Diseño e implementación de un simulador software basado en el procesador MIPS32

Manuel Rivas Pérez[1], Manuel Domínguez Morales[1], Francisco Gómez Rodríguez[1], Alejandro Linares Barranco[1], Gabriel Jiménez Moreno[1], Antón Civit Balcells[1]

[1] Departamento de Arquitectura y Tecnología de Computadores
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla
Avenida Reina Mercedes s/n
Sevilla, España
{mrivas, mdominguez, gomezroz, alinares, gaji, civit}@atc.us.es

Resumen. La arquitectura de computadores es una asignatura de gran importancia actualmente en las titulaciones de Informática. Pero en muchas ocasiones, los estudiantes tienen problemas para comprender la materia debido a la falta de herramientas que muestren el funcionamiento de los componentes internos de la arquitectura de los computadores de manera fácil e intuitiva. En este trabajo se expone un simulador del procesador MIPS32 desarrollado en .NET que puede ser controlado a través de línea de comandos o desde una interfaz gráfica versátil e intuitiva para facilitar a los alumnos el estudio de la arquitectura de los procesadores segmentados. La interfaz gráfica ofrece un entorno de desarrollo integrado en el que editar y ensamblar los programas, así como mostrar el funcionamiento del procesador a través de sus registros, memoria, pipeline y el cronograma de ejecución. En este trabajo se expondrá un simulador como producto que responde a las necesidades de los alumnos en asignaturas relacionadas con el estudio de la arquitectura de los computadores. En primer lugar se expondrá una comparativa de simuladores MIPS, posteriormente se mostrarán las características del procesador que se simula, se describirá la implementación del ensamblador y del propio simulador y finalmente se mostrará su funcionamiento a través de la interfaz gráfica desarrollada denominada VisualMips32.

Palabras Clave: MIPS32, simulador, segmentación, .NET, arquitectura de computadores.

Abstract. Nowadays, computer architecture is a very important subject in Computer Science degrees. But often, students have problems understanding the topic due to the lack of tools to show the behavior of the internal computer architecture components in an easy and intuitive way. This paper presents a MIPS32 processor simulator developed in .NET that can be controlled via command line orders or using a versatile and intuitive graphical interface that makes the study of segmented processor architecture easier. The GUI (graphical user interface) offers an integrated development environment, where assembly programs can be assembled and run, in addition to being able to watch a step-

by-step execution through its registers, memory, pipeline and execution chronogram. This paper presents a simulator as a software tool developed to meet the students' needs in subjects related to the computer architecture. First, a comparative of several MIPS simulators is shown. After that, the implementation of the assembler and the simulator itself will be described; and, finally, its operation is displayed through the developed graphical interface, called VisualMips32.

Keywords: MIPS32, simulator, segmentation, .NET, computer architecture.

1 Introducción

La electrónica digital ha evolucionado rápidamente en muy poco tiempo llegando a integrar más de mil millonesde transistores en un mismo chip. La arquitectura también ha evolucionado con nuevas estructuras como la segmentación, el uso de cachés, procesamiento multi-hilo o el multiprocesamiento ayudados por este aumento de la integración de los transistores. El estudio de la arquitectura de los computadores es de gran importancia en las titulaciones universitarias relacionadas con la informática. Estas arquitecturas, fruto de su evolución, pueden resultar complejas lo que provoca que muchos estudiantes encuentren dificultades para comprender algunos aspectos de la asignatura de arquitectura de computadores, como la segmentación de cauce (*pipelining*) o el cálculo del rendimiento del procesador. La segmentación es una técnica muy importante en arquitectura de computadores en la que varias instrucciones pueden ejecutarse simultáneamente manteniendo cada una de ellas en una etapa diferente de la ruta de datos (*datapath*). El uso de procesadores ejemplo ayuda al alumno a comprender los conceptos clave de la arquitectura y las mejoras de los procesadores. Este es el caso del procesador MIPS descrito en los libros de Henessy y Patterson [1] [2] ampliamente conocidos por la comunidad universitaria, que han convertido a este procesador de ejemplo en uno de los más utilizados para enseñar la asignatura de arquitectura de computadores en las universidades [3].

Pero aun así, estos contenidos son difíciles de asimilar si se exponen únicamente haciendo uso de la pizarra o con lápiz y papel. Es por ello que muchos los profesores emplean simuladores gráficos como herramienta para mostrar los conceptos de forma intuitiva e interactiva simplificando la forma en la que los profesores enseñan y los alumnos aprenden la arquitectura de computadores. Hay numerosos artículos que tratan sobre el aprendizaje de la arquitectura de computadores [4–7] y la mayoría defienden la importancia de los simuladores para el aprendizaje. De hecho estos simuladores se han convertido en una herramienta indispensable para que los alumnos puedan entender las complejas arquitecturas difíciles de asimilar sin la interfaz gráfica que los simuladores actuales pueden ofrecer [8].

Hay una gran variedad de simuladores actualmente, desde los más simples hasta complejos simuladores utilizados en investigación y que son analizados en [9]. Algunos dirigidos a simular los componentes hardware y otros la arquitectura del juego de instrucciones cada uno de ellos con distinto nivel de detalle desde simplemente modelar el comportamiento externo a mostrar las interacciones de los componentes internos. A pesar de haber gran variedad de ellos, algunos no son

adecuados para el aprendizaje ya que son difíciles de usar y comprender, o bien son demasiado específicos [6].

En este trabajo se presenta un software que simula y visualiza el procesamiento de instrucciones del procesador segmentado MIPS para mejorar la calidad de la enseñanza y dar a los estudiantes un entorno en el que experimentar. Este trabajo se organiza de la siguiente forma: En la sección 2 se exponen los requerimientos necesarios para desarrollar el simulador académico adecuado. La sección 3 describe muchos de los simuladores que existen y se especifican los aspectos más relevantes de cada uno de ellos. La sección 4 muestra las características del procesador MIPS que se ha simulado. Posteriormente se describe la implementación del ensamblador y del simulador en la sección 5, el funcionamiento del entorno visual VisualMIPS32 en la sección 6 y, terminando el presente trabajo, con las mejoras futuras y las conclusiones.

2 Objetivos

Para conseguir el objetivo principal de obtener una herramienta que facilite la enseñanza y el aprendizaje de la arquitectura de computadores, el desarrollo de este software se basa en los siguientes requerimientos:

- Permitir reconfigurar y parametrizar el procesador.
- Realizar ejecución paso a paso y control de puntos de interrupción (*Breakpoints*).
- Simular la versión monociclo, multiciclo y segmentada del procesador.
- Ofrecer una interfaz gráfica intuitiva, clara y fácil de usar.
- Integrar un editor de ensamblador en el entorno de desarrollo.
- Incluir un amplio número de instrucciones del repertorio tanto de enteros como de coma flotante además de pseudo-instrucciones.
- Mostrar información clara y detallada sobre la codificación de cada instrucción, la segmentaciónde las instrucciones, los tipos de bloqueos que se producen, la activación de desvíos (*bypasses*) y las estadísticas sobre el rendimiento obtenido.

3 Otros simuladores

Como se indicó anteriormente, existen actualmente muchas herramientas para simular el funcionamiento de un procesador y de muy diversos tipos, algunas de ellas poco adecuadas para la enseñanza debido a su dificultad o por ser muy específicas [7]. A continuación se describen los simuladores orientados a la enseñanza más conocidos especificando algunos puntos a favor o en contra de ellas:

- *QtSpim* [10, 11]. Es un simulador de MIPS32 programado en C++ y Qt que fue muy utilizado tanto en educación como en la industria [12]. Este

simulador implementa un amplio repertorio de instrucciones incluyendo algunas en coma flotante [13]. Es una buena herramienta de depuración y es intuitiva pero sólo simula la versión del procesador monociclo.

- *MARS* [12, 14]. Está escrito en java y se utiliza de muchas universidades de todo el mundo. Simula la ejecución del programa ensamblador paso a paso y muestra el resultado de los registros y de la memoria. Su entorno de desarrollo integra un editor capaz de mostrar la sintaxis en una ventana emergente durante la programación. Es una buena herramienta de simulación y depuración aunque, al igual que QtSpim, sólo es capaz de simular la versión monociclo del MIPS. Una de las características más destacable de este simulador es que admite la inclusión de plug-ins lo que permite ampliar su funcionalidad. Buen ejemplo de ello es el MIPS X-Ray [15], un plug-in que representa gráficamente la ruta de datos del procesador.
- *ProcessorSim* [16]. Es una herramienta desarrollada en java que simula el MIPS R2000 monociclo. Este simulador incluye varias rutas de datos diferentes, e incluso el usuario puede crear más, que se muestran gráficamente en forma de animación. Respecto a la animación, un problema del que adolece es que sólo un componente puede enviar un mensaje en cada instante de tiempo cuando en realidad todos los componentes deberían trabajar concurrentemente [17]. Otras de sus debilidades son que no muestra ruta de datos de la versión segmentada, que su editor es muy simple y que el juego de instrucciones que implementa es muy reducido.
- *MIPS-Datapath* [18]. Es un simulador desarrollado en C++ que muestra la ruta de datos gráficamente tanto de la versión monociclo como de la segmentada, con y sin adelantamientos. Permite la ejecución paso a paso, pero posee un juego de instrucciones muy reducido y no soporta el bloqueo del cauce.
- *WebMIPS* [19]. Este simulador fue programado en ASP y simula el MIPS segmentado con detección de riesgos. Es un simulador educativo interesante ya que se ejecuta desde un explorador web por lo que no precisa instalación. Es capaz de ejecutarse paso a paso mostrando la ruta de datos gráficamente. Es un buen simulador educativo pero no implementa la versión monociclo, su editor es muy simple y la representación de la ruta de datos es estática.
- *EduMIPS64* [20, 21]. Esta herramienta es una re-implementación del simulador WinMips64 [22][23]. Simula el procesador MIPS64 segmentado, posee una interfaz muy intuitiva, posee un amplio repertorio de instrucciones que incluye operaciones en punto flotante, llamadas al sistema e implementa la detección de bloqueos. Por el contrario, no posee un editor integrado, no soporta simulación monociclo ni muestra la ruta de datos.
- *MiniMIPS* [24]. Es un simulador implementado en C para Unix. Muestra la ruta de datos durante la simulación pero no es a nivel RT ni es animada y el único dato de rendimiento que ofrece es el número de ciclos.
- *DrMips* [25]. Es un simulador de MIPS32 implementado en java que visualiza paso a paso la ruta de datos incluyendo la latencia de los componentes cuando usa el modo performance. Muestra varias rutas de datos basadas en libros de Henessy y Patterson [1, 2] incluyendo la ruta de datos monociclo y segmentada. Posee un editor que muestra la sintaxis en un menú

emergente. Su característica más destacable es que posee una versión para la plataforma Android.

