ReVisión
Revista de Investigación en Docencia Universitaria de la Informática
AENUI Asociación de Enseñantes Universitarios de la Informática

Artículo invitado

Utilizando ARMSim y QtARMSim para la docencia de Arquitectura de Computadores

Sergio Barrachina Mir, Germán Fabregat Llueca,
Juan Carlos Fernández Fernández, Germán León Navarro
Dpto. de Ingeniería y Ciencia de los Computadores
Universitat Jaume I
Castellón
barrachi@uji.es, fabregat@uji.es, jfernand@uji.es, leon@uji.es

Resumen

Muchos de los objetivos formativos de las asignaturas de introducción a la Arquitectura de Computadores se centran en aquellos aspectos que conforman la visión que un programador en lenguaje ensamblador tiene de un computador. Por regla general, para definir dichos objetivos se suele utilizar una arquitectura de computador concreta, que normalmente se selecciona con el doble criterio de que sea lo más sencilla posible y, a la vez, motive al estudiantado.

La arquitectura ARM es una candidata idónea como vehículo conductor en la docencia de Arquitectura de Computadores. Por un lado, al estar basada en la arquitectura RISC (*Reduced Instruction Set Computer*), es relativamente sencilla. Por otro, se trata de una arquitectura actual y ampliamente difundida (especialmente en dispositivos móviles, *smartphones* y tabletas), lo que motiva al estudiantado.

Para poder realizar prácticas sobre ARM es conveniente disponer de un simulador o de una herramienta de desarrollo sobre una máquina ARM. Puesto que dicha materia se explica en los primeros cursos, conviene que la aplicación seleccionada sea sencilla de utilizar y lo suficientemente flexible. Por otro lado, conviene que sea *software* libre, para poder adaptarla en caso necesario, y también multiplataforma y gratuita, para facilitar que el estudiante que lo desee pueda instalarla en su propio equipo. Tras evaluar distintas opciones, finalmente se optó por desarrollar y liberar un simulador propio de ARM, ARMSim, y una interfaz gráfica para dicho simulador, QtARMSim.

El motor de simulación, ARMSim, y su interfaz, QtARMSim, han sido utilizados durante el curso 2014–15. Las críticas recibidas, tanto por los estudiantes como por los profesores de laboratorio, han sido muy positivas.

Palabras clave: Computadores y educación, Ensañanza asistida por computador, Arquitectura de Computadores, ARM Thumb, Simulador, Ensamblador.

1. Motivación

Históricamente, el contenido de los cursos de Arquitectura de Computadores ha seguido el frenético ritmo marcado primero por los avances tecnológicos y arquitectónicos en el diseño de grandes computadores, y a partir de los 80, por la evolución en el diseño de los microprocesadores. Así pues, se puede decir que la docencia en Arquitectura de Computadores ha pasado por seis grandes eras: *mainframes*, minicomputadores, primeros microprocesadores, microprocesadores, RISC y post RISC. Cada vez que las universidades han podido acceder a *hardware* específico a un coste razonable, este ha sido utilizado ampliamente como material de referencia. Algunas de las máquinas y microprocesadores que han disfrutado de una mayor popularidad en la docencia de Arquitectura de Computadores han sido: el PDP-11, el 68000 de Motorola, el 80x86 de Intel, el MIPS y el SPARC. Aunque en ocasiones también se ha recurrido a computadores hipotéticos dedicados en exclusiva a la enseñanza de los conceptos arquitectónicos [6].

Las asignaturas relacionadas con la Arquitectura de Computadores de la Universitat Jaume I también han pasado por varias de las opciones indicadas: se comenzó con el

68000 de Motorola, más adelante se utilizó brevemente un computador hipotético y posteriormente se cambió a la arquitectura MIPS, que ha sido la utilizada hasta el curso 2013–14. A principios de 2013 los profesores implicados en la docencia de dichas asignaturas se plantearon cambiar de la arquitectura MIPS a la ARM por los motivos que se enumeran a continuación.

La arquitectura ARM presenta muchas características que la distinguen de otras arquitecturas contemporáneas y, por otro lado, al estar basada en RISC, es relativamente sencilla [4]. Además, el hecho de que ARM sea una arquitectura actual y ampliamente difundida, especialmente en dispositivos móviles, *smartphones* y tabletas, es un factor especialmente motivador para el estudiantado [5].

Una vez tomada la decisión de realizar dicho cambio, se comenzó a redefinir las guías docentes y qué materiales se deberían utilizar en la enseñanza tanto teórica como práctica de las distintas asignaturas relacionadas con la materia de Arquitectura de Computadores.

En el caso de la asignatura *Estructura de Computadores*, del primer semestre del primer curso, uno de los primeros objetivos fue seleccionar un simulador o entorno de ARM que pudiera utilizarse en las prácticas de la asignatura. Había que tener en cuenta, además, que otras asignaturas iban a utilizar la plataforma Arduino Due [3], que está dotada de un microprocesador que implementa la arquitectura ARM Thumb 2.

Hasta ese momento, en las prácticas de dicha asignatura [1] se utilizaba el simulador de la arquitectura MIPS, QtSpim,[1] antes llamado *xspim* en su versión para GNU/Linux. El simulador *xspim* nos parecía un entorno adecuado para las prácticas de la asignatura debido a que: (1) permitía simular la ejecución de programas en ensamblador y visualizar el contenido de los registros y la memoria de forma sencilla, y (2) era *software* libre, gratuito y multiplataforma.

Debido a que *xspim* era lo suficientemente sencillo, los estudiantes podían centrarse en el desarrollo y simulación de los programas, más que en el manejo del propio simulador. Además, al ser *software* libre, gratuito y multiplataforma, los estudiantes podían descargarlo gratuitamente para realizar prácticas por su cuenta en sus propios ordenadores.

La primera de las opciones que se consideró para sustituir a *xspim* por un entorno para ARM, fue el entorno integrado de desarrollo y simulación μVision de la empresa Keil.[2] Dicho entorno es uno de los más utilizados en la programación de microcontroladores basados en ARM (la empresa Keil Software fue adquirida en 2005 por ARM) y se proporciona como ejemplo en libros sobre ARM [7, 8]. Incluye compiladores de C y C++, ensamblador, enlazador, depurador y simuladores de varios microcontroladores basados en ARM.

Adoptar μVision hubiera proporcionado un entorno realista de programación de sistemas empotrados. Sin embargo, su manejo hubiera sido ligeramente más complejo de lo que se quería para una asignatura de primero en la que se pretende explicar los fundamentos de la arquitectura más que la programación de sistemas empotrados. Por otro lado, solo hay versión para Windows, no es *software* libre y sí que es gratuito pero con ciertas limitaciones: 32 KB de código (lo cual no hubiera presentado ningún problema) y requiere registrarse para su descarga (lo que obligaría a los estudiantes a dar sus datos personales para poder disponer de él).

Relacionado con dicho IDE, también se evaluó la adquisición del producto *Lab-in-a-Box* de ARM University, que incluye[3]: (1) licencias de Keil *MDK-ARM Professional Software Tool* (del que forma parte el IDE μVision); (2) tarjetas Freescale basadas en el procesador ARM Cortex-M0+; y (3) material didáctico, de formación y de prácticas. Una vez se constató que el curso se centraba exclusivamente en la programación en C de sistemas empotrados, se descartó como opción válida para esta asignatura.

La siguiente alternativa que se consideró fue la de utilizar Eclipse DS-5.[4] Así como μVision está orientado a los sistemas empotrados, Eclipse DS-5 es el entorno de desarrollo que proporciona ARM para procesadores ARM y dispositivos System-on-Chip. Permite programar en C/C++ y en ensamblador, así como depurar el código que se esté ejecutando en el computador para el que se desarrolla.

