WepSIM: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador

Alejandro Calderón Mateos, Félix García Carballeira, Javier Prieto Cepeda

Computer Architecture Group, Computer Science and Engineering Department
Universidad Carlos III de Madrid, Leganés, Madrid, Spain.
acaldero,fgcarbal@inf.uc3m.es, javpriet@pa.uc3m.es

Resumen Lo que diferencia WepSIM[1] de otros simuladores usados en la enseñanza de Estructura de Computadores está en tres aspectos importantes. Primero, ofrece una visión integrada de la microprogramación y de la programación en ensamblador, dando la posibilidad de trabajar con distintos juegos de instrucciones. Segundo, permite al estudiante una mayor movilidad al poder usarse también en dispositivos móviles.
Tercero, tiene un diseño modular, es posible añadir, quitar o modificar los elementos existentes. Busca un equilibrio entre simplicidad, para facilitar la enseñanza, y el detalle, para imitar la realidad. Una de las grandes ventajas del simulador WepSIM es que no está limitado a un juego de instrucciones concreto, permitiendo definir un amplio juego de instrucciones de procesadores reales o inventados.
Este artículo describe WepSIM y los resultados en la primera experiencia de su uso en la asignatura de Estructura de Computadores impartida en el Grado en Ingeniería Informática de la Universidad Carlos III de Madrid.

Palabras Clave: Estructura de Computadores, Procesador Elemental, Microprogramación, Programación en ensamblador

Abstract There are three differences between WepSIM[1] and other simulation tools used in Computer Structure Teaching. First one, it offers an integrated vision of microprogramming and assembly programming where different instructions sets can be defined. Second one, it provides more flexibility to students by using mobile devices. Third one, it was created with a modular design. It is possible to add, remove or modify the existing elements, and it is simple enough to be used for teaching but with enough details to mimic the reality.
This paper introduces WepSIM, and the results obtained in the first experience of usage in the Computer Structure course from the Bachelor's

[1] http://www.arcos.inf.uc3m.es/~ec-2ed

Degree in Computer Science and Engineering in the Universidad Carlos III de Madrid.

Keywords: Computer Structure, Elemental Processor, Microprogramming, Assembly programming

1 Introducción

Hay distintos simuladores que se pueden utilizar para trabajar con los principales aspectos que se tratan en las asignaturas de Estructura y Arquitectura de Computadores: ensamblador, caché, etc. Aunque la idea de usar distintos simuladores cae dentro de la estrategia de "divide y vencerás", hay dos principales problemas con estos simuladores: cuanto más realistas son más compleja se hace la enseñanza (tanto del simulador como de la tarea simulada), y cuantos más simuladores se usan más se pierde la visión de conjunto.

Hay otro problema no menos importante: la mayoría de los simuladores están pensados para PC. Uno de los objetivos que nos planteamos con WepSIM es que pudiera ser utilizado en dispositivos móviles (*smartphones* o *tablets*), para ofrecer al estudiante una mayor flexibilidad en su uso.

Además de tener un simulador portable a distintas plataformas, el simulador ha de ser lo más autocontenido posible de manera que integre la ayuda principal para su uso (no como un documento separado que sirva de manual de uso para ser impreso).

Por todo ello, nos hemos planteado cómo ofrecer un simulador que sea simple y modular, y que permita integrar la enseñanza de la microprogramación con la programación en ensamblador. En concreto, puede utilizarse para microprogramar un juego de instrucciones y ver el funcionamiento básico de un procesador, y para crear programas en ensamblador basados en el ensamblador definido por el anterior microcódigo. Esto es de gran ayuda, por ejemplo, para la programación de sistemas dado que es posible ver cómo interactúa el software en ensamblador con el hardware en el tratamiento de interrupciones. La idea es ofrecer un simulador que ofrezca una visión global de lo que pasa en hardware y software.

En este artículo vamos a introducir el simulador WepSIM (*Web Elementary Processor SIMulator*) que hemos diseñado e implementado buscando alcanzar los anteriores objetivos. El simulador WepSIM está evolucionando con la experiencia de su uso, y queremos compartir los pilares del mismo, las ideas que hay detrás de la propuesta y los primeros resultados alcanzados.

El resto del documento se organiza como sigue: la segunda sección del documento introduce la arquitectura de WepSIM y el modelado de hardware que hemos propuesto. La tercera sección introduce el procesador elemental que se simula usando la arquitectura anteriormente presentada. También se describe la forma de definir un determinado juego de instrucciones así como el microcódigo asociado al mismo. Con este formato los estudiantes pueden diseñar e implementar código fuente en cualquier ensamblador previamente definido. La cuarta

sección resume los principales aspectos de su implementación. La quinta sección repasa el estado del arte, y finalmente la sexta sección muestra las conclusiones y trabajos futuros.

2 La arquitectura de WepSIM y el modelo hardware

La arquitectura de la solución presentada (véase la Figura 1) en este trabajo tiene tres elementos principales: (1) un modelo hardware que permite definir el hardware a usar, (2) un modelo software que permite definir el microcódigo/-lenguaje máquina a utilizar y (3) un *motor* que simula el funcionamiento del hardware ejecutando el microcódigo/lenguaje máquina definido anteriormente.

El modelo hardware permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular. La forma de definir estos elementos equilibra dos objetivos contrapuestos: es suficientemente completa como para imitar los principales aspectos de la realidad, pero es lo suficientemente mínima para facilitar su uso. Ante todo se persigue que sea una herramienta didáctica.

El modelo software permite definir el microcódigo y el ensamblador basado en este microcódigo de la forma tan intuitiva posible. El ensamblador a usar viene dado por un conjunto de instrucciones que puede ser definido por el profesor/alumno e intenta ser lo suficientemente flexible como para poder definir diferentes tipos y juegos de instrucciones, como por ejemplo MIPS o ARM.

El tercer elemento de la arquitectura propuesta es un *motor* que toma como entrada el modelo hardware descrito y el modelo software de trabajo, y se encarga de mostrar el funcionamiento del hardware con el software dado.

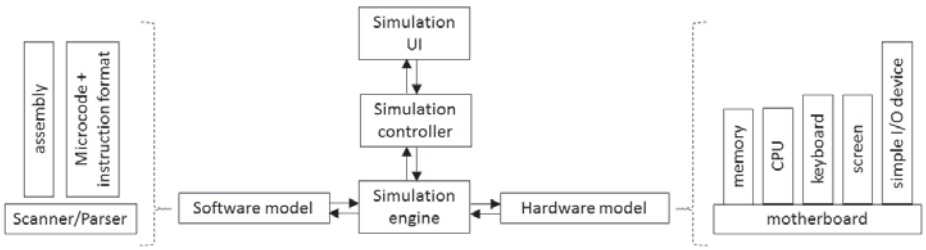

Figura 1. Arquitectura del simulador WepSIM.

