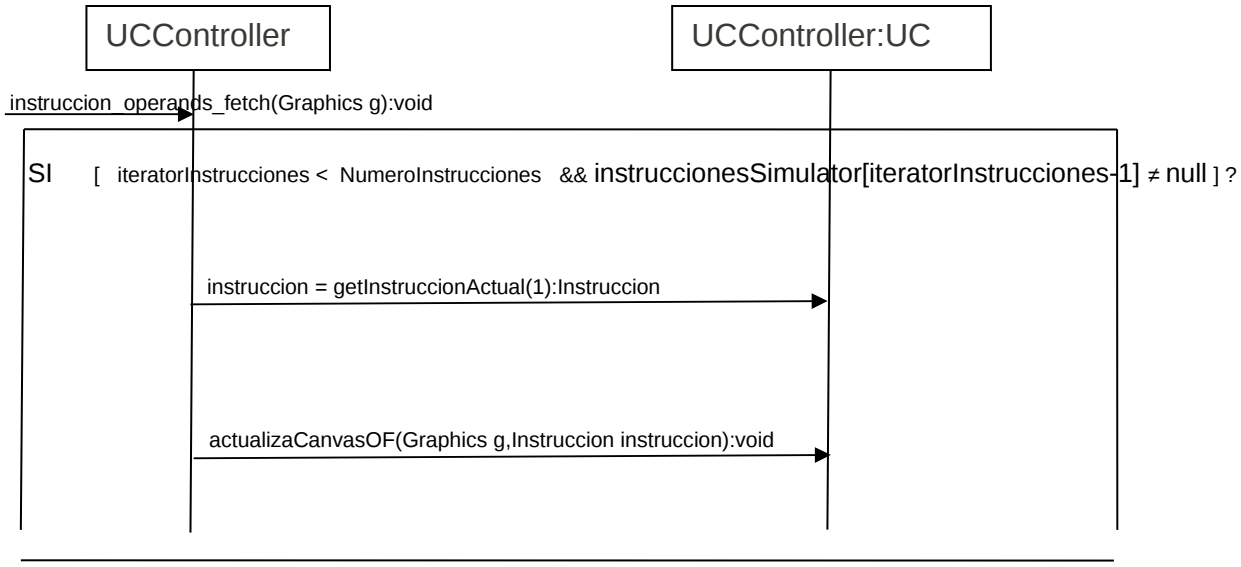
4.3. Diagramas de secuencia
UC3:Ejecucion de las instrucciones



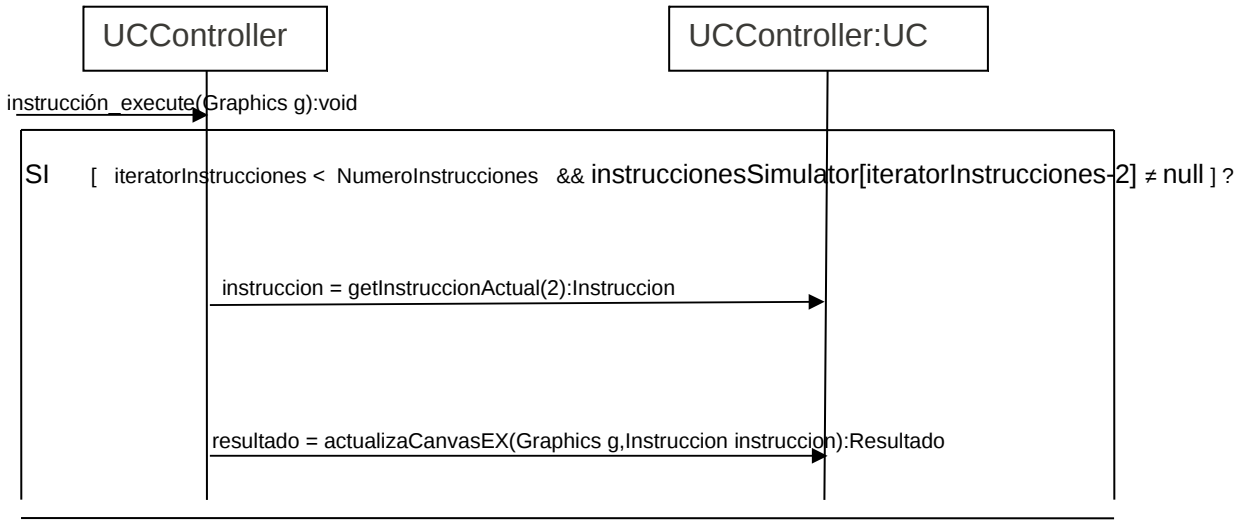
UC9: IF(Instruction Fetch)



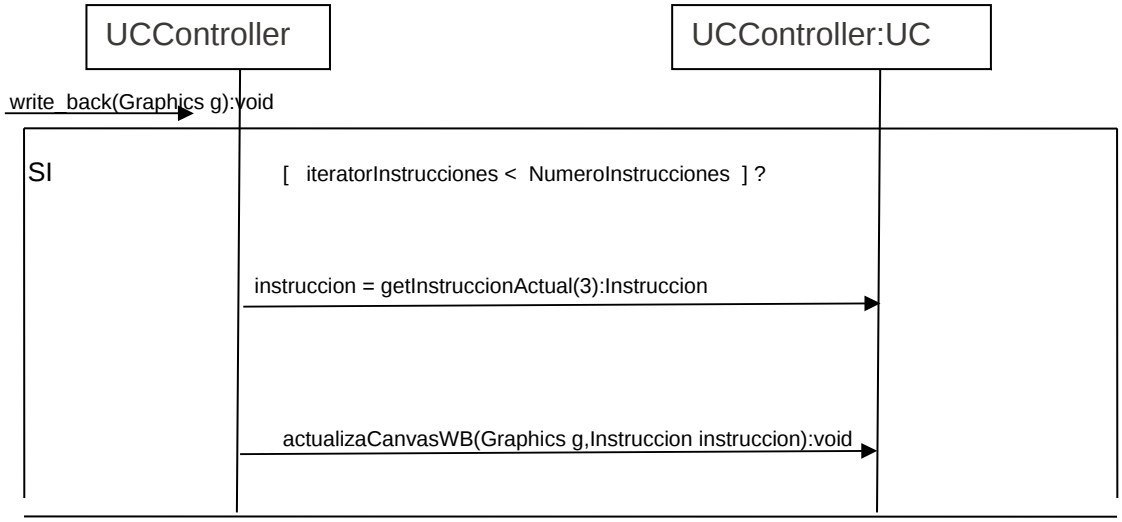
UC10: OF(Operand Fetch)



UC11: EX(Execute)



UC12: WB(Write back)



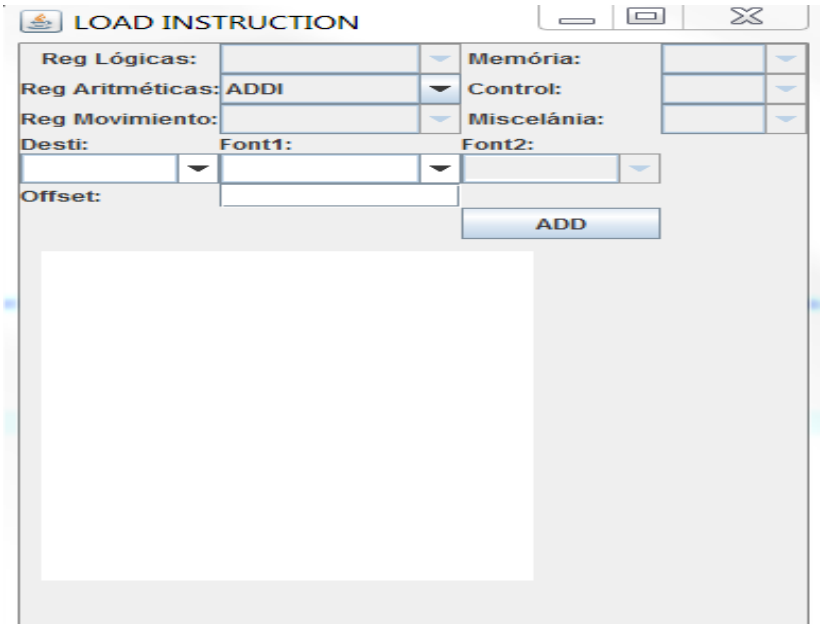
4.4. Formato de instrucción

-Al seleccionar un código de operación de cualquiera de los bloques se ha de poder habilitar o deshabilitar los siguientes campos destino , font1 ,font2 y offset dependiendo de la operación seleccionada.
A continuación se explicara en seudocódigo solo para el caso de las operaciones aritméticas inmediatas donde le campo font2 se oculta. Para los demás casos el procedimiento es similar.