Aunque la versión habitual de Eclipse DS-5 no es ni libre ni gratuita, existe una versión llamada Community Edition que sí que lo es. Además, al tratarse en realidad de un *plugin* para Eclipse (desarrollado en Java), se trata de una solución multiplataforma.

Sin embargo, uno de las primeros puntos débiles que se detectó es que no proporciona un entorno de simulación propiamente dicho. No obstante, es posible utilizar las opciones de depuración para emular un entorno de simulación. Basta con crear una máquina virtual ARM (por ejemplo, con QEMU), instalar un sistema operativo GNU/Linux para ARM sobre dicha máquina virtual y configurar dicha máquina y Eclipse DS-5 para comunicarse entre sí por medio de SSH y el servidor de GDB. De esta forma, las capacidades de depuración de Eclipse servirían para observar la ejecución simulada de un programa sin necesidad de recurrir a *hardware* específico. Por otro lado, también es posible seguir el mismo procedimiento, pero utilizando *hardware* específico, como podría ser una Raspberry Pi.

Aunque se consiguió depurar código utilizando ambos métodos, contra una máquina virtual y contra una Raspberry Pi, la complejidad de todo el montaje era excesiva para lo que se pretendía que fueran unas prácticas de primer curso.

Por último, también se evaluó el simulador ARMSim#.[5] Dicha aplicación permite simular la ejecución de programas en ensamblador en un sistema basado en el procesador ARM7.

[1] http://spimsimulator.sourceforge.net/
[2] http://www.keil.com/product/brochures/uv4.pdf
[3] http://arm.com/support/university/educators/embedded/
[4] http://ds.arm.com/
[5] http://armsim.cs.uvic.ca/

Está escrito en C#, funciona en Windows y, en teoría, sobre Mac OSX y GNU/Linux utilizando la implementación Mono del entorno .NET. Este simulador se acercaba más a lo que se estaba buscando para la asignatura y presenta características interesantes, como la simulación y visualización de periféricos, e incluso la posibilidad de desarrollar *plugins* para extenderlo con nuevos dispositivos de E/S. Sin embargo, en la actualidad no soporta el modo Thumb de ARM, que era necesario utilizar para poder enlazar las prácticas con simulador de esta asignatura, con prácticas posteriores en la misma asignatura y en otras del grado, en las que el estudiante iba a programar tarjetas Arduino Due.

Así que, puesto que ninguno de los sistemas evaluados cubrían todas nuestras necesidades, se optó por desarrollar nuestro propio simulador de ARM[6] con la intención de que también pudiera ser utilizado en cursos introductorios a la Arquitectura de Computadores de otras universidades.

El simulador desarrollado consta de un motor de simulación, ARMSim, y una interfaz gráfica, QtARMSim [2]. Puesto que ambas aplicaciones están totalmente desacopladas, alguien interesado únicamente en el simulador podría desarrollar su propia interfaz gráfica o, en el caso contrario, modificar ligeramente la interfaz gráfica para conectarla a otro simulador.

Además, se ha escrito un libro de prácticas, Introducción a la Arquitectura de Computadores con QtARMSim y Arduino,[7] en el que se describe con detalle el simulador y se proponen una serie de prácticas de introducción a la arquitectura de computadores utilizando la arquitectura ARM Thumb.

El simulador ARMSim y QtARMSim fue descrito por primera vez en un artículo anterior [2]. El presente artículo describe con mayor detalle el motor de simulación, la interfaz gráfica y la interrelación entre ambos. También incorpora las mejoras que se han realizado en la interfaz gráfica con respecto a la versión publicada en la obra citada. Por último, proporciona más detalles sobre la encuesta realizada a los estudiantes y sobre la evolución de los resultados académicos obtenidos antes y después de utilizar el simulador.

2. Objetivos

El objetivo que se tuvo en mente cuando se comenzó a desarrollar el simulador fue proporcionar a los estudiantes un simulador de ARM Thumb que les permitiera alcanzar fácilmente los objetivos formativos de arquitectura de computadores relacionados con el funcionamiento del procesador.

Para cumplir con dicho objetivo docente, el simulador debería ser lo más sencillo posible, gratuito, multiplataforma y libre, que contemplara todo el proceso de desarrollo y simulación de código en ensamblador. Además de lo anterior, también se planteó que sería conveniente que el motor de simulación estuviera desacoplado de la interfaz gráfica.

El simulador debía ser lo más sencillo posible para que los estudiantes pudieran utilizarlo en las sesiones de prácticas sin mayores problemas, centrándose en el desarrollo y simulación de programas en ensamblador, más que en el manejo y configuración del simulador.

También se quería que el simulador fuera gratuito y multiplataforma, para que el estudiante tuviera la oportunidad de instalarlo gratuitamente en su propio computador, independientemente del sistema operativo que utilizase.

El objetivo al hacerlo *software* libre fue incentivar al estudiantado a que pudieran inspeccionar el código, tanto del motor, como de la interfaz gráfica. Otra ventaja de ser *software* libre, es que permite que terceras personas puedan involucrarse en su desarrollo o lo adapten a sus propias necesidades.

Por otro lado, el simulador debía contemplar todo el proceso tanto de desarrollo como de simulación del código con la intención de que dicho proceso fuera lo más ágil y sencillo posible. De esta forma, se pretendía evitar que el estudiante se viera obligado a alternar entre un editor externo —para escribir el código en ensamblador— y el simulador —para las fases de compilación y simulación del código—. Así pues, debería proporcionar todos los aspectos necesarios para el desarrollo y simulación de código en ensamblador: desde la edición del código, pasando por su compilación, hasta su ejecución, ya fuera completa o paso a paso.

El último de los objetivos fue el de que el motor de simulación y la interfaz estuvieran desacoplados. Esta decisión permitirá en el futuro que el simulador pueda ser utilizado de forma remota o desde distintas interfaces gráficas. Debido a esta característica, el simulador consta en realidad de dos aplicaciones: ARMSim, un motor de simulación de la arquitectura ARM Thumb, y QtARMSim, una interfaz gráfica para el simulador.

3. ARMSim

ARMSim es un motor de simulación de la arquitectura ARM Thumb. Proporciona un modelo de procesador —que incluye registros e indicadores de estado— y un modelo de memoria —tanto RAM como ROM—. El modelo de procesador es capaz de decodificar y ejecutar el juego completo de instrucciones de la arquitectura ARM Thumb. El modelo de memoria permite definir la cantidad de memoria disponible, así como inicializar, consultar y modificar su contenido.