La figura 1 resume la arquitectura de WepSIM. El punto de inicio es el modelo hardware que describe el procesador a ser simulado. Ello incluye el procesador, la memoria y algunos dispositivos de E/S: teclado, pantalla y un dispositivo de E/S simple que genera interrupciones. El modelo hardware describe el estado global del procesador. A partir del estado global del procesador, el motor de simulación actualiza el estado en cada ciclo de reloj.

La unidad de control simulada almacena las señales de control de cada ciclo en una memoria de control. La memoria de control tiene todos los microprogramas para las instrucciones con las que trabaja el procesador, y el *fetch* para leer la instrucción de memoria y decodificarla.

El microcódigo (el contenido de la memoria de control) junto con el formato de cada instrucción (campos de la instrucción y su longitud) se describe en un fichero de texto. El modelo software lee este fichero, lo traduce a binario y lo carga en el procesador. La definición del lenguaje ensamblador a utilizar se describe junto con el microcódigo, y el modelo software permite traducir a binario programas escritos en dicho ensamblador.

El motor de simulación pregunta al subsistema del modelo software por el microcódigo definido, la descripción del formato de instrucción y el contenido de la memoria principal. Los binarios se cargan en los elementos del modelo hardware, y a continuación el motor de simulación actualiza el estado global en cada ciclo de reloj.

WepSIM dispone de un controlador de simulación que se encarga de actualizar el ciclo de reloj y mostrar el estado global. El subsistema de interfaz de simulación actualiza la interfaz de usuario. Cuando el usuario usa la interfaz de usuario para solicitar una operación, el subsistema de interfaz de simulación traslada la petición al controlador de simulación. Como se puede ver, se usa un Modelo-Vista-Controlador (MVC) básico para la arquitectura de WepSIM.

2.1 El modelo hardware en WepSIM

El modelo hardware que usa WepSIM permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular y de manera que sea posible añadir, quitar o modificar estos elementos.

La figura 2 introduce el modelo propuesto. Cada elemento del circuito se describe como una caja negra con posibles entradas, posibles salidas y señales de control (que controlan las posibles transformaciones de las entradas a las salidas). El subsistema del modelo hardware transforma esta caja negra en dos conjuntos de objetos: estados y señales. Un estado tiene un identificador (el nombre), el valor (un valor entero) y un valor inicial (el valor por defecto). Los valores que puede tomar son valores naturales dentro de un rango, dado por el número de bits con los que se representa el estado. Una señal es un estado especial que controla el valor de otros estados o señales. Hay dos atributos asociados a las señales (y no a los estados): el tipo de señal (por nivel o por flanco) y su comportamiento. Para cada valor de señal, una cadena de caracteres describe en un Lenguaje Simple lo que la señal mueve o transforma. Este Lenguaje Simple se compone principalmente de instrucciones que representan las operaciones elementales.

Las figuras 3 y 4 muestran dos ejemplos: un triestado y un registro.

El triestado controla dos estados: el estado del bus al que se conecta BUS_IB y el estado del registro de entrada, REG_RT1 en este caso. Ambos representan el valor a la salida de la puerta (BUS_IB) y el valor del registro RT1 (REG_RT1). La señal T4 se encarga de indicar cuándo el valor del registro RT1 se envía a la salida. Esta señal T4 es una señal por nivel (tipo: L), con valor cero no

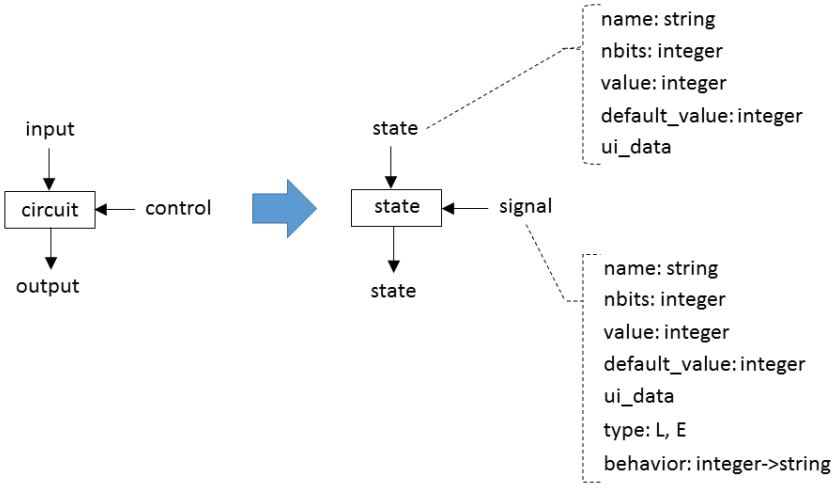

Figura 2. Modelado del hardware.

tiene efecto (comportamiento "NOP"). Cuando el valor de la señal es uno entonces el comportamiento es el de copiar el valor del registro RT1 a la salida (comportamiento "MV BUS_IB REG_RT1").

El ejemplo con el registro (véase la figura 4) es similar. En este caso trabaja con dos estados: el contenido del registro RT1 y el contenido situado a la entrada (BUS_IB). La señal C4 controla cuándo se almacena en el registro RT1 el valor que hay en la entrada. La diferencia está en el tipo de señal: C4 es una señal por flanco de bajada (tipo: E), por lo que al final del ciclo de reloj (pasa de uno a cero) si la señal vale uno entonces el comportamiento es el de copiar el valor situado a la entrada al registro (comportamiento "MV REG_RT1 BUS_IB").

El Lenguaje Simple usado para definir los comportamientos añade a las operaciones elementales otras operaciones necesarias. Por ejemplo disparar una señal ("FIRE C4") que ayuda a propagar el efecto de una señal al re-evaluar la señal inmediata que podría verse afectada. Otro ejemplo lo encontramos en dos operaciones que pueden ser muy útiles a la hora de depurar: imprimir el valor de un estado ("PRINT_E BUS_IB") e imprimir el valor de una señal ("PRINT_S C4").

3 El procesador elemental WepSIM

Usando el modelo simplificado introducido anteriormente es posible definir todos los elementos de nuestro procesador elemental. La figura 5 muestra la estructura de Procesador Elemental WepSIM.