En [19] se muestra una tabla que compara varios simuladores MIPS teniendo en cuenta si implementan el MIPS de Hennessy y Patterson [1, 2] a nivel RT, si muestran datos estadísticos del rendimiento, si realizan animaciones en tiempo de simulación que muestren cómo se transfieren las señales o si emplean una plataforma independiente que los haga más portables. En [25] también se expone una tabla comparativa muy completa de muchos de los simuladores anteriormente citados teniendo en cuenta muchos aspectos como, si integra un editor, si implementa instrucciones en coma flotante, si la ruta de datos es configurable, si es segmentado, o si permite la edición de datos durante la simulación.

4 Características del procesador simulado: MIPS32

MIPS (acrónimo de Microprocessor without Interlocked Pipeline Stages) es un procesador con un juego reducido de instrucciones (RISC) desarrollado por MIPS Computer Systems, que pasó a llamarse MIPS technologies y que ha sido vendida a Imagination technologies.

Los procesadores MIPS forman parte de una gran familia de procesadores RISC desde su primer procesador R2000 en el 1986 cuyo juego de instrucciones ha evolucionado desde la versión original denominada MIPS I hasta las actuales MIPS32 y MIPS64.

Gracias a la sencillez de su arquitectura y al hecho de que actualmente sea bastante utilizado en sistemas embebidos, lo ha convertido en un procesador muy adecuado para el estudio de la arquitectura de computadores en las universidades.

El simulador presentado en este trabajo está basado en el procesador MIPS32 cuya arquitectura ISA puede consultarse en [26, 27, 28]. Las características principales de este procesador son:

- Procesador RISC segmentado de 32 bits y direcciones de 32 bits
- Instrucciones de tamaño fijo para una rápida decodificación de las instrucciones.
- Almacenamiento en memoria en *big-endian* o *little-endian*.
- 32 registros de propósito general de 32 bits para enteros.
- Unidades funcionales independientes para el producto y la división. Estas unidades funcionales contienen además los registros HI y LO utilizados para almacenar el resultado de operaciones (véase Figura 1).
- Coprocesador de coma flotante. Este coprocesador es identificado como coprocesador 1 y posee un banco de 32 registros de 32 bits (Figura 1) que pueden emplearse para almacenar números en coma flotante de simple precisión (32 bits) o bien pueden agruparse por pares para almacenar números en coma flotante de doble precisión (64 bits).
- Unidad de detección y control de riesgos.
- Implementa adelantamientos (*forwarding*) para reducir los bloqueos de datos.
- Saltos retardaos para reducir los bloqueos de control.

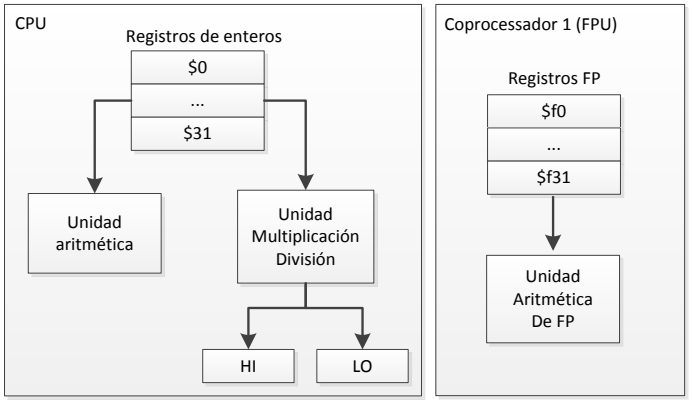

Figura 1. Unidades de procesamiento del MIPS32

Todas estas características han sido implementadas en el procesador siendo configurables la utilización de adelantamientos, los saltos retardados, el número de ciclos que duran las etapas y cuáles de ellas son segmentadas (ver Figuras 11-12).

4.1 Modos de direccionamiento

Las arquitecturas RISC admiten un conjunto muy reducido de modos de direccionamiento y todos ellos han sido implementados en el simulador. Los modos de direccionamiento son:

- Registro. El operando está en un registro del banco de registros de enteros o de coma flotante según el tipo de instrucción. Para especificar los registros de enteros en ensamblador pueden utilizarse indistintamente dos nomenclaturas. La nomenclatura empleada para cada registro aparece en la Tabla 1 y también se muestra el uso al que va destinado cada registro. Una nomenclatura consiste en emplear el símbolo '$' seguido del número del registro (por ejemplo *add $1, $2, $3*) y otra utiliza el símbolo '$' seguido de una palabra que define su utilización, por ejemplo *add $at, $v0, $v1*.

Tabla 1. Nomenclatura y uso de los registros de enteros del procesador.

Nombre	Registro	Uso
$zero	0	Constante 0
$at	1	Reservado para el ensamblador
$v0-$v1	2-3	Valores de retorno
$a0-$a3	4-7	Argumentos
$t0-$t7	8-15	Temporales
$s0-$s7	16-23	Valores Guardados
$t8-$t9	24-25	Temporales
$k0-$k1	16-27	Reservados para el kernel
$gp	28	Puntero Global
$sp	29	Puntero de Pila
$fp	30	Puntero de Marco
$ra	31	Dirección de retorno

- Inmediato. El operando es una constante de 16 bits almacenada en la propia instrucción, por ejemplo *addi $1,$2, 2000*.
- Registro base más desplazamiento. Direccionamiento a memoria en el que la dirección efectiva viene determinada por el contenido de un registro denominado registro base más una constante que actúa de desplazamiento. Este modo de direccionamiento sólo se usa en instrucciones de acceso a memoria (cargas y almacenamientos). Su sintaxis es **inm(reg)**, siendo *reg* el registro base e *inm* el desplazamiento, por ejemplo *lw $1, 24($2)*.
- Relativo al Contador de Programa. Se emplea exclusivamente en los saltos condicionales y la dirección destino de salto se obtiene sumando una constante de 16 bits en complemento a 2 al registro contador de programa. Este modo de direccionamiento queda oculto en el ensamblador gracias al uso de etiquetas. Un ejemplo de este modo sería *beqz $1, $2, etiqsigue*.
- Pseudodirecto: Utiliza una constante para especificar los 26 bits menos significativos de la dirección. Los 6 bits más significativos que faltan, de los 32 bits que forman la dirección, se toman del registro PC. Es por ello por lo que su nombre contiene el prefijo pseudo. Este modo se emplea exclusivamente para indicar la dirección de salto en las instrucciones de salto incondicional sin registro (instrucciones *j* y *jal*). Al igual que en el modo de direccionamiento anterior, el editor oculta este detalle mediante etiquetas (por ejemplo *j sigue*).

4.2 Repertorio de instrucciones

De todo el repertorio de instrucciones de la arquitectura MIPS32, formado por algo más de 200 instrucciones incluyendo las que operan en coma flotante, este simulador implementa 80 de ellas que son las que comúnmente se encuentran en la mayoría de los programas en ensamblador.

Las tablas 2a-2d muestran el repertorio de las instrucciones implementadas en el simulador clasificadas según el tipo de operación que realizan.

Tabla 2a. Instrucciones de carga y almacenamiento.

Instrucción	Descripción
lb reg,inm16(reg)	Carga byte con signo en reg. entero desde mem (extiende signo).
lbu reg,inm16(reg)	Carga byte sin signo en reg. entero desde mem (extiende ceros).
lh reg,inm16(reg)	Carga 16 bits con signo en reg. entero desde mem. (extiende signo)
lhu reg,inm16(reg)	Carga 16 bits sin signo en reg. entero desde mem. (extiende ceros)
lw reg,inm16(reg)	Carga 32 bits en reg. entero desde mem.
sb reg,inm16(reg)	Guarda byte en mem. desde reg. entero
sh reg,inm16(reg)	Guarda 16 bits en mem. desde reg. entero
sw reg,inm16(reg)	Guarda 32 bits en mem. desde reg. entero
lwc1 freg,inm16(reg)	Carga 32 bits en reg. FP simple prec. desde mem.
ldc1 freg,inm16(reg)	Carga 64 bits en reg. FP doble prec. desde mem.
swc1 freg,inm16(reg)	Guarda 32 bits en mem. desde reg. FP
sdc1 freg,inm16(reg)	Guarda 64 bits en mem. desde reg FP doble prec.
mfc1 reg,freg	Carga 32 bits en reg. entero desde reg. FP

mtc1 reg,freg	Carga 32 bits en reg. FP desde reg. entero
mflo reg	Carga 32 bits en reg. entero desde reg. LO
mfhi reg	Carga 32 bits en reg. entero desde reg. HI
lui reg,inm16	Carga inm de 16 bits en parte alta de reg. entero
li reg,imm32	Carga inm de 32 bits en reg. entero (pseudo-instrucción).
la reg,etiq	Carga en reg. entero la dirección de una etiqueta

Tabla 2b. Instrucciones de operaciones aritméticas y lógicas con enteros.