Aparte de modelar los componentes anteriormente citados, el simulador permite indicar un fichero fuente para su ensamblado. En lugar de implementar un ensamblador propio, se ha optado por una solución más versátil y potente: ARMSim ejecuta un ensamblador de ARM (por ejemplo, GCC de GNU) e interpreta el código objeto generado por dicho ensamblador para inicializar la ROM y la RAM a partir de dicha información. Además, también extrae la información sobre qué líneas

[6] El simulador puede descargarse de `http://lorca.act.uji.es/projects/qtarmsim`

[7] El libro «Introducción a la Arquitectura de Computadores con QtARMSim y Arduino» se puede descargar desde `http://lorca.act.uji.es/book/practARM/`

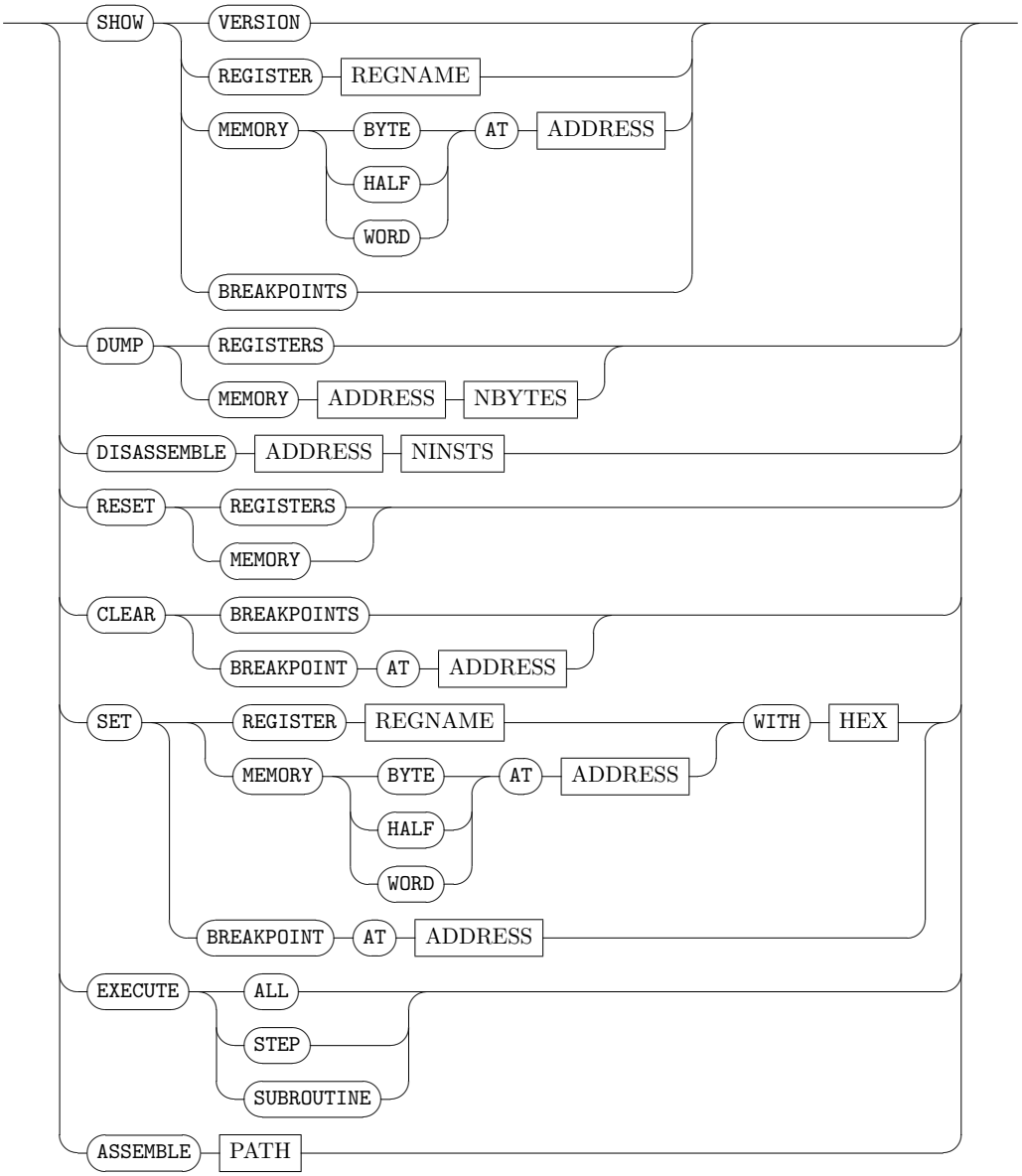

Figura 1: Sintaxis de los instrucciones soportados por el motor de simulación ARMSim

de código fuente han generado cada instrucción máquina.

ARMSim no proporciona una interfaz gráfica. En lugar de eso, cuando se ejecuta, se pone a la escucha en el puerto que se haya indicado en la línea de instrucciones. Para interactuar con ARMSim es necesario establecer una conexión con dicho puerto (por ejemplo, utilizando la aplicación telnet) e indicar por medio de instrucciones de texto, qué acciones se quieren llevar a cabo.

La sintaxis de las instrucciones reconocidas por el motor de simulación ARMSim se describe en la figura 1. Como se puede observar, las instrucciones aceptadas por ARMSim permiten: (1) ensamblar un fichero; (2) consultar y modificar el contenido de registros y memoria; (3) desensamblar posiciones de memoria; (4) definir y consultar puntos de ruptura; (5) ejecutar el código a partir de una posición dada hasta que se acabe el programa o se encuentre un punto de ruptura; y (6) ejecutar paso a paso el programa (entrando o no en las subrutinas).

4. QtARMSim

Tal y como ya se ha comentado, QtARMSim es una interfaz gráfica para el motor de simulación ARMSim. La interrelación entre la interfaz gráfica, QtARMSim, que se describe en este apartado, el motor de simulación, ARMSim, descrito en el apartado anterior, y el compilador GCC de GNU se muestra esquemáticamente en la figura 2.

Como se puede ver en esta figura, la interfaz gráfica, QtARMSim, se encarga de la edición de los ficheros fuente en ensamblador y delega la simulación del código objeto correspondiente al motor de simulación, ARMSim. La comunicación entre la interfaz gráfica y el simulador se realiza utilizando el protocolo telnet. La interfaz gráfica envía las acciones que debe realizar el motor de simulación y recibe como respuesta los resultados de dichas acciones. La otra interacción que tiene lugar ocurre entre el motor de simulación y el compilador GCC. El motor de simulación, cuando es requerido para ello, ejecuta la instrucción *gcc* de la forma adecuada para ensamblar el código fuente indicado por la interfaz gráfica. Una vez ensamblado el código fuente, el motor de simulación carga en memoria el código objeto resultante, informa a la interfaz gráfica de su nuevo estado y queda a la espera de nuevas peticiones.

Además de lo anterior, la interfaz gráfica también es la encargada de arrancar el motor de simulación y de gestionar su configuración —puerto en el que debe escuchar, ruta de la instrucción *gcc* y parámetros de compilación—.

Naturalmente, todo lo descrito previamente es transparente al usuario de la interfaz gráfica. Cuando se ejecuta el simulador, el usuario simplemente interactúa con una ventana gráfica similar a la mostrada en la figura 3. En dicha figura se puede ver el aspecto de la ventana principal de QtARMSim en la que se está editando un pequeño fragmento de código. La parte central de dicha ventana corresponde al editor de código fuente en ensamblador. Alrededor de dicha parte central se distribuyen una serie de paneles. En la disposición por defecto, a la izquierda del editor se encuentra el panel de registros; a su derecha, el panel de memoria, y debajo, el panel de mensajes. Los paneles de registros y memoria están desactivados en el modo de edición y serán descritos más adelante. En cuanto al panel de mensajes, este se encarga de ir mostrando los mensajes relacionados con las acciones que se vayan llevando a cabo: si el código se ha ensamblado correctamente, si por el contrario ha ocurrido algún error al ensamblarlo, qué línea se acaba de ejecutar, etc.

Cuando se instala QtARMSim por primera vez puede ser necesario configurarlo para indicar cómo ejecutar el motor de simulación ARMSim o dónde está instalado el compilador cruzado de GCC para ARM. Generalmente no será necesario modificar manualmente la configuración, ya que en su primera ejecución QtARMSim intenta averiguar de forma automática todos los parámetros necesarios.

