SISA-EMU: *feedback* automático para ensamblador

Carlos Álvarez, Daniel Jiménez-González, David López, Javier Alonso, Ruben Tous, Joan M. Parcerisa, Pere Barlet, Montse Fernández, Jordi Tubella, Christian Pérez
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
C/Jordi Girona, 1 i 3. Campus Nord. Mòduls C6 i D6
{calvarez, djimenez, david, alonso, rtous, jmanel, pbarlet, montsef, jordit, christip}@ac.upc.edu

Resumen

Un estudiante de primer curso de Ingeniería en Informática debe adquirir la capacidad de analizar y depurar códigos, tanto a alto nivel como en ensamblador. Este proceso requiere una participación activa por parte de los estudiantes, sobre todo en el laboratorio. Sin embargo, un entorno de laboratorio desconocido y la falta de conocimientos previos llevan a los alumnos a basar su aprendizaje en la realimentación (*feedback*) ofrecida por el profesor. Esta dependencia hace que los alumnos no utilicen el software de laboratorio para solucionar los problemas de teoría y trabajar de forma autónoma. Además, la mayoría de las preguntas de los estudiantes son siempre las mismas, y pueden ser resueltas de manera fácil y rápida. Sin embargo, el profesor no está disponible en todo momento, por lo que una pregunta que podía resolverse en segundos mantiene al estudiante atascado durante minutos, hasta que dispone del profesor. Para solucionar esta situación se ha desarrollado el entorno SISA-EMU, que permite comprobar rápidamente si una solución es correcta. Además incorpora un sistema de realimentación automático que orienta a los alumnos sobre los errores que han cometido. Con esta herramienta se fomenta que los alumnos trabajen de forma semiautónoma y obtengan un mayor provecho de los problemas realizados.

1. Motivación

El objetivo de las primeras asignaturas de arquitectura y tecnología de computadores en las carreras de Ingeniería en Informática es introducir y analizar conceptos como la arquitectura de un procesador, el lenguaje ensamblador, la sincronización con el subsistema de entrada/salida o la estructura de la memoria y su gestión.

El uso de simuladores en lugar de procesadores reales es bastante común, dado que ofrecen un entorno más amigable y controlado que permite explicar los conceptos clave que atañen a un procesador, sin tener que lidiar con todos los pequeños detalles de programación del procesador real y sin necesidad de un hardware de prácticas específico, añadiendo nuevas funcionalidades conforme se necesitan. El lector puede encontrar unas reflexiones sobre las características necesarias para un buen entorno de simulación en los trabajos de Sánchez [7] y Vega et al [9]. Existen en la actualidad diversos simuladores, desde el tradicional DLX [3], hasta simuladores adaptados a las necesidades pedagógicas de muchos centros [1][2][4][5][6][10].

La arquitectura SISA (*Simple Instruction Set Architecture*), ha sido desarrollada con fines didácticos en el Departamento de Arquitectura de Computadores de la Universidad Politécnica de Cataluña. Su objetivo principal es facilitar la enseñanza del funcionamiento de un procesador completo a lo largo de diferentes asignaturas de la carrera, de manera que en cada asignatura se construya sobre un sistema conocido, con dificultad creciente. Se empieza en primero por un sistema básico (uniciclo), pasando por un sistema medio (multiciclo en orden) hasta un sistema con las técnicas más actuales (superescalar fuera de orden para la asignatura Arquitecturas de Computadores Avanzadas).

La asignatura Estructura de Computadores I está en segundo cuatrimestre de primer curso de las ingenierías en informática que se imparten en la Facultad de Informática de Barcelona. Es la segunda asignatura de arquitectura de computadores y la máquina a estudiar es la denominada SISA-F (multiciclo en orden con unidad de coma flotante). Para desarrollar las prácticas disponemos de un procesador simulado, denominado SISP-F, de 16 bits, RISC

y multiciclo. Para trabajar sobre este procesador se ha desarrollado un entorno de simulación (SISA-EMU) que permite al alumno desarrollar, probar y depurar sus programas. Hay, sin embargo, algo que distingue esta herramienta de otras: la capacidad de ofrecer realimentación.