Instrucción	Descripción
add reg,reg,reg	Suma entre registros de enteros
addu reg,reg,reg	Suma entre registros de enteros sin desbordamiento
addi reg,reg,inm16	Suma de enteros con inmediato
addiu reg,reg,inm16	Suma de enteros con inmediato sin desbordamiento
sub reg,reg,reg	Resta entre registros de enteros
subu reg,reg,reg	Resta entre registros de enteros sin desbordamiento
mul reg,reg,reg	Multiplicación de enteros con signo.
mult reg,reg	Multiplicación de enteros con signo de 32 bits.
multu reg,reg	Multiplicación de enteros sin signo de 32 bits.
div reg,reg	División de enteros con signo.
divu reg,reg,reg	División de enteros sin signo.
and reg,reg,reg	Operación AND entre registros
andi reg,reg,inm16	Operación AND entre registro e inmediato
nor reg,reg,reg	Operación NOR entre registros.
or reg,reg,reg	Operación OR entre registros.
ori reg,reg,inm16	Operación OR entre registro e inmediato
xor reg,reg,reg	Operación OR exclusiva entre registros.
xori reg,reg,inm16	Operación OR exclusiva entre registro e inmediato
sll reg,reg,inm16	Desplazamiento lógico a la izquierda
srl reg,reg,inm16	Desplazamiento lógico a la derecha
sra reg,reg,inm16	Desplazamiento aritmético a la derecha
sllv reg,reg,reg	Desplazamiento lógico a la izquierda variable
srlv reg,reg,reg	Desplazamiento lógico a la derecha variable
srav reg,reg,reg	Desplazamiento aritmético a la derecha variable
rol reg,reg,reg	Rotación a la izquierda variable
ror reg,reg,reg	Rotación a la derecha variable
slt reg,reg,reg	Comparación “menor que” entre reg. con signo
sltu reg,reg,reg	Comparación “menor que” entre reg. sin signo
slti reg,reg,inm16	Comparación “menor que” con inm y con signo
sltiu reg,reg,inm16	Comparación “menor que” con inm sin signo
nop	No operación.

Tabla 2c. Instrucciones de operaciones aritméticas y lógicas en coma flotante.

Instrucción	Descripción
add.d freg,freg,freg	Suma en FP de simple precisión
sub.d freg,freg,freg	Resta en FP de simple precisión
mul.d freg,freg,freg	Multiplicación en FP de simple precisión
div.d freg,freg,freg	División en FP de simple precisión
abs.d freg,freg	Valor absoluto en FP de simple precisión
add.d freg,freg,freg	Suma en FP de doble precisión
sub.d freg,freg,freg	Resta en FP de doble precisión
mul.d freg,freg,freg	Multiplicación en FP de doble precisión

div.d freg,freg,freg	División en FP de doble precisión
abs.d freg,freg	Valor absoluto en FP de doble precisión

Tabla 2d. Instrucciones de control.

Instrucción	Descripción
beq reg,reg,etiq	Salto rel. a PC si ambos registros son iguales
bne reg,reg,etiq	Salto rel. a PC si ambos registros son distintos
beqz reg,etiq	Salto rel. a PC si el registro es igual a "0"
bnez reg,etiq	Salto rel. a PC si el registro es distinto de "0"
bltz reg,etiq	Salto rel. a PC si el registro es menor que"0"
blez reg,etiq	Salto rel. a PC si el registro es menor o igual a "0"
bgtz reg,etiq	Salto rel. a PC si el registro es mayor que "0"
bgez reg,etiq	Salto rel. a PC si el registro es mayor o igual a "0"
bgt reg,reg,etiq	Salto rel. a PC si es mayor con signo
bge reg,reg,etiq	Salto rel. a PC si es mayor o igual con signo
blt reg,reg,etiq	Salto rel. a PC si es menor con signo
ble reg,reg,etiq	Salto rel. a PC si es menor o igual con signo
bgtu reg,reg,etiq	Salto rel. a PC si es mayor sin signo
bgeu reg,reg,etiq	Salto rel. a PC si es mayor o igual sin signo
bltu reg,reg,etiq	Salto rel. a PC si es menor sin signo
bleu reg,reg,etiq	Salto rel. a PC si es menor o igual sin signo
j etiq	Salto incondicional a dirección de 26 bits (etiq)
jal etiq	Salto incondicional a dir. de 26 bits con retorno.
jr reg	Salto incondicional a dir. dada por registro
jalr reg	Salto incondicional a dir. por registro con retorno.

5 Implementación

Esta herramienta se ha implementado en C# .NET y consta fundamentalmente de los siguientes elementos:

- *Ensamblador* Se ha desarrollado una librería dinámica (.dll) encargada de analizar el código ensamblador de las 80 instrucciones enumeradas en las Tablas 2a-2d y generar el código máquina incluyendo la información del analizador.
- *Simulador MIPS32*. Se ha implementado el simulador del MIPS32 en otra librería dll independiente incluyendo la memoria principal. Básicamente se encarga de simular paso a paso el procesamiento de las instrucciones e informar de las etapas del pipeline. Esta librería utiliza directamente el componente anterior por lo que abarca todo el proceso desde el ensamblado del programa hasta la simulación.
- *ConsolaMips32*. Es una interfaz que muestra por pantalla el cronograma de ejecución en modo texto.
- *VisualMips32*. Interfaz gráfica del simulador que integra el editor y que muestra el estado del procesador desde distintos puntos de vista como es el pipeline, cronograma, registros, memoria, etc. durante la simulación de forma gráfica.

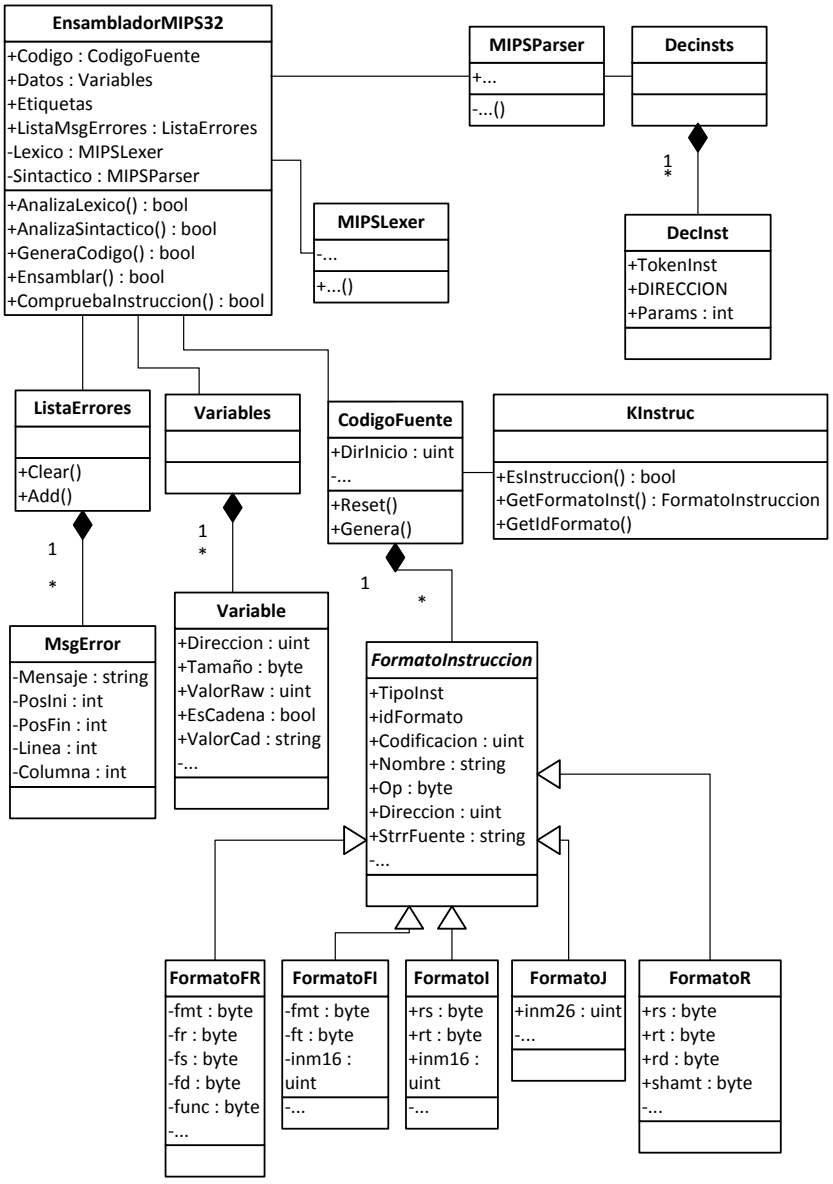

Figura 2. Diagrama de clases básico del ensamblador

5.1 Desarrollo del ensamblador

Los analizadores léxico y sintáctico, así como el generador de código, se encuentran implementados en la librería *AnalizadorMIPS32_ANTLR3.dll* cuyo diagrama de clases se muestra de forma muy simplificada en la figura 2. Este simulador verifica la sintaxis de las instrucciones basándose en el repertorio del instrucciones (ISA) descrito en [26] [27]. En la Figura 2 destaca la clase *EnsambladorMIPS* encargada de gestionar el análisis, el control de errores y la generación de código. A partir del programa en ensamblador, la clase *MIPSLexer* es la responsable de realizar el análisis léxico y generar la lista de *tokens*. Posteriormente, el analizador sintáctico *MIPSParser* realiza el parsing de los *tokens* del analizador léxico y genera una lista de declaraciones de instrucciones (clase *DecInsts*). Cada objeto de la clase *DecInst* guarda el *token* que identifica la operación de la instrucción, una lista de sus parámetros y la dirección que le correspondería en memoria. Durante la fase de generación de código, se crea un objeto de la clase *CodigoFuente* el cual genera una lista de objetos *FormatoInstruccion* a partir de la lista de declaración de instrucciones obtenida por el analizador sintáctico y la clase estática *Kinstruc*. La clase abstracta *FormatoInstruccion* es implementada por cada uno de los formatos de instrucción que posee el MIPS: *FormatoR (registro)*, *FormatoI (inmediato)*, *FormatoJ (salto incondicional) y FormatoFR (operaciones en punto flotante)*. *FormatoInstruccion* representa a una instrucción máquina a simular y mantiene, además del código de la instrucción, toda la información relacionada con el programa fuente, los campos de cada formato, dirección de memoria, mnemónico, etc. Aunque podría haberse generado gran parte del código (salvo la resolución de etiquetas) durante el análisis sintáctico, realizar la generación de código en un paso posterior reduce el tiempo del análisis sintáctico y por consiguiente el de la detección de errores del programa.

