ISBN: 978-99920-70-10-9

Un simulador de arquitectura MIPS para el estudio del procesamiento paralelo de instrucciones

Belen Bermejo, Carlos Guerrero, Isaac Lera, Catalina Lladó
Departament de Matemàtiques i Informàtica
Universitat de les Illes Balears
07122 Palma de Mallorca
{belen.bermejo,carlos.guerrero,isaac.lera,cllado}@uib.es

Resumen

Presentamos una herramienta docente para la explicación del conjunto de técnicas de paralelismo a nivel de instrucción, conocido con el acrónimo en inglés de ILP. Hemos programado un diseño modular de la arquitectura MIPS que permite la ejecución segmentada de una instrucción, con el objetivo de ofrecer la posibilidad de comparar diferentes escenarios según los parámetros y elementos deseados. La herramienta ofrece un volcado de métricas de rendimiento para estudios comparativos y su programación orientada a objetos, relativamente sencilla, permite implementar nuevos módulos como algoritmos de especulación de salto o planificación de código. Además, adjuntamos unos recursos didácticos para mostrar su funcionalidad y potencial, junto con una valoración de experiencia docente en el uso de dicha herramienta por parte de los alumnos.

Abstract

We present a teaching tool for explaining the set of techniques for instruction level parallelism (ILP). We have designed a simulator of the MIPS architecture that allows the pipeline execution of instructions. The aim is to be able to compare different scenarios depending on input parameters and the inclusion of desired elements. The tool provides a set of performance metrics for comparative studies. Its object-oriented programming is also relatively simple, which allows implementing new modules as branch speculation algorithms or scheduling code. In addition, we include a set of teaching activities to show its functionality and potential. Finally, we describe the teaching experience of using this tool in a couple of courses.

Palabras clave

Recurso docente, Arquitectura MIPS, Paralelismo a Nivel de Instrucción.

1. Introducción

En este trabajo presentamos una herramienta educativa para mostrar y ejercitar en la práctica el conjunto de técnicas de paralelismo a nivel de instrucción. Este conocimiento común en la rama de informática se enmarca dentro de la capacidad de conocer, comprender y evaluar la estructura y arquitectura de los computadores. Específicamente, aunque dependiendo del centro, el nombre de la asignatura donde se imparte es Arquitectura de Computadores. Las técnicas de paralelismo a nivel de instrucción, llamadas en inglés *instruction level parallelism* (ILP) se utilizan para incrementar el grado de paralelismo en la ejecución segmentada de instrucciones. Los ciclos por instrucción de una arquitectura dependen de como se traten las dependencias estructurales, las dependencias entre datos y las de control [1]. El objetivo es incorporar funciones y estructuras para mitigar el efecto de las dependencias en el número de ciclos destinados a la ejecución de instrucciones. Entre estas técnicas podemos destacar el adelantamiento y anticipación de valores, la planificación estática y dinámica de código, la predicción de saltos con múltiples algoritmos de predicción, el grado de paralelismo, especulación, segmentación software, estaciones de reserva, buffer de reorden, bus de datos común, etc.

Creemos que es importante a nivel docente explicar y promover actividades sobre cada una de estas técnicas, aisladamente o en combinación con otras, para la comprensión de un grupo de decisiones que cambiaron los cimientos en el diseño de procesadores. Vimos la necesidad de crear y diseñar un simulador de la arquitectura MIPS con soporte a tal combinación de técnicas y, en donde las diferentes fases de segmentación se adapten a la activación y parametrización de las mismas. Otro objetivo ambicioso es la difusión de esta herramienta a nivel nacional e internacional para que puede ser usada en el ámbito educativo. Por ello, se ha liberado el código y el conjunto de actividades docentes complementarias con la herramienta en un re-

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

positorio público. Cualquiera puede usar y colaborar en incrementar las prestaciones del simulador. Una posible actividad de mejora es la capacidad de implementar nuevos algoritmos de predicción de saltos y evaluar su rendimiento; el simulador tiene una clase abstracta de planificador donde pueden implementarse algoritmos específicos. En este sentido, otro de los objetivos es que el alumno pueda programar extensiones y alternativas a los diferentes módulos, es decir, que sean capaces de diseñar arquitecturas y tenga la capacidad de evaluarlas.