Lógica del simulador para la comprobación del formato de instrucciones:

Si operacionSelecciona == ADDI || operacionSelecciona == SUBI ||
operacionSelecciona == MULI || operacionSelecciona == DIVI

deshabilita campo font2



Formato instrucción (Figura 21)

-En el campo Offset solo está permitido valores enteros de 16 bits , en caso se introdujera otro valor que no sea un entero se mostrara un dialogo especificando que el formato del parámetro introducido no es correcto al aceptar el dialogo nos retornara a la ventana anterior hasta que introduzcamos un valor correcto. Estas comprobaciones varían dependiendo del tipo de instrucción.

Si operacionSelecciona == ANDI || operacionSelecciona ==ORI ||

operacionSelecciona == XORI || operacionSelecciona == ADDI ||
operacionSelecciona == MULI || operacionSelecciona == DIVI ||
operacionSelecciona == SUBI

Si destino está vacío

Lanza un dialogo con el siguiente mensaje “Desti no puede estar vacio”
Retornara a la ventana anterior para ingresar el destino

Si font1 esta vacío

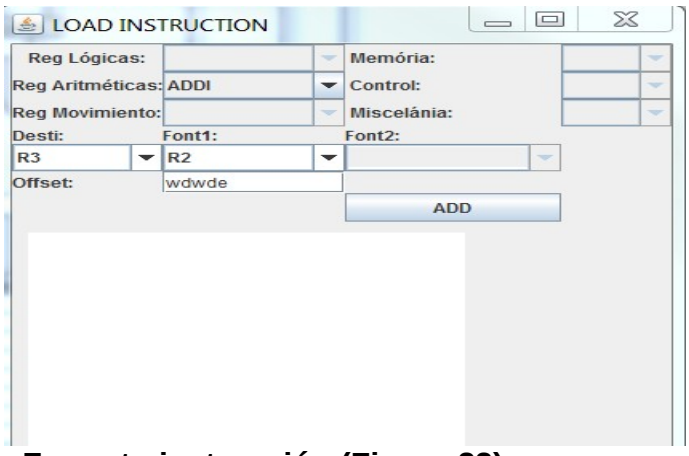
Lanza un dialogo con el siguiente mensaje “Font1 no puede estar vacío”
Retornara a la ventana anterior para ingresar el font1

Si valorCorrecto(offset) es falso

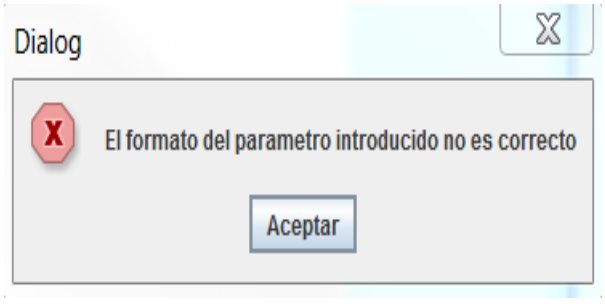
Lanza un dialogo con el siguiente mensaje “El formato del parámetro introducido no es correcto”
Retornara a la ventana anterior para ingresar el offset

Guarda los campos obtenidos en un objeto Instruction
Agrega el objeto instrucción a la lista de instrucciones
Muestra la instrucción agregada + un salto de línea para la siguiente instrucción en el área Texto de la ventana

Ejemplo:



Formato instrucción (Figura 22)



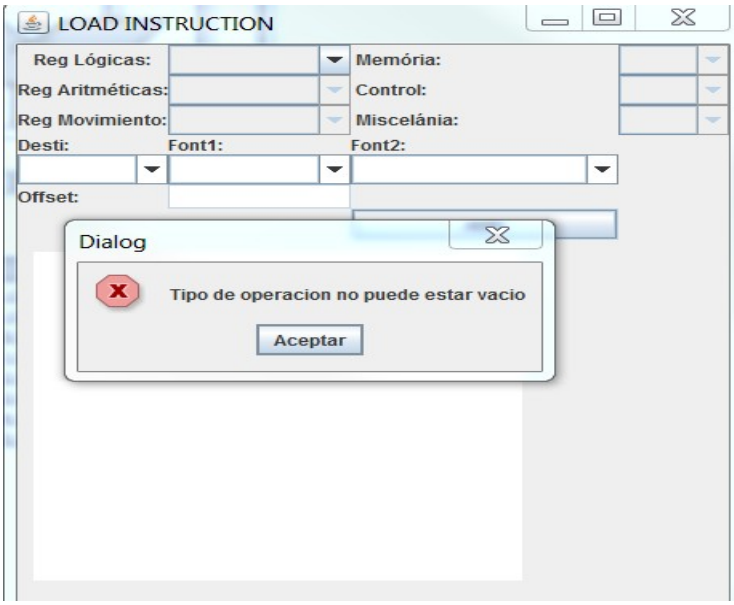
Offset incorrecto (Figura 23)

-Si seleccionamos un código de operación vacío mostrara un dialogo que el código de operación no puede estar vacío al aceptar nos retornara a la ventana anterior hasta que demos un código de operación que no sea vacío y completar con los parámetros dependiendo de la operación seleccionada.

Si operacionSeleccionada == “ ”

Lanza un dialogo con el siguiente mensaje “Tipo de operación no puede estar vacío”

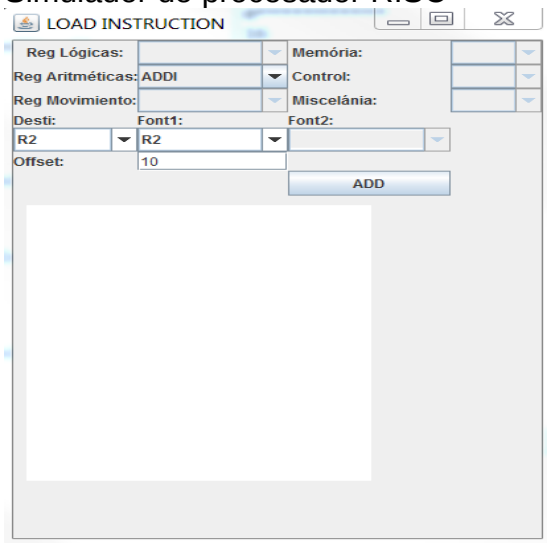
Retornara a la ventana anterior para ingresar la operación

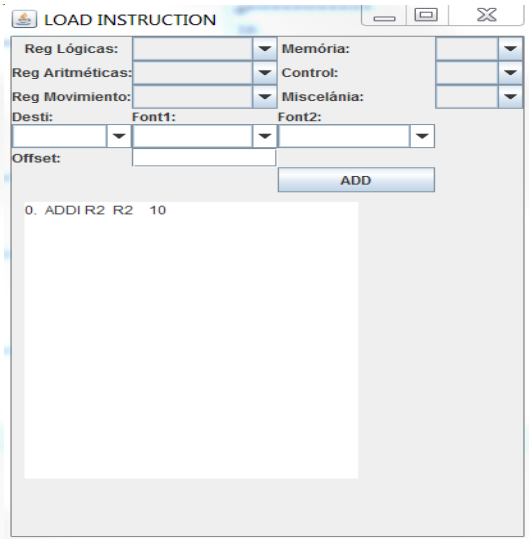


Código de operación vacío (Figura 24)

-Al agregar la instrucción se podrá visualizar así como su índice de la instrucción en el área de texto. Con la siguiente secuencia de órdenes.