La figura 4 muestra el cuadro de diálogo de preferencias. En él se pueden diferenciar dos cuadros. El cuadro superior corresponde a la configuración relacionada con el simulador ARMSim y permite configurar: el servidor y el puerto al que se debe conectar la interfaz gráfica; la línea de instrucciones para ejecutar el simulador (en el caso de que el simulador se ejecute en la misma máquina que la interfaz), y el directorio de trabajo del simulador. El cuadro inferior, por su parte, gestiona la configuración relacionada con el compilador de GCC para ARM. En dicho cuadro se debe indicar la ruta al ejecutable del compilador de GCC para ARM y las opciones que se le deben pasar para que invoque el ensamblador de la forma adecuada.

QtARMSim distingue entre dos modos de funcionamiento: el modo de edición y el de simulación. Dichos modos se describen con más detalle en los dos apartados siguientes.

4.1. Modo de edición

En el modo de edición, como ya se ha comentado, la parte central de la ventana es un editor de código fuente en ensamblador, que permite escribir el programa en ensamblador que se quiere simular posteriormente. La figura 3, referenciada anteriormente, muestra la ventana de QtARMSim en la que se está editando un pequeño programa en ensamblador. Como se puede observar en dicha figura, el editor reconoce el lenguaje ensamblador ARM Thumb y resalta convenientemente el código fuente.

Con respecto a la versión anterior de QtARMSim [2], conviene destacar que el editor de código fuente se ha reescrito desde cero para eliminar las dependencias de la versión anterior con el componente QScintilla,[8] lo que obligaba a utilizar PyQt[9] e impedía la utilización de PySide,[10] que resulta ser

[8] `http://www.riverbankcomputing.co.uk/software/qscintilla/intro`
[9] `http://www.riverbankcomputing.co.uk/software/pyqt/intro`
[10] `https://wiki.qt.io/Category:LanguageBindings::PySide`

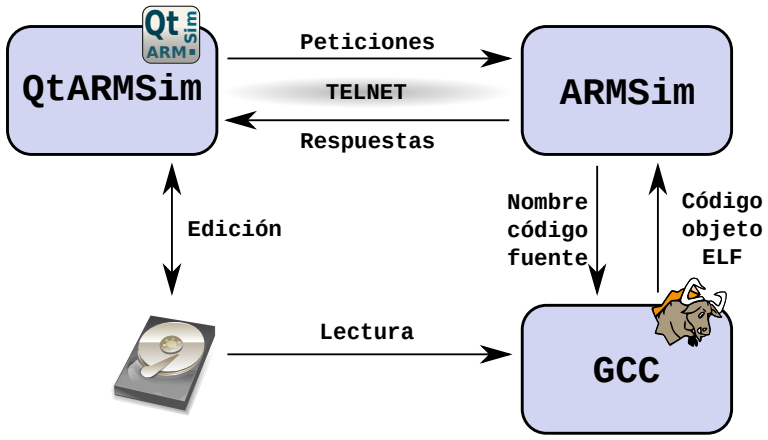

Figura 2: Interrelación entre los componentes del simulador (*el dibujo del disco duro es de Andrew Fitzsimon*)

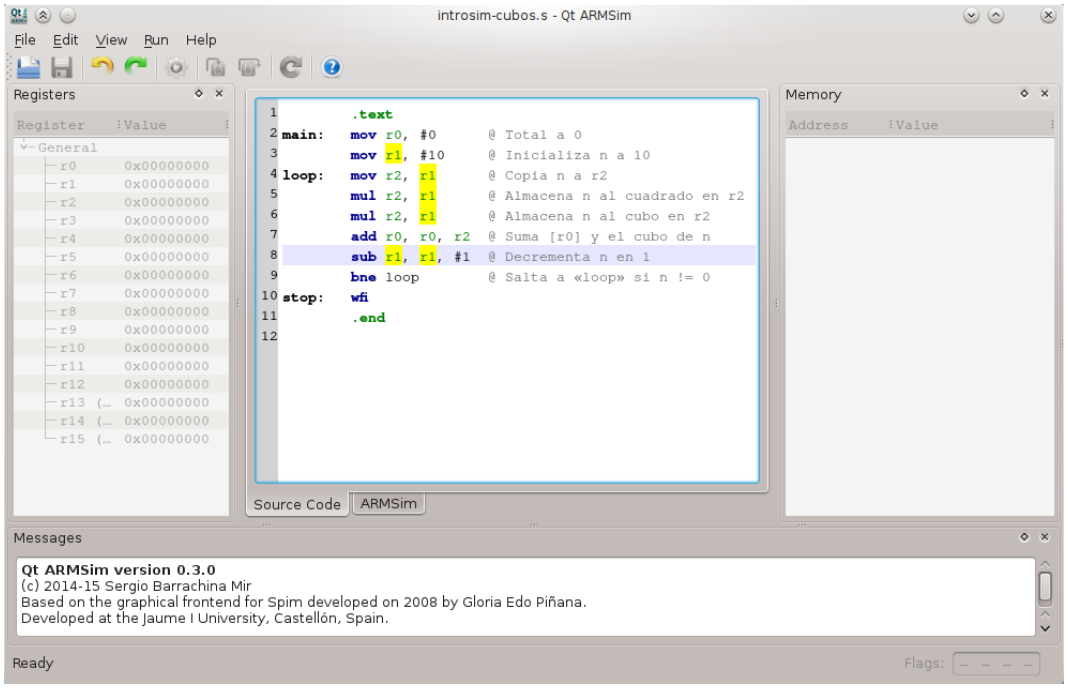

Figura 3: Ventana principal de QtARMSim editando el código introsim-cubos.s

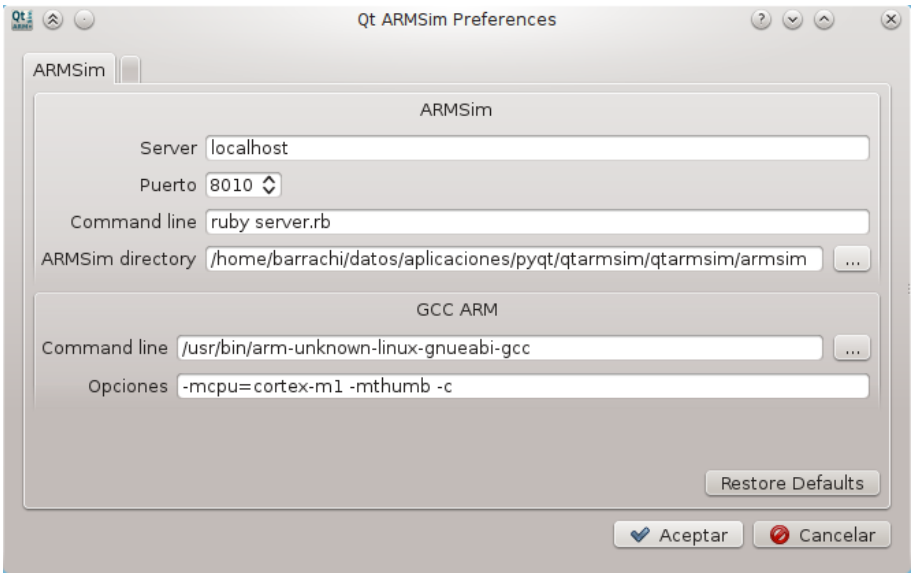

Figura 4: Cuadro de diálogo de preferencias de QtARMSim

mucho más sencillo de instalar que PyQt. Gracias a la reescritura de dicho componente, la actual versión de QtARMSim puede basarse en PySide (en lugar de en PyQt), lo que a su vez facilita enormemente la instalación del simulador.