Habíamos observado que los alumnos no eran capaces de extraer el máximo provecho de esta herramienta. La falta de conocimientos y el hecho de enfrentarse a un entorno desconocido hacía que los estudiantes pasaran a depender en el laboratorio de la realimentación ofrecida por el profesor. Por otro lado, hemos observado una tendencia por parte de los alumnos a separar la teoría del laboratorio, considerándolos como asignaturas desconectadas, de manera que no utilizan la herramienta fuera del laboratorio para comprobar si sus soluciones a los problemas de teoría son o no correctas.

La mayoría de las preguntas realizadas por los estudiantes en el laboratorio son siempre las mismas, y pueden ser resueltas rápidamente por el profesor. Sin embargo, el profesor no está disponible fuera de las horas de laboratorio, y dentro de éstas puede tardar en responder si está ocupado contestando a otros grupos. Si la propia herramienta pudiera ofrecer esta realimentación, el alumno tendría más información inmediata sobre su ejercicio sin depender del profesor, por lo que podría avanzar más deprisa y podría utilizar la herramienta fuera de las horas de laboratorio para comprobar sus propios ejercicios. Además, el profesor dedicaría su tiempo a responder dudas más conceptuales y menos mecánicas.

Por tanto, hemos cambiando de enfoque del simulador para transformarlo en una herramienta que ofrece una serie de guías y consejos al alumno a la hora de validar y depurar sus programas, consiguiendo un entorno mucho más amigable para programadores noveles. Así el alumno se sentirá más cómodo y seguro para usarlo fuera del aula de laboratorio sin el apoyo constante del profesor, de manera que se eliminaría la frontera entre teoría y laboratorio.

El resto de este artículo se organiza de la siguiente forma: en la sección 2 se explican los objetivos de la asignatura y su organización; en la sección 3 se explican las características de la herramienta. Finalmente, en la sección 4 se presentan las conclusiones y el trabajo futuro.

2. Descripción de la asignatura

2.1. Objetivos de la asignatura

Los objetivos de la asignatura se han dividido en tres categorías según lo descrito por F. Sánchez y R. Gavaldà [1], diferenciando entre objetivos generales y específicos cuando ha sido posible.

Dentro de los relacionados con contenidos técnicos, el objetivo general es que los alumnos que cursan esta asignatura han de ser capaces de entender el funcionamiento de un procesador escalar en orden y su relación con el resto de elementos que conforman un computador en el nivel de lenguaje ensamblador, así como su interrelación con el lenguaje de alto nivel y con el nivel de bloques lógicos.

Como objetivos específicos un alumno ha de:

1. Ser capaz de describir el funcionamiento de cualquier código escrito en lenguaje ensamblador de la máquina SISA-F.
2. Ser capaz de traducir cualquier código expresado en lenguaje de alto nivel C a lenguaje ensamblador SISA-F, de varias formas posibles.
3. Ser capaz de usar diferentes alternativas para gestionar estructuras de datos en ensamblador, con especial énfasis en la alternativa utilizada por C.
4. Ser capaz de explicar las distintas formas de comunicación del procesador con elementos externos de entrada salida.
5. Ser capaz de describir el funcionamiento general de los periféricos de entrada/salida más habituales.
6. Ser capaz de gestionar la comunicación del procesador con los periféricos de entrada/salida, tanto usando sincronización por encuesta como por interrupción.
7. Ser capaz de describir el funcionamiento de la memoria del procesador y enumerar las ventajas de la jerarquía de memoria.
8. Ser capaz de describir el funcionamiento de una memoria caché de mapeo directo.