Para finalizar el trabajo exponemos una pequeña valoración personal en el uso del simulador en dos asignaturas diferentes: tercero de grado en informática y en el máster de ingeniería en informática.

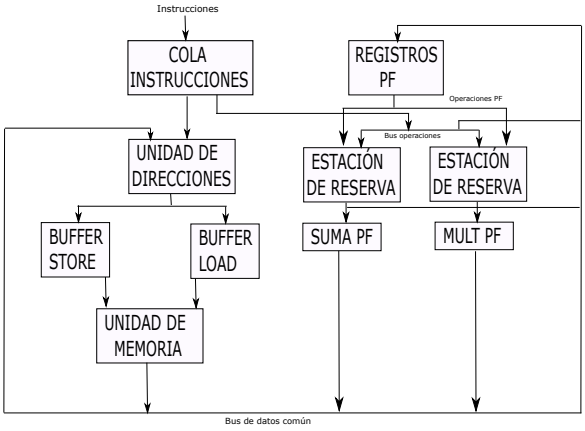

Figura 1: Estructura básica con unidades de punto flotante MIPS usando el algoritmo de Tomasulo [1].

2. Contexto

El objetivo de la segmentación y de las técnicas ILP es maximizar los recursos y solapar la ejecución de un mayor número de instrucciones manteniendo las excepciones bajo control y, por ende, la coherencia del resultado. Por tanto, se pretende disminuir el número de interrupciones que penalicen la carga y la parada de instrucciones. La mayor causa se debe a las dependencias entre datos: dependencias reales (*Read after Write*, RAW) , cuando el operando de una instrucción depende del resultado de otra instrucción previa, y falsas dependencias (WAR, WAW). Otra causa de interrupciones son las dependencias de control causadas por instrucciones de salto. Éstas actualizan el *contador del programa* con un nuevo índice pero para evitar paradas innecesarias hasta que se calcula ese valor se puede decidir por seguir ejecutando especulativamente una rama; y si en algún punto se falla sobre la predicción, hay que deshacer los resultados intermedios. Por último, están las dependencias estructurales causadas por las características de los elementos: cantidad, capacidad, puertos de entrada y de salida, elementos compartidos o individualizados, etc. Todos los componentes de la arquitectura se pueden sintetizar en el algoritmo de Tomasulo (ver figura 1), sin incluir las correspondientes adaptaciones como el buffer de reorden (ROB) cuando se incorpora ejecución especulativa en la predicción de saltos.

Los elementos y su parametrización quedan definidos de la siguiente manera: el grado de escalabilidad, número de registros para el renombre, estaciones de reserva, tipos, capacidad y *bypassing*; unidades funcionales, tipos, capacidad; capacidad del bus de datos común; tamaño de la ROB; especulación de saltos y algoritmos; planificación estática y dinámica de código.

El número de combinaciones es elevado y las diferentes propuestas ofrecen un repertorio amplio de casos y ejercicios. A nivel académico es interesante analizar el rendimiento de la arquitectura, la evolución de los elementos según la existencia de estos y sus correspondientes parámetros y el flujo de ejecución de las instrucciones.

3. Trabajos relacionados

Existen varias herramientas que simulan con mayor o menor flexibilidad los componentes de una arquitectura MIPS.

DLXview fue uno de los simuladores más utilizados desarrollado en Purdue University. Es un simulador de la segmentación de un repertorio de instrucciones DLX, simulando tres versiones diferentes del flujo de instrucciones: básico, *scoreboard* y Tomasulo. Esto posibilita la parametrización estructural, la capacidad, de los componentes en cada tipo de flujo; pero los componentes han de estar. Es decir, no hay capacidad de desactivar el tipo de estación de reserva, u otros detalles, como la existencia de la ROB durante la especulación. De la herramienta, destacamos la interfaz gráfica con el flujo de instrucciones donde se puede apreciar el estado de la ejecución de cada instrucción. En la web oficial no hay ningún enlace valido para su descarga y hay que recurrir a otras fuentes donde aún se conserva alguna versión[1]. Las versiones disponibles - desconociendo si son las últimas- son poco fiables pues presentan fallos graves a la hora de guardar documentos y se cierran inesperadamente.