Los elementos del procesador se describen con 57 estados y 62 señales, de forma similar a los ejemplos comentados anteriormente (de hecho T4 y C4 están

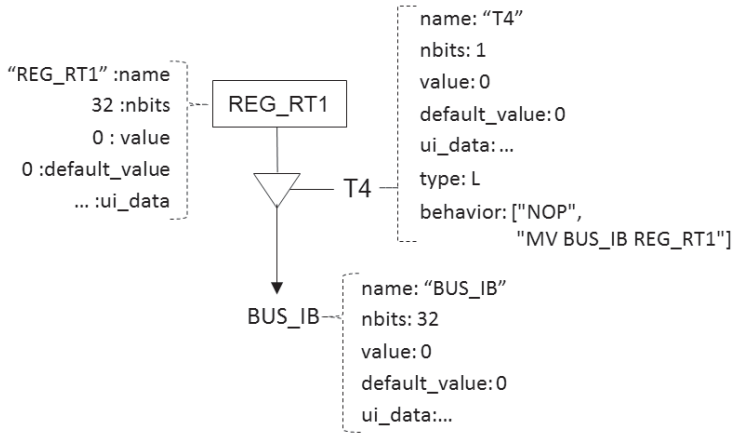

Figura 3. Ejemplo de modelado de una puerta triestado.

incluidas entre las 62 señales, así como REG_RT1 y BUS_IB (bus interno) están incluidos entre los 57 estados). El lenguaje que describe los comportamientos incluye 46 operaciones. La ALU precisa de 15 de ellas, y la memoria y dispositivos de E/S precisan alrededor de 15 operaciones más. WepSIM tiene un módulo de memoria, un dispositivo teclado, un dispositivo pantalla y un dispositivo de E/S genérico que se usa para trabajar con interrupciones. También incluye un procesador que está etiquetado en la figura 5 como "Processor".

WepSIM es un procesador de 32 bits que direcciona la memoria por bytes, que tiene un banco de registros (con 32 registros) y dos registros adicionales (RT1 y RT2) que no son visibles para el programador de ensamblador. Desde los registros es posible enviar los valores para operar en una ALU con 15 operaciones aritmético-lógicas, que incluyen las más comunes. El registro PC tiene su propio operador de sumar cuatro, de forma que la ALU no es necesaria para esta operación. El resultado se puede almacenar en un registro temporal (RT3) transparente también para el programador de ensamblador, o se puede enviar directamente al bus interno a través del triestado correspondiente.

El registro de estado (SR) se puede actualizar con los flags resultantes de la última operación de la ALU (O, N y Z). Para ello SELEC/SelP representa un bloque de circuitos (de más alto nivel que un multiplexor, demultiplexor, etc.) que permite indicar qué parte del registro de estado (SR) actualizar. A la derecha de SELEC/SelP llega los bits del registro de estado SR como entrada (Input=O N Z I U) y SelP permite seleccionar qué grupo de estos bits se actualizará en el registro de estado: los bits O, N y Z con los valores procedentes de la ALU, el bit I con el valor indicado o el bit U con el valor indicado para el mismo.

El registro de instrucción (IR) tiene asociado un módulo selector (circuito de más alto nivel que un multiplexor, etc.) que permite seleccionar una porción del valor binario almacenado en el registro de instrucción que pasará hacia T3.

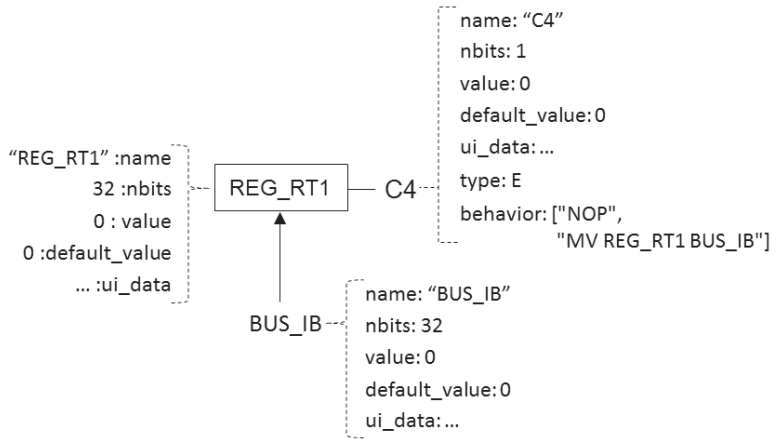

Figura 4. Ejemplo de modelado de un registro.

En concreto, se indica la posición (*Offset*, donde 0 represente el bit menos significativo del registro IR) inicial y el número de bits (*Size*) a tomar a partir de dicha posición inicial, así como si se desea hacer extensión de signo (*SE*) antes de pasar el valor a la entrada de T3.

Los registros MAR y MBR se usan para almacenar la dirección y el contenido asociado a esta dirección en las operaciones de lectura/escritura con la memoria. La memoria está diseñada para un funcionamiento síncrono o asíncrono. Actualmente funciona de forma síncrona, pero dispone de la señal MRdy para en un futuro trabajar de forma asíncrona. El circuito de selección permite indicar qué porción de la palabra de memoria es la que se desea (un byte, dos bytes o una palabra completa de cuatro bytes).

Se dispone también de tres dispositivos de E/S: un teclado, una pantalla y un dispositivo genérico que se puede configurar para generar diversos tipos de interrupciones.

Finalmente, la Unidad de Control genera las señales de control para cada ciclo de reloj.

La figura 6 muestra la Unidad de Control en más detalle. Se trata de una unidad de control microprogramada con secuenciamiento implícito. Las señales de control para el ciclo de reloj actual se almacenan en el registro de microinstrucción (aquel con los campos A0, B, C, SelA, etc.). El contenido de este registro proviene de la memoria de control, concretamente del contenido en la posición a la que apunta el registro de microdirección ($\mu Addr$). La microdirección almacenada en este registro puede modificarse usando el multiplexor "MUX A". Hay cuatro opciones: la microdirección actual más uno, una microdirección indicada en la propia microinstrucción (que se solapa con SelA, SelB y parcialmente con SelE), la primera microdirección asociada al campo de código de operación de la instrucción del registro IR, y finalmente el valor cero, que es la dirección de la

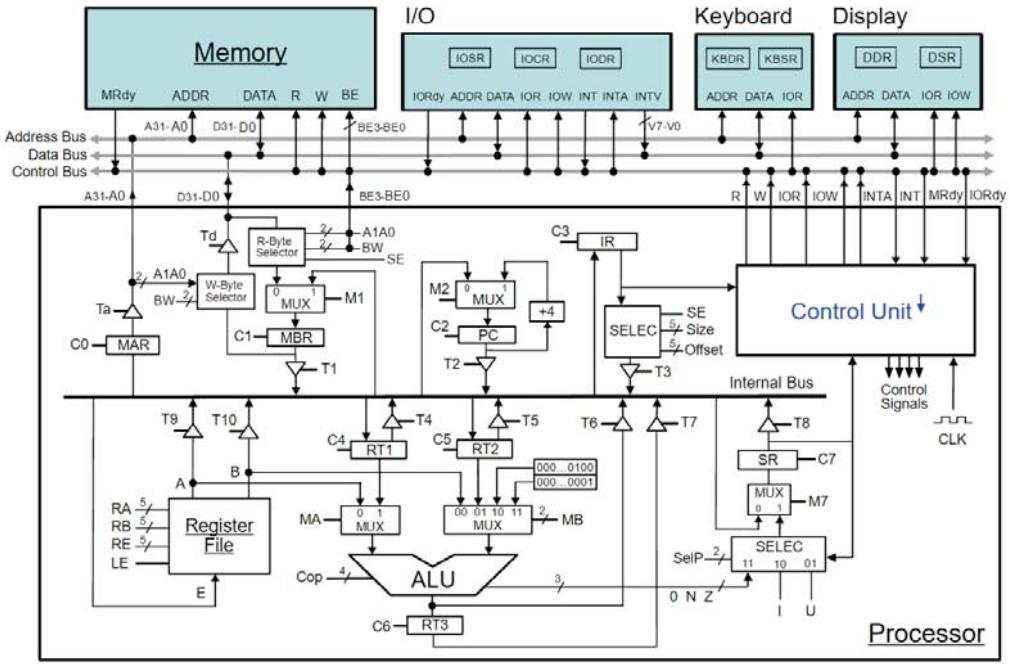

Figura 5. Estructura del Procesador Elemental WepSIM.

memoria de control donde se almacena la microrrutina correspondiente al *fetch*. La microdirección se puede seleccionar de forma condicional, para ello se usa el multiplexor "MUX C"que permite seleccionar los bits del registro de estado (SR) o valores de las señales de control de E/S.