Por lo que se refiere a objetivos relacionados con capacidades y aptitudes, el estudiante ha de:

9. Incrementar su capacidad de análisis de códigos.
10. Incrementar su capacidad de programación.

11. Mejorar sus habilidades de detección de errores en códigos programados.
12. Mejorar su capacidad de comprensión de los procesos internos de un sistema computador.

Finalmente, los objetivos relacionados con actitudes, valores y normas, el estudiante ha de capaz de:
13. Trabajar en equipo.
14. Aprender de forma autónoma.

2.2. La máquina SISA-F

La máquina SISA-F es una máquina multiciclo en orden con unidad de coma flotante. Concretamente, como ya se ha comentado en los objetivos, la máquina de arquitectura SISA-F se utiliza para enseñar lenguaje ensamblador y el sistema de entrada/salida del procesador. Los alumnos que llegan a esta asignatura han superado previamente otra asignatura en la que construyen por medio de unidades lógicas la máquina SISA-I (uniciclo) e implementan programas en ensamblador muy sencillos (de menos de 10 instrucciones). El Figura 1 muestra un ejemplo de programa sencillo que suma en la variable RES todos los elementos del vector V realizado en ensamblador SISA-F.

```
.include "macros.s"
       N = 5
.data
       RES: .word 0
       V:   .word -1, -2, -3, -4, -5
.text
 main:
  MOVI R1, 0     ; Contador
  MOVI R2, N     ; Límite
  MOVI R3, 0     ; Suma parcial
  $MOVEI R4, V  ; @ inicial de V
 bucle:
  CMPLT R5, R1, R2
  BZ R5, fibucle ; Acaba en 5
  ADD R5, R1, R1 ; Cada dato
                 ;ocupa 2 bytes
  ADD R5, R4, R5
  LD R6, 0(R5)   ; Leemos un dato
  ADD R3, R3, R6 ; Acumulamos
  ADDI R1, R1, 1 ; Actualizamos
                 ; el índice
  BNZ R1, bucle  ; Salta siempre
 fibucle:
  $MOVEI R4, RES; Resultado
  ST 0(R4), R3
  HALT           ; Acaba
```

Figura 1. Programa que recorre y suma los elementos de un vector en lenguaje ensamblador SISA-F.

2.3. Organización de la asignatura

Esta asignatura tiene 6 horas de clase presencial semanal, de las cuales 5 horas son de teoría (donde también se realizan problemas), y una hora es de laboratorio. Además, se espera que los estudiantes empleen unas 5 horas semanales de estudio y trabajo personal, más 15 horas de preparación de exámenes en el total del curso. Esto hace, para un cuatrimestre de 15 semanas un total de 180 horas de dedicación por parte del alumno. Por ello, nuestra escuela ha asignado a esta asignatura 7.2 créditos ECTS (a 25 horas el crédito).

La evaluación de la asignatura se realiza mediante un sistema de evaluación continuada que depende de 3 exámenes, un parcial con el 20% del peso, un examen de laboratorio con otro 20% y un examen final con el 60% restante. Los exámenes parcial y de laboratorio nunca pueden empeorar la nota obtenida en el final.

Los grupos de teoría son de 60 alumnos, y se dividen en 3 subgrupos de 20 en las prácticas de laboratorio. La hora semanal de prácticas se organiza en torno a seis sesiones de prácticas de 2 horas que los alumnos realizan cada quince días. Una última sesión adicional de 2 horas se utiliza para realizar el examen de laboratorio. En la Tabla 1 se pueden observar los contenidos de cada sesión.

Sesión de prácticas	Contenido
0	Introducción al entorno de prácticas
1	Aritmética entera y real
2	Datos estructurados
3	Subrutinas
4	Encuesta
5	Interrupciones
Examen Laboratorio	Todo

Tabla 1. Contenidos de las prácticas de la asignatura.