Además de reescribir el editor para cambiar las dependencias de la interfaz gráfica, se han realizado las siguientes mejoras en el editor: (1) se ha mejorado el motor de reconocimiento de la sintaxis de ARM, que ahora es mucho más preciso que el utilizado en la versión anterior; (2) se ha incorporado la funcionalidad de escalar el tamaño de la letra (lo que es especialmente útil, por ejemplo, para realizar exposiciones en clase), y (3) se ha incorporado la funcionalidad de resaltar aquellos registros o etiquetas presentes en el código fuente y que coincidan con el texto situado en ese momento bajo el cursor (lo que facilita seguir las dependencias de datos durante el desarrollo o depuración del código). Un ejemplo de esta última característica se puede ver en la figura 3: debido a que el cursor de texto se ha situado sobre una aparición del registro `r1`, todas las apariciones de dicho registro aparecen resaltadas en amarillo.

4.2. Modo de simulación

Para pasar al modo de simulación, basta con pulsar sobre la pestaña *ARMSim*, que se encuentra debajo de la sección central de la ventana principal. La figura 5 muestra el aspecto de QtARMSim cuando está en el modo de simulación.

Con respecto a qué versión del código que se está editando debería simularse, se evaluaron las dos opciones posibles: (1) simular el código editado aunque no se haya guardado, o (2) solo simular la última versión guardada del código editado. Finalmente, se optó por implementar la segunda de dichas opciones, para de esta forma permitir al estudiante alternar libremente entre el modo de simulación y el de edición mientras esté editando un determinado código y que fuera él quien decidiera cuándo dichos cambios deberían forzar el comienzo de una nueva simulación.

De forma transparente para el usuario, al cambiar al modo de simulación, la interfaz gráfica instruye al motor de simulación ARMSim, para que se encargue de realizar las siguientes acciones: (1) llamar al ensamblador cruzado para ensamblar el código fuente; (2) actualizar el contenido de la memoria ROM con las instrucciones máquina generadas por el ensamblador; (3) inicializar, si es el caso, el contenido de la memoria RAM con los datos indicados en el código fuente, y, por último, (4) inicializar los registros del procesador simulado.

Si se produjera algún error al intentar pasar al modo de simulación, se mostrará un cuadro de diálogo informando del error, se volverá automáticamente al modo de edición y en el panel de mensajes se mostrarán las causas del error. En caso contrario, se completa el cambio al modo de simulación.

Una vez en el modo de simulación, cada línea de la ventana central muestra la información correspondiente a una instrucción máquina (ver figura 5). Para cada una de ellas, se tiene (de izquierda a derecha):

1. La dirección de memoria en la que está almacenada la instrucción máquina.
2. La instrucción máquina expresada en hexadecimal.
3. La instrucción máquina expresada en ensamblador.
4. La línea original en ensamblador que ha dado lugar a la instrucción máquina.

Así por ejemplo, la información mostrada en la primera línea de la ventana de desensamblado de la figura 5 (dicha información se muestra también en la figura 6) indica que:

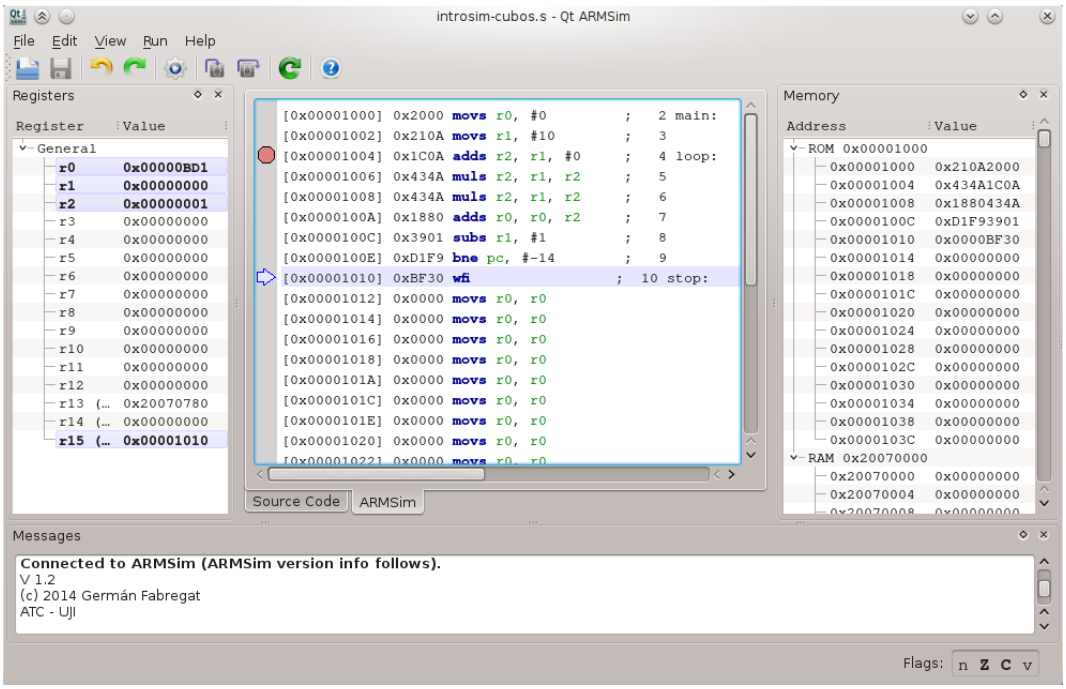

Figura 5: QtARMSim en el modo de simulación después de ejecutar un código máquina

1. La instrucción máquina está almacenada en la dirección de memoria `0x0000 1000`.
2. La instrucción máquina expresada en hexadecimal es `0x2000`.
3. La instrucción máquina expresada en ensamblador es `movs r0, #0`.
4. La instrucción se ha generado a partir de la línea número 2 del código fuente original, que es:

```
main: mov r0, #0 @ Total a 0
```

Por otro lado, el contenido de la memoria del computador simulado se muestra en el panel de memoria, tal y como se puede ver en la parte derecha de la figura 5. En este ejemplo se puede observar que el computador dispone de dos bloques de memoria: un bloque de memoria ROM que comienza en la dirección `0x0000 1000` y un bloque de memoria RAM que comienza en la dirección `0x2007 0000`. También se puede ver cómo las celdas de la memoria ROM contienen algunos valores distintos de cero (que corresponden a las instrucciones máquina del programa ensamblado) y las celdas de la memoria RAM están todas a cero.

De forma similar, el contenido de los registros del `r0` al `r15` se muestra en el panel de registros (a la izquierda en la figura 5). El registro `r15` merece una mención especial, ya que se trata del contador de programa (PC). Para facilitar su uso, el simulador resalta la línea correspondiente a la dirección indicada por el PC, de tal forma que el usuario puede identificar fácilmente qué línea de código será la siguiente en ser ejecutada.

Si se compara la apariencia de QtARMSim cuando está en el modo de edición (figura 3) con la de cuando está en el modo de simulación (figura 5), se puede observar que al cambiar al modo de simulación se han habilitado los paneles de registros y memoria que estaban desactivados en el modo de edición. De hecho, si se vuelve al modo de edición pulsando sobre la pestaña *Source Code*, se podrá comprobar que dichos paneles se desactivan automáticamente. De igual forma, si se vuelve al modo de simulación, aquellos volverán a activarse.

En cuanto al manejo de la interfaz, puesto que los paneles del simulador son empotrables, es posible cerrarlos de manera individual, reubicarlos en una posición distinta, o desacoplarlos y mostrarlos como ventanas flotantes. A modo de ejemplo, la figura 6 muestra la ventana principal del simulador tras cerrar los paneles en los que se muestran los registros y la memoria. No obstante, y pensando en su uso en el aula, se ha incorporado al simulador la funcionalidad de restaurar la disposición por defecto (basta con pulsar la tecla F3 o su correspondiente entrada en el menú). Naturalmente, también es posible volver a mostrar alguno o todos los paneles que han sido cerrados previamente sin necesidad de restaurar la disposición por defecto.