Para generar los valores correspondientes a las señales selectoras del banco de registros RA, RB y RE se usan los circuitos selectores SelRA, SelRB y SelRE. Estos selectores toman como entrada los 32 bits del registro de instrucción (IR) por un lado y el campo SelA, SelB y SelE por otro, de forma que toman SelX como el desplazamiento dentro del registro de instrucción (de 0 a 32) desde donde tomar los siguientes 5 bits correspondientes a las señales RA, RB o RE. Permiten por tanto seleccionar 5 bits consecutivos de los 32 bits del registro de instrucción.

El multiplexor MR permite indicar si las señales RA, RB y RE serán literalmente los valores almacenados en SelA, SelB y SelE (MR=1) o bien si SelA, SelB y SelE indican el desplazamiento dentro de la instrucción donde están los valores a utilizar para RA, RB y RE. Esto último permite que en la instrucción se pueda indicar los registros a usar como operandos en el banco de registros, en lugar de indicarlos desde la microinstrucción.

Para la señal Cop (código de operación en la ALU) la señal MC permite tomar el valor SelCop de la microinstrucción (MC=1) o bien los 4 bits menos significativos del registro de instrucción (MC=0), es decir IR3-IR0.

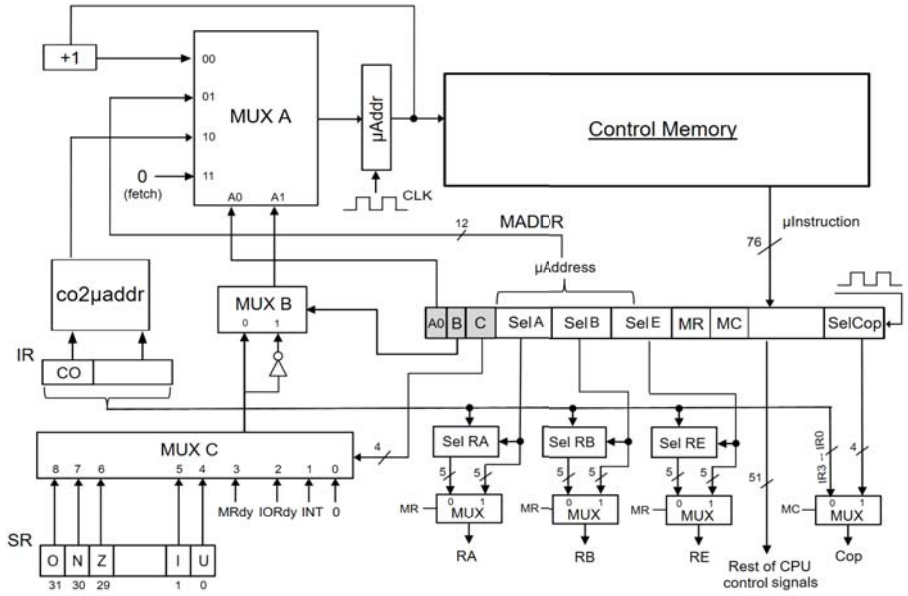

Figura 6. Unidad de control del procesador WepSIM.

En [1] se puede encontrar una descripción más detallada del procesador descrito anteriormente.

3.1 Definición del microcódigo y del conjunto de instrucciones a usar en WepSIM

Una vez definido el procesador elemental usando el modelo hardware propuesto, toca describir el conjunto de instrucciones que es capaz de ejecutar así como el microcódigo que lo orquesta. En un fichero de texto se define el formato de las instrucciones máquina junto con el cronograma asociado a la ejecución de cada una de las instrucciones máquina. El listado 1.1 muestra un ejemplo de definición para la instrucción **li** (*load inmmediate*), que almacena un valor inmediato en un registro.

El fichero con el cronograma de fetch y todos los cronogramas de las instrucciones define el microcódigo para la plataforma WepSIM. El simulador permite la definición de diferentes juegos y formatos de instrucciones. Inicialmente hemos implementado un subconjunto de las instrucciones del MIPS, pero es posible definir instrucciones de otros conjuntos de forma similar. En este fichero se pueden asignar códigos simbólicos a los registros del banco de registros, lo que permite que en los programas escritos en ensamblador se puedan usar dichos símbolos (por ejemplo, registro `$t3` en el listado 1.2).

El campo "co" identifica el código de instrucción máquina, que es un número binario de 6 bits. Esto permite definir hasta 64 instrucciones distintas. Dado

```
li reg val
{
       co=000010,
       nwords=1,
       reg=reg(25,21),
       val=inm(15,0),
       {
          (SE=0, OFFSET=0, SIZE=10000, SE=1, T3=1,
           LE=1, MR=0, SELE=10101, A0=1, B=1, C=0)
       }
}
```

Listado 1.1. Ejemplo de formato de instrucción y su cronograma asociado.

que los últimos 4 bits de la instrucción pueden usarse para seleccionar la operación en la ALU, es posible seleccionar hasta 16 operaciones aritmético-lógicas con un mismo código de instrucción, por lo que se podrían tener 79 (63+16) instrucciones en total.

Cuando WepSIM carga el microcódigo, cada código de instrucción tiene asociado una dirección de comienzo en la memoria de control donde se almacena el cronograma asociado. Esta tabla con dos columnas (el código de instrucción y su dirección de comienzo asociada en la memoria de control) se carga en la ROM $co2\mu Addr$ mostrada en la figura 6.

El campo "nwords" define cuantas palabras precisa la instrucción para su definición y carga en memoria. Una palabra en WepSIM son 4 bytes.

Para cada campo de la instrucción (*reg* y *val* del ejemplo del listado 1.1) se define el bit inicial, el bit final (ambos incluidos) y el tipo de campo (registro, valor inmediato, dirección absoluta y dirección relativa a PC). Una vez definido el formato, se definen todas las microinstrucciones que necesita la instrucción máquina definida para su ejecución. Todas las microinstrucciones se encuentran encerradas entre llaves y cada microinstrucción está formada por una lista de tuplas (señal, valor) encerradas entre paréntesis. Para la instrucción definida en el listado 1.1 se precisa de una sola microinstrucción, en la que se indican qué señales de activan durante un ciclo de reloj. Para las señales no indicadas se asume que su valor es 0 durante el ciclo de reloj correspondiente.