2.4. El laboratorio y su entorno

Para cada sesión de prácticas los alumnos deben realizar un trabajo previo individual, que consiste en escribir una serie de programas de dificultad creciente que cubren los temas explicados en teoría. Este estudio previo repasa los conceptos que se van a tratar en la sesión. Ya

en el laboratorio, los alumnos se agrupan en parejas y proceden a poner en común su trabajo.

El entorno para realizar estas prácticas consiste en tres programas: un ensamblador, un enlazador, y un programa depurador y simulador. Gracias a este entorno los alumnos pueden comprobar el funcionamiento de cualquier programa realizado, ya que el simulador incorpora entre sus funcionalidades la simulación de los dispositivos de entrada/salida más usuales como la impresora, el teclado, el reloj, la pantalla e incluso el disco.

Para comprobar que el programa funciona correctamente el simulador ofrece la posibilidad de ejecución continua o paso a paso, y de inspeccionar valores de memoria, tal y como se muestra en la Figura 2. También se puede observar la ejecución del programa "desde fuera" si el programa interacciona con los dispositivos de entrada/salida, tal y como muestra la Figura 3. En los laboratorios siempre está presente un profesor al que los alumnos pueden pedir ayuda para aclarar conceptos.

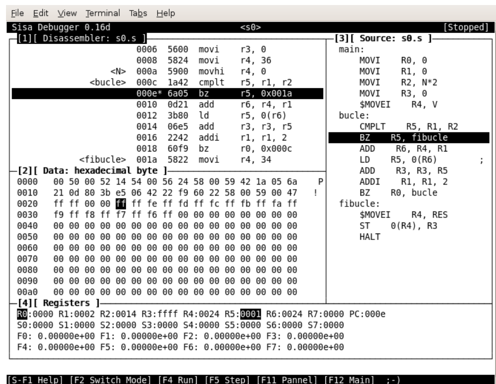

Figura 2. El simulador SISA-F. Vista del depurador.

2.5. El problema

Gracias a este entorno los alumnos pueden realizar las prácticas de una forma casi autónoma. La realidad observada, sin embargo, contradice este punto: resulta muy habitual que los alumnos no sean capaces de seguir las instrucciones de comprobación de un resultado de una forma correcta, sobre todo a medida que los programas se complican. Ante un resultado equivocado suelen ser incapaces de determinar si el fallo está en el programa que han realizado o en la forma de comprobar el resultado. Lo que es peor, si realmente se han equivocado realizando el programa, mayoritariamente son incapaces de investigar con el depurador qué es lo que han hecho mal y arreglarlo. En este punto su único recurso es acudir al profesor de laboratorio.

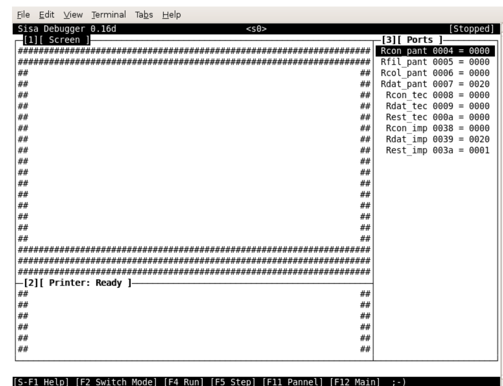

Figura 3. El simulador SISA-F. Vista de la simulación.

Además, este sistema se ha comprobado inútil a la hora de trascender el aula de laboratorio: los alumnos no utilizan este software para comprobar que sus problemas están bien hechos. Ante un resultado negativo no son capaces de encontrar el fallo de sus programas por sí mismos, por lo que el entorno de simulación no les resulta útil para apoyar los conocimientos de las clases de teoría. La dependencia de las "pistas" que proporciona el profesor es casi absoluta, lo que limita el uso del software simulador al laboratorio. Ciertamente, el proceso de depuración de un programa con el simulador en modo de ejecución paso a paso requiere habilidades deductivas y a menudo resulta un proceso tedioso y complejo que causa el abandono por parte de los alumnos. Por el contrario, les resulta mucho más fácil si pueden acotar el problema con la ayuda de las pistas ofrecidas por el profesor.