5.2 Desarrollo del simulador

La implementación del simulador está subdividida básicamente en 3 elementos: La memoria, los bancos de registros y el procesador. Éste último se encarga de la parte más compleja (implementar cada etapa del pipeline, decodificar instrucciones, implementar los desvíos para los adelantamientos y registrar el estado del pipeline en cada ciclo), permitiendo crear un historial de acontecimientos con el fin de simplificar el desarrollo de interfaces de usuario. Un diagrama de clases simplificado del simulador se muestra en la Figura 3. En ella sólo se han representado los atributos, métodos y relaciones entre clases más importantes para mantener la claridad del diagrama.

El sistema simula una memoria de 4GB, esto es, la capacidad máxima admisible para el procesador MIPS32. Puesto que implementar todas las posiciones de la memoria sería muy costoso, se ha optado por diseñar un método que almacene sólo los rangos de posiciones de memoria que contengan valores distintos de cero. Con este método, al escribir datos en memoria fuera de los rangos ya definidos, se crea un nuevo rango de memoria o se amplía un rango ya existente en función de lo alejados que se encuentren los nuevos datos de dicho rango.

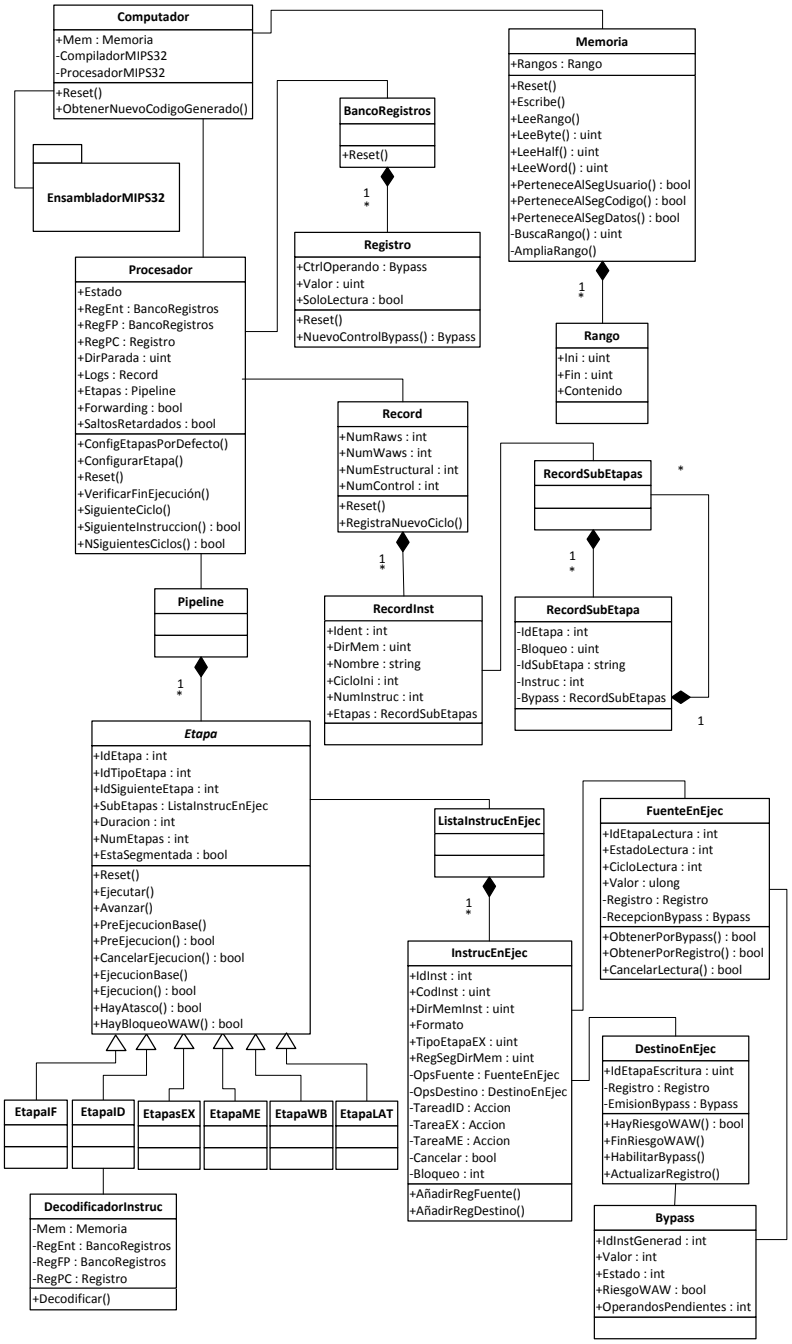

Figura 3. Diagrama de clases básico del simulador. Nota: EtapaLAT no es del diseño MIPS, pero es necesaria en el simulador (detallado más adelante).

La clase *Procesador* es la responsable de controlar la ejecución paso a paso. Para la simulación de un ciclo del reloj, el procesador realiza un proceso de 3 pasos con cada etapa del pipeline empezando desde la última hasta la primera etapa. Tales pasos se realizan de forma intercalada, esto es, se realiza el paso 1 en todas las etapas antes de continuar con el paso 2.

Cada paso consiste en lo siguiente:

- *Paso 1*. *Avanzar*: el procesador llama a la función *Avanzar()* de cada etapa encargada de transferir la instrucción en ejecución (implementada en la clase InstrucEnEjec) a la siguiente etapa o subetapa (en caso de ser una etapa segmentada, ver Figuras 11-12) si se encuentra libre.
- *Paso 2*. *Pre-ejecución*: se llama a la función *PreEjecucionBase()* de la superclase Etapa encargada de verificar si la instrucción alojada en la etapa cumple las condiciones para esa etapa. Para tener en cuenta las consideraciones propias de la etapa, la función *PreEjecucionBase()* llama a su vez a *PreEjecucion()* que es propia para cada subclase de la superclase Etapa. Este último método se encarga habitualmente de verificar si los operandos fuente están disponibles pues son necesarios para la ejecución de la etapa permitiendo así detectar bloqueos RAW. Para realizar la verificación el método deberá inspeccionar el atributo *OpsFuente* de la clase *InstruccEnEjec*. De no cumplir alguna de las condiciones, la ejecución de la etapa será cancelada hasta el siguiente ciclo, momento en el que volverán a ser reevaluadas las condiciones. En este paso también podrían llevarse a cabo las acciones propias del primer semiperiodo de un ciclo, como es el caso de la lectura de los bancos de registros o la decodificación de la instrucción en la etapa ID. El proceso de decodificación de la instrucción se realiza a través de la clase DecodificadorInstruc.
- *Paso 3*. *Ejecución*. Del último paso se encarga la función *EjecucionBase()* de la superclase Etapa. Ella será la responsable de verificar si se cumplen las condiciones relacionadas con los operandos destino, esto es, verificar si hay bloqueos WAW; y también debe realizar las tareas específicas de esa etapa. Para ejecutar tales operaciones, *EjecucionBase()* llama a la función *Ejecución()* de la subclase de Etapa correspondiente. Esta última función verifica las condiciones particulares de la etapa y lanza la tarea que la instrucción en ejecución *InstrucEnEjec* tiene asociada a esa etapa. Concretamente los atributos *TareaId*, *TareaEx* y *TareaME* de *InstrucEnEjec* contienen las tareas a realizar mediante delegados.

Para la implementación de los desvíos cuando hay adelantamientos (forwarding), se utiliza la clase Bypass que es mantenida por *Pipeline* a través de las clases *Registro*, *FuenteEnEjec* y *DestinoEnEjec*.

La estrategia básica para controlar los desvíos se basa en las dos reglas siguientes:

- Cada registro destino de la instrucción (atributo *OpsDestino* de *InstrucEnEjec*) creará un nuevo objeto Bypass que se asociará al registro correspondiente del banco de registros durante la decodificación de la instrucción.
- Cada registro fuente de la instrucción (atributo *OpsFuente* de *InstrucEnEjec*) se enlazará con el objeto Bypass perteneciente al registro fuente del banco de

registro cuando la instrucción es decodificada. Este proceso se conoce como suscripción al bypass.