Una vez cargado el microcódigo en WepSIM, es posible cargar cualquier fichero ensamblador que haya sido codificado usando las instrucciones máquina definidas anteriormente en el microcódigo.

En el listado 1.2 se muestra un ejemplo de código fuente en ensamblador que se puede usar en WepSIM. Este ejemplo en particular (listado 1.2) muestra un código estilo MIPS. Para que un programa en ensamblador pueda utilizar la instrucción de carga inmediata li (load inmmediate) y de suma add (addition),

```
.text
main:  li  $t3 8
       li  $t5 10
       add $t6 $t3 $t5
```

Listado 1.2. Ejemplo de código fuente en ensamblador.

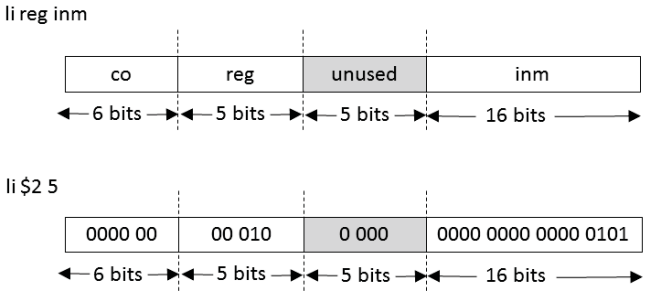

Figura 7. Formato de instrucción descrito en el microcódigo y ejemplo de su traducción en binario.

deben haber sido definidas previamente en el microcódigo. WepSIM puede comprobar los errores de sintaxis y construir el binario mediante el rellenado de los campos descritos en la definición del microcódigo correspondiente a la instrucción. La figura 7 muestra un ejemplo de traducción a binario para la instrucción `li $2 5` en función del formato definido en el listado 1.1. También se debe haber definido en el fichero de microcódigo el valor del registro asociado a la etiqueta `$2` (00100 en este caso).

Una de las grandes ventajas del simulador WepSIM es que no está limitado a un conjunto de instrucciones concreto. Se puede definir un amplio conjunto de instrucciones de procesadores reales o inventados. Se puede usar para añadir, por ejemplo, a un conjunto de instrucciones MIPS, otras instrucciones diferentes no incluidas en dicho conjunto de instrucciones.

4 El diseño y la implementación de WepSIM

Para acercar la docencia al alumno se ha tenido que buscar una forma de implementar y desplegar el simulador de forma que pueda ejecutarse en el mayor número de plataformas posibles, priorizando las plataformas móviles de forma que, por ejemplo, cuando un estudiante viaje en transporte público pueda usar el simulador desde un dispositivo móvil (con pantalla de cinco o más pulgadas) sin

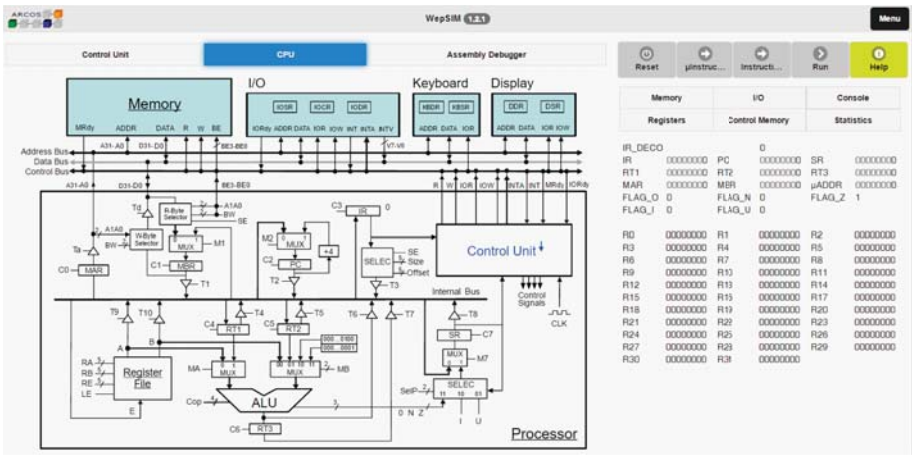

Figura 8. Pantalla principal de WepSIM.

necesitar recursos adicionales (apuntes, papel, etc.). Este acercamiento también implica de forma implícita que hay que ofrecer una ayuda en línea, que sea lo más autocontenida posible

El prototipo presentado en este artículo se implementó en HTM5 (HTML + JavaScript + CSS) de forma que es posible ejecutarlo en cualquier plataforma (*smartphones*, *tablet*, portátil, PC, etc.) en que pueda ejecutarse Mozilla Firefox o Google Chrome. El prototipo inicial se compone de 13 ficheros, con cerca de 5 000 líneas de código disponible con licencia LGPL 3.0 en GitHub: `https://github.com/wepsim/wepsim`. El prototipo depende de los siguiente frameworks/bibliotecas bien conocidos: JQuery, JQueryUI, JQuery Mobile, Knockout y BootStrap.

También recientemente hemos creado una aplicación móvil para Android y Windows Phone gracias al proyecto Apache Cordova. Para el próximo curso el simulador podrá ser ejecutado desde un navegador Web y desde la aplicación móvil correspondiente. La segunda opción es interesante para poder trabajar sin conexión a Internet.

4.1 La interfaz de usuario

La figura 8 muestra la pantalla principal del simulador. Hay tres áreas de trabajo bien diferenciadas: la de ejecución, la pantalla de trabajo con el microcódigo y la pantalla de trabajo con el ensamblador.

La pantalla de trabajo con el microcódigo (ver figura 9) permite indicar el microcódigo que tendrá la memoria de control y que controla el comportamiento del procesador. Se puede cargar un microcódigo de ejemplo (botón *Example*), se puede modificar el contenido con el editor, guardar con *Save* y cargar con *Load*.