Para centrar el problema, se pasó una encuesta a los alumnos de la asignatura donde se les pedía que valoraran del 1 al 4 (1 muy en desacuerdo y 4 muy de acuerdo) las

afirmaciones que se pueden observar en la Tabla 2 sobre el entorno actual de simulación. La encuesta fue contestada por 59 alumnos y sus resultados se reflejan en la Figura 4. Como se puede observar, los estudiantes valoran muy positivamente el simulador del que ya disponen (preguntas 1 y 2). Sin embargo, la gran mayoría admite que no utiliza el simulador para resolver los ejercicios de problemas (pregunta 3) y piensa que el simulador sería más útil si les proporcionase más información (pregunta 4).

#	Afirmación
1	El simulador me ayuda a entender la asignatura
2	El simulador me resulta útil para encontrar un fallo en un programa
3	Utilizo el simulador de laboratorio para resolver los problemas de la colección de ejercicios
4	El simulador sería más útil si me diera pistas de los errores que cometo.

Tabla 2. Afirmaciones de la encuesta.

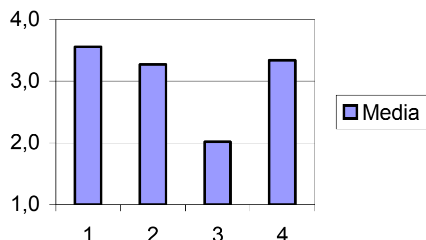

Figura 4. Resultados encuesta.

La encuesta incluía una quinta pregunta que pedía a los estudiantes que describiesen su forma de depurar programas. Nadie afirmó depurar mediante prueba y error (pese a que es un sistema que se puede observar habitualmente en clase). Un 55% contestó que lo hacían ejecutando paso a paso, pero un 45% dijo que simplemente miraban el código a ver lo qué habían hecho mal. La experiencia (y la nota de laboratorio) demuestra que este último método de depuración no funciona.

3. SISA-EMU

3.1. Objetivos a mejorar

Como ya se ha comentado, el sistema actual resulta pobre en lo que se refiere a la enseñanza de los objetivos 9 y 11 de la asignatura. Esto implica, además, que los objetivos 1 a 6 (que se trabajan en laboratorio) se cubren de una forma menos satisfactoria de lo que nos gustaría. Mejorar estos puntos requiere fomentar el trabajo práctico de los alumnos en el entorno de laboratorio. Sin embargo, como se ha explicado en el punto anterior, los alumnos encuentran este entorno poco útil y se resisten a utilizarlo excepto cuando se ven obligados a ello.

Con el objetivo de mejorar la percepción de los alumnos y su rendimiento se ha decidido incorporar un nuevo elemento al entorno de trabajo. Este elemento es un programa de corrección automática que los alumnos podrán usar para obtener, en la mayoría de los casos, la información que anteriormente les proporcionaba el profesor después de revisar su programa.

El nuevo entorno, por tanto, debería ser capaz de devolver, a partir de una solución realizada por los alumnos, una evaluación de si esta solución es correcta o no, y en caso de que no lo sea, debería proporcionar orientaciones acerca de en qué puede estar equivocada, de forma que el proceso de aprendizaje se vuelva independiente del profesor.

3.2. El sistema de autocorrección

El sistema de autocorrección debe permitir a los alumnos obtener de una forma sencilla una evaluación de su programa. Si además, el programa no funciona total o parcialmente, debería ofrecer una orientación de qué tipo de error es el que provoca el fallo. Es decir, se trata de un sistema de respuesta a los errores más frecuentes. Este sistema permitirá por un lado que los alumnos sean capaces de desenvolverse en el entorno de prácticas en ausencia del profesor y, por otro que éste pueda dedicar más tiempo a resolver los problemas de concepto.