MIPS Assembler and Runtine Simulator (MARS) [3] es una aplicación bajo licencia MIT (*open-source*) mantenido por la Missouri State University. Es multiplataforma, basado en Java, con una interfaz gráfica elaborada donde se visualizan los valores de los registros, un editor de código y

[1] `http://goo.gl/KKd5tR`

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

diferentes opciones de ejecución. MARS contiene algunos complementos como: una tabla de histórica de saltos (BHT), la carga de las unidades funcionales, estado de la caché y su organización, etc. Consideramos que es una herramienta completa con un diseño muy funcional. En cambio, no permite la elección y parametrización de diferentes unidades funcionales, estaciones de reserva, otros registros de renombre, algoritmos de predicción, etc. Además, el código es complejo y está estrechamente relacionado con la interfaz gráfica. Su última actualización data de agosto del 2014. A grandes rasgos, MARS está enfocado a la ejecución y depuración de código ensamblador.

QtSpim[2] es la nueva versión del simulador SPIM, bajo licencia BSD. Es un simulador para la ejecución de código MIPS32 con una interfaz gráfica similar al MARS pero con menor funcionalidad. Pero al igual que éste, la parametrización de los diferentes elementos es inviable, sirviendo sólo para ejecutar y depurar código. En los repositorios la última modificación data del 2013.

La herramienta *Basic MIPS Simulator in JavaScript (Accurate Simulation)*[3] está desarrollada por un estudiante de la Washington University in St. Louis. Destacamos su entorno de ejecución basado en javascript, pudiendo ejecutarlo en cualquier navegador. De esta manera, no requiere de ninguna descarga, ni de configuraciones de librerías, ni el servidor web ha de tener ciertas características técnicas. Su funcionamiento nos abre una vía alternativa en el desarrollo futuro pues consideramos muy interesante poder explicar conceptos teóricos y que rápidamente se puedan aplicar desde el navegador, proyector o incluso en el móvil facilitando la dinámica de la clase.

En la tabla 1 mostramos una serie de requisitos deseables en cualquier herramienta docente de los simuladores. Nos interesa desde el punto de vista académico que los alumnos sean capaces de modificar y añadir fases a nivel de programación. Por lo que el código ha de poseer las correspondientes licencias de uso y modificación; más ha de ser fácil de interpretar y complementar por un alumno de grado. El material complementario no sólo ha de contener ejemplos de código para la ejecución sino ejercicios docentes: comparativas entre múltiples configuraciones para mostrar el avance tecnológico que supusieron. La flexibilidad de añadir múltiples configuraciones. La última característica hace referencia al soporte, refiriéndonos a la disponibilidad del código, ejecutable y del material en algún repositorio público. Ninguno de estos simuladores cumple con estos requerimientos

[2] http://spimsimulator.sourceforge.net/

[3] http://morriswmz.jit.su/static/simple-mips-pipelined.html

Simulador	Licencia	Material	Flexible	Soporte
MARS	MIT	✗	✗	✓
QtSPIM	BSD	✗	✗	✗
DLXview	✗	✓	✓	✗
MIPS JS.	✗	✗	✗	✗

Cuadro 1: Características de los simuladores analizados

Figura 2: Fases de la simulación

4. Arquitectura de la aplicación

La aplicación permite la configuración de la arquitectura a partir de tres ficheros de entrada que contienen el código fuente a ejecutar, el valor inicial de los registros de propósito general y los parámetros propios de una arquitectura segmentada. El código fuente a ejecutar se describirá en el lenguaje de programación diseñado para tal fin a partir de la modificación del propuesto por Kane [2].