- En la etapa en la que se calcule el resultado de la operación deberá habilitarse el bypass del registro destino para notificar que se encuentra disponible para todos los operandos fuente relacionados con dicho bypass. Ello se consigue llamando a la función *HabilitarBypass()* del objeto *DestinoEnEjec*.
- En la etapa en la que la instrucción precise un operando fuente, lo tomará por bypass siempre que éste se encuentre disponible. En caso de tomarlo, deberá notificarlo para actualizar convenientemente el número de envíos que el bypass tiene pendiente.
- En cuanto se actualice el resultado de la operación en el registro destino correspondiente, deberá notificarse que el desvío ya no será necesario para nuevos operandos fuente aunque los operandos ya suscritos sí seguirán tomando el operando a través del desvío.
- A pesar de haber ejecutado todas las etapas hasta WB, una instrucción no podrá abandonar el pipeline mientras tenga envíos por bypass pendientes, permaneciendo mientras tanto en la *EtapaLAT* diseñada para tal fin.

La Figura 4 muestra un ejemplo del mecanismo descrito anteriormente indicándose el instante en el que se crean los desvíos, se suscribe a ellos, se habilitan, se envían operandos por bypass y se actualiza el registro. Por ejemplo, el ciclo de vida del bypass B1:

- *Ciclo 2 (ID de I1)*: se crea el bypass B1 asociado al registro de destino $1 de la instrucción 1.
- *Ciclo 3 (EX de I1)*: el bypass B1 es habilitado por lo que el valor de $1 se encuentra disponible por bypass.
- *Ciclo 3 (ID de I2)*: el operando fuente $1 de I2 se suscribe a B1 para tomar el valor desde éste cuando la instrucción I2 lo precise.
- *Ciclo 4 (EX de I2)*: la instrucción I2 recibe el dato correspondiente por bypass gracias a la suscripción que se realizó a B1.
- *Ciclo 4 (ID de I3)*: el operando destino de la instrucción I3 es $1 por lo que se crea un nuevo bypass (bypass B3) que remplaza a B1. A partir de este momento, las nuevas suscripciones a $1 irán dirigidas a B3.
- *Ciclo 5 (WB de I1)*: se actualiza el valor del bypass B1 en el registro.

	Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**
I1	**addi $1,$0,1**	IF	**ID**	**EX**	ME	WB		
I2	**addi $5,$1,3**		IF	**ID**	**EX**	ME	WB	
I3	**lw $1,0($5)**			IF	**ID**	EX	ME	WB
I4	**add $9,$1,$7**				IF	ID	--	EX
Bypass creado			$1→B1	$5→B2	$1→B3	$9→B4		
Bypass suscrito				B1@I2	B2@I3	B3@I4		
Bypass habilitado				B1	B2		B3	B4
Bypass enviado					B1→I2	B2→I3		B3→I4
Bypass actualizado						B1	B2	B3

Figura 4. Ejemplo de funcionamiento de los desvíos

El simulador implementa un modelo para registrar los sucesos que se van produciendo en cada etapa para cada ciclo de reloj durante la simulación, como son los bloqueos o los desvíos que se activan. El objetivo de mantener este historial de acontecimientos es ofrecer un mayor soporte a las interfaces que podrían necesitar estos datos, por ejemplo cuando el usuario navegue por el cronograma. Las clases que implementan esta funcionalidad se muestran en la Figura 3.

La clase Record mantiene además de la estadística de rendimiento, una lista de las instrucciones que se han ejecutado. Cada instrucción ejecutada se registra en un objeto de la clase *RecordInst* que incluye toda la información que pudiera ser relevante posteriormente, como es el ciclo en el que comenzó la ejecución, un identificador de la línea con la que se corresponde en el programa ensamblador, la dirección de memoria y una lista de lo ocurrido en cada ciclo de reloj mientras estuvo la instrucción en ejecución. Esta lista de sucesos está implementada en la clase *RecordSubEtapas*, la cual contiene objetos de *RecordSubEtapa* que son los responsables de guardar la información del estado de la instrucción en cada ciclo, como es el tipo de bloqueo que se produjo o desde o hacia dónde se activó el desvío.

6 Funcionamiento del entorno VisualMips32

La interfaz gráfica VisualMips32 para Windows ofrece un entorno de simulación integrado con las herramientas necesarias para la edición, depuración y simulación de un código ensamblador para MIPS. El objetivo principal de este entorno de desarrollo es simplificar la interacción con el simulador y visualizar de forma clara e intuitiva el estado del procesador en cualquier momento a través del cronograma de ejecución de las instrucciones, el contenido de la memoria, los registros del procesador y la representación del pipeline.

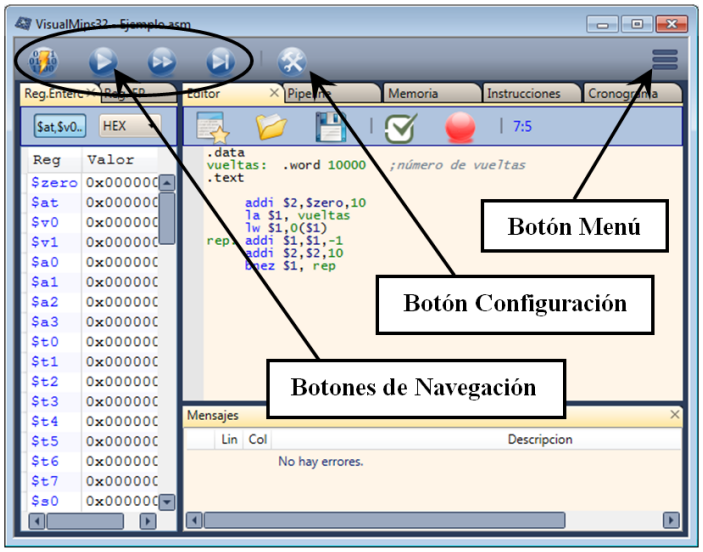

Figura 5. Visión general del entorno, la ventana principal

El entorno se compone fundamentalmente de siete ventanas que el usuario puede acoplar a la ventana principal en forma de pestañas o subdividiendo el área de otras ya acopladas. Al iniciar la aplicación las ventanas se encuentran distribuidas en la ventana principal como muestra la Figura 5.

La ventana principal del entorno, que sirve de contenedor para el resto de ventanas, posee los siguientes botones:

- *Botón de Menú*. para acceder a todas las funciones del entorno.
- *Botón de Configuración*. Muestra la ventana de configuración del procesador y simulador.
- *Botones de navegación*. Para reiniciar y avanzar en la simulación ciclo a ciclo, instrucción a instrucción o hasta el final.

6.1 Ventana Editor

Esta venta se utiliza principalmente para cargar, editar y ensamblar el código ensamblador a simular. Los programas se guardan con la extensión *asm*.

Como se aprecia en la Figura 6, además de los botones propios de cualquier editor de texto, incluye el botón Ensamblar encargado de detectar los errores del código y mostrarlos por la ventana de mensajes. Si la opción *AutoEnsamblar* de la ventana de configuración está activa los errores se mostrarán automáticamente durante la edición del programa.

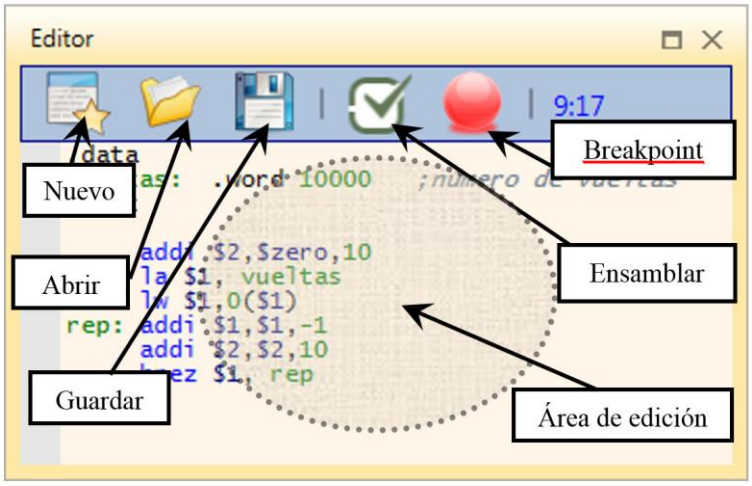

Figura 6. Ventana Editor

El entorno tiene soporte para gestionar puntos de interrupción (*breakpoints*) durante la simulación. Desde la ventana Editor se pueden insertar estos puntos de interrupción bien a través del botón *Breakpoint* mostrado en la Figura 6 o haciendo doble clic con el ratón en el margen izquierdo del área de edición. Al comenzar la simulación, los puntos de interrupción que se hayan definido en el editor se transfieren a la ventana Instrucciones.

6.2 Ventana Mensajes

Esta ventana se encarga de detallar los errores de ensamblado que presenta el código durante la fase de edición o bien las excepciones lanzadas por el procesador al ejecutar las instrucciones durante la simulación.

6.3 Ventana Memoria

Como se aprecia en la Figura 7, esta ventana representa el contenido de la memoria en hexadecimal y permite agrupar las posiciones de memoria formando bytes, medias palabras, palabras o dobles palabras. Tanto el contenido del segmento de código como el del segmento de datos puede ser también modificado directamente por el usuario. Cuando se realiza una escritura en memoria, bien sea por el usuario o por el procesador, la ventana muestra en rojo el último dato que ha sido modificado. A continuación se describen los elementos de la ventana memoria:

- *Selector Dirección de inicio*: permite al usuario acceder rápidamente a una zona determinada de la memoria. En ella puede indicarse, una dirección de memoria en hexadecimal (por ejemplo 0x00400000), o el nombre de un segmento (como puede ser *.text* para ir al comienzo del segmento de código) o bien el nombre de una variable definida en el programa.
- *Selector Bytes por dato*: especifica en cuántos bytes se agrupan los datos contenidos en la memoria. Pueden agruparse por bytes, medias palabras (16 bits), palabras (32 bits) o dobles palabras (64 bits).