Para conseguir estos objetivos, la tarea del profesor debe ser definir tanto los errores más habituales de cada problema como la forma de comprobar inequívocamente si se ha cometido

dicho error en la ejecución del programa. Los pasos que permiten llegar a un pseudocódigo de comprobación son los siguientes:

1. Definición del resultado correcto del problema.
2. Definición de unos valores de comprobación inequívocos para saber si el resultado es correcto.
3. Definición de los errores generales más habituales que realizan los alumnos.
4. Definición de los criterios de comprobación de dichos errores generales.
5. Definición de los errores específicos del problema más habituales.
6. Definición de los criterios de comprobación de dichos errores específicos.

Una vez hemos definido estos criterios, el resto del proceso consiste en programar un código de validación que compruebe dichos criterios y devuelva mensajes de validación adecuados al contexto.

3.3. *Scripts* para realimentación

Para ofrecer realimentación, el nuevo entorno de prácticas incluye un soporte desarrollado en JAVA para realizar ejecución bajo línea de comandos del simulador, lo que permite realizar *scripts* de comandos a través de una serie de directivas predefinidas. La Tabla 3 muestra algunos de los ejemplos más representativos de estas instrucciones.

Tipos de directivas	Ejemplo
Ejecución	run, stop, reset, reload
Comprobación de valores del procesador	mem_read, reg_read, sym_value, pc, disasm
Comprobación de valores del sistema	clock_time, keyb_data, video_data, ptr_data
Comprobación de eventos	io_access_subscribe, clk_add_timer
Modificación de valores del procesador	mem_write, reg_write
Ejecución de eventos	keyb_send, clk_pause, clk_resume
Simulación detallada	stepi, breakpoint_add, returnpoint_add

Tabla 3. Grupos de directivas del simulador.

Por ejemplo, si se invoca la directiva Run(); el simulador ejecutará todo el programa hasta el final. Otras directivas permiten obtener información de los valores de cualquier elemento del procesador en un momento concreto, así como de los eventos del procesador. A su vez, estos eventos pueden ser provocados por instrucciones (el programa de prueba puede “introducir pulsaciones de teclado”). Finalmente el simulador permite programar todas las actuaciones que podríamos realizar en un depurador convencional, como introducir puntos de parada, ejecutar paso a paso, etc.

El elemento que da la mayor potencia a este sistema es que estas directivas están perfectamente integradas en el lenguaje Python, de forma que podemos realizar cualquier programa de comprobación de resultados, tan complejo como sea necesario. En estos *scripts* es donde incorporaremos las comprobaciones comentadas anteriormente.

3.4. Ejemplo de implementación

Como ejemplo de implementación vamos a mostrar el proceso seguido para obtener, a partir de un problema que tiene como resultado el código mostrado en la Figura 1, la realización de todo el proceso. La Figura 5 muestra el problema tal y como se le plantearía a los alumnos.

```
Traduce a ensamblador SISA-F el
siguiente programa en alto nivel:

#define N 5
int RES=0;
int V[N]={-1,-2,-3,-4,-5};

main(){
   register int i; // R1

   for (i=0; i<N; i++)
      RES=RES+V[i];
}

Si el programa es correcto el valor
final de la variable RES será de -
15.
```

Figura 5. Problema propuesto a los alumnos.

Como se puede ver en la Figura 5 el problema informa a los alumnos de lo que ha de hacer el programa, además de una posible comprobación de la solución.