Los pasos de la arquitectura están representados en la figura 2. Los parámetros configurables de la arquitectura son la dirección de inicio y fin de la memoria de datos, la latencia de cada una de las fases de la segmentación expresada en ciclos, el algoritmo de planificación que se quiere ejecutar sobre las instrucciones, el número de registro de renombre, la escalabilidad del procesador, la presencia de ROB, así como su tamaño expresado en instrucciones y la velocidad de retirada de las mismas expresada en instrucciones por ciclo. También se permite configurar el número de unidades funcionales de propósito general y de propósito específico, así como la latencia de cada una de ellas expresada en ciclos. Una vez se ha definido la configuración pertinente, se procede a la creación de la arquitectura definida y se realiza el análisis del código fuente, a fin de que las instrucciones sean correctas y no haya inconsistencias. El siguiente paso es la ejecución segmentada de las instrucciones. Esta fase se compone de

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

los siguientes módulos: un módulo de unidades funcionales de propósito general y específico, así como el correspondiente controlador, el cual permite o deniega el acceso de instrucciones a dichas unidades. También se dispone del módulo de la ejecución segmentada, el cual está compuesto por cada una de las fases propias de la arquitectura MIPS y del módulo ROB, el cual permite la ejecución fuera de orden de las instrucciones. En cuanto al almacenamiento, se disponen de una serie de registros de propósito general los cuales pueden ser utilizados por el programador del código fuente a ejecutar, así como los correspondientes registros intermedios. La memoria está dividida en dos: datos e instrucciones, las cuales son accedidas por las distintas fases de la segmentación.

Sobre la arquitectura descrita correrán los diferentes programas bajo la configuración definida. Desde que las instrucciones son escritas por el programador, hasta que finalizan pasan por una serie de etapas: análisis de dependencias, planificación y ejecución segmentada.

En primer lugar se realiza un análisis de las dependencias existentes entre las instrucciones. Se detectan las dependencias reales y no reales, generando un grafo que se deposita en un fichero de texto. A partir del grafo generado se da paso a la planificación de las instrucciones. Para facilitar cualquier algoritmo de planificación (planificación de lista, camino más corto según la latencia de las unidades funcionales, etc.), se ha implementado una clase llamada *Scheduler*. Desde esa clase se tiene acceso a toda la configuración de la arquitectura, ya sea a la definida por el usuario como al grafo de dependencias generado. El acceso a la configuración de la arquitectura, se hará mediante la llamada a las funciones estáticas de la clase *ConfigFile*.

Una vez se han planificado las instrucciones, se procede a la ejecución segmentada de las mismas. El módulo de segmentación puede estar en dos estados: con ROB o sin ROB, haciendo que la presencia de ROB añada una fase más a la segmentación, la fase de *commit*. La clase *BranchManager* se encarga de la resolución de los saltos, en función de la arquitectura definida y el código fuente a ejecutar. Para hacer uso de este módulo se invocará un objeto de dicha clase con la instrucción de salto a revolver por parámetro. Este hecho permite que los saltos puedan ser resueltos en cualquiera de las fases de la segmentación; las cuales se explican a continuación:

4.1. Fase de *Fetch*

En esta fase se capturan las instrucciones de la memoria de instrucciones. En caso de presencia de ROB, las instrucciones capturadas -en función del grado de escalabilidad- serán introducidas en la ROB. Si la ROB no dispone del espacio suficiente, se detendrá dicha fase hasta que se disponga de espacio. En el caso de no tener ROB, la resolución de los saltos se realiza en esta fase, pero como ya se ha comentado, este hecho es modificable. Mientras se está resolviendo el salto, la ejecución se paraliza.