Guarda los campos obtenidos en un objeto Instruction
Agrega el objeto instrucción a la lista de instrucciones
Muestra la instrucción agregada + un salto de línea para la siguiente instrucción en el área Texto de la ventana .





Formato instrucción (Figura 25) **Formato instrucción (Figura 26)**

4.5. Carga de instrucciones

-Al presionar el boton de WRITE INSTRUCTION se muestran todas las instrucciones cargadas en la ventana LOAD INSTRUCTION si no hay instrucciones lanza un dialogo con el siguiente mensaje “No hay instrucciones a cargar”

Si numeroInstrucciones > 0

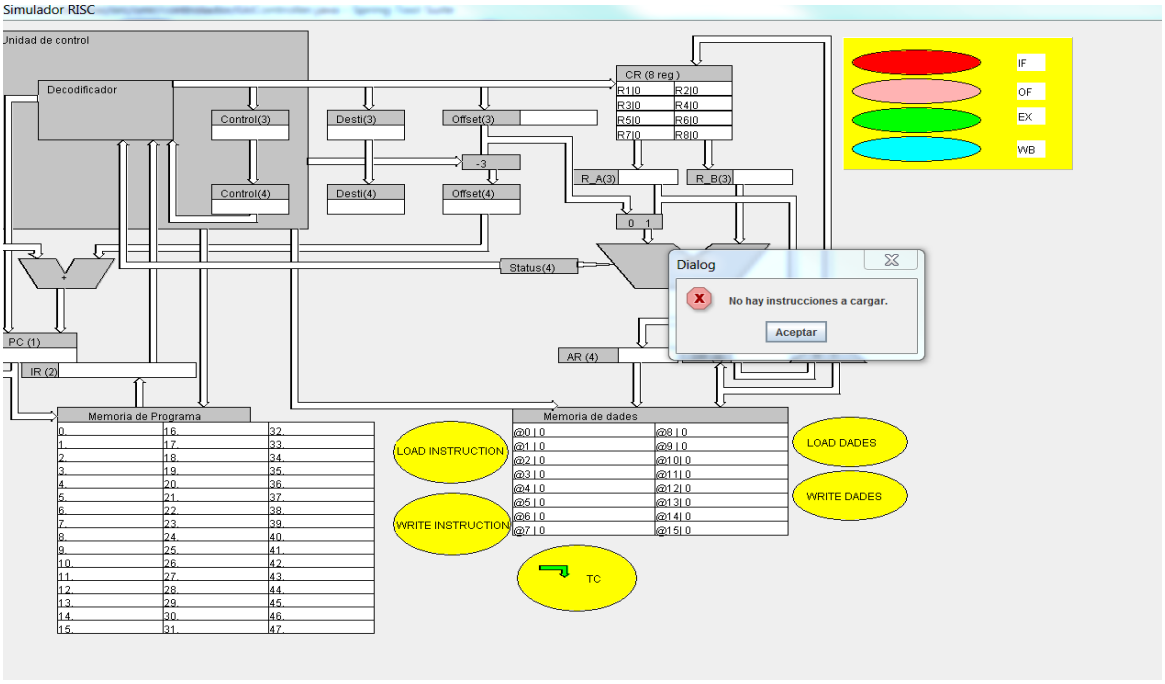
instruction = instruction.set(“NOOP”)
Desde i = 0 hasta i < 3
Inserta instruction a la listaInstrucciones
i = i + 1

Desde i = 0 hasta i < numeroInstrucciones
Imprime listaInstrucciones[i]
i = i + 1

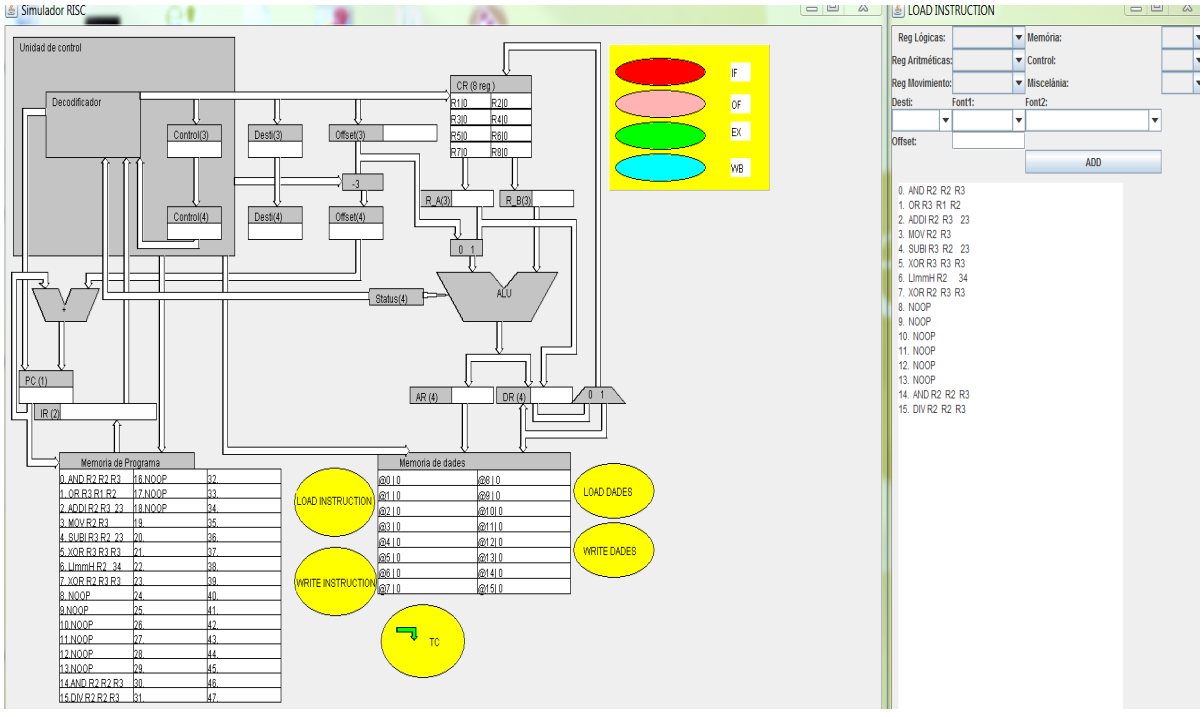
SiNo

Lanza un dialogo con el siguiente mensaje “No hay instrucciones a cargar”
Retorna a la interfaz grafica

Ejemplos:



Carga vacía de instrucciones (Figura 27)



Carga de instrucciones (Figura 28)

4.6. Inicializar memoria y conjunto de registros

-Al inicializar el simulador las direcciones de memoria y conjunto de registros deben de estar inicializadas a 0.

