Herramienta software para la enseñanza del paralelismo a nivel de instrucciones (ILP)

Manuel Rivas-Pérez*, Manuel Domínguez-Morales*, Alejandro Linares-Barranco*, Antón Civit-Balcells*

Resumen—**La arquitectura de computadores es una asignatura de gran importancia en las titulaciones de Informática. Debido a la dificultad de poder acceder directamente a los componentes internos de un procesador, es común en la enseñanza hacer uso de una o varias herramientas software que simulen el comportamiento interno del mismo. Sin embargo, suelen ser herramientas limitadas u obsoletas. Esto provoca una gran dificultad a la hora de transmitir los conocimientos en este tipo de asignaturas. Este caso se puede aplicar directamente al paralelismo a nivel de instrucciones (ILP), que precisa poder visualizar el comportamiento a nivel interno del camino de datos del procesador. Para ello, en este trabajo se presenta una herramienta software desarrollada íntegramente a medida para la enseñanza de arquitectura de computadores, que permite la visualización detallada del pipeline del procesador, permitiendo modificaciones sobre su ejecución y aplicar optimizaciones en tiempo de ejecución para poder obtener medidas de rendimiento. Seguidamente, tras tres años de uso en docencia, se presenta un estudio sobre la bondad del uso de la herramienta atendiendo a la evolución de las calificaciones y a encuestas realizadas sobre el alumnado.**

Palabras clave—**ILP, segmentación, arquitectura de computadores, enseñanza, MIPS32.**

I. INTRODUCCIÓN

La electrónica digital ha evolucionado rápidamente en muy poco tiempo llegando a integrar más de mil millones de transistores en un mismo chip. La arquitectura también ha evolucionado con nuevas estructuras como la segmentación, el uso de cachés, procesamiento multihilo 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].

Dpto. Arquitectura y Tecnología de Computadores, E.T.S. Ingeniería Informática, Universidad de Sevilla.

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 por lo 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].

En este trabajo se presenta una herramienta software que simula y visualiza el procesamiento paralelo 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 describe la implementación y características del simulador; en la sección 3, se expone el funcionamiento del entorno visual VisualMIPS32; a continuación, se evalúa la herramienta en base a las calificaciones del alumnado y a las encuestas de satisfacción; para, finalmente, presentar las mejoras futuras y conclusiones del trabajo.

II. CARACTERÍSTICAS E IMPLEMENTACIÓN

Se describirá en primera instancia las características básicas del procesador MIPS para, posteriormente, introducir los detalles de implementación.

A. *Características*

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 [9]–[11]. Las características principales de este procesador son, entre otras: Procesador RISC segmentado de 32 bits y direcciones de 32 bits, instrucciones de tamaño fijo, 32 registros de propósito general, unidades funcionales independientes para producto y división, detección y control de riesgos, implementa adelantamientos (*forwarding*) y permite

saltos retardaos. 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.

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 147 de ellas que son las que comúnmente se encuentran en la mayoría de los programas en ensamblador:

- Todas las cargas con y sin signo del MIPS I.
- Todos los almacenamientos con y sin signo del MIPS I.
- Todas las instrucciones aritméticas y lógicas con enteros del MIPS I.
- Gran parte de las instrucciones aritméticas y lógicas en coma flotante del MIPS I.
- Gran parte de las instrucciones de control del MIPS I.

B. 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 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.

1) Implementación 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 Fig. 1. Este simulador verifica la sintaxis de las instrucciones basándose en el repertorio de instrucciones (ISA) descrito en [26] [27]. En la Fig.1 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.