Partiendo de este problema, debemos definir los criterios de corrección así como los criterios para detectar los errores más habituales y el mensaje de información sobre los mismos. En la

#	Criterio	Comprobación	Mensaje si se cumple
1	Código correcto	RES = -15 El programa acaba en el fin del main	Tu programa es correcto.
2	Errores habituales: Acceso no alineado	El programa genera una excepción por acceso no alineado	Tu programa realiza un acceso de 2 bytes a una dirección impar. Comprueba si calculas bien la dirección de los elementos V[i].
3	Error habitual: acceden a los elementos correctos pero no los suman en la variable destino	Acceden a los elementos que deben acceder pero RES no vale -15	Tu programa lee los valores que tocan pero no los sumas bien o no los guardas en memoria.
4	Error habitual: acceden a memoria pero no al vector V	Acceden a elementos cuya dirección no pertenece al vector	Lees direcciones de memoria que no tocan. Verifica el cálculo de la dirección que lees.
5	Error habitual: El programa no acaba	El programa no acaba	Tu programa no acaba, has programado un bucle infinito.
6	Error específico: Realizan 5 sumas en vez de un bucle	El código contiene 5 loads.	No debes sumar los 5 elementos uno a uno sino hacerlo con un bucle. Repite el problema.
7	Error específico: No leen de memoria los valores del vector, sólo calculan su dirección.	No ejecutan ninguna instrucción LOAD	Tu programa no lee los elementos V[i] de memoria. Debe hacerlo con instrucciones LOAD.
8	Error específico: No realizan correctamente el bucle (este no se ejecuta 5 veces)	El código debe ejecutar la instrucción de suma exactamente 5 veces.	Tu código tiene un bucle que se ejecuta N veces en vez de 5. Compruébalo.
9	El sistema no es capaz de determinar que hace el programa.	Ninguno de los anteriores.	Error indefinido, consúltalo con el profesor.

Tabla 4. Especificación del *script* de corrección

Tabla 4 se puede ver cómo se aplicarían los pasos explicados en el apartado 3.2 al programa de ejemplo. En el Apéndice A se muestra el *script*, escrito por los profesores, para realizar las comprobaciones y dar los mensajes descritos en la Tabla 4. En este apartado, la experiencia de los docentes resulta clave para determinar qué fallos típicos deben ser tratados y qué fallos no suelen ser tan habituales y por tanto no precisan tanta atención. Además, este es un proceso abierto ya que conforme los alumnos utilicen el sistema se pueden encontrar nuevas necesidades que requieran reevaluar estos aspectos.

Como se puede ver el sistema no sólo informa del resultado de la prueba sino que orienta a los estudiantes sobre dónde pueden haber cometido el fallo de programación. Estos mensajes deben ayudar a los alumnos a corregir el fallo en la siguiente versión y a la vez animarles a empezar a depurar sus programas.

4. Conclusiones y trabajo futuro

En este artículo hemos presentado un simulador para una asignatura relacionada con arquitectura de computadores de primer curso de Ingeniería Informática. La herramienta tiene un enfoque novedoso: además de las funciones habituales de un simulador, permite que el profesor defina los errores más frecuentes de cada ejercicio, de manera que el simulador ofrece realimentación automática (e inmediata) al estudiante. El entorno es simple y amigable para el estudiante, mientras que para el profesor es muy fácil de utilizar.

Un objetivo logrado es haber diluido la frontera entre teoría y laboratorio: los estudiantes encuentran la herramienta sencilla y útil, por lo que se animan a usarla para comprobar si los ejercicios que realizan en sus horas de estudio son correctos. Al disminuir la dependencia del estudiante respecto del profesor, logramos aumentar su capacidad de aprendizaje autónomo.

La herramienta está actualmente en fase de pruebas, y aunque ya lo usan los alumnos, tenemos como objetivo a medio plazo integrarla en la plataforma Moodle, por lo que todavía no es pública. Sin embargo, nuestra intención es ponerla a disposición de la comunidad educativa, así que animamos a quien esté interesado a ponerse en contacto con los autores.

Agradecimientos

Este trabajo ha sido desarrollado con el apoyo del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya (proyecto 2007 MQD 00203) y de la Facultad de Informática de Barcelona (FIB).