4.2. Fase de *Decode*

Para cualquier configuración de la arquitectura se ejecutarán los siguientes pasos. En primer lugar se comprobará la existencia de dependencias reales entre la instrucción en cuestión y una predecesora. En caso afirmativo, el caudal de la segmentación se parará hasta que dicha dependencia sea resuelta. Superada esta restricción, se asignará a la instrucción el tipo de unidad funcional en la cual se ejecutará. Este hecho da pie a que se pueda implementar la estación de reserva. Para ello se crea una clase nueva *ReserveStation*, desde la cual se tendrá acceso a la configuración de la arquitectura mediante la clase *ConfigFile* y al estado de cada ciclo de ejecución de la segmentación a partir de la información de cada objeto *Instruction* y la clase *Pipeline*. Al disponer de varias unidades funcionales y diversos tipos, una vez se haya implementado una estación de reserva -o las que convenga-, se podría desarrollar un bus de datos común, pudiendo dar pie a la del algoritmo de Tomasulo.

4.3. Fase de *Execute*

Una vez decodificada la instrucción se pasa a la ejecución de la misma. En caso de no tener ROB se ejecutará la operación correspondiente. Pero en caso contrario, si la instrucción es un salto, éste se resolverá en esta fase. Si se ha producido una ejecución especulativa, se actualizará el estado de la máquina al punto correspondiente al salto y se ejecutará la operación asociada. Además, en esta fase entra en juego el controlador de las unidades funcionales, el cual deniega o da acceso a las instrucciones al tipo de unidad funcional asignada en la fase de decodificación. Cuando una instrucción quiere ser ejecutada en una unidad funcional, ésta es asignada a la primera unidad funcional disponible. En caso de no disponer de unidades libres en un momento concreto, la instrucción permanecerá en espera. La política de asignación de una instrucción a una unidad funcional también puede ser modificada, permitiendo así la comparativa de diferentes políticas. Para implementar otra política de asignación, se deberá modificar la función *get* de la clase *FUManager*. Esta función requiere el tipo de unidad funcional en la que se deberá ejecutar la instrucción. Desde esta clase se tiene acceso, de nuevo, a la configuración de la arquitectura y se podrán hacer pronósticos de utilización de las unidades funciones, dando pie a la implementación de políticas de asignación de recursos basadas en la optimización de la utilización de dichas unidades funcionales vistas

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

durante otras asignaturas del grado. Por otra parte, también es posible implementar más unidades funcionales de propósito específico.

4.4. Fase de *Memory*

En esta fase, las instrucciones que requieran de un acceso a la memoria de datos lo realizarán, tanto si hay, o no, presencia de ROB. A la hora de almacenar un valor en la memoria de datos se realiza de forma secuencial en la palabra correspondiente. Al igual que en la fase anterior, la política de alojamiento de las variables en la memoria de datos podrá ser modificada, de forma que se podría optimizar el uso de la memoria en función de su tamaño. Para modificar dicha política es necesario modificar la función *addVariable* de la clase *Memory*. Desde esta clase se tiene acceso a la configuración de la arquitectura y a las variables declaradas en el código fuente.

4.5. Fase de *Write*

En esta etapa, las instrucciones harán el volcado del resultado de la fase de ejecución y memoria en los registros destino correspondiente. En el caso de disponer de ROB, la escritura se realizará en la estructura de registros intermedios, y en caso contrario en los registros de propósito general.

4.6. Fase de *Commit*

Esta fase solo se ejecutará si la arquitectura está dotada con una ROB. El objetivo de la ROB es mantener la consistencia en el orden de finalización de las instrucciones. La política de ejecución y finalización de las instrucciones implementada por defecto es "lanzar en orden, acabar en orden", pero dicha política podría ser modificada, permitiendo lanzar y terminar las instrucciones fuera de orden, o cualquiera de las combinaciones. Para ello, se deberá de modificar la función *doCommit* de la clase ROB. Al igual que en el resto de casos, desde esta clase se tiene acceso a la configuración de la arquitectura y al código fuente, así como al estado de la segmentación en cada ciclo.