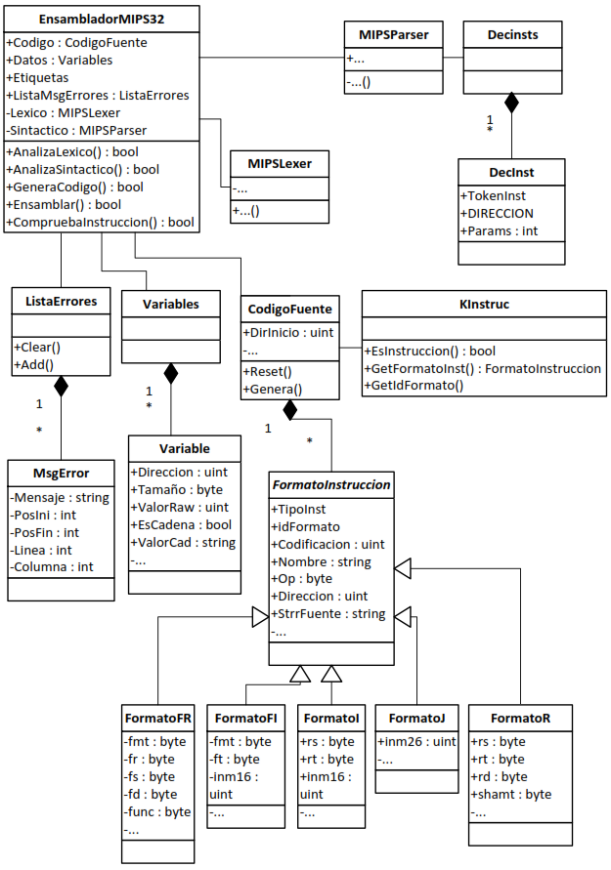

Fig. 1. Diagrama de clases básico del ensamblador.

2) Implementación del simulador

La implementación del simulador está subdividida básicamente en 3 elementos: La memoria, los bancos de registros y el procesador. Este 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 Fig. 2. 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.

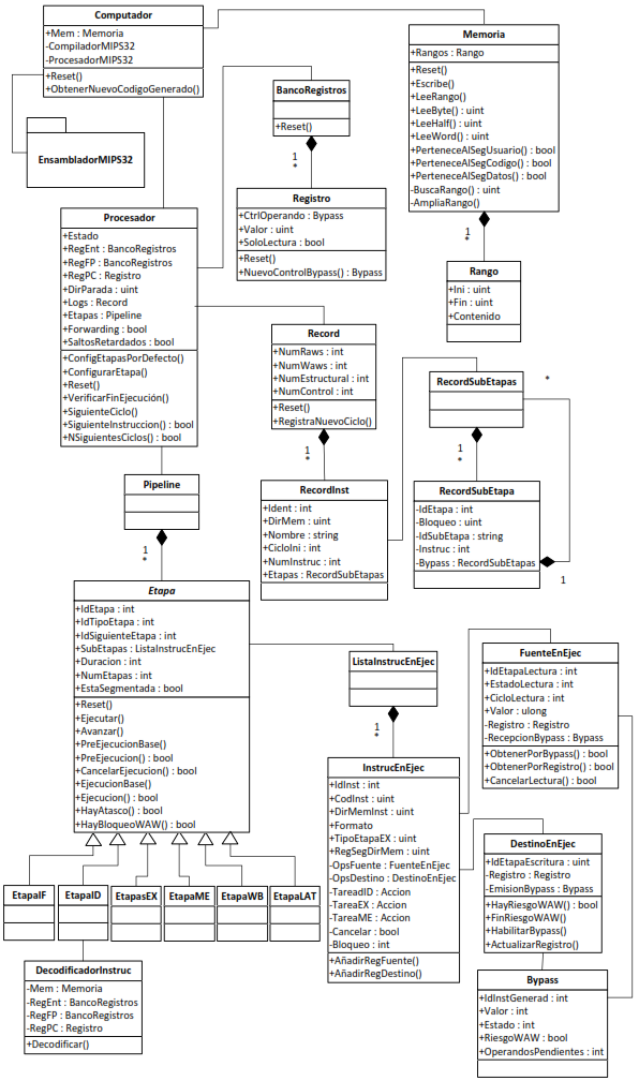

Fig. 2. Diagrama de clases básico del simulador.

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. Estos pasos consisten en: Avanzar (transferir la instrucción en ejecución a la siguiente etapa si se encuentra libre), Pre-ejecución (verificar si la instrucción reúne las condiciones para la etapa en la que se encuentra; por ejemplo, disponibilidad de operandos); y Ejecución (verificar condiciones relacionadas con operandos destino y realizar las tareas específicas de esta etapa).