Referencias

[1] R. Conchiero, M. Loureiro, M. Amor, P. González. *Simula3MS: simulador pedagógico de un procesador*. JENUI 2005.

[2] J. Cortés Ayala, T. Ruiz Vázquez, I. Etxeberria Uztarroz. *Guia Visual y dinámica del funcionamiento de un procesador didáctico sencillo*. JENUI 2005.

[3] J.L. Hennessy, D.A. Patterson. *Computer Architecture, A quantitative approach*. 3rd. ed. Morgan Kauffmann Publishers. 2003

[4] R. Laza, D. Glez-Peña, J.R. Méndez, F. Fdez-Riverola, J. Baltasar García, M. Reboiro. *ALT: Algorithm Learning Tool*. JENUI 2007.

[5] M. Prieto, A.J. de Vicente, R. Aldea, E. Bonillo. *Soluciones para las prácticas de E/S*. JENUI 2006.

[6] Ll. Ribas Xirgo, D. Rodríguez Jurado. *jsYASP: Un simulador de un procesador educativo en Javascript*. JENUI 2005

[7] F.Sánchez. *Características deseables en un procesador pedagógico para la enseñanza básica de la arquitectura de computadores*. JENUI 2002.

[8] F. Sánchez y R. Gavaldà. *Objetivos formativos del primer curso de las ingenierías informáticas y estrategias docentes relacionadas*. SINDI2005.

[9] M.A. Vega, J.A. Gómez, J.M. Sánchez. *Innovación docente en la arquitectura de computadores mediante el uso de simuladores*. JENUI 2000.

[10] A.J. de Vicente, R.M. Estriégala, V. Escuder. *Ensambla-T*. JENUI 2006.

Apéndice A

```
def hand_mem(addr,mode,size):
  global lastaddr, stride, stride_wrong
  global addr_ok, addr_wrong
  global num_reads_ok, num_reads
  global lastread, oneload

  if mode == MemAccessMode.READ:
    num_reads = num_reads + 1
    addr_ok = 0
    for i in range(5):
      if addr == sym_value("V")+i*2:
        addr_ok = 1
    if addr_ok == 0:
      addr_wrong = 1
    else:
      num_reads_ok = num_reads_ok + 1
      if lastread == 0:
        lastread=pc()
      else:
        if lastread != pc():
          oneload = 0
      if lastaddr == -1:
        lastaddr = addr
      elif stride == -1:
        stride = addr - lastaddr;
        lastaddr = addr
      elif stride != addr - lastaddr:
        stride_wrong = 1
      else:
        lastaddr = addr
  else:
    if addr != sym_value("RES"):
      addr_wrong = 1;

#Inicialización de variables
num_reads_ok = 0;num_reads = 0
addr_wrong = 0; lastaddr = -1
stride = -1; stride_wrong = 0
lastread = 0; oneload = 1

# Activamos handler de acceso a memoria
mem_access_subscribe(hand_mem)

# Ejecutamos y analizamos resultados
stepi(1000)

if pc() == sym_value("RSE_default_halt"):
  print "Mensaje 2 de Tabla 4"
elif disasm(pc()) != 'halt  ':
  print "Mensaje 5 de Tabla 4"
elif num_reads == 0:
  print "Mensaje 7 de Tabla 4"
elif oneload != 1:
  print "Mensaje 6 de Tabla 4"
elif sym_read16("RES") == -15 and \
   num_reads_ok == 5 and (stride == 2 or\
   stride == -2) and addr_wrong == 0:
  print "Tu programa es correcto."
elif num_reads_ok == 5 and (stride == 2 \
   or stride == -2) and addr_wrong == 0:
  print "Mensaje 3 de Tabla 4"
elif num_reads != 5:
  print "Mensaje 8 de Tabla 4"
elif addr_wrong == 1:
  print "Mensaje 4 de Tabla 4"
else:
  print "Mensaje 9 de Tabla 4"
```