ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN DE EMPRESAS

Práctica 2

Introducción a la microprogramación

UNIVERSIDAD CARLOS III DE MADRID

Grupo de Arquitectura de Computadores, Comunicaciones y Sistemas

Curso 2017/2018

Versión 2.3

Contenido

Objetivos de la práctica

El objetivo de este trabajo es permitir practicar los conocimientos vistos en la asignatura, así como ofrecer un complemento de gran valor a los ejercicios disponibles en cada tema. De esta forma se incrementa las distintas habilidades de los estudiantes.

Los principales conocimientos que se practicarán son los de microprogramación, programación en ensamblador y representación de la información. Otros conocimientos que también se ejercitan son los de evaluación de alternativas de diseño y el trabajo en grupo.

Por ello, para el desarrollo de la práctica es necesario que el estudiante repase si fuera necesario:

- La representación de números enteros, cadenas de caracteres, etc.
- Los principales aspectos del lenguaje ensamblador de MIPS32.
- El formato de instrucciones y los tipos de direccionamiento.
- El funcionamiento de un procesador, incluyéndose las etapas de ejecución, microprogramación, etc.

El estudiante podrá usar el simulador WepSIM para poder ejercitar de una forma interactiva los conceptos y conocimientos anteriormente indicados, completando de esta forma los ejercicios de la asignatura, y añadiendo la posibilidad de mejorar la visión global de cómo encaja las diferentes partes de la materia introducidas en la asignatura.

Ámbito de trabajo

Imagine la siguiente situación. En la empresa para la que estamos trabajando se usa el hardware que es simulado por la herramienta WepSIM. Un importante cliente de la empresa está usando dicho hardware con software que ha escrito para él.

Debido al éxito del tratamiento de grandes volúmenes de datos en redes sociales, nuestro cliente nos pide mejorar la capacidad del sistema para el tratamiento de cadenas de caracteres (*strings*) para poder facilitar el diseño y desarrollo de los programas que procesen mensajes intercambiado en dichas redes sociales.
El soporte que busca el cliente es disponer de dos funcionalidades:

- Conocer la longitud de una cadena.
- Eliminar los espacios en blanco iniciales de una cadena.

Pero como puede comprobar, el hardware simulado por WepSIM no dispone de una unidad específica para operaciones con cadenas de caracteres. Por ello la empresa baraja dos posibles formas de implementar las anteriores dos funcionalidades:

1) Mediante subrutinas escritas en el ensamblador que ejecuta WepSIM.
2) Mediante nuevas instrucciones máquina microprogramadas en WepSIM (como instrucciones especiales que son parte de un ASE -*Application-Specific Extensions*-).

La empresa nos asigna el trabajo de implementar y estudiar estas dos posibilidades y realizar un informe comparando el rendimiento en ambos casos (usando el simulador WepSIM).
A la empresa la versión microprogramada le supone pagar un 30% más de dinero por lo que quiere conocer si compensa. Si se reduce el número de ciclos de reloj necesarios para ejecutar en, al menos, un 20% entonces compensará el mayor coste. El informe que realicemos deberá comparar ambas versiones y las conclusiones servirán a los directivos para tomar la decisión.

Las cadenas se almacenan en memoria en direcciones crecientes y se terminan en el carácter '\0' (o valor 0x0 en decimal).

Ejercicio 1

Para comparar la posibilidad de ofrecer tratamiento de cadenas de caracteres como subrutinas de ensamblador o bien como instrucciones a añadir a las existentes en el procesador WepSIM, la empresa nos facilita un juego inicial de instrucciones con las que trabajar, que son un subconjunto de las instrucciones en ensamblador MIPS32.

El primer ejercicio consiste en revisar dicho juego (o conjunto) de instrucciones para asegurarse que cumplen las especificaciones asociadas que la empresa nos facilita (dado que el estudio pedido por la empresa se ha de hacer con dichas instrucciones). Si se detectara algún problema entonces podrán ser modificadas **siempre y cuando el resultado sea lo mínimo necesario a modificar (que no añadir) para conseguir la funcionalidad descrita dentro del conjunto de instrucciones de MIPS32**. Puede que no sea preciso modificar las instrucciones (no se debe añadir nuevas instrucciones), pero es conveniente comprobar las mismas.