Desde i = 0 hasta i < numeroDireccionesMemoria

dato.setDireccion(i)

dato.setValor(0)

agregar dato a direccionesMemoria

i = i +1

Desde i = 1 hasta i < 9

Agrega al mapaRegistros “R” + i como llave y 0 como valor

i = i +1

Memoria de dades	
@0 | 0	@8 | 0
@1 | 0	@9 | 0
@2 | 0	@10 | 0
@3 | 0	@11 | 0
@4 | 0	@12 | 0
@5 | 0	@13 | 0
@6 | 0	@14 | 0
@7 | 0	@15 | 0

Memoria de datos(Figura 29)

CR (8 reg)	
R1|0	R2|0
R3|0	R4|0
R5|0	R6|0
R7|0	R8|0

Conjunto de registros(Figura 30)

4.7. Carga de la vista

-Carga de registros

Al inicializar el simulador se podrá visualizar una serie de registros , los cuales servirán como registros temporales que guardan resultados después de cada microinstrucción.

-Carga de botones

El mismo proceso se utilizara para los botones ya que estos tiene diferentes características se guardaran en otro mapa de datos.

-Carga de buses

El mismo proceso se utilizara para los buses ya que estos tiene diferentes características se guardaran en otro mapa de datos.

Para almacenar todos los componentes he usado HashMap para poder llamarlos por su llave y modificar sus atributos cuando sea necesario. Hay

cuatro HashMap para las vistas uno es para componentes , para los buses , para botones y registros.

5. Pruebas y resultados:

Ejemplos:

Programa con instrucciones de memoria, miscelánea y registro:

A continuación se cargara un pequeño programa en el cual podemos observar que la carga de valores a memoria se aplicaran en las instrucciones desde la dirección 0 → 7.
La carga desde memoria al conjunto de registros internos se hacen desde las direcciones 8 → 11.
Las operaciones aritméticas se hacen en las direcciones 12 → 14.
Y finalmente se guardaran los resultados la memoria de datos del simulador y podrán verse en la interfaz gráfica estas instrucciones se aplican en las direcciones 15 → 17.

Dirección	Instrucción	Que hace
0	LImmL R1 10	CR[R1[15..0]] ← 10
1	LImmL R2 20	CR[R2[15..0]] ← 20
2	LImmL R3 30	CR[R3[15..0]] ← 30
3	LImmL R4 40	CR[R4[15..0]] ← 40
4	STORE R1 R8 0	Mem[CR[R8]+0] ← CR[R1]
5	STORE R2 R8 1	Mem[CR[R8]+1] ← CR[R2]
6	STORE R3 R8 2	Mem[CR[R8]+2] ← CR[R3]
7	STORE R4 R8 3	Mem[CR[R8]+3] ← CR[R3]
8	LOAD R1 R8 0	CR[R1]← Mem[base + 0]
9	LOAD R2 R8 1	CR[R2]← Mem[base + 1]
10	LOAD R3 R8 2	CR[R3]← Mem[base + 2]
11	LOAD R4 R8 3	CR[R4]← Mem[base + 3]
12	ADD R5 R1 R2	CR[R5] ← CR[R1] + CR[R2] = 30
13	SUB R6 R2 R3	CR[R6] ← CR[R2] - CR[R3] = -10
14	ADD R7 R3 R4	CR[R5] ← CR[R1] + CR[R2] = 70
15	STORE R5 R8 10	Mem[CR[R8]+10] ← CR[R5] = @10 ← 30
16	STORE R6 R8 11	Mem[CR[R8]+11] ← CR[R6] = @11 ← -10

17	STORE R7 R8 12	Mem[CR[R8]+12] ← CR[R7] = @12 ← 70

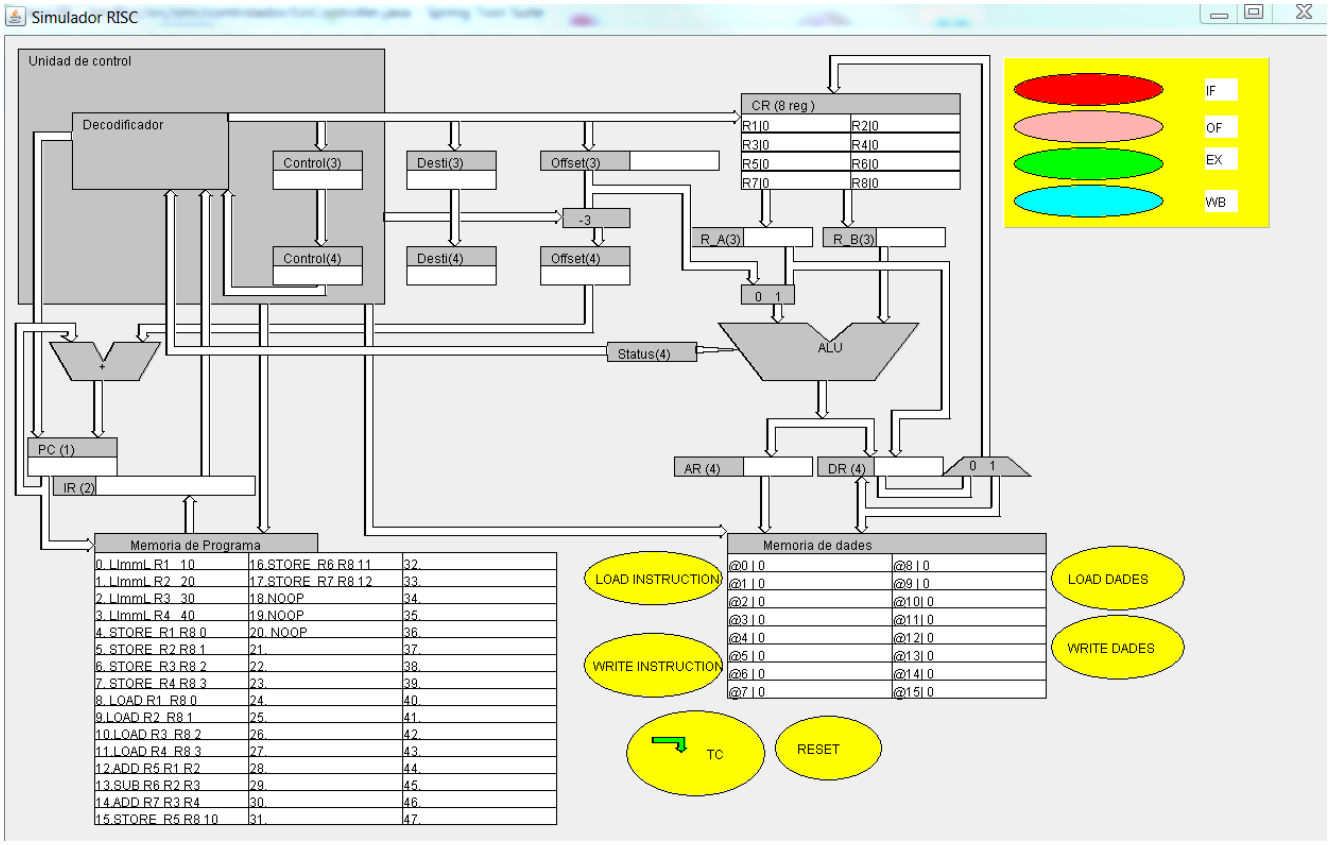
Programa inicial:

X = A + B

Y = B – C

Z = C + D

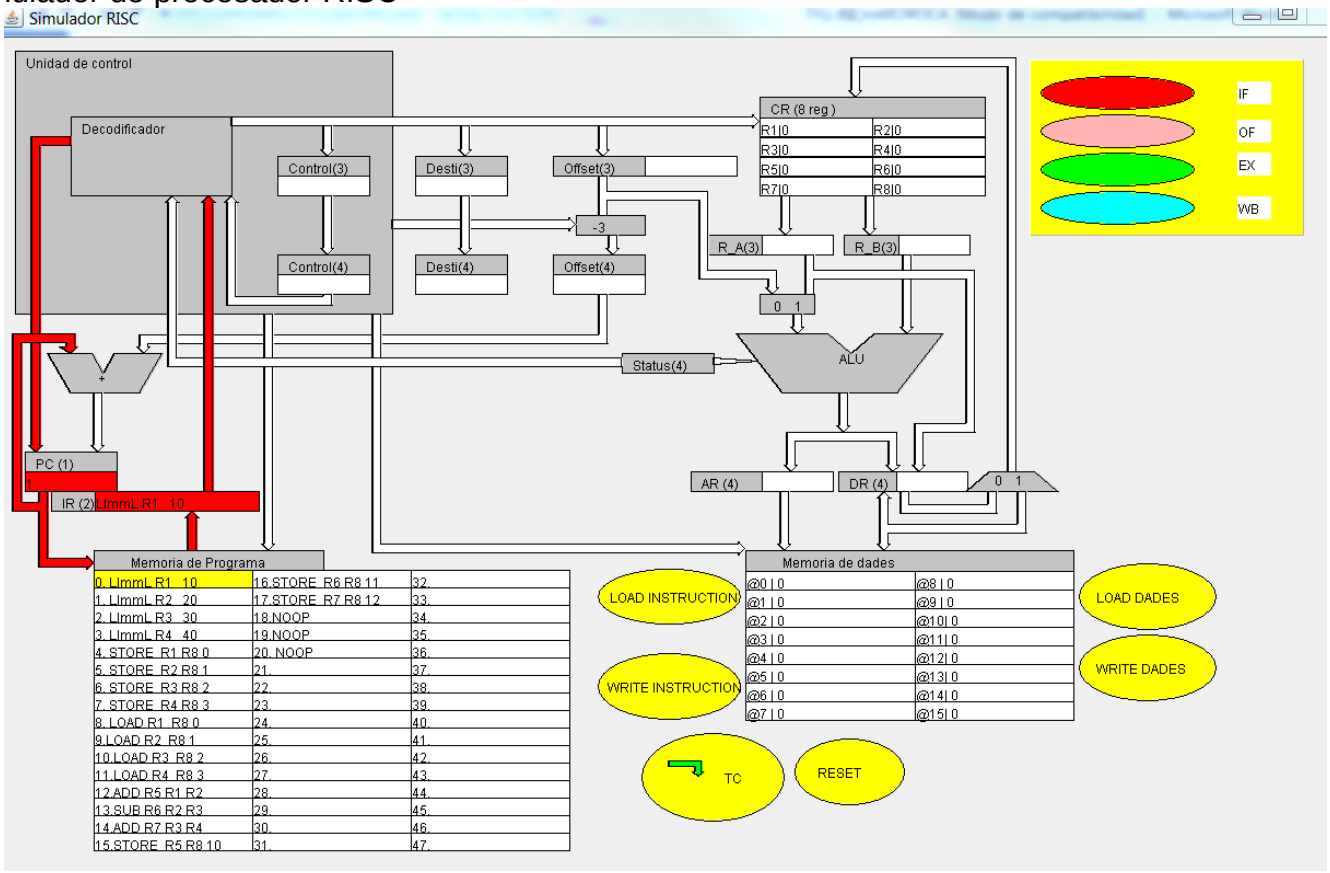
Donde X, Y, Z indican direcciones de memoria A, B, C, D los registros de propósito general.



Programa inicial operaciones aritméticas (Figura 31)

Etapa IF (Instruction Fetch):

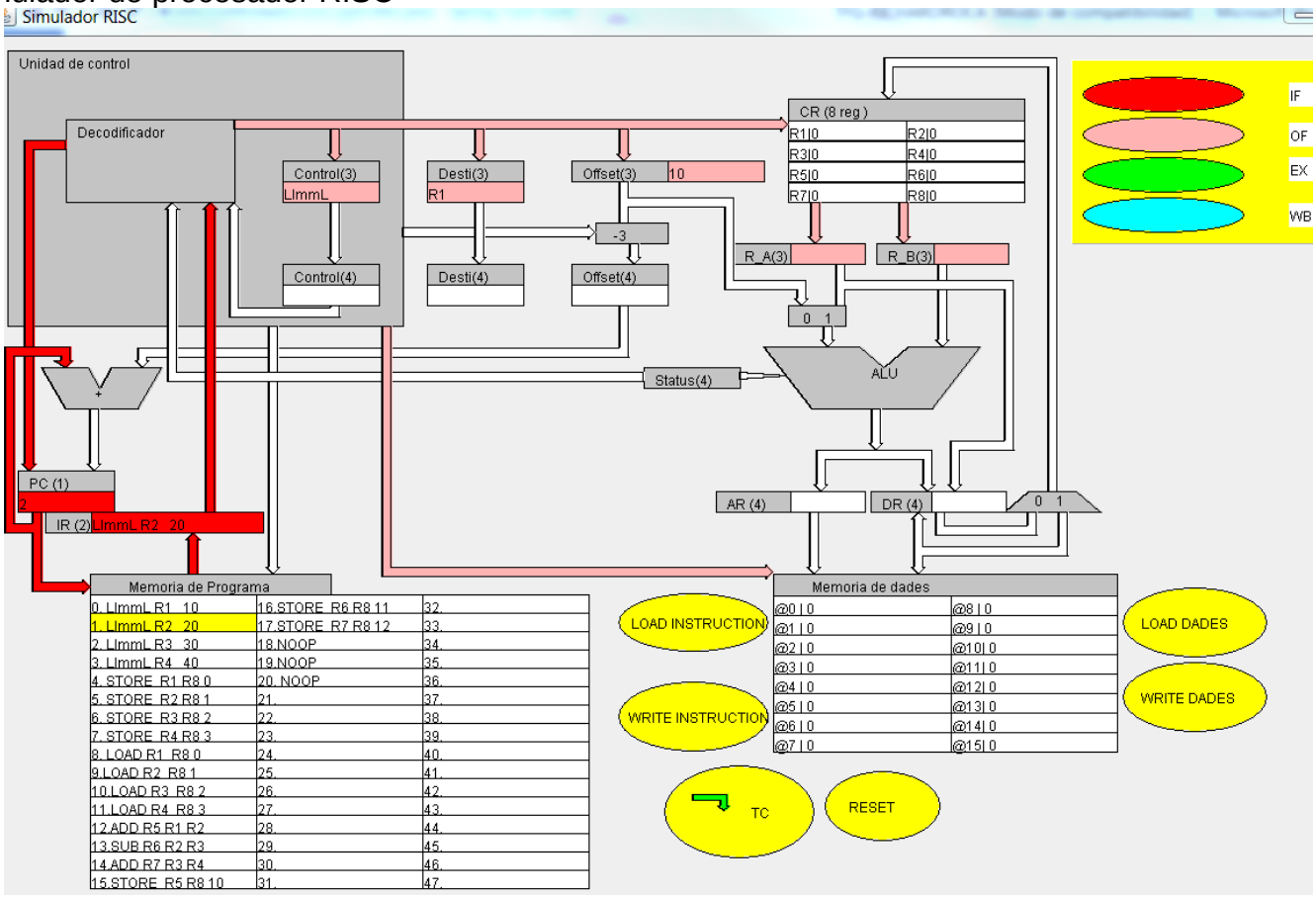
Instrucción LImmL R1 10



Programa operaciones aritméticas etapa IF(Figura 32)

Etapa OF (Operand Fetch):