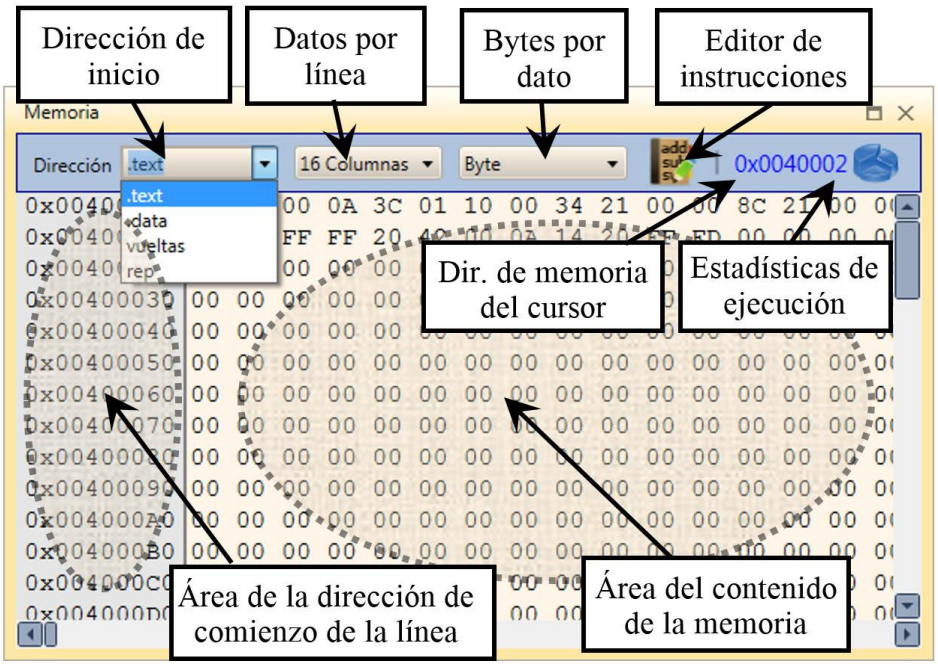

Figura 7. Ventana Memoria

- *Selector Datos por línea*: establece cuantos datos se muestran en cada línea del área del contenido de la memoria, siendo el tamaño de cada dato especificado por el selector anteriormente descrito.
-
- *Botón editor de instrucciones*: al pulsar este botón cuando el cursor se encuentra en el segmento de código, el entorno toma los datos de la memoria sobre los que se encuentra el cursor y los interpreta como el código de una instrucción. A continuación se abre la ventana *Editor de Instrucciones* (Figura 8) que permite modificar el contenido de la memoria en función de la instrucción que el usuario defina.

6.4 Ventana Editor de instrucciones

La función principal del editor de instrucciones es codificar instrucciones de forma sencilla con el fin de modificar el programa cargado en memoria durante su ejecución. Por otro lado, proporciona una herramienta didáctica de interés para estudiar los formatos de instrucción del procesador MIPS.

Una vez abierto el editor, el usuario puede modificar directamente los bits del código de la instrucción, especificar el mnemónico de una nueva instrucción o incluso modificar cada uno de los campos que contiene el formato de la instrucción. En función del código de operación que se especifique los campos que forman el formato de instrucción cambiarán automáticamente. Cada una de las alternativas anteriores puede especificarse en base decimal, binaria o hexadecimal.

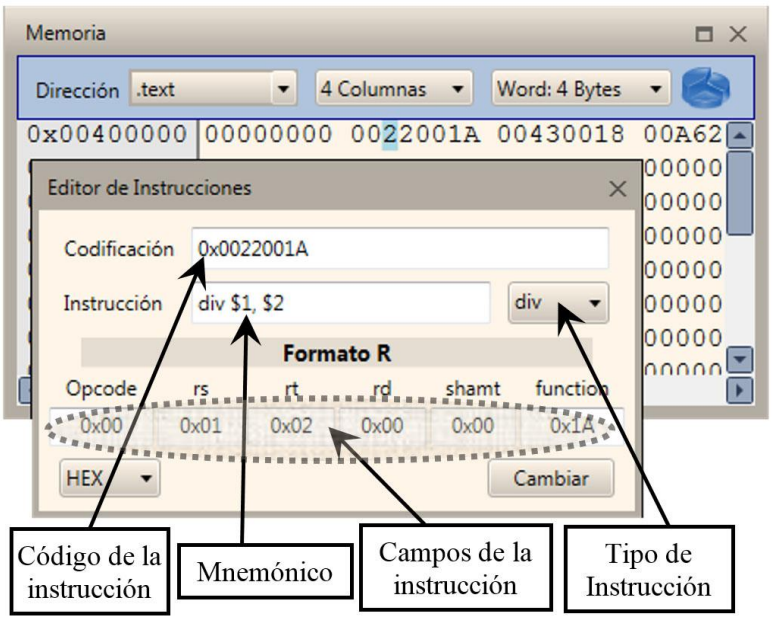

Figura 8. Ventana Editor de Instrucciones

6.5 Ventanas de edición de registros

Consisten en dos ventanas que visualizan y permiten modificar el contenido de los registros del banco de registros de enteros y de coma flotante durante la simulación.

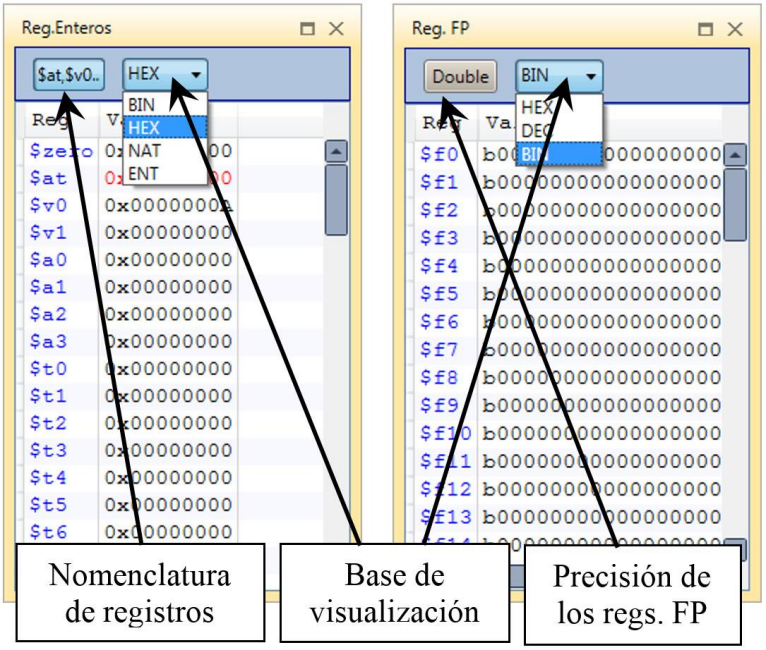

Figura 9. (Izquierda) Ventana del banco de registros de enteros; (Derecha) Ventana del banco de registros de coma flotante.

En estas ventanas se encuentran los siguientes elementos:

- *Selector de la base de visualización*: especifica la base en la que se mostrarán los registros: binario, hexadecimal, entero o natural (entero sin signo). Independientemente de cuál sea la base de visualización seleccionada, el usuario podrá modificar el contenido de los registros en cualquier base simplemente especificando el prefijo 0x para referirse a la base hexadecimal, el prefijo b para describir que se trata de base binaria o ninguno de los anteriores para tratar el valor introducido como base decimal.
- *Botón Precisión de los registros en FP*: Cada registro de FP de los 32 que posee el banco puede utilizarse para almacenar un número en coma flotante de simple precisión, o bien pueden agruparse dos a dos para formar registros de 64 bits con el fin de almacenar números en coma flotante de doble precisión. El botón Precisión conmutará entre la visualización del banco como 32 registros de simple precisión o 16 registros de doble precisión.
- *Botón Nomenclatura de registros*: en MIPS32 pueden utilizarse dos nomenclaturas distintas para referirse a los registros, una consistente en especificar el símbolo $ seguido del número que ocupa en el banco de registros (por ejemplo $1), o bien emplear el símbolo $ junto al nombre que

denota la función del registro en el procesador (por ejemplo $t2). La Tabla 1 recoge ambas nomenclaturas, su equivalencia y uso.

6.6 Ventana Instrucciones

Lista las instrucciones que se encuentran almacenadas en el segmento de código y las decodifica.

Para cada instrucción de la lista se especifican los siguientes campos:

- *Dirección de memoria donde comienza la instrucción.*
- *Codificación de la instrucción expresada en hexadecimal.*
- *Nemónico de la instrucción.*
- *Texto de la instrucción según aparece en el editor del código de usuario.*
- *Etapa en la que se encuentra cada instrucción durante la simulación.*

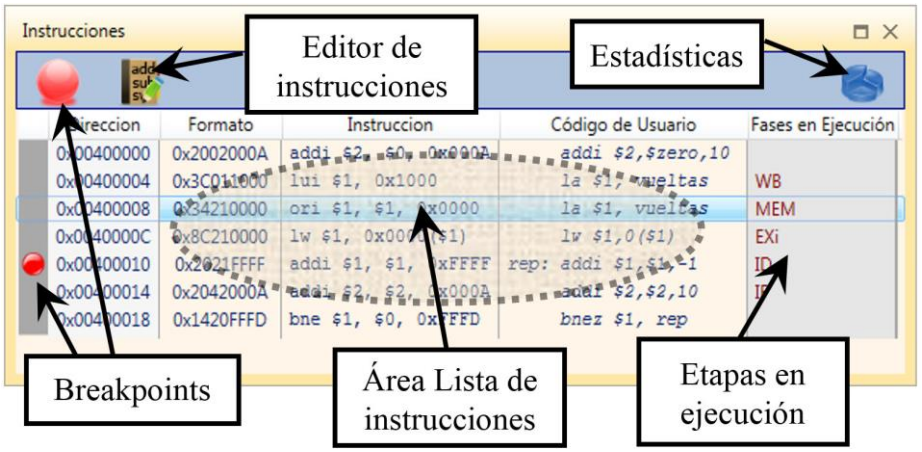

Figura 10. Ventana Instrucciones

Esta ventana permite además editar las instrucciones y establecer puntos de interrupción durante la ejecución.