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 Fig. 3 muestra un ejemplo del funcionamiento del simulador 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 destino $1 de la instrucción 1.
- Ciclo 3 (EX de I1): el *bypass* B1 es habilitado, así pues, el valor de $1 está 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

Fig. 3. Ejemplo de funcionamiento del simulador.

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 Fig. 2.

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.

III. 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.

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 Fig. 4.

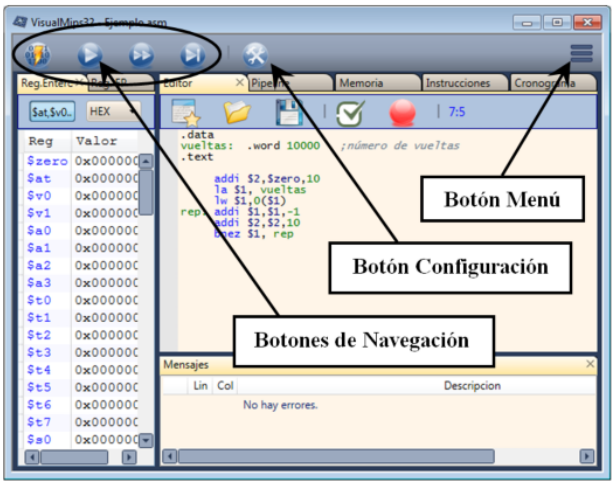

Fig. 4. Visión general del entorno, la ventana principal.

La ventana principal del entorno, que sirve de contenedor para el resto de las ventanas, posee los siguientes botones: Botón de Menú (funciones del entorno), Botón de Configuración (configuración del procesador y simulador), Botones de navegación (reiniciar y avanzar en la simulación).

A. *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*. Los errores se mostrarán automáticamente durante la edición del programa.

El entorno tiene soporte para gestionar puntos de interrupción (*breakpoints*) durante la simulación.

B. *Ventana Memoria*

Esta ventana representa el contenido de la memoria en hexadecimal y permite agrupar las posiciones de memoria formando bytes, medias palabra, 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.

C. *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, y Etapa en la que se encuentra cada instrucción durante la simulación.

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

D. *Ventana Pipeline*

La arquitectura MIPS32 dispone de varias unidades funcionales de cálculo, unas integradas en el procesador y otras alojadas en un coprocesador independiente, cada una de ellas dedicada a realizar ciertos tipos de operaciones. Las unidades funcionales que posee el MIPS32 y que han sido implementadas en el simulador son: Unidad de enteros principal (*EXi*), Unidad para multiplicación de enteros (*EXm*), Unidad para la división de enteros (*EXd*), Unidad de FP principal (*EXfp*), Unidad para la multiplicación en coma flotante (*EXfpm*), y Unidad para la división en coma flotante (*EXfpd*).

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.

Un ejemplo de la ventana Pipeline se encuentra en la Fig. 5. En ella se representa el estado del pipeline mediante cajas. 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.

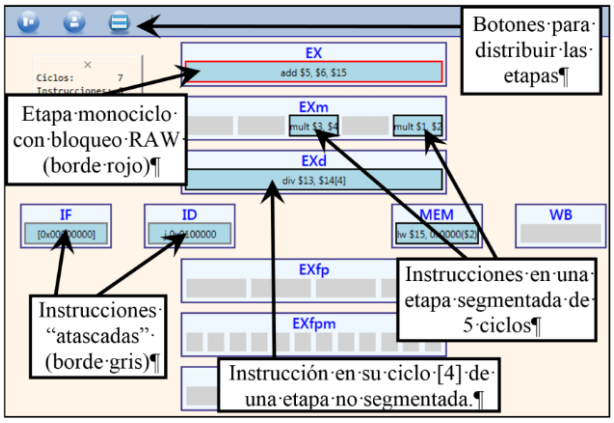

Fig. 5. Ventana Pipeline.

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.

E. Ventana Cronograma

Una de las ventanas que aporta más información al simular el procesamiento segmentado es la ventana Cronograma. Como puede apreciarse en la Fig. 6, 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.