Una vez disponible la especificación del microcódigo deseado, se microensambla con el botón *µassemble* y es posible ver el contenido de la memoria de control

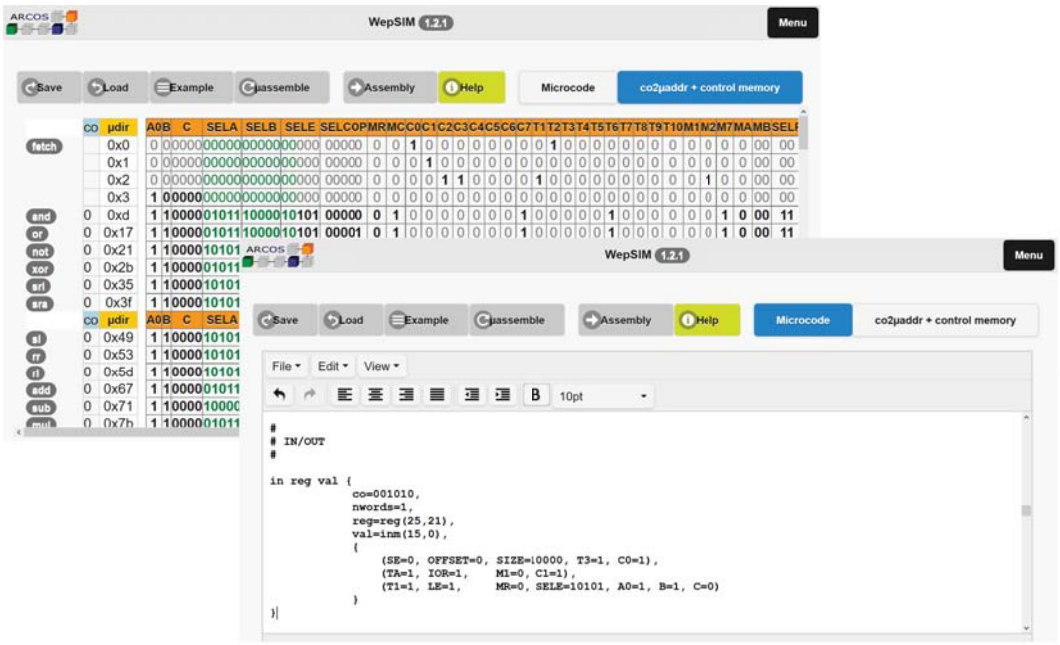

Figura 9. Pantallas de carga de microcódigo y visualización en binario.

y de la ROM de traducción (*co2µaddr + control memory*) en binario, resultado del microensamblado.

El microcódigo define, al menos, el tratamiento para el *fetch* de cada instrucción así como las señales a generar para poder ejecutar cada instrucción de ensamblador. Es posible definir instrucciones de distintos tipos de procesador (MIPS, ARM, Intel, Z80, etc.) lo que facilita posteriormente cargar un programa que use el conjunto de instrucciones definido.

La pantalla de trabajo con ensamblador (ver figura 10) permite cargar en memoria un programa en ensamblador, de acuerdo a la sintaxis definida en la definición del microcódigo. De igual forma que antes, se puede cargar un programa de ejemplo (botón *Example*), puede editarse, guardar con *Save* y cargar posteriormente con *Load*.

A partir de un determinado programa escrito en ensamblador, se puede ensamblar (botón *Assemble*) y es posible ver el contenido de la memoria principal en binario (*Main Memory*), resultado del proceso de ensamblado del código ensamblador dado, usando el ensamblador definido en el microcódigo.

Una vez cargado el microcódigo y el programa en ensamblador, es posible ejecutar dicho programa instrucción a instrucción de ensamblador o bien ciclo a ciclo de cada microprograma correspondiente a cada instrucción. Esta integración permite a los estudiantes ver cómo se integran ambos niveles (véase la Figura 11).

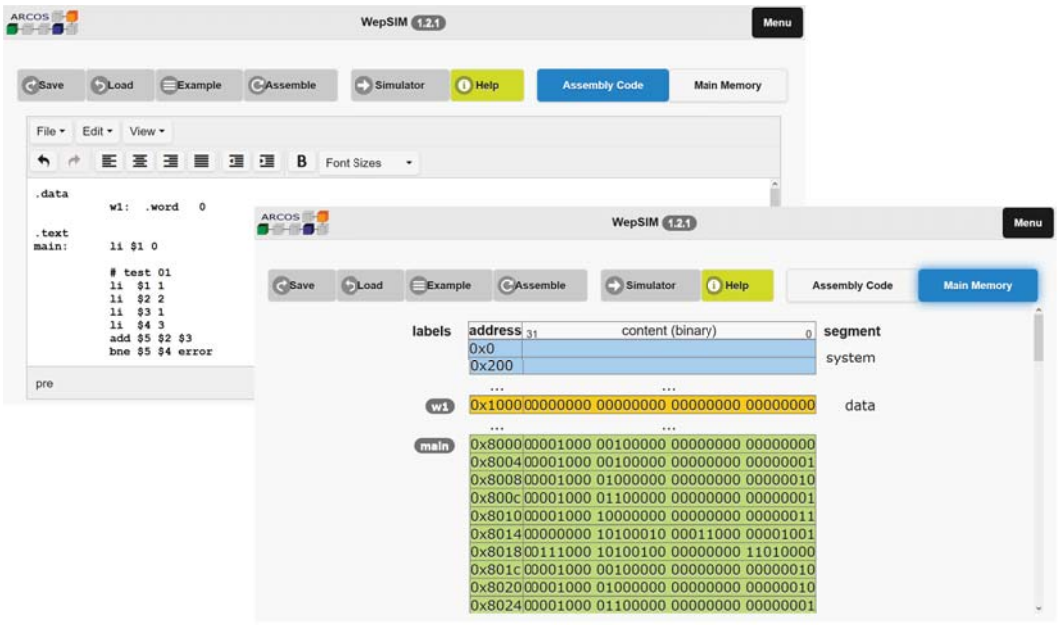

Figura 10. Pantallas de carga de código ensamblador y visualización en binario.

El diagrama del procesador elemental muestra con diferente color las señales activas (rojo), así como los caminos de datos activos (azul). Una señal está activa si aparece en la definición de la microinstrucción, y el camino de datos que dicha señal selecciona es el camino de datos activo asociado. De esta forma es más fácil e intuitivo seguir la pista de lo que está pasando en el procesador ciclo a ciclo. Destacar que el simulador es interactivo por defecto (es una opción configurable) de manera que si se quiere cambiar el valor de una señal sobre la marcha, solo se precisa hacer clic con el botón izquierdo del ratón en el nombre de la señal en el diagrama del procesador. Como se muestra en la figura 12, se puede seleccionar el valor (cero y uno) y se puede pedir una breve ayuda sobre la señal.

En el caso de ejecución instrucción a instrucción en ensamblador es posible usar la pestaña *Assembly Debugger* tal y como se muestra en la figura 8. Dicha pestaña permite ver el código en ensamblador, señalando la instrucción en curso y permitiendo el uso de puntos de ruptura (*Breakpoints*).

La implementación se ha realizado de forma que el simulador pueda ejecutarse sin mostrar el diagrama del procesador elemental. Ello permite que pueda utilizarse para una ejecución por lotes de pruebas consecutivas, útil por ejemplo para la corrección de distintos microcódigos.