6.7 Ventana Pipeline

La arquitectura MIPS32 dispone de varias unidades funcionales de cálculo, unas integradas en el procesador y otras alojadas en un coprocesador independiente (Figura 1), cada una de ellas dedicada a realizar ciertos tipos de operaciones. A continuación se describen las unidades funcionales que posee el MIPS32 y que han sido implementadas en el simulador:

- *Unidad de enteros principal (EXi)*: encargada de las cargas, almacenamientos, saltos y las operaciones ALU enteras excepto las de multiplicación y división. Por ejemplo la instrucción add $1, $2,$3 se realizaría en Exi.

- *Unidad para multiplicación de enteros (EXm)*.
- *Unidad para la división de enteros (EXd)*.
- *Unidad de FP principal (EXfp)*: realiza todas las operaciones de coma flotante a excepción de la multiplicación y la división. Ejemplo *add.s $f2, $f4, $f5*.
- *Unidad para la multiplicación en coma flotante (EXfpm)*.
- *Unidad para la división en coma flotante (EXfpd)*.

Teniendo en cuenta estas unidades funcionales, el simulador representa un pipeline según se muestra en la Figura 11.

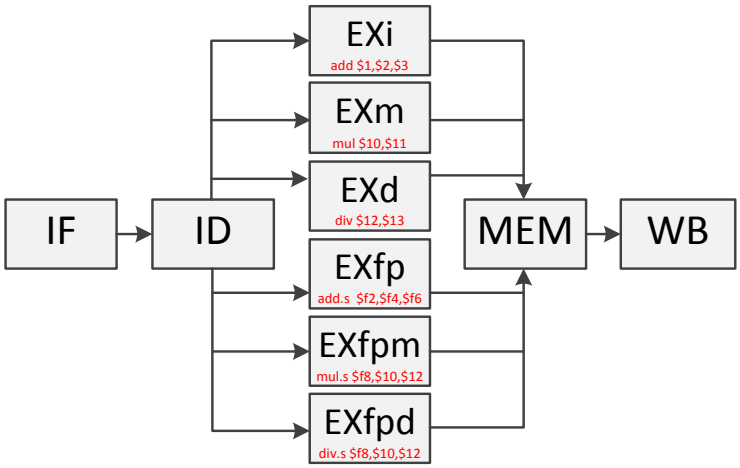

Figura 11. Pipeline con las unidades funcionales independientes

Salvo la unidad funcional *EXi*, estas unidades realizan operaciones complejas por lo que suelen precisar de varios ciclos de reloj para realizar el cálculo, así que la etapa EX de tales instrucciones dura más de un ciclo. Además, algunas de estas unidades de cálculo multiciclo pueden estar a su vez segmentadas, esto es, que el cálculo de la operación puede estar dividido en varias subetapas. La ventaja de las unidades segmentadas es que permiten comenzar una nueva operación en cada ciclo de reloj aunque otras anteriores no hayan terminado.

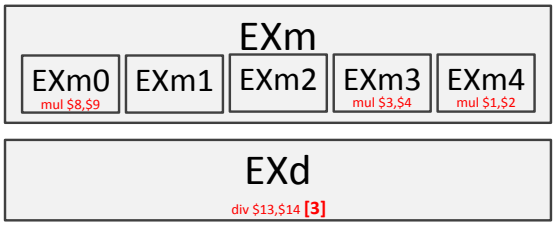

Figura 12. (Arriba) Ejemplo de unidad segmentada multiciclo; (Abajo) Ejemplo de unidad no segmentada multiciclo

La Figura 12 muestra una posible forma de representar las unidades segmentadas y no segmentadas multiciclo. El ejemplo de la Figura 12a corresponde a una unidad de la multiplicación segmentada con 5 etapas y que contiene 3 instrucciones en ejecución cada una en subetapas distintas. La Figura 12b muestra una unidad funcional de la división configurada como no segmentada en la que hay una instrucción que se encuentra en el ciclo 3 de su ejecución.

Un ejemplo de la ventana Pipeline se encuentra en la Figura 13. En ella se representa el estado del pipeline mediante cajas siguiendo un esquema similar al mostrado en la Figura 11. Para especificar el estado de cada etapa, el borde del recuadro que representa a la etapa se colorea según el resultado de la ejecución de la instrucción alojada en ella. Un borde de color negro especifica que la ejecución se ha realizado con éxito y otro color representa el tipo del bloqueo según la configuración de colores especificada en la configuración del simulador.

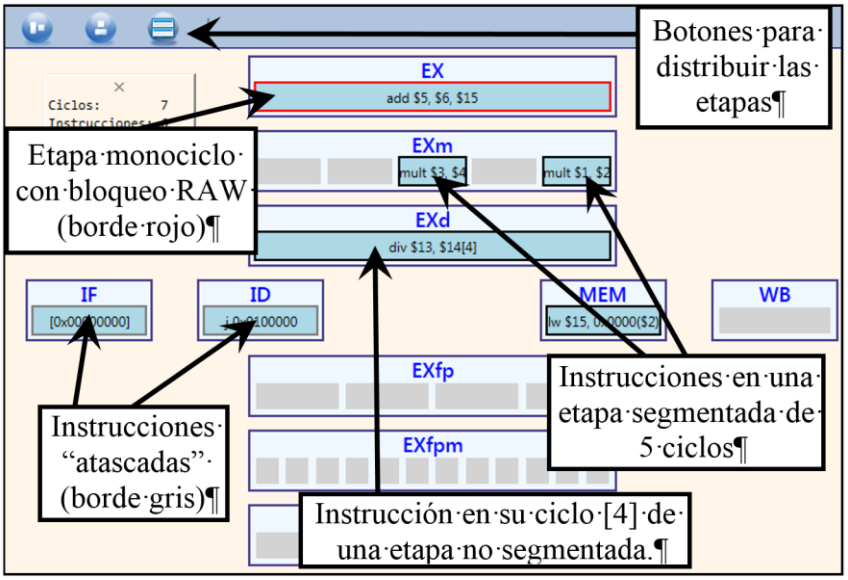

Figura 13. Ventana Pipeline

Cada caja que representa a una etapa se puede mover y cambiar de tamaño para adaptarse a las necesidades del usuario. También dispone de botones que simplifican la distribución de las etapas en el espacio de trabajo y reorganizan las subetapas de cada etapa horizontal o verticalmente.

6.8 Ventana Cronograma

Una de las ventanas que aporta más información al simular el procesamiento segmentado es la ventana Cronograma. Como puede apreciarse en la Figura 14, en esta ventana se representa el diagrama de tiempos del estado de cada etapa en cada ciclo de reloj durante la ejecución de las instrucciones del código en ensamblador.

Como suele ser lo habitual, el eje de ordenadas indica las instrucciones y las abscisas el ciclo de ejecución. Para cada instrucción se especifica un número de secuencia, que ocupa en la ejecución de la instrucción, la dirección de memoria y su representación en ensamblador. Para cada intersección instrucción-ciclo se muestra el nombre de la etapa y el estado en el que se encuentra. Cuando la etapa ha sido bloqueada se utiliza un código de colores configurable que identifica el tipo de bloqueo. Para una mejor identificación del bloqueo, es posible opcionalmente tachar el nombre de la etapa y añadir un carácter especial junto al nombre. El significado de estos caracteres especiales es el siguiente:

- *Asterisco como prefijo (*)*: denota que la etapa posee un bloqueo tipo lectura tras escritura (RAW). Por defecto se representan en rojo. Ejemplo ~~*EXi~~
- *Asterisco como sufijo (*)*: especifica que la etapa posee un bloqueo tipo escritura tras escritura (WAW) y se representan en verde. Por ejemplo ~~EX*~~.
- *Acento circunflejo (^)*: representa a los bloqueos estructurales y suelen aparecer en color amarillo. Sirva de ejemplo ~~EX^~~.
- *Barra ascendente (/)*. Indica un bloqueo de control y es dibujado en color azul. Por ejemplo ~~IF/~~.

Cuando la instrucción no puede avanzar de etapa debido a la detención del cauce no se utiliza ningún carácter especial, tan sólo se muestra la etapa en gris por defecto y aparece tachada.