Instrucción LImmL R1 10

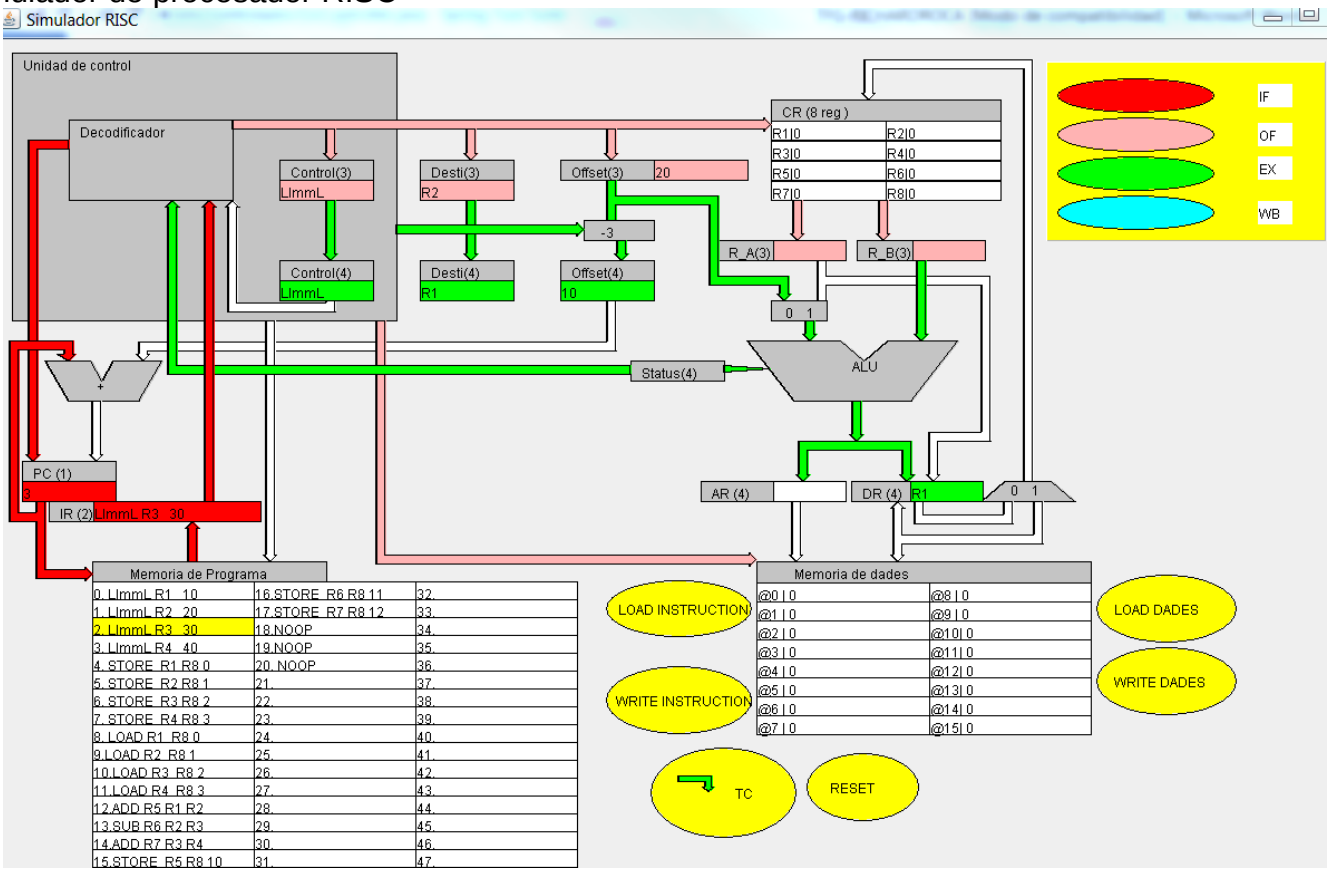


Programa operaciones aritméticas etapa OF(Figura 33)

Etapa EX (Execute):

Instrucción LImmL R1 10

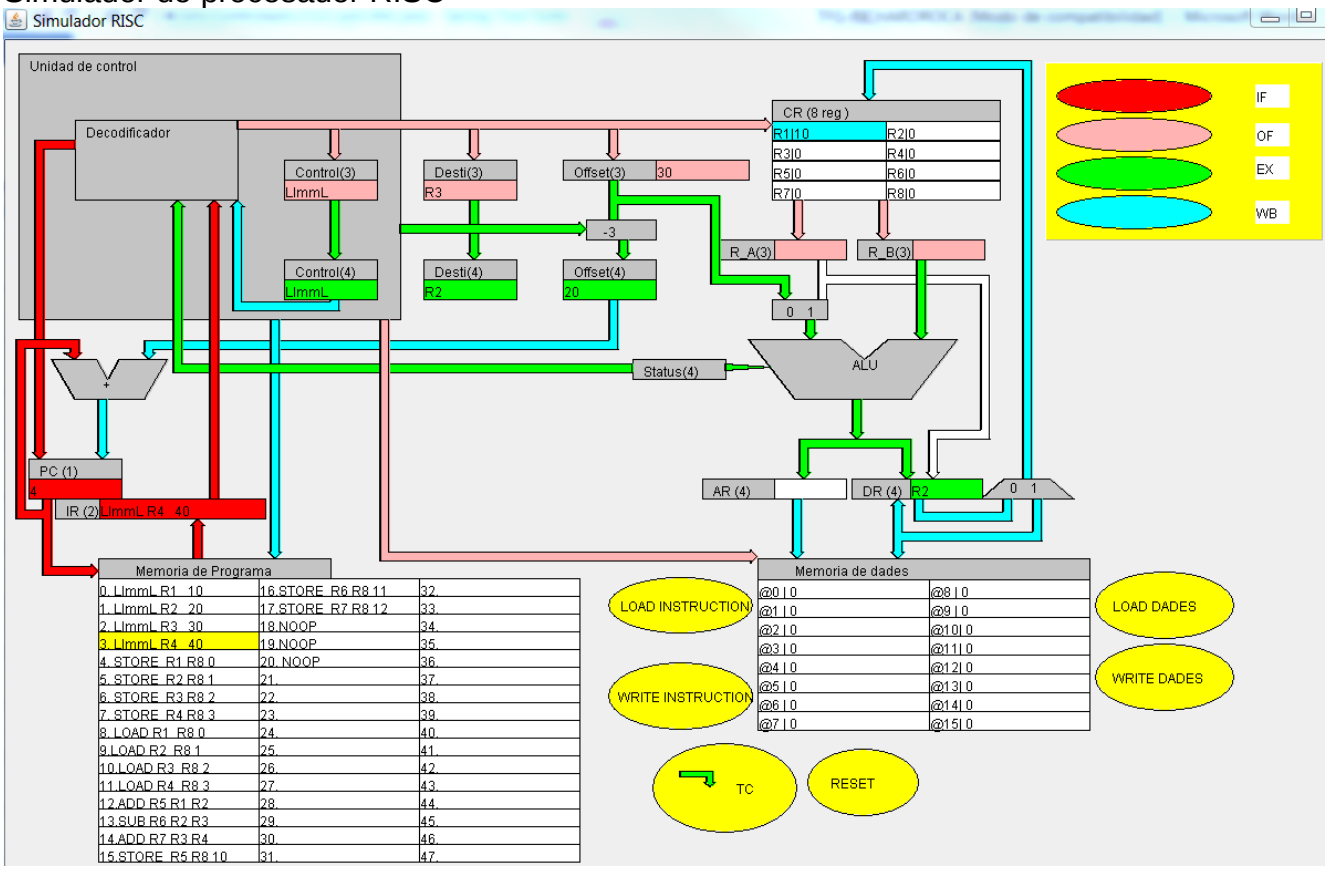
Simulador de procesador RISC



Programa operaciones aritméticas etapa EX(Figura 33)