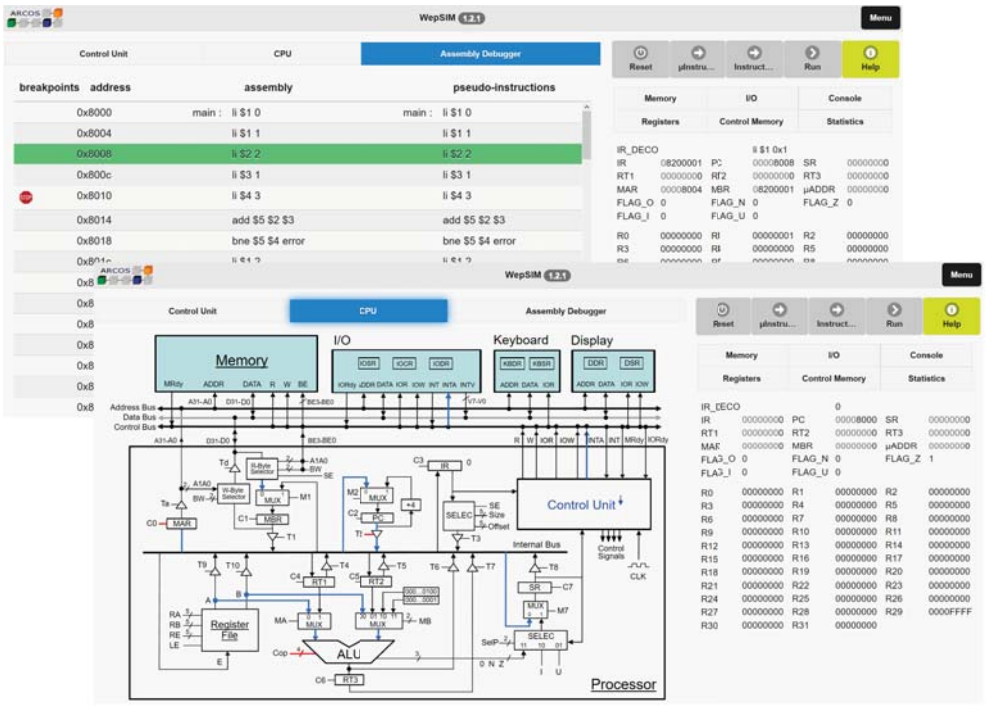

Figura 11. Pantallas que muestran la ejecución del simulador.

5 Estudio del uso del simulador WepSIM

Hemos analizado los registros del servidor Web desde donde se accede al simulador a través de Web para estudiar los dispositivos que acceden al simulador (ordenadores cliente). Durante la última semana de la entrega de la actividad docente el simulador WepSIM ha sido usado un 7 % con direcciones IP de la Universidad y un 93 % con direcciones IP de fuera de la Universidad. Estos resultados demuestran por tanto que hemos facilitado el uso de nuestra herramienta remotamente.

El segundo análisis realizado sobre los registros del servidor Web estudia el sistema operativo usado en los ordenadores cliente (tal y como se identifican ellos mismos). Queremos con ello estudiar si nuestro simulador WepSIM facilita a los estudiantes usar distintas plataformas de trabajo. Los resultados se muestran en la Figura 13, donde el eje y representa el número de accesos y el eje x la plataforma desde la cual provienen los accesos. El eje y usa una escala logarítmica. El grupo a la izquierda del eje x son los sistemas operativos de escritorio (Linux, Windows-NT y MacOS) mientras que el grupo de la derecha son los sistemas operativos para plataforma móviles (Android, iOS y Windows Phone). Estos resultados muestran la diversidad de los sistemas operativos usados, y por tanto que los estudiantes han podido usar distintas plataformas.

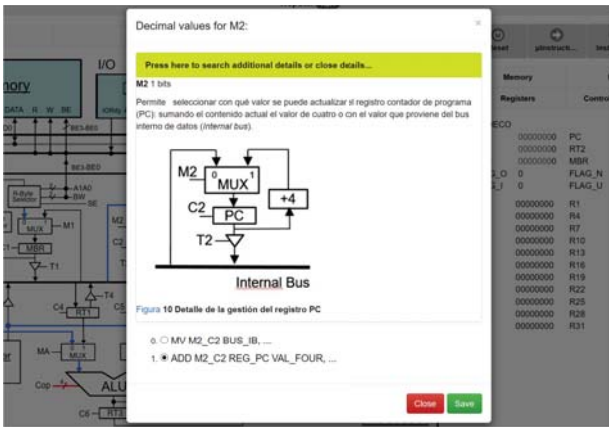

Figura 12. Detalle de la ventana emergente para cambiar interactivamente el valor de una señal. Incluye una breve ayuda de aprendizaje sobre la señal.

Para estudiar en qué horas se pide acceder al simulador, hemos analizado las horas de acceso en los registros del servidor Web. Los resultados se muestran en la Figura 14, donde el eje x representa la línea temporal del día sobre la que se señalan los accesos en cada instante del día producidos en la última semana de entrega de la práctica.

Salvo de la una de la noche hasta las ocho de la mañana donde los accesos son mínimos, el resto del día se realizaron accesos a WepSIM. Estos resultados nuevamente demuestran que hemos facilitado el uso de nuestra herramienta en las distintas horas del día, y desde distintas plataformas.

6 Estado del arte

Hay distintas herramientas muy útiles para labores docentes relacionadas con la asignatura de Estructura de Computadores, aunque como se comentó en la introducción, no hemos encontrado una herramienta como la presentada en este trabajo, especialmente en los siguientes aspectos:

- Permite a los estudiantes ofrecerles una visión interrelacionada de la microprogramación y la programación en ensamblador.
- Permite una flexibilidad en la plataforma usada (pensando en dispositivos móviles).
- Es lo más autocontenida posible (incorpora ayuda en línea y asociada en lo posible al contexto presentado).
- Presenta un modelo hardware que puede modificarse o ser ampliado si se desea.
- Permite definir un amplio conjunto de instrucciones máquina.
- Se puede usar como herramienta de microprogramación o de programación en ensamblador.

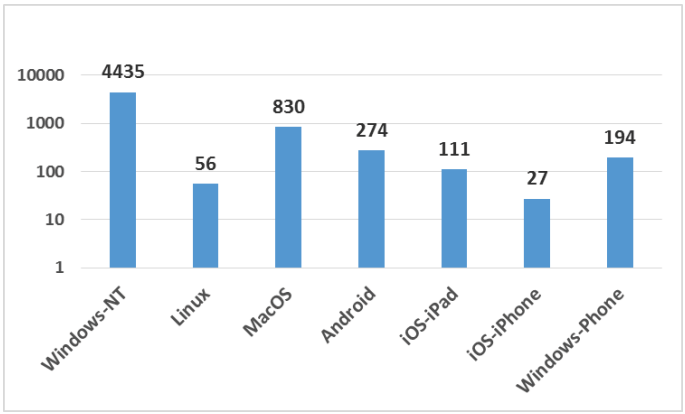

Figura 13. Sistema operativo de los ordenadores cliente que han usado WepSIM.