Las instrucciones dadas inicialmente como base de trabajo son:

Instrucción MIPS32	Descripción
reti	Recupera de la pila SR y PC, y continua la ejecución
syscall	Produce evento tipo #2
and R1 R2 R3	R1 = R2 & R3
or R1 R2 R3	R1 = R2 | R3
xor R1 R2 R3	R1 = R2 ^ R3
not R	R = ^R
add R1 R2 R3	R1 = R2 + R3
sub R1 R2 R3	R1 = R2 - R3
mul R1 R2 R3	R1 = R2 * R3
div R1 R2 R3	R1 = R2 / R3
rem R1 R2 R3	R1 = R2 % R3
srl R1 R2 inm	R1 = R2 >> inm
move R1 R2	R1 = R2
li R inm	R = inm (con extensión de signo)
liu R inm	R = (R << 16) & 0xFFFF0000
la R addr	R = addr
lw R addr	R = MEM[addr] (palabra)
sw R addr	MEM[addr] = R (palabra)
lb R addr	R = MEM[addr] (byte)
sb R addr	MEM[addr] = R (byte)
lb R1 (R2)	R1 = MEM[R2] (byte, ext. sig.)
lbu R1 (R2)	R1 = MEM[R2] (byte)
in R addr	R = IO[addr]
out R addr	IO[addr] = R
b addr	PC = addr
beq R1 R2 addr	si R1==R2 -> PC=addr
bne R1 R2 addr	si R1!=R2 -> PC=addr
bge R1 R2 addr	si R1>=R2 -> PC=addr
blt R1 R2 addr	si R1<R2 -> PC=addr
bgt R1 R2 addr	si R1>R2 -> PC=addr
ble R1 R2 addr	si R1<=R2 -> PC=addr
j addr	PC = addr
jal addr	RA = PC; PC = addr
jr R	PC = R

Para comprobar una instrucción se ha de usar, al menos, un ejemplo de uso de ella. A continuación, se ejecutará dicho ejemplo obteniendo las diferencias entre los estados antes de ejecutar la instrucción y después de ejecutar la instrucción con ese ejemplo de uso. Estas diferencias se registrarán en una tabla, y finalmente se analizará que el comportamiento de la instrucción se ajusta a la descripción dada.

Es importante recordar que el comportamiento de la instrucción queda descrito por su especificación, pero una forma de ver que es así es comprobar que en la diferencia de estados la instrucción modifica los elementos (registros, posiciones de memoria, etc.) que se han de modificar y que no se modifican los elementos que están fuera de los indicados a modificar.

El "Apéndice 1. Ejemplo de validación de la instrucción li" describe a título de ejemplo el proceso detrás de una posible validación de dicha instrucción. **Ha de completar** de la siguiente tabla **las** instrucciones **que no han sido validadas** (con un No en columna 'Validada'):

Instrucción	Ejemplo	Estado modificado	Validada
`add R1 R2 R3`			**No**
`sub R1 R2 R3`			**No**
`mul R1 R2 R3`			**No**

`li R inm`	`li $t0 100`	`Type Id. Clipoboard Selected`
		`cpu PC = 0x8000 0x8004`
		`cpu R8 = 0 0x64`
			Si

`lb R1 (R2)`			**No**
`lbu R1 (R2)`			**No**
`sb R1 (R2)`			**No**
`b addr`			**No**
`beq R1 R2 addr`			**No**

Los resultados de este ejercicio están tanto en la parte de la memoria correspondiente como en el fichero con la funcionalidad pedida asociado.

El fichero con la funcionalidad pedida será:

- **cadenas_1_mc.txt**:
  Contiene los microprogramas para las instrucciones asociadas iniciales que se hayan visto obligados a modificar según lo indicado anteriormente. **Si no ha modificado microprogramas alguno entonces ha de entregar un archivo vacío.**

La sección en la memoria para el ejercicio 1 ha de contener:

- Si ha tenido que modificar algún microprograma de las instrucciones originalmente dadas entonces una tabla con el diseño en lenguaje RT correspondiente.
- Tabla indicada a completar, que incluya ejemplo usado, estado modificado y validación correspondiente.