4.7. Generación de resultados

Uno de los objetivos de la implementación de esta herramienta, es poder comparar diferentes escenarios basándonos en los resultados obtenidos de la ejecución de un programa respecto a una configuración concreta. Por este motivo, una vez se ha finalizado la ejecución segmentada, se da paso al cálculo de las medidas de rendimiento. Para cada una de las ejecuciones se muestra el orden final de las instrucciones decidido por el planificador, el porcentaje de instrucciones ejecutadas

```
/ Main program - this a coment
/Variables for main
.data
age_ .word -256 /decimal
gpa_ .half 0xA02 /hex
nl_ .byte 0%12 /octal
bin_ .word 0b11000111100001111
010010101011111 /binary

/Main body
.text
LWR r30,#150
ADD r2,r3,r4
SB r30,gpa
```

Figura 3: Código de ejemplo

de cada tipo (computacional, de salto o de memoria), el porcentaje de utilización de cada una de las unidades funcionales, el CPI (ciclos por instrucción), el número de saltos ejecutados, el número de accesos a memoria y el diagrama temporal resultante de la ejecución, donde se muestra para cada uno de los ciclos en qué fase se encuentra cada instrucción. Estos resultados son volcados en el fichero de salida.

5. Manual de usuario

Junto con la herramienta se adjuntan una serie de ficheros que contienen la configuración de la aplicación. En el fichero *reg.properties* se ha de indicar el valor inicial de cada uno de los registros, siempre en formato decimal. Los valores admisibles serán los correspondientes al rango de valores con signo de 32 bits. Por otra parte, se deberá crear un fichero de extensión .txt que contenga el código fuente a ejecutar, por ejemplo el mostrado en la figura 3.

Además, se deberá dar valor a los parámetros de la arquitectura, los cuales están contenidos en el fichero *config.properties* y tendrán que seguir las siguientes restricciones de valor. El fichero con el código fuente ha de existir en el sistema, el número de bits de la arquitectura ha de ser 32, las direcciones de inicio y fin de la memoria de datos comprenderán el rango de números enteros, al igual que el número de registros de propósito general, la latencia de cada una de las fases de segmentación, el número de registros de renombre, la escalabilidad del procesador, el tamaño y la velocidad de la ROB, el número de unidades funcionales y su latencia. Por otra parte, el tipo de planificación se indicará con el nombre del algoritmo implementado y la presencia de ROB o no con 'y' o 'n' respectivamente.

Se han realizado diferentes pruebas de ejecución mediante un generador de código aleatorio que pue-

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

de configurarse al gusto. Su ejecución puede realizarse desde cualquier entorno de desarrollo de Java o bien a partir del ejecutable generado tras la compilación del código fuente. De la ejecución se obtendrán como salida los siguientes ficheros: *Dependence.rtf,* el cual tiene el grafo de dependencias, *SimResult.txt*, el cual contiene las medias de rendimiento de la ejecución, *schedResult.txt*, que dispone del orden final de las instrucciones tras su planificación y *code.txt*, donde se almacena el código fuente generado de forma aleatoria. Todos estos archivos pueden ser consultados y utilizados como entrada para otras posibles aplicaciones que el usuario disponga.

La herramienta y las actividades complementarias están disponibles en un repositorio GitHub en: `https://github.com/belen20/MIPS-Simulation-Environment`

6. Actividades complementarias

Mediante el uso del simulador, se podrá trabajar en clase algunos aspectos de los procesadores ILP de forma más amplia y profunda. Para ello será interesante el estudio comparativo entre distintas configuraciones de una arquitectura ILP, haciendo al alumno que reflexione sobre las ventajas e inconvenientes de cada una de esas configuraciones antes de llevar a cabo la ejecución en el simulador. Y una vez obtenidos los datos de la ejecución con el simulador, hacerle comparar y reflexionar sobre sus conclusiones iniciales. Hemos adjuntado una serie de actividades en el repositorio. A continuación pasamos a comentar algunos aspectos comparativos que se pueden estudiar.