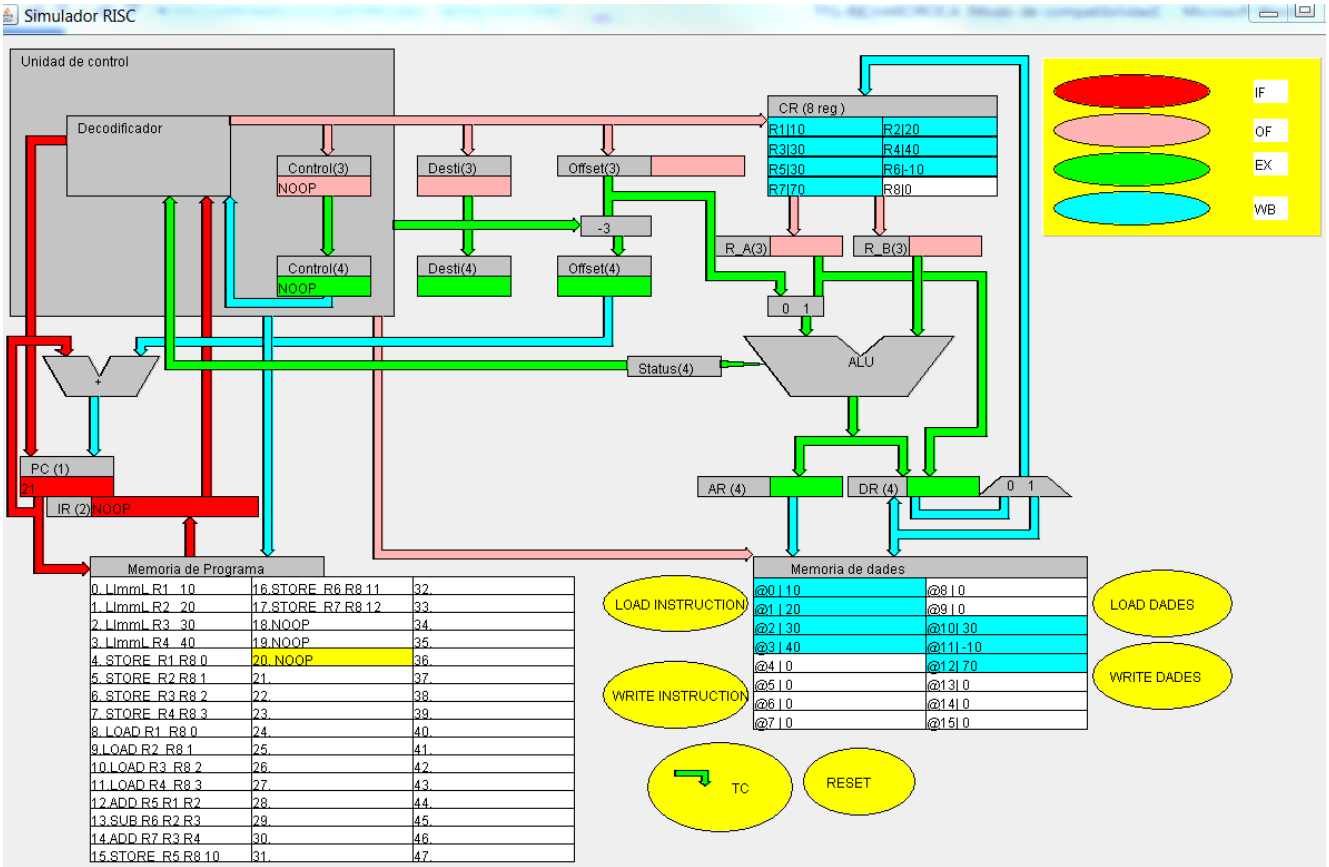
Etapa WB (Write back):

Instrucción LImmL R1 10



Programa operaciones aritméticas etapa WB(Figura 34)

Fin del programa:



Programa operaciones aritméticas final(Figura 34)

Programa con instrucciones de memoria, registro ,control y miscelánea :

A continuación se cargara un pequeño programa que calcula máximo y mínimo entre 2 valores y se mostrara cuando cumple la condición y cuando no la cumple, el valor máximo se guardara en R7 y el mínimo en R8:

Programa:

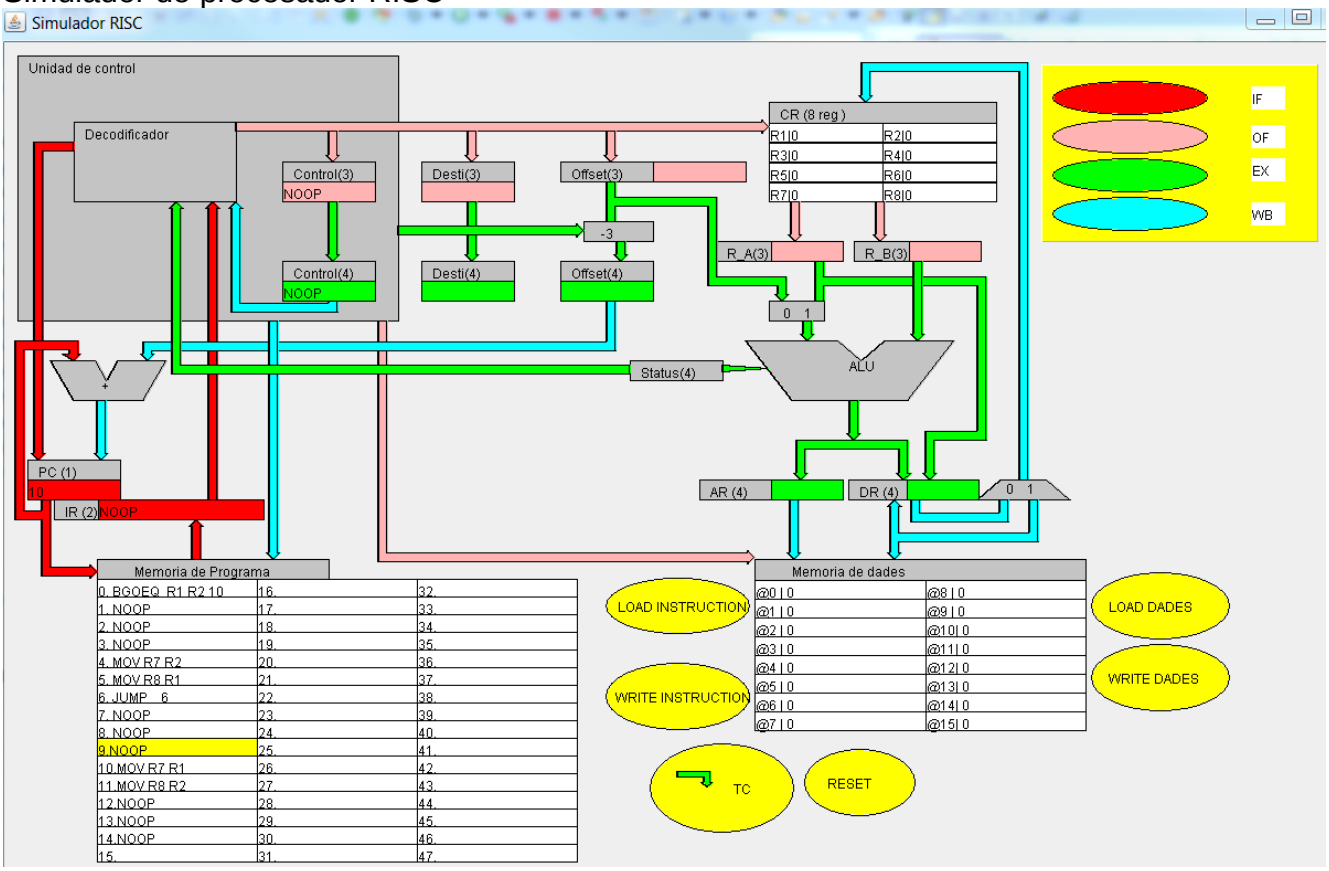
Si R1 >= R2 se cumple
Guarda en R7 ← R1 y
Guarda en R8 ← R2