De entre los simuladores más conocidos para labores docentes, desde el punto de vista de programación en ensamblador, está SPIM y Mars. SPIM [7] es un simulador de un procesador MIPS de 32 bits que permite ejecutar (y depurar) programas en ensamblador para esta arquitectura. Este simulador creado por James R. Larus tiene versiones compiladas para Windows, Mac OS X y Unix/-Linux e incluso tiene una versión básica para Android [8]. Desafortunadamente no integra la visión para microprogramación, no está pensado su diseño para dispositivos móviles (la versión existente busca parecerse a la versión para ordenador) y se basa en el conjunto de instrucciones de MIPS32 (con la extensión de la instrucción syscall).

Mars [12] es también un simulador de un procesador MIPS de 32 bits similar a SPIM desarrollado en Java. Sus autores destacan la interfaz gráfica, la edición integrada y la reciente incorporación de una utilidad para acceder al hardware desde ensamblador. De igual forma que antes, no ofrece una visión integrada, no se encuentra versión alguna para dispositivos móviles y se centra en MIPS32.

Otro simulador conocido en entornos docentes es el em88110 descrito en [10], y como en casos anteriores, se centra en una arquitectura específica (MC88110) y no ofrece una ayuda en la movilidad del estudiante. Aunque integra el efecto de un procesador superescalar, no integra la microprogramación. Detrás de este simulador hay un trabajo muy interesante descrito en [5], donde se muestra la metodología detrás del simulador pc88110 para ser usado en prácticas. Ello incluye la descripción del entorno de prácticas, el sistema de entrega, el corrector automático, la detección de copias, etc.

Figura 14. Instantes en el tiempo del día en los que se ha pedido usar WepSIM.

Fuera del ámbito docente hay simuladores/emuladores como por ejemplo OVPsim [13] o GXemul [6] que permiten trabajar con distintas arquitecturas como por ejemplo ARM, MIPS, PowerPC, etc. En el caso de OVPsim, se ha usado para la investigación de plataformas de computación paralelas [11], co-diseño de hardware/software [9], etc. y aunque es posible su uso para labores docentes, por su nivel de detalle es posible que no sea la mejor herramienta con la que empezar a aprender.

Respecto a simuladores para labores docentes en microprogramación destacamos P8080E [3] desarrollado en el DATSI de la Facultad de Informática de la Universidad Politécnica de Madrid. A diferencia del P8080E nuestra propuesta (a) presenta una interfaz gráfica portable e interactiva, y (b) el ensamblador es definido en el microcódigo de forma que luego es posible ensamblarlo y generar el binario asociado. En el P8080E esta última labor se hace a mano, en la definición de las instrucciones no se incluye su formato y no está pensado para poder usarse para enseñar tanto microprogramación como ensamblador con la misma herramienta. Otros simuladores para microprogramación son el UT1000 [2] de 1989 y MicMac [4] de 1987. MicMac está pensado para usar su propio código máquina denominado Mac1 (no tanto diseñar nuevos), y UT1000 es descrito por sus autores como un simulador de una CPU de 16 bits con un secuenciador de AMD2910. Desafortunadamente UT1000 tras más de 25 años, es un proyecto a día de hoy no accesible.

7 Conclusiones y trabajos futuros

Este trabajo presenta un nuevo simulador que es intuitivo, portable y extensible. Permite definir diferentes juegos de instrucciones, y ejecutar y depurar código fuente que use el conjunto de instrucciones definido. Se basa en un modelo simple pero suficientemente capaz de mimetizar el funcionamiento básico de un procesador. También permite definir el comportamiento del procesador mediante microprogramación.

WepSIM permite a los estudiantes entender cómo funciona un procesador elemental de una forma fácil. Puede usarse desde un *smartphone*, una *tablet*, un portátil o un ordenador de escritorio con un navegador Web moderno, sin necesidad de instalación. Los estudiantes pueden interactuar con el simulador y aprender de esta forma cómo funciona un procesador elemental, incluyendo los mecanismos de interacción con el software de sistema. Integra tanto la programación en ensamblador como la microprogramación.

Hay varias líneas de trabajos futuros con las que ya estamos trabajando:

- Completar con una herramienta de pruebas basada en el motor de WepSIM que permita ejecutar un microcódigo con varios programas en ensamblador y un programa de ensamblador con varios microcódigos.
- Estamos trabajando en una aplicación móvil basada en el proyecto Apache Cordova para iOS, de forma que sea capaz de trabajar incluso sin conexión a Internet.

Hay más líneas de trabajos futuros que están siendo consideradas:

- Queremos estudiar el ensamblador de MIPS/ARM generado con gcc/clang de forma que pueda ser usado directamente en WepSIM.
- Introducir más elementos hardware, como una memoria caché.

Referencias

1. Carballeira, F.G., Pérez, J.C., Sánchez, J.D.G., Singh, D.E.: Problemas resueltos de estructura de computadores, segunda edición, vol. 1, pp. 1–307. Ediciones Paraninfo (2015)
2. Cornett, F.: The ut1000 microprogramming simulator: An educational tool. SIGARCH Comput. Archit. News 17(4), 111–118 (Jun 1989), http://doi.acm.org/10.1145/71317.71325
3. DATSI.FI.UPM.ES: P8080E (Apr 2016), http://www.datsi.fi.upm.es/docencia/Estructura/U_Control/
4. Donaldson, J.L.: Micmac: A microprogram simulator for courses in computer organization. SIGCSE Bull. 19(1), 428–431 (Feb 1987), http://doi.acm.org/10.1145/31726.31800
5. Dopico, A.G., de la Fuente, S.R., García, F.J.R.: Automatización de prácticas en entornos masificados. In: Actas de las IX Jornadas de Enseñanza universitaria de la Informática. pp. 119–126. Jenui 2003, Thomson-Paraninfo, Spain (2003), https://books.google.es/books?id=U07SAAAACAAJ
6. Gavare, A.: GXemul (Mar 2016), http://gxemul.sourceforge.net/
7. Larus, J.R.: SPIM (Feb 2016), http://spimsimulator.sourceforge.net/
8. Matak, J.: Assembly Emulator (Feb 2016), https://play.google.com/store/apps/details?id=gr.ntua.ece.assembly.emulator
9. Nita, I., Lazarescu, V., Constantinescu, R.: A new hw/sw co-design method for multiprocessor system on chip applications. In: Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and system Synthesis (ISSCS). pp. 1–4. IEEE Computer Society (2009)
10. Pérez Villadeamigo, J.M., de la Fuente, S.R., Cavanillas, R.M., García Clemente, M.I.: The em88110: Emulating a superscalar processor. SIGCSE Bull. 29(4), 45–50 (Dec 1997), http://doi.acm.org/10.1145/271125.271153
11. Pinto, C., Raghav, S., Marongiu, A., Ruggiero, M., Atienza, D., Benini, L.: Gpgpu-accelerated parallel and fast simulation of thousand-core platforms. In: The 11th International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 53–62. IEEE Computer Society (2011), http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2011.html#PintoRMRAB11
12. Sanderson, P., Vollmar, K.: MARS (Feb 2016), http://courses.missouristate.edu/kenvollmar/mars/
13. Software, I.: Open Virtual Platforms simulator (Mar 2016), http://www.ovpworld.org/