Desde el modo de simulación QtARMSim permite ejecutar el programa de forma completa, ejecutarlo paso a paso, modificar manualmente el contenido de registros y memoria, y establecer puntos de ruptura. En los siguientes apartados se describen dichas funcionalidades.

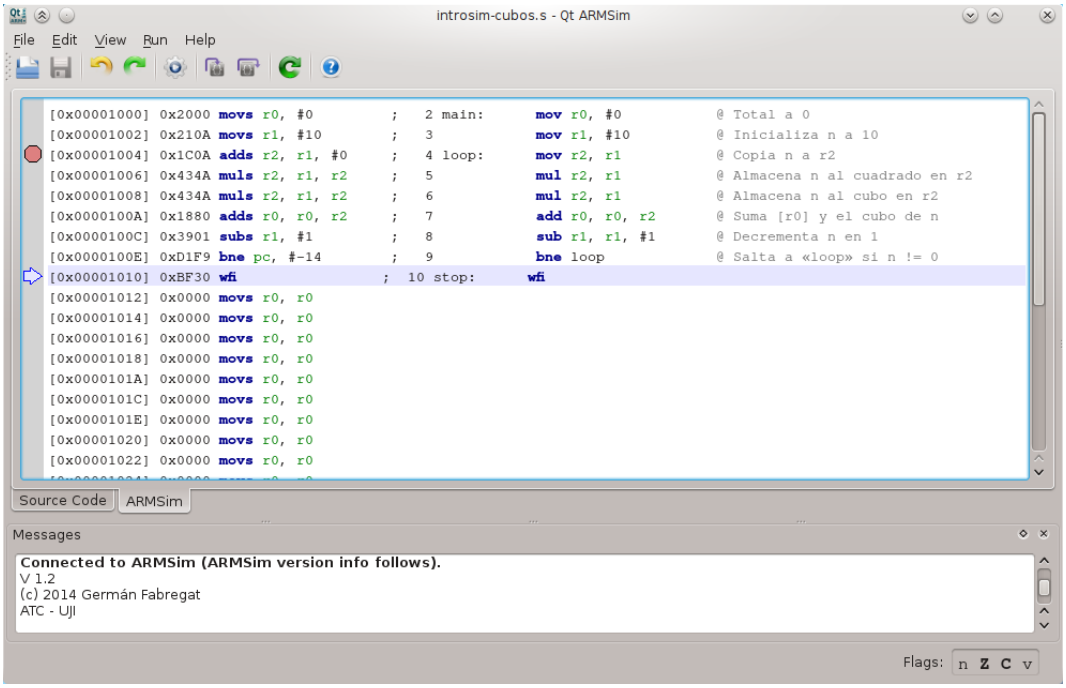

Figura 6: QtARMSim con los paneles de registros y memoria cerrados

4.2.1. Ejecución del programa completo

La opción más sencilla de simulación es la de ejecutar el programa completo. La figura 5 muestra la ventana de QtARMSim después de ejecutar el código máquina generado al ensamblar el fichero *introsim-cubos.s*. Como se puede observar en dicha figura, el simulador resalta aquellos registros y posiciones de memoria que se han modificado tras la última ejecución, con el objetivo de que el estudiante identifique fácilmente los cambios que ha ocasionado la ejecución del código. Así, se puede observar en la figura 5 que los registros `r0`, `r1`, `r2` y `r15` tienen ahora fondo azul y están en negrita, puesto que han sido los registros que se han visto afectados por la ejecución completa del código.

4.2.2. Ejecución paso a paso

Aunque la ejecución completa de un programa pueda servir para comprobar si el programa hace lo que se espera de él, no permite ver con detalle cómo se llega a dicho resultado. Tan solo es posible comparar el estado inicial del computador simulado y el estado al que se llega cuando concluye la ejecución del programa.

Para poder ver qué es lo que ocurre al ejecutar cada instrucción, el simulador proporciona la opción de ejecutar paso a paso. Esta opción suele utilizarse para ver por qué un determinado programa o una de sus partes no está haciendo lo que se espera de ella. Pero también suele utilizarse para evaluar cómo afectaría al programa el que en un momento dado se modificara el contenido de determinados registros o posiciones de memoria.

Si por ejemplo se quisiera evaluar cómo afectaría a la ejecución del resto del programa el que el registro `r1` tuviera un valor distinto al actual, habría que modificar el contenido de dicho registro antes de proseguir con la ejecución del programa.

El simulador permite modificar el contenido de un registro simplemente haciendo doble clic sobre la celda en la que está su contenido actual, teclear el nuevo número y pulsar la tecla Retorno (ver figura 7). Además, el nuevo valor[11] puede introducirse en decimal, en hexadecimal (si se precede de «0x», por ejemplo, «0x3»), o en binario (si se precede de «0b», por ejemplo, «0b11»).

Naturalmente, una vez modificado el contenido de un registro o de una posición de memoria RAM, se podría ejecutar el resto del código de golpe, no haría falta seguir ejecutándolo paso a paso.

Otra característica del simulador es que proporciona dos modalidades de ejecución paso a paso, la primera de ellas ejecuta siempre una única instrucción, pasando el PC a apuntar a la siguiente instrucción. La segunda modalidad tiene en cuenta el hecho de que los programas suelen estructurarse por medio de rutinas. Si el código en ensamblador incluye llamadas a rutinas, al utilizar el primer modo de ejecución paso a paso sobre una instrucción de llamada a una rutina, la siguiente ins-

[11] También es posible introducir cadenas de como mucho 4 caracteres. Para hacerlo, basta con introducirlas entre comillas simples o dobles (por ejemplo, `"Hola"` o `'Hola'`). Conviene tener en cuenta que al codificar los caracteres de la cadena introducida, se utilizará la codificación UTF-8 y que los bytes resultantes se ordenarán siguiendo la organización *Little-Endian*.

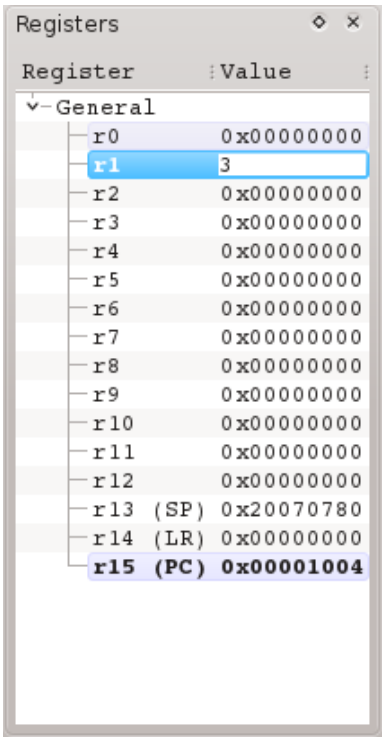

Figura 7: Edición del contenido del registro `r1`

trucción que se ejecutará será la primera instrucción de dicha rutina.

Sin embargo, en ocasiones no interesa tener que ejecutar paso a paso todo el contenido de una determinada rutina, puede ser preferible ejecutar la rutina entera como si de una única instrucción se tratara y que una vez ejecutada la rutina, el PC pase a apuntar directamente a la siguiente instrucción a la de la llamada a la rutina. De esta forma, sería fácil para el estudiante ver y comparar el estado del computador simulado antes de llamar a la rutina y justo después de volver de ella. Para poder hacer lo anterior, QtARMSim también proporciona la opción de ejecución paso a paso llamada *por encima*.