Sí No se cumple la condición
Guarda en R7 ← R2
Guarda en R8 ← R1

Dirección	Instrucción	Que hace
0	BGOEQ R1 R2 10	Compara los 2 valores
1	NOOP	No hace nada
2	NOOP	No hace nada
3	NOOP	No hace nada
4	MOV R7 R2	No se cumplió la condición R2 es el máximo
5	MOV R8 R1	No se cumplió la condición R1 es el mínimo
6	JUMP 6	Salta a final, a la posición
7	NOOP	No hace nada
8	NOOP	No hace nada
9	NOOP	No hace nada
10	MOV R7 R1	Si se cumple la condición R1 es el máximo
11	MOV R8 R2	Si se cumple la condición R2 es el mínimo

Se cumple la condición ya que inicialmente los registros tienen el valor 0 y al hacer la comprobación Si R1 >= R2 salta a la dirección 10 puesto que ambos valen 0.

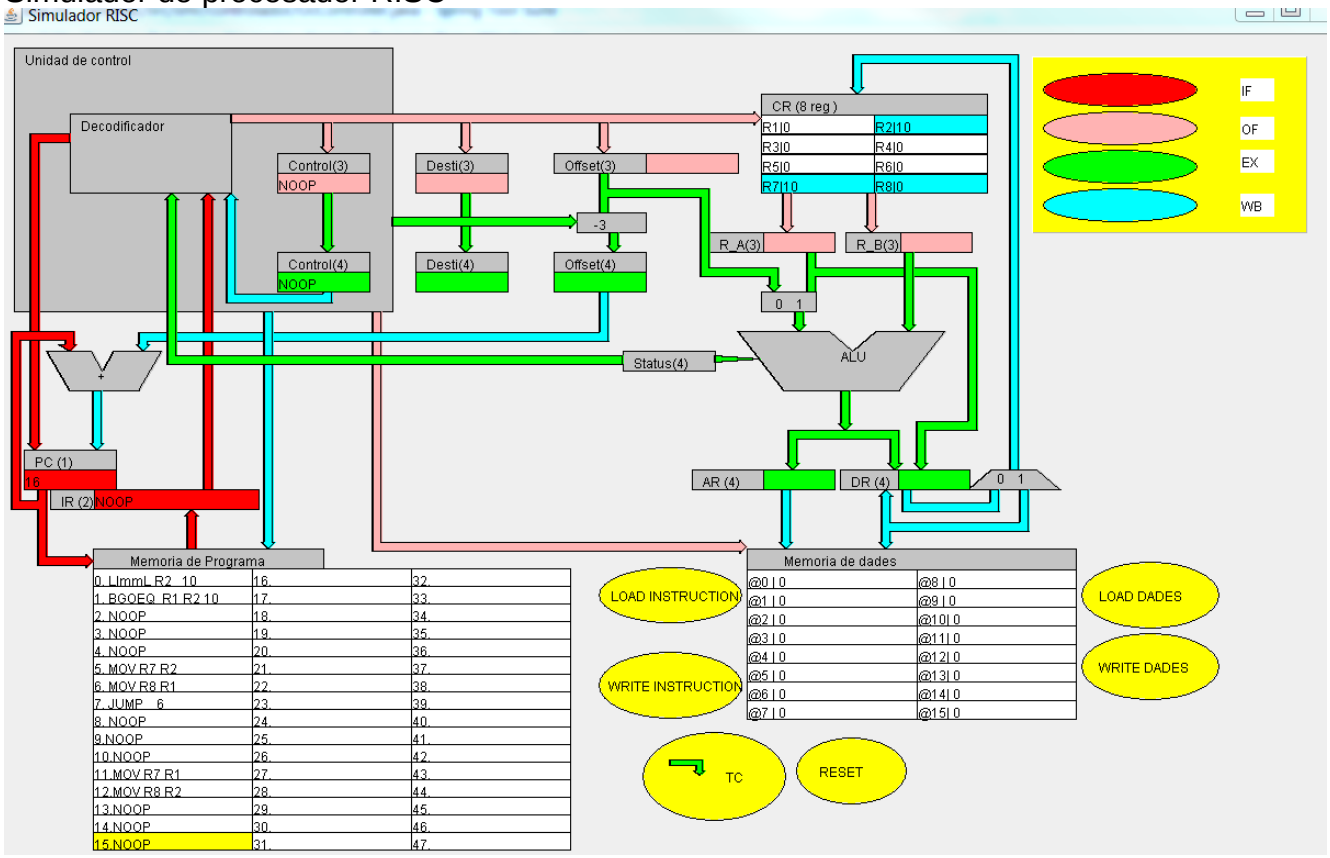
Se cumple la condición:



Programa operaciones de salto BGOEQ(Figura 35)

No cumple la condición previamente se ha agregado una instrucción para cargar un valor en R2 y no se cumpla la condición. No hay salto a 10 solo se cumple el salto inmediato JUMP 6 R2 es el máximo y R1 es el mínimo.

No se cumple condición:



Programa operaciones de salto JUMP(Figura 36)

6. Conclusiones y trabajos futuros

- **Conclusiones**

Se pudo cumplir con todos los objetivos planteados al inicio del proyecto.

Se ha podido conseguir que el interfaz se ha idéntico al que se da en clases de teoría así como el conjunto de instrucciones.

El simulador está escrito en lenguaje Java funciona para cualquier sistema operativo que tenga java instalado.

Las etapas del pipeline se pueden visualizar por medio de diferentes colores, pulsando el boton TC se ejecutara instrucción a instrucción de la memoria de programa.

También se pudo controlar el formato de instrucción a la hora de introducir el programa a probar.

Poder introducir programas pequeños máximo de 45 instrucciones ya que esta es la cantidad que se pueden mostrar en la interfaz.

- La posibilidad de ejecutar el programa nuevamente una vez se haya ejecutado(reset del programa).

Se cumplió con el objetivo de guardar los Programas introducidos con el simulador en un fichero de texto para poder cargar nuevamente el programa.

Se cumplió también con el objetivo de abrir fichero con el programa y ejecutarlo.

Se cumplió con el objetivo de agregar un botón que tenga la función de iniciar el programa nuevamente después de haberlo ejecutado.

- **Futuras mejoras**

Algunas mejoras posibles que se han podido observar son :

-Que al cargar las instrucciones a memoria de programa vaya cargándose unas cuantas , mientras estas se van ejecutando se vayan borrando de la memoria de programa y se carguen las nuevas así ya no haría falta un límite de instrucciones.

-Se podría agregar un botón más en la ventana LOAD INSTRUCTION que pueda borrar la instrucción seleccionada, para no tener que cambiar todo el programa.

-El botón RESET reinicia el programa esto se podría ampliar de tal forma que se pueda introducir un número y no tenga que iniciar desde 0 sino a la posición que se introduzca.

-Podría agregarse un compilador al simulador para que compile las instrucciones antes de cargarlas al simulador.

7. Bibliografía:

http://www.tucows.com/preview/31786/Klogic

https://es.wikipedia.org/wiki/Reduced_instruction_set_computing

http://www.azc.uam.mx/publicaciones

https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento#Operaci.C3.B3n