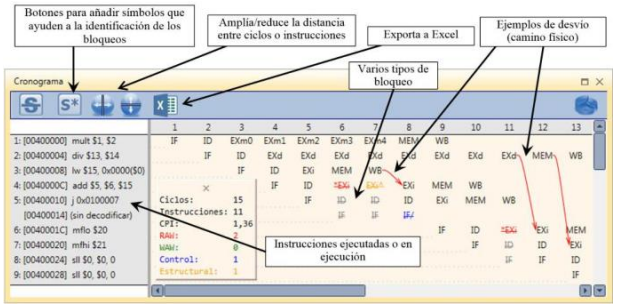

Fig. 6. Ventana Cronograma.

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. Por 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. Por 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.

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 Fig. 6; 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 Fig. 6 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*.

Llegados a este punto, queda descrito en profundidad el simulador implementado. A continuación, se describirá el mecanismo utilizado para su evaluación y los resultados obtenidos.

IV. EVALUACIÓN

En la asignatura Arquitectura de Computadores, en la cual participan los autores de este trabajo, se estudia la arquitectura *Von Neumann* al completo, haciendo hincapié en la ejecución de instrucciones en el procesador. Puesto que la implementación de este simulador se llevó a cabo para satisfacer las necesidades docentes de esta asignatura, las funcionalidades que integra tienen relación directa con el temario impartido en ésta.

La parte del procesador de esta asignatura engloba casi dos meses de docencia presencial: seis sesiones teóricas de dos horas y cinco sesiones prácticas de dos horas. Lo que hace un total de 22 horas de las 60 que constituyen la asignatura. Aunque el simulador está a libre disposición del alumnado para que pueda trabajar con él desde casa e, incluso, utilizarlo para resolver ejercicios realizados en las sesiones teóricas, éste se utiliza de manera activa en las sesiones prácticas indicadas anteriormente.

En ellas, el alumno realiza una serie de ejercicios entregables utilizando el simulador para comprender el funcionamiento del paralelismo a nivel de instrucciones.

Descrito resumidamente el entorno en el que se utiliza el simulador, indicamos a continuación las métricas que se tienen en cuenta para evaluar la mejoría en el aprendizaje mediante el uso del simulador. Todas ellas, evalúan datos obtenidos durante 4 cursos académicos

consecutivos y aplicadas sobre una población que abarca 12 subgrupos prácticos en dos titulaciones diferentes (una media de más de 200 alumnos por año). Estas métricas son:

- Las calificaciones en las sesiones prácticas.
- Asistencia a las sesiones prácticas correspondientes.
- Encuestas de satisfacción del alumnado

A. Evolución de las calificaciones en las sesiones prácticas.

Hasta el curso académico 2014-15 se hacía uso de otro simulador que ayudaba en la explicación de una parte del contenido teórico. Esta herramienta software era muy limitada y por tanto no cubría todos los aspectos estudiados del procesador, por lo que era necesario complementarlo con un segundo simulador. El nuevo simulador integra todos los contenidos que el alumno debe estudiar, lo que facilita la comprensión del funcionamiento global del procesador al poder examinar todos los detalles de la simulación en un solo vistazo. La Fig. 7 muestra una gráfica con la evolución de las notas media obtenidas por los alumnos desde el año académico 2011-2012, 4 años antes de utilizar el nuevo simulador en las sesiones prácticas.

Fig. 7. Evolución de las calificaciones de prácticas.

Obteniendo la media de los cursos anteriores a la implantación de la herramienta VisualMIPS se obtiene una calificación general de 5.43, significativamente inferior a la media de los cuatro cursos donde se utilizó la herramienta (6.28).

B. Evolución de la Asistencia a las sesiones prácticas.

La asistencia a las sesiones de prácticas es un factor indispensable en este estudio, pues permite cuantificar el grado de motivación del alumnado y su capacidad para mantener su atención en la asignatura al trabajar con un simulador con una interfaz más atractiva.

Desde la implantación de los nuevos grados de Informática, existía un porcentaje preocupante de abandono en la asignatura, a pesar de que las prácticas eran obligatorias. Como se aprecia en la gráfica de la figura 8, antes de la utilización de VisualMips, la tasa de abandono era alta y con tendencia a aumentar cada año.