Ejercicio 2

El objetivo de este ejercicio es estudiar la implementación de las dos funcionalidades descritas al inicio mediante subrutinas escritas en ensamblador.

> Se utilizarán las instrucciones microprogramadas dadas para empezar el trabajo, con las modificaciones que haya realizado en el ejercicio 1. Recuerde una vez más que podrán modificarse los microprogramas **siempre y cuando el resultado sea lo mínimo necesario a modificar (que no añadir) para conseguir la funcionalidad descrita dentro del conjunto de instrucciones de MIPS32**.

Las rutinas que deberá crear tienen las siguientes especificaciones:

- `strlen_1`:
  La cual recibe un parámetro:
    - La dirección de comienzo de la cadena de caracteres STR1.

  El objetivo de esta subrutina es calcular el número de caracteres de la cadena de caracteres STR1 (longitud de la misma) y devolver dicha longitud.
  Como ejemplos de dicha subrutina:
  - strlen_1(“HOLA”) -> 4
    En este caso la longitud de “HOLA” es de 4 caracteres (aunque se precise un carácter adicional para almacenar en memoria el fin de cadena, este es transparente al programador).

- `skipspaces_1`:
  La cual recibe un parámetro:
    - La dirección de comienzo de la cadena de caracteres STR1.

  El objetivo de esta subrutina es saltar los espacios en blanco al principio de la cadena STR1. La subrutina **devuelve la dirección del primer carácter** en STR1 tras saltar todos los espacios en blanco iniciales en STR1. No hay que hacer nada con la cadena inicial. Si no hay espacios en blanco, devuelve la dirección de comienzo de STR1.
  Como ejemplos de dicha subrutina:
  - skipspaces_1(“ HOLA”) -> “HOLA”
    la función devolvería, para la cadena pasada como entrada, la dirección donde se almacena el carácter ‘H’, una vez saltados todos los espacios en blanco.

Tenga presente que deberá de diseñar y realizar las pruebas que estime necesario para asegurar el cumplimiento de los requisitos pedidos para las nuevas subrutinas.
También deberá realizar una prueba con la que probar este ejercicio y el siguiente, y medir el número de ciclos de reloj totales en la ejecución de la prueba. Dicha medida servirá para comparar para el mismo objetivo qué opción (microprogramar o programar en ensamblador) sería recomendable.

Los resultados de este ejercicio están tanto en la parte de la memoria correspondiente como en el fichero con la funcionalidad pedida asociado.

El fichero con la funcionalidad pedida será:

- **cadenas_1_mp.txt**:
  Contiene la implementación de las subrutinas pedidas implementadas con el juego de instrucciones anteriormente entregado. **Solo ha de incluir las subrutinas indicadas (no ha de incluir la subrutina main, sección .data, etc.)**

La sección en la memoria para el ejercicio 2 ha de contener:

- Tabla con (al menos) dos columnas: una con las comprobaciones usadas para asegurar que se cumplen los requisitos, y otra indicando el resultado de la prueba.
- Los ciclos de reloj totales generados por la prueba de comparativa de las dos posibles soluciones (prueba a usar para los ejercicios 2 y 3).

Ejercicio 3

Para la segunda posibilidad se ha ofrecer las dos funcionalidades mediante dos nuevas instrucciones máquinas a diseñar e incorporar al simulador WepSIM.

> Se partirá (y podrá utilizar) las instrucciones microprogramadas dadas para empezar el trabajo, con las modificaciones que haya realizado en el ejercicio 1.

Las instrucciones a microprogramar tienen las siguientes:

- `strlen_2 $r1 $r2`:
  La instrucción guardará en el registro indicado por `$r1` la longitud de la cadena cuya dirección de comienzo está en el registro indicado por `$r2`. Es decir, el número de caracteres de la cadena que es apuntada por la dirección contenida en el registro `$r2`.
  Esta instrucción una vez ejecutada solo puede modificar `$r1`.
  La instrucción se codificará con el formato: **`011011 R1 R2 00000 00000000000`**