La ejecución paso a paso entrando (*step into*, en inglés) y la ejecución paso a paso por encima (*step over*, en inglés) se comportarán de forma diferente únicamente cuando la instrucción que se vaya a ejecutar sea una instrucción de llamada a una rutina. Ante cualquier otra instrucción, las dos ejecuciones paso a paso harán lo mismo.

4.2.3. Puntos de ruptura

La ejecución paso a paso permite ver con detenimiento qué es lo que está ocurriendo en una determinada parte del código. Sin embargo, puede que para llegar a la zona del código que se quiere inspeccionar con detenimiento haya que ejecutar muchas instrucciones. Por ejemplo, se podría estar interesado en una parte del código a la que únicamente se llega después de completar un bucle con cientos de iteraciones. No tendría sentido tener que ir paso a paso hasta conseguir salir del bucle y llegar a la parte del código que en realidad se quiere ver con más detenimiento.

Por tanto, es conveniente disponer de una forma de indicarle al simulador que ejecute aquellas partes del código que no interesa ver con detenimiento y que se detenga cuando llegue a aquella instrucción a partir de la cual se quiere realizar una ejecución paso a paso (o en la que se quiere poder observar el estado del simulador).

Un punto de ruptura (*breakpoint* en inglés) sirve justamente para eso, para indicar al simulador que tiene que detener la ejecución del programa cuando se alcance la instrucción en la que se haya definido un punto de ruptura.

QtARMSim permite definir y desmarcar puntos de ruptura. Para definir un punto de ruptura, basta con hacer clic sobre el margen izquierdo de la ventana de desensamblado, en la línea en la que se quiere definir el punto de ruptura. Al hacerlo, aparecerá un símbolo hexagonal rojo en el margen para señalar que en esa línea se ha definido un punto de ruptura. En la figura 5 se puede observar que se ha definido un punto de ruptura en la dirección de memoria `0x0000 1004`.

Para desmarcar un punto de ruptura definido previamente, se procede de la misma forma, haciendo clic sobre la marca de dicho punto de ruptura.

5. Resultados

Se ha desarrollado un simulador de ARM que cumple con los objetivos descritos en el apartado 2 y que puede descargar-

se gratuitamente. También se ha redactado y publicado bajo licencia Creative Commons el material de prácticas utilizado durante el curso 2014–15. En dicho material se describe el simulador y se proponen ejercicios de introducción a la Arquitectura de Computadores que lo utilizan.

Como este curso ha sido el primero en el que se ha utilizado dicho simulador, se ha preguntado constantemente a los estudiantes por su opinión sobre el funcionamiento del simulador para que nos dijeran qué aspectos consideraban que deberían mejorarse. La respuesta ha sido en general muy positiva y, gracias a las sugerencias recibidas, se ha tenido la oportunidad de ir mejorando el simulador durante el curso. La crítica más acuciante fue la de mejorar el proceso de instalación, ya que dependiendo del sistema operativo podía ser más o menos engorroso. Debido a dicha crítica, tal y como se comenta en el apartado 4.1, se ha reescrito desde cero el código del editor para modificar las dependencias de la interfaz gráfica y facilitar así su instalación (aprovechando la ocasión para mejorarlo y añadirle nuevas funcionalidades).

Conviene destacar que la respuesta de los estudiantes repetidores ha sido especialmente gratificante, ya que pese a que el curso pasado hicieron las prácticas con MIPS y este curso cambiaron a ARM, su percepción de la facilidad de uso de la nueva herramienta les ha llevado a comentarnos que el lenguaje ensamblador de ARM es mucho más sencillo que el de MIPS (cuando desde nuestro punto de vista no es así).

Además de lo anterior, una vez finalizado el curso se realizó una encuesta a los estudiantes con las preguntas que se muestran a continuación, donde las primeras 8 preguntas eran de respuesta cerrada y utilizaban una escala Likert de 1 a 5, siendo 1 «totalmente en desacuerdo» y 5 «totalmente de acuerdo»; y las dos últimas preguntas eran de respuesta abierta:

1. He instalado QtARMSim sin problemas en mi computador.
2. La interfaz gráfica de QtARMSim me parece adecuada.
3. La edición de código en QtARMSim me resultaba sencilla.
4. Las funcionalidades de depuración de código (ejecución paso a paso, puntos de ruptura, etc.) me han resultado útiles.
5. La información mostrada en los distintos paneles de QtARMSim es la adecuada.
6. Considero que QtARMSim me ha servido para entender mejor cómo funciona el procesador ARM.
7. Recomendaría que se siguiera utilizando QtARMSim en la docencia de arquitectura de computadores.
8. Considero más interesantes las prácticas sobre arquitectura de computadores de este curso que las que se realizaron el curso pasado con el simulador XSPIM.
9. ¿Qué aspectos positivos destacarías de QtARMSim?
10. ¿Qué crees que convendría mejorar de QtARMSim?

La encuesta fue contestada por 28 estudiantes (un $19\,\%$ de los estudiantes que siguieron la asignatura).

Las cinco primeras preguntas de la encuesta versaban sobre los aspectos técnicos del simulador (instalación, interfaz, funcionalidad de depuración e información en paneles). En dichas preguntas se obtuvo una mediana de 4, salvo en la referida a la instalación, en que la mediana fue de 3.

Las siguientes tres preguntas, más relacionadas con el objetivo docente de la aplicación y la satisfacción del estudiante, «Considero que QtARMSim me ha servido para entender mejor cómo funciona el procesador ARM», «Recomendaría que se siguiera utilizando QtARMSim en la docencia de arquitectura de computadores» y «Considero más interesantes las prácticas sobre arquitectura de computadores de este curso que las que se realizaron el curso pasado con el simulador XSPIM», obtuvieron una mediana de 5.

Las respuestas dadas a las dos últimas preguntas, de respuesta abierta, «¿Qué aspectos positivos destacarías de QtARMSim?» y «¿Qué crees que convendría mejorar de QtARMSim?» incidieron por un lado en la facilidad de uso y en la utilidad del simulador para facilitar su aprendizaje, y por otro, en que convendría simplificar el procedimiento de instalación. Algunas de las respuestas obtenidas para dichas preguntas se recogen en los cuadros 1 y 2.

En cuanto al rendimiento académico, comparando el rendimiento obtenido el curso pasado con el actual, tal como se puede observar en el cuadro 3, el porcentaje de estudiantes que han seguido la asignatura ha aumentado en un $5\,\%$, el número de aprobados sobre matriculados, en un $14\,\%$ y el número de aprobados sobre presentados, en un $12\,\%$. Hay que tener en cuenta que no es posible concluir que las mejoras en los resultados estén directamente relacionadas con la utilización del simulador, puesto que se han realizado más cambios en la asignatura que también podrían haber contribuido a dicha mejora.

Por último, la valoración de los profesores de laboratorio también ha sido muy positiva y su impresión ha sido la de que los estudiantes se daban más cuenta de lo que estaba ocurriendo conforme simulaban los ejercicios de prácticas, en comparación con lo observado en cursos anteriores, en los que les daba la impresión de que los estudiantes tenían menos claro qué estaba ocurriendo cuando simulaban ejercicios similares.