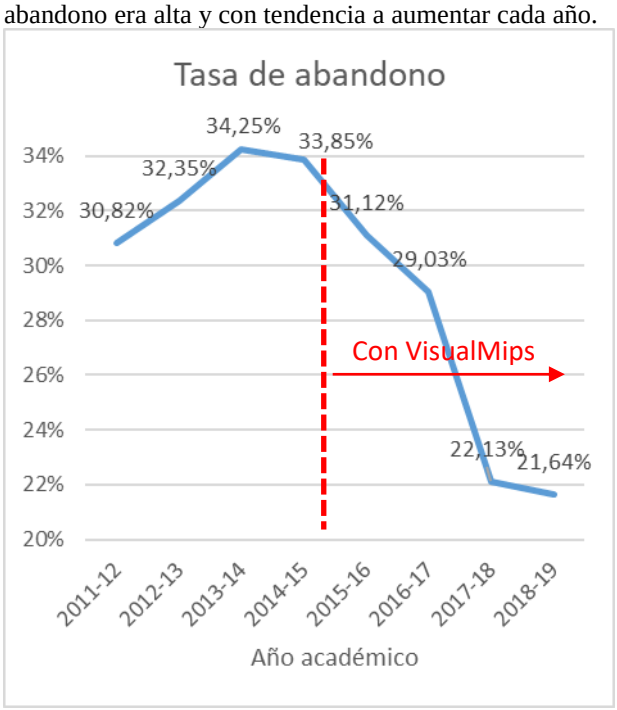

Fig. 8. Evolución del porcentaje de asistencia a prácticas.

Desde el año en el que se empezó a utilizar el nuevo simulador, la tasa de abandono ha descendido paulatinamente, a pesar de que la calificación de prácticas tiene un peso en la evaluación de la asignatura inferior que años anteriores.

C. Encuestas de satisfacción del alumnado.

Otro factor a tener en cuenta para cuantificar la usabilidad, la utilidad y la experiencia del usuario con la herramienta es la opinión de los usuarios que la usan. De manera subjetiva y anónima los alumnos realizan una encuesta personal en la que evalúan ciertos aspectos y aportan comentarios personales acerca de la herramienta. Para ello, al final de la última sesión práctica de procesadores, el alumnado rellenó una encuesta de 8 preguntas evaluadas de 1 a 8 cuyo enunciado son los siguientes:

1. Me ha ayudado a solventar dudas que poseía en las clases teóricas.
2. La notación de la que hace uso es intuitiva y está relacionada al 100% con el contenido teórico.
3. Posee todos los aspectos necesarios para comprender el funcionamiento del procesador (no le falta nada).
4. Sería interesante ampliar el simulador con módulos para acceso a memoria y entrada/salida.
5. El diseño de la aplicación es usable y fácilmente accesible para una persona que lo utilice por primera vez.
6. He hecho uso del simulador para comprobar los resultados de ciertos ejercicios teóricos.

7. El simulador utilizado me ha ayudado a comprender claramente el funcionamiento de la ejecución en pipeline.
8. En general, me parece una herramienta muy útil y completa.

La encuesta, al ser voluntaria, ha sido realizada por 133 alumnos entre los cursos 15-16, 16-17 y 17-18 (en este curso no se ha realizado aún) y se obtuvo la siguiente calificación en cada una de las preguntas:

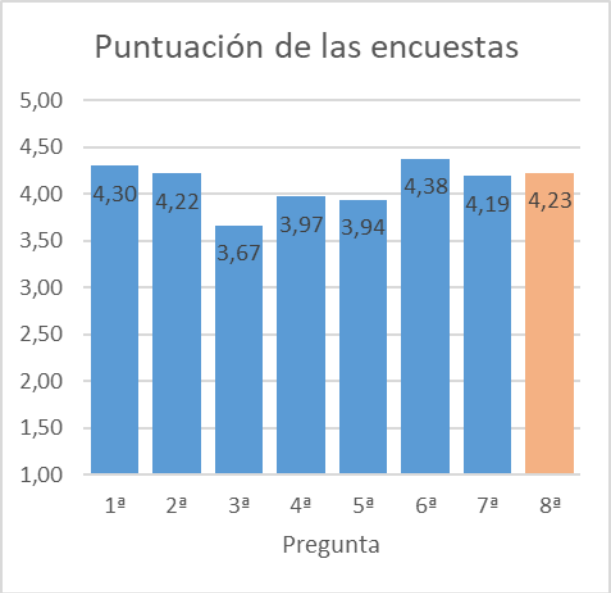

Fig.9. Resultado de las encuestas de los alumnos.