- `skipasciicode_2 $r1 $r2 n`:
  Esta instrucción usará dos registros: el resultado se guardará en `$r1` y la dirección de comienzo de una cadena de caracteres STR1 estará en el registro indicado por `$r2`. La instrucción guardará como resultado la dirección del primer carácter que no sea el código ASCII pasado como valor inmediato 'n' en STR1 comenzando desde el principio. Esta instrucción una vez ejecutada solo puede modificar `$r1`. No obstante, si lo considera oportuno puede modificar los registros `$k0` y `$k1` (definidos en el microcódigo inicial).
  Por ejemplo, la instrucción `skipasciicode_2 $r1 $r2 'a'` saltar todas las primeras ocurrencias del carácter 'a '.
  La instrucción se codificará con el formato: **`011010 R1 R2 00000 000 nnnnnnnn`**, donde el valor inmediato n se codifica en los ocho bits inferiores de la instrucción.

Tenga presente que deberá de diseñar y realizar las pruebas que estime necesario para asegurar el cumplimiento de los requisitos pedidos para las nuevas instrucciones.
También deberá realizar una prueba con la que probar este ejercicio y el siguiente, y medir el número de ciclos de reloj totales en la ejecución de la prueba. Dicha medida servirá para comparar para el mismo objetivo qué opción (microprogramar o programar en ensamblador) sería recomendable.

Para la funcionalidad correspondiente a saltar blancos se utilizará la instrucción `skipasciicode` con el código ASCII correspondiente al espacio en blanco.

Los resultados de este ejercicio están tanto en la memoria como en el fichero con la funcionalidad pedida.

El fichero con la funcionalidad pedida será:

- **cadenas_2_mc.txt**:
  Los microprogramas (únicamente) de las instrucciones descritas.

La sección en la memoria para el ejercicio 3 ha de contener:

- El diseño en lenguaje RT de los microprogramas asociados a las instrucciones descritas en una tabla (con, al menos dos columnas: instrucción y microprograma asociado).
- Tabla con tres columnas: ejemplo usado para la comprobación de que se cumple uno (o varios) requisito(s), diferencia de estados (antes de ejecutar y después de ejecutar) al ejecutar dicha instrucción, y validación (Si o No) de los requisitos.
- Los ciclos de reloj totales generados por la prueba de comparativa de las dos posibles soluciones (prueba a usar para los ejercicios 2 y 3).

Para comprobar que las instrucciones modifican los elementos (registros, posiciones de memoria, etc.) que se han de modificar y que no se modifican los elementos que están fuera de los indicados a modificar, **tendrá que mostrar la diferencia de los estados antes de ejecutar la instrucción y después de ejecutar la instrucción** (y comprobar que dicha diferencia cumple lo especificado). El "Apéndice 1. Ejemplo de validación de la instrucción li" describe a título de ejemplo una posible validación de dicha instrucción.

Ejercicio 4

Los resultados de este ejercicio estarán en la memoria. No hay que entregar código asociado a este apartado.

La sección en la memoria para el ejercicio 4 ha de contener:

- Una tabla con dos filas (una por funcionalidad: strlen y skipspace) y dos columnas (una por alternativa: ensamblador o microprogramado) donde se muestre el tiempo se precisa para ejecutar medidos como en ciclos de reloj la prueba que se diseñe para comparar las alternativas.
- Un análisis de los anteriores resultados donde ha de comparar dichos valores e indicar cuál es mejor justificando su respuesta
- Una recomendación brevemente justificada para la empresa.

Se valorará positivamente que se haya realizado una implementación válida que minimice el número de ciclos de reloj en ambos casos, justificando brevemente en la memoria las decisiones de diseño que ha tomado para lograrlo.

Aspectos importantes a tener en cuenta

Ha de revisar cuidadosamente que se cumple todos los requisitos indicados en el enunciado. Se recomienda a partir del enunciado generar una lista de comprobación (*checklist*) que ayude al grupo de práctica a revisar todo.

Es importante recordar que el nombre de las subrutinas, ficheros, etc. han de ser los indicados en este enunciado. No respetar dichos nombres supondrá un cero de nota en el apartado correspondiente por lo que se recomienda añadir a la lista de comprobaciones las comprobaciones de los nombres.