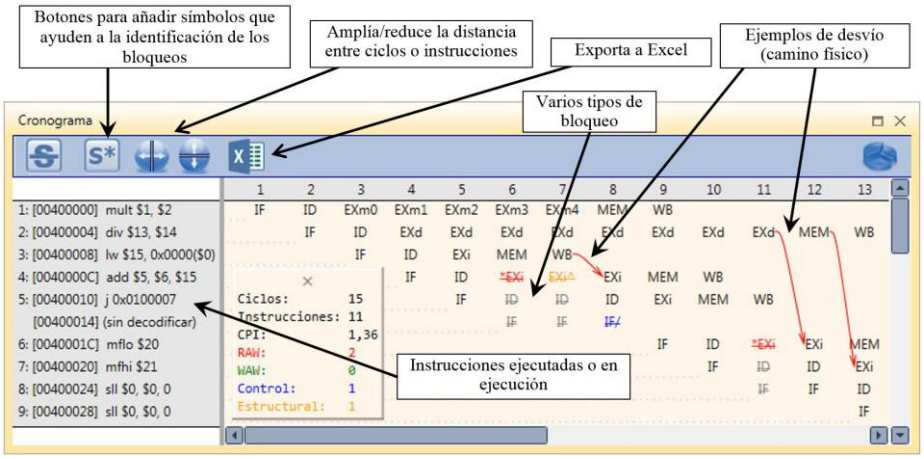

Figura 14. Ventana Cronograma

En una etapa multiciclo, la instrucción deberá permanecer en la unidad funcional durante varios ciclos. Si la etapa está segmentada, el cronograma muestra junto al nombre de la etapa el número de la subetapa en la que se encuentra la instrucción. Véase como ejemplo la instrucción 1 que aparece en la Figura 14; en ella puede verse que la instrucción *mult* realizó su fase EX en la unidad de multiplicación de enteros EXm, que está segmentada y que dura 5 ciclos. Dicha instrucción accedió a la subetapa 0 de EXm en el ciclo 3 y fue avanzando a las siguientes subetapas hasta alcanzar la última subetapa en el ciclo 7. En cambio las etapas configuradas como

multiciclo no segmentadas no añaden ningún identificador de la subetapa en la que se encuentran ya que en realidad la subetapa es única, sólo que dura varios ciclos. La Figura 14 muestra un ejemplo de ello en la instrucción 2, la cual entra en la subetapa que posee EXd en el ciclo 4 y permanece en ella hasta pasar a la etapa de MEM.

Para finalizar este artículo, se exponen a continuación los posibles trabajos futuros y las conclusiones.

7 Trabajos futuros

Aunque el simulador presentado en este trabajo ya ha alcanzado el fin de su fase de desarrollo según los requerimientos establecidos inicialmente, son innumerables las mejoras que se le están aplicando para conferirle mayor estabilidad, fiabilidad y versatilidad. Además de estas continuas mejoras, claves para conseguir un software de calidad, se han añadido a la lista de tareas por hacer nuevas características y herramientas. Algunas de ellas encaminadas a ampliar el sistema que se simula con nuevos componentes como cachés y BTB; algunas orientadas a representar el sistema con mayor detalle, como es mostrar la ruta de datos a nivel RT, pero sin sacrificar su facilidad de uso.

Por último, en aspectos relacionados con la docencia, este simulador abarca un contenido íntimamente relacionado con la asignatura *Arquitectura de Computadores* de segundo curso de los grados de Ingeniería Informática impartidos por la Universidad de Sevilla. Por consiguiente, se pretende utilizarlo este curso académico en las sesiones prácticas relacionadas con esta materia, así como dejarlo a libre disposición del alumnado para su descarga gratuita y uso para prácticas en casa.. De esta manera, se pretende realizar un análisis de la mejora del rendimiento académico en el alumnado mediante dos mecanismos diferenciados:

- *Encuesta anónima al alumnado*: en la que se evaluarán aspectos más subjetivos de la utilidad del simulador ampliamente detallado en las secciones anteriores. De esta manera se pueden obtener unas conclusiones rápidas sobre la utilidad y fidelidad del simulador, de cara a realizar las mejoras oportunas para el siguiente curso académico.
- *Evaluación de resultados académicos*: este mecanismo pretende aportar unas métricas objetivas acerca de la mejoría de comprensión del procesador MIPS mediante una comparativa entre las calificaciones obtenidas por el alumnado en cursos anteriores y las obtenidas en el curso académico actual.

Mediante la evaluación de las métricas anteriores se espera obtener suficiente información para mejorar aquellos aspectos del simulador que más problemas ocasionen al aprendizaje del alumnado. De esta forma, el simulador será sometido a un ciclo de mejora continua; obteniéndose una versión definitiva del simulador en pocos cursos académicos.

8 Conclusiones

El estudio de la arquitectura de computadores puede convertirse en una ardua tarea para profesores y alumnos si no se dispone de un simulador adecuado. En este trabajo se ha descrito algunos de los simuladores académicos más utilizados y se ha presentado un nuevo software de simulación del procesador MIPS32 segmentado orientado a la enseñanza y al aprendizaje de la arquitectura de computadores que se imparte en las universidades. Este simulador implementa más de 80 instrucciones y simula el comportamiento del procesador con capacidad para detectar y resolver los riesgos en la segmentación. Implementa algunas mejoras del procesador MIPS como son los adelantamientos (forwarding) y los saltos retardados. También se ha propuesto una interfaz gráfica muy intuitiva y versátil con el propósito de hacer más atractivo el estudio del procesador a los estudiantes. El simulador está implementado en una librería independiente de la interfaz por lo que podría diseñarse nuevas interfaces de usuario adaptadas a las necesidades de los alumnos.

Referencias

1. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design - The Hardware/Software Interface”, 3rd ed. Morgan Kaufmann, 2005
2. John L. Hennessy and David A. Patterson, “Computer Architecture: A Quantitative Approach”. San Francisco, CA: Morgan Kaufman, 2007.
3. J. L. S. C. Pereira, “Educational package based on the MIPS architecture for FPGA platforms”, Master Thesis, Faculdade de Engenharia da Universidade do Porto, June 2009. http://repositorio-aberto.up.pt/bitstream/10216/59975/1/000135086.pdf
4. A. Clements, “The undergraduate curriculum in computer architecture”, IEEE Micro, vol. 20, no. 3, pp. 13–21, 2000
5. J. Djordjevic, B. Nikolic, T. Borozan, and A. Milenkovie, “CAL2: Computer aided learning in computer architecture laboratory”, Comput. Appl. Eng. Educ., vol. 16, pp. 172–188, 2008
6. B. Nikolic, Z. Radivojevic, J. Djordjevic and V. Milutinovic, "A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization", Education, IEEE Transactions on , vo1.52, no.4, pp.449-458, 2009
7. H. Oztekin, F. Temurtas, and A. Gulbag, “BZK.SAU: Implementing a hardware and software-based computer architecture simulator for educational purpose”, Proc. 2nd Int. Conf. Comput. Design Appl., pp. 490–497, 2010
8. B. Mustafa, "Modem computer architecture teaching and learning support: An experience in evaluation", International Conference on Information Society (i-Society), pp.411-416, 27-29, 2011
9. W. Yurcik, G. S. Wolffe and M. A. Holiday, “A Survey of Simulators used in Computer Organization/Architecture Courses”, Procs. of Summer Conf. on Computer Simulation, pp. 524-529, Orlando, Florida, 2001.
10. Página web del simulador QtSpim: http://spimsimulator.sourceforge.net

11. Página web del simulator SPIM: http://cs.wisc.edu/~larus/spim.html
12. D. K. Vollmar and D. P. Sanderson, "MARS: An Education-Oriented MIPS Assembly Language Simulator", Procs. of 37th SIGCSE technical symposium on computer science education, 2006.
13. J. Larus, "SPIM S20: A MIPS R2000 Simulator", Computer Sciences Department, University of Wisconsin, Tech. Rep., 1990. http://pages.cs.wisc.edu/~larus/SPIM/spim_documentation.pdf
14. Página web del simulador MARS: http://mars-sim.sourceforge.net/
15. G. C. R. Sales, M. R. D. Araujo, and F. L. C. Padua, "MIPS X-Ray: A Plug-in to MARS Simulator for Datapath Visualization", 2nd Intern. Conf. on Education Technology and Computer (ICETC), 2010
16. Página web del simulador ProcessorSim: http://jamesgart.com/procsim.
17. H. Sarjoughian, Y. Chen, & K. Burger, (2008). "A component-based visual simulator for MIPS32 processors". In Proceedings – 38th Frontiers in Education Conference (FIE), 2008.
18. Página web del simulador MIPS-Datapath: https://github.com/acecil/MIPS-Datapath
19. I. Branovic, R. Giorgi, and E. Martinelli, "WebMIPS: A New WebBased MIPS Simulation Environment for Computer Architecture Education", Workshop on Computer Architecture Education, 31st International Symposium on Computer Architecture, 2004.
20. Página web del simulador EduMIPS64: http://www.edumips.org.
21. Página web del cuestionario sobre impacto de EduMIPS64 en los estudiantes: http://www.diit.unict.it/users/spadaccini/edumips64-survey.html
22. Página web del simulador WinMIPS64: http:// http://indigo.ie/~mscott/
23. P. S. Coe, F. W. Howell , R. N. Ibbett and L. M. Williams, "Technical Note: A hierarchical computer architecture design and simulation environment". ACM Trans. Model. Com. Simul., pp. 431–446, 1998.
24. E. Z. Bem and L. Petelczyc, "MiniMIPS: a simulation project for the computer architecture laboratory". Procs. of the 34th SIGCSE technical symposium on Computer science education, pp. 64–68, 2003
25. B. Nova, J.C. Ferreira and A. Araujo. "Tool to support computer architecture teaching and learning". 1st International Conference of the Portuguese Society for Engineering Education (CISPEE), 2013
26. MIPS32® Instruction Set Quick Reference, Revision 1.01. http://www.imgtec.com/downloads/factsheets/MD00565-2B-MIPS32-QRC-01.01.pdf
27. MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set, Revision 5.04. http://www.imgtec.com/mips/architectures/mips32.asp
28. MIPS® Architecture for Programmers Volume I-A: Introduction to the MIPS32® Architecture. Revision v6.01. http://www.imgtec.com/mips/architectures/mips32.asp