Como se aprecia en la gráfica de la figura 9, el grado de satisfacción del alumnado con la herramienta es alta, tanto en su efectividad para el aprendizaje (preguntas 1, 2, 3 y 7), como herramienta de apoyo a los ejercicios de clase (pregunta 6), o respecto a su usabilidad, claridad y sencillez (pregunta 5). De forma global obtiene una calificación de 4,23 sobre 5, por lo que se convierte en una herramienta idónea para el aprendizaje.

Además de las 8 preguntas evaluables, la encuesta incluye un apartado de observaciones para que el alumno pudiese expresar algún aspecto a destacar, sobre el que quiera incidir o que desee criticar de la herramienta. La mayoría de las observaciones de los alumnos van encaminadas a posibles mejoras o para indicar algunos errores que fueron detectados. Todos los errores y muchas de las mejoras fueron tenidas en cuenta en posteriores versiones.

V. MEJORAS FUTURAS

Desde el año 2015, en el que se implementó la primera versión del simulador, el programa está en constante evolución, corrigiéndose posibles errores, mejorando la interfaz gráfica, ampliando el repertorio de instrucciones, añadiendo nuevos elementos que forman parte del computador.

Las últimas versiones del simulador añaden la jerarquía de la memoria en fase beta para ser integrado al proyecto educativo en los próximos años.

VI. CONCLUSIONES

En este trabajo se presenta una herramienta software para facilitar el aprendizaje del paralelismo a nivel de instrucciones (ILP) en un procesador segmentado con juego de instrucciones RISC.

La herramienta es completamente configurable y ha demostrado, mediante encuestas y resultados, ser de gran utilidad en la enseñanza de Arquitectura de Computadores en los grados de Ingeniería Informática.

Los resultados presentados demuestran la reducción significativa de la tasa de abandono, así como la mejora de los resultados ligados a la parte en la que se centra la herramienta.

Es, además, una herramienta en continuo desarrollo y en uso en los diversos grados de Ingeniería Informática de la Universidad de Sevilla.

AGRADECIMIENTOS

Este trabajo ha sido desarrollado y financiado dentro del grupo de investigación TEP-108: Robótica y Tecnología de Computadores de la Universidad de Sevilla.

REFERENCIAS

[1] D. Patterson and J. Hennessy, “Computer Organization and Design : The Hardware / Software Interface,” *Comput. Organ. Des. Hardw. / Softw. Interface*, 2014.

[2] J. L. Hennessy and D. a Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. 2007.

[3] J. L. S. C. Pereira, “Educational package based on the MIPS architecture for FPGA platforms,” Faculdade de Engenharia da Universidade do Porto, 2009.

[4] A. Clements, “Undergraduate curriculum in computer architecture,” *IEEE Micro*, 2000.

[5] J. Djordjevic, B. Nikolic, B. Tanja, and A. MilenkoviĆ, “Cal2: Computer aided learning in computer architecture laboratory,” *Comput. Appl. Eng. Educ.*, 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,” *IEEE Transactions on Education*. 2009.

[7] H. Oztekin, F. Temurtas, and A. Gulbag, “BZK.SAU: Implementing a hardware and software-based computer architecture simulator for educational purpose,” in *2010 International Conference on Computer Design and Applications, ICCDA 2010*, 2010.

[8] B. Mustafa, “Modern Computer Architecture Teaching and Learning Support : An Experience in Evaluation,” in *Information Society (i-Society), 2011 International Conference on*, 2011.

[9] “MIPS32® Instruction Set Quick Reference, Revision 1.01.” [Online]. Available: http://www.imgtec.com/downloads/factsheets/MD00565-2B-MIPS32-QRC01.01.pdf.

[10] “MIPS® Architecture for Programmers Volume I-A: Introduction to the MIPS32® Architecture. Revision v6.01.” [Online]. Available: http://www.imgtec.com/mips/architec%0Atures/mips32.asp.

[11] “MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set, Revision 5.04.” [Online]. Available: http://www.imgtec.com/mips/architectures/mip%0As32.asp.