En primer lugar, podría ser interesante comparar como la técnica de renombre de registros permite la aceleración de la ejecución de instrucciones ya que evitará el bloqueo de instrucciones que tengan falsas dependencias y dependencias de salidas. Puede ser muy interesante comparar el beneficio de la utilización del renombre en un código en el que haya muchas falsas dependencias, con otro en el que no existan. De esta forma, se buscará que el alumno sea capaz de detectar ambos casos y que sea capaz de ver en cual de ellos se obtendrá una mayor mejora.

Igual de interesante que el análisis de la utilización de las técnicas de renombre de registro, puede ser el caso de estudiar la utilización del ROB, incluso en combinación con la técnica anterior. De esta forma el alumno deberá reflexionar sobre los tipos de bloqueos producidos en el código, y cuando la utilización de un lanzamiento fuera de orden, habilitado por la utilización del ROB, mejorará el rendimiento del procesador.

El simulador también puede ser muy útil para el análisis de técnicas de planificación y de desarrollo de bucles. La idea sería buscar ejemplos de códigos que mejoren el tiempo de ejecución una vez desarrollados y planificados, con otros que no muestren dicha mejora, por ejemplo, porque dispongan de un número menor de dependencias reales. Igualmente, son dos escenarios adecuados para mostrar como, de nuevo, el uso de ROB y de renombre también influirá en los tiempos de ejecución.

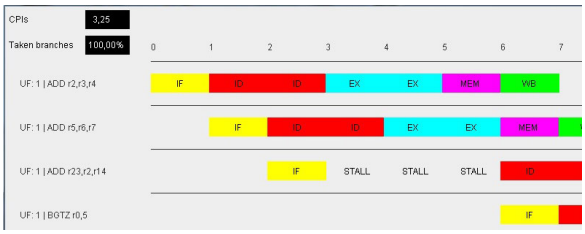

Figura 4: Interfaz desarrollada por un grupo de alumnos usando el API del simulador.

De esta forma, se conseguirá que el alumno trabaje su capacidad de análisis y de abstracción. Haciéndole analizar casos antes de llevar a cabo el análisis de los resultados en el simulador.

7. Experiencia docente

El simulador de la arquitectura MIPS descrito ha sido usado por primera vez como herramienta docente en el primer cuatrimestre del curso 2014-2015 en la asignatura Arquitectura de Computadores del tercer curso del Grado de Informática, ya que esta asignatura contempla el estudio de las arquitecturas segmentadas como uno de sus temas principales. El diseño modular del simulador ha permitido ofrecer a los estudiantes el entorno de simulación *incompleto* para que sean ellos los que tengan que completar la parte directamente vinculada con la asignatura. Así, se les ha ofrecido un conjunto de librerías documentadas para que puedan usarlas en la implementación de las clases *Pipeline* y *Branch manager*, descritas anteriormente. En la clase *Pipeline* deberán implementar las diferentes tareas de cada una de las etapas de segmentación usando, entre otros, el módulo de análisis de dependencias para detectar cuando se deben insertar paradas debidas a las dependencias reales. Es importante destacar que el número de ciclos de duración o latencia de cada una de las etapas debe de ser configurable así como el valor de escalabilidad, que puede ser mayor que uno (diferentes tuberías de segmentación). Finalmente, una vez ejecutado el código, deberán mostrar el diagrama temporal de la ejecución de las instrucciones (la figura 4 muestra el diagrama presentado por uno de los grupos de prácticas de este curso), así como el CPI medio y el porcentaje de saltos realizados.

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

La experiencia de la realización de dicha práctica fue muy positiva ya que generó muchas discusiones en relación a la gestión de la segmentación que los estudiantes simplemente no se plantean en una clase teórica o incluso de ejercicios en la pizarra. La implementación concreta de algunos casos depende de las suposiciones realizadas del funcionamiento e implementación de la arquitectura y diferentes grupos de prácticas realizaron diferentes suposiciones. Esto permitió mostrar a toda la clase los diferentes resultados obtenidos cuando se usaban cada una de ellas. Finalmente, una vez corregidos los exámenes, se ha podido comprobar el conocimiento de todos los alumnos del tema de la segmentación ya que, un $90\,\%$ tendrían una calificación $>= 6$ (sobre 10) de los ejercicios de segmentación, con una media de $6,2$ de dichos ejercicios, claramente relacionado con la realización de la práctica (el $100\,\%$ de los alumnos presentó la práctica y obtuvo una calificación de esta $>= 7$ (sobre 10)). Las calificaciones de otros años no están disponibles en relación a los diferentes temas de la asignatura, así que no podemos hacer comparaciones en este sentido.