6. Conclusiones y trabajo futuro

En este artículo se ha presentado la última versión de las aplicaciones ARMSim y QtARMSim, que proporcionan un entorno didáctico de simulación de la arquitectura ARM Thumb multiplataforma, libre, gratuito y fácil de utilizar. Dichas aplicaciones se han desarrollado con el objetivo de facilitar la comprensión del funcionamiento de un proce-

Sencillo y básico para aprender.
Facilidad de uso. Información detallada.
Está bien diseñado en todos sus aspectos.
El uso de la función paso a paso y la clara posición de los flags.
Sencillez de la interfaz y utilización. Muy útil a la hora de estudiar.
Poder ejecutar código de ensamblador de ARM y poder ver registros y memoria.
Se trata de una forma muy visual de observar el funcionamiento de la memoria y los registros del procesador ARM mientras va ejecutando cada instrucción.
Se ve muy claramente qué se guarda en cada registro, si son datos o direcciones de memoria. También es muy gráfico y nada lioso, como el pasado simulador, es mucho más "entendible" todo lo que sucede.
Representa gráficamente muy bien la ejecución de programas de ARM y dónde se guardan los datos, de dónde se cogen, qué se hace con ellos, etc. Una vez se entiende la teoría, el simulador es muy visual y ayuda a acabar de comprender los conceptos.

Cuadro 1: Algunas de las respuestas a la pregunta «¿Qué aspectos positivos destacarías de QtARMSim?»

Facilitar la instalación en Windows.
La instalación y configuración tendría que estar automatizada.
Mejorar un poco su editor, pero sobretodo el proceso de instalación.
Su instalación. Al principio fue bastante difícil instalarlo, tanto en Windows como en Linux. Con el tiempo aparecieron algunas guías, pero aún así convendría hacerlo más sencillo.
Una breve explicación en la parte inferior de cada instrucción paso por paso o algún gráfico, de hecho en arquitectura de computadores, el programa muestra mediante un esquema como van progresando las instrucciones en el procesador.

Cuadro 2: Algunas de las respuestas a la pregunta «¿Qué crees que convendría mejorar de QtARMSim?»

	2013	2014
Matriculados	160	165
Seguimiento de la asignatura	83 %	88 %
Aprobados sobre matriculados	45 %	59 %
Aprobados sobre presentados	55 %	67 %

Cuadro 3: Evolución del rendimiento académico

sador y se ha constatado que tanto los estudiantes como los profesores perciben que se ha cumplido con dicho objetivo.

Como trabajo futuro se contempla implementar nuevas funcionalidades en la interfaz gráfica, como la ayuda en línea y los mensajes de ayuda contextual o la inclusión de una consola simulada. En lo que respecta al motor de simulación, se pretende añadir soporte completo para la arquitectura Thumb 2 y ampliar el módulo de gestión de accesos a memoria para incluir regiones asociadas a dispositivos de entrada/salida simulados e incluso implementar simulación de memorias caché. Todo el entorno de simulación se quiere ampliar para permitir la compilación de código en C y el uso de ARMSim de forma remota para la evaluación automatizada de ejercicios, así como incorporar una animación gráfica de la ejecución de cada instrucción.

Referencias

[1] Sergio Barrachina Mir, Maribel Castillo Catalán, José M. Claver Iborra y Juan C. Fernández Fernández. *Prácticas de introducción a la arquitectura de computadores con el simulador SPIM.* Pearson Educación. Madrid, 2013.

[2] Sergio Barrachina Mir, Germán Fabregat Llueca, Juan Carlos Fernández Fernández y Germán León Navarro. *ARMSim y QtARMSim: simulador de ARM para docencia.* En Actas de las XXI Jornadas de Enseñanza Universitaria de la Informática, Jenui 2015, pp. 2–9. Andorra, julio de 2015.

[3] Sergio Barrachina Mir, Germán Fabregat Llueca y José Vicente Martí Avilés. *Utilizando Arduino DUE en la docencia de la entrada/salida.* En Actas de las XXI Jornadas de Enseñanza Universitaria de la Informática, Jenui 2015, pp. 58–65. Andorra, julio de 2015.

[4] Alan Clements. *Selecting a processor for teaching computer architecture.* Microprocessors and Microsystems, vol. 23, núm. 5, pp. 281–290. Octubre de 1999.

[5] Alan Clements. *ARMs for the poor: Selecting a processor for teaching computer architecture.* En Actas de Frontiers in Education Conference (FIE), pp. T3E 1–T3E 6. Octubre de 2010.

[6] Alan Clements. *The Undergraduate Curriculum in Computer Architecture.* IEEE Micro, vol. 20, núm. 3, pp. 13–22. Mayo/Junio de 2000.

[7] Alan Clements. *Computer Organization and Architecture. Themes and Variations.* Cengage Learning. EEUU, 2014.

[8] William Hohl y Christopher Hinds. *ARM Assembly Language: Fundamentals and Techniques.* Second Edition. Crc Press. EEUU, 2014.

Sergio Barrachina Mir es Ingeniero de Telecomunicaciones, especialidad electrónica, por la Universitat Politécnica de Valencia y Doctor en Ingeniería Informática por la Universitat Jaume I. Ha sido profesor universitario desde 1997 y actualmente es profesor titular del Área de Arquitectura y Tecnología de Computadores de la Universitat Jaume I.
Ha ejercido de coordinador de Estancias en Prácticas de la titulación de Ingeniería Técnica en Informática de Sistemas (ITIS) y, posteriormente, de Vicedirector de la Escuela Superior de Tecnología encargado de la titulación de ITIS.
Ha dirigido y participado en 17 proyectos de innovación docente, es autor de 14 comunicaciones en congresos nacionales e internacionales sobre innovación docente y ha recibido dos premios relacionados con la innovación docente.

Germán Fabregat es licenciado en Física, especialidad Electricidad, Electrónica e Informática, por la Universidad de Valencia desde 1989 y doctor en la misma disciplina desde 1996. Es profesor de la Universidad Jaume I desde 1991, siendo Titular del Área de Arquitectura y Tecnología de Computadores desde 2001.
Sus trabajos se ha desarrollado en tolerancia a fallos, sistemas empotrados y redes de sensores, con especial interés en sus aplicaciones a la industria. Además su labor docente se caracteriza por el continuo desarrollo de aplicaciones de soporte a la docencia como dispositivos para las prácticas, simuladores, entornos de programación de sistemas, etcétera.

Juan Carlos Fernández Fernández es Diplomado en Informática por la Universidad de Oviedo, Licenciado y Doctor en Informática por la Universidad Politécnica de Valencia. Se incorporó a la Universidad Politécnica de Valencia en 1992 como Profesor Asociado. En 1994 se incorpora a la Universitat Jaume I, donde es Profesor Titular de Universidad en el área de Arquitectura y Tecnología de Computadores del Departamento de Ingeniería y Ciencia de los Computadores desde el año 2003.
Su actividad docente se ha centrado principalmente en las antiguas titulaciones de informática y, actualmente, en el Grado de Ingeniería Informática y en el de Diseño y Desarrollo de Videojuegos. Ha participado en diferentes proyectos de innovación docente, es autor de varias comunicaciones en congresos sobre innovación docente, así como de una serie de libros de apuntes de diferentes asignaturas relacionadas con el área de Arquitectura y Tecnología de Computadores. Ha sido Presidente de la Comisión Docente del Departamento de Ingeniería y Ciencia de los Computadores.

Germán León Navarro es Licenciado en Informática, especialidad sistemas físico, por la Universitat Politécnica de Valencia y Doctor por la Universitat Jaume I. Ha sido profesor universitario desde 1989 y actualmente es profesor titular del Área de Arquitectura y Tecnología de Computadores de la Universitat Jaume I.
Ha ejercido de Vicedirector de la Escuela Superior de Tecnología encargado de la titulación de ITIG. Ha dirigido y participado en 12 proyectos de innovación docente, 3 proyectos de formación de noveles, es autor de 8 comunicaciones en congresos nacionales e internacionales sobre innovación docente y de 6 libro docente, y ha recibido un premio relacionado con la innovación docente.