Por otra parte, el simulador también ha sido usado como herramienta docente de la asignatura Informática de altas prestaciones del máster de ingeniería en informática. En este caso un $78\,\%$ de los alumnos presentó la practica en la convocatoria de febrero con una calificación $>= 8$.En este caso, en el examen final había una cuestión larga relacionada directamente con la practica cuya respuesta fue puntuada $>= 7$ a todos los alumnos que habían entregado la práctica y $<= 2$ a los que no la habían entregado, cosa que relaciona claramente ambas cosas. Éste ha sido el primer curso en el que se ha impartido esta asignatura, así que no se pueden hacer comparaciones con otros años. En futuros cursos se irán haciendo diferentes usos de la herramienta de prácticas y se estudiará su repercusión en las calificaciones de cuestiones relacionadas en el examen.

8. Conclusiones

La evolución histórica de las diferentes fases de la segmentación durante la ejecución de instrucciones y el paralelismo a nivel de instrucción son conceptos con un gran potencial docente para fomentar el espíritu crítico de un futuro ingeniero. En ese sentido, nuestro diseño modular de un simulador de la arquitectura MIPS facilita la comprensión e interpretación de las técnicas básicas de ILP implicadas. Además, el repertorio de actividades docentes mediante cambios en las configuraciones e implementaciones incrementa la utilidad de la herramienta. Se pueden hacer estudios para valorar la aceleración del CPI en función del tipo de instrucciones y dependencias en el código y, también, por dependencias de control y estructurales.

Esta herramienta destaca del resto por ser parametrizable en la activación y configuración de elementos e incluso altera las fases si hay o no presencia de dichos elementos. Un claro ejemplo de tal hecho, se produce con la implementación por defecto, de la incorporación de la ROB. Respecto a la implementación de alternativas, hemos pretendido crear un código intuitivo en Java con un nivel similar al de alumnos de grado. La interfaz es básica para no añadir complejidad al código. Lo cual no imposibilita proyectos paralelos para la visualización, tal como hicieron alumnos durante las prácticas por iniciativa propia. Las pruebas, aunque no son suficientes, ya han demostrado un aumento significativo de comprensión respecto al grupo de control del curso anterior.

Como es un proyecto con fines docentes, más deseamos generar un grupo de trabajo mayor con el poder mejorar el repertorio de servicios y actividades del simulador, el código está disponible en un repositorio público[4].

Como futuro trabajo, pretendemos simplificar el entramado de clases y funciones. No consideramos que el número de clases sea elevado pero hemos de simplificar el tipado de variables para facilitar la visibilidad y, por ende, la creación de nuevos elementos. Pretendemos incluir soporte a *plugins* con lo que no habrá necesidad de compilar el proyecto global y, por último, ofrecer una implementación alternativa en javascript dando cabida a resoluciones inmediatas de ejercicios propuestos en clase.

Agradecimientos

Trabajo cofinanciado por el Departamento de Matemáticas y por el proyecto ”Anàlisis i estudi de les eines de gamificació per a la seva utilització en estudis universitaris” del Vicerrectorado de Docencia y Calidad de la Universitat de les Illes Balears.

Referencias

[1] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[2] Gerry Kane and Joe Heinrich. *MIPS RISC Architectures*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[3] Kenneth Vollmar and Pete Sanderson. MARS: An Education-oriented MIPS Assembly Language Simulator. *SIGCSE Bull.*, 38(1):239–243, March 2006.

[4] https://github.com/belen20/MIPS-Simulation-Environment