Normas generales

1) La entrega de la práctica se realizará a través de los entregadores habilitados. No se permite la entrega a través de correo electrónico.
2) La entrega se realizará en el plazo dado por los entregadores. Es posible que para un entregador de Aula Global el fin del plazo para una entrega a las 24:00 termine 10 minutos antes. Revise el soporte de Aula Global para confirmar el plazo.
3) Se prestará especial atención a detectar funcionalidades copiadas entre prácticas.
   En caso de detectar copia en dos prácticas (o en la memoria), todos los grupos implicados (copiados y copiadores) obtendrán una calificación de 0 (cero), pudiendo abrirse un expediente dependiendo de la gravedad.
   La copia de porciones de Internet o de prácticas de otros cursos es también considerado copia, y todos los grupos implicados podrán ser expedientados.

Códigos de la práctica

1) Se valorará positivamente que se haya realizado una implementación válida que minimice el número de ciclos de reloj.
2) Un programa no comentado adecuadamente obtendrá una calificación de 0.
   Los comentarios han de buscar describir cada paso que se pretenden implementar antes del bloque de código que lo implementa.
3) Hay que tener en cuenta que un programa que compile correctamente no es garantía de que funcione correctamente. Por ello tendrá que realizar aquellas pruebas que garanticen el correcto funcionamiento de la práctica.
4) En el código pedido para entregar **no** han de imprimirse mensajes por pantalla, ni contener ningún otro código adicional usado para diagnóstico.
5) Todos los ejercicios han de funcionar en todo momento con la versión del simulador WepSIM dada en la URL: **https://wepsim.github.io/wepsim/**

Memoria de la práctica

1) **La longitud de la memoria no deberá superar las 15 páginas** (portada e índice incluidos).
2) **Se entregará en formato PDF con texto seleccionable (ha de permitir que turnitin pueda realizar el control de copia en la memoria)**.
3) La memoria (un único documento) tendrá que contener al menos los siguientes apartados:
    - Portada donde figuren los autores (incluyendo nombre completo, NIA y grupo de cada autor), titulación, asignatura y práctica.
    - Índice de contenidos.
    - Sección para el ejercicio 1 donde se incluya lo pedido para dicho ejercicio.
    - Sección para el ejercicio 2 donde se incluya lo pedido para dicho ejercicio.
    - Sección para el ejercicio 3 donde se incluya lo pedido para dicho ejercicio.
    - Sección para el ejercicio 4 donde se incluya lo pedido para dicho ejercicio.
    - Conclusiones y problemas encontrados. Incluya un resumen del número de horas destinada a la práctica.

NOTA: NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA.

Aprobar la memoria es tan imprescindible para aprobar la práctica, como el correcto funcionamiento de la misma. Si al evaluarse la memoria de su práctica, se considera que no alcanza el mínimo admisible, su práctica estará suspensa.

Procedimiento de entrega de la práctica

La entrega de la práctica 2 se realizará de forma electrónica a través de Aula Global, para lo que se habilitará el entregador asociado a dicha práctica.

La fecha límite de entrega para ambos es el día **3 de diciembre de 2017 a las 23:50 horas.**

Es posible entregar tantas veces como quiera dentro del plazo dado, la única versión registrada de su práctica es la última entregada. La valoración de la práctica es la valoración del contenido de esta última entrega. Por favor revise siempre lo que entregue (usted y resto del grupo).

A entregar: Se deberá entregar un único archivo comprimido en formato **zip** con el nombre AAAAAAAAA_BBBBBBBBB.zip donde A…A y B…B son los NIA de los integrantes del grupo.

El archivo **zip** debe contener solo los siguientes archivos:

- **cadenas_1_mc.txt**
- **cadenas_1_mp.txt**
- **cadenas_2_mc.txt**
- **memoria.pdf**
- **autores.txt**

Donde el fichero "autores.txt" contendrá una línea por autor con su NIA correspondiente, y todos los archivos serán de tipo ASCII salvo la memoria que será en formato PDF.

Evaluación de la práctica

La evaluación de la práctica se va a dividir en dos partes:

- **Código (8 puntos)**
- **Memoria (2 puntos)**

La puntuación de cada ejercicio:

- Ejercicio 1 (1.0 *puntos*)
- Ejercicio 2 (2.0 *puntos*)
- Ejercicio 3 (6.0 *puntos*)
- Ejercicio 4 (1.0 *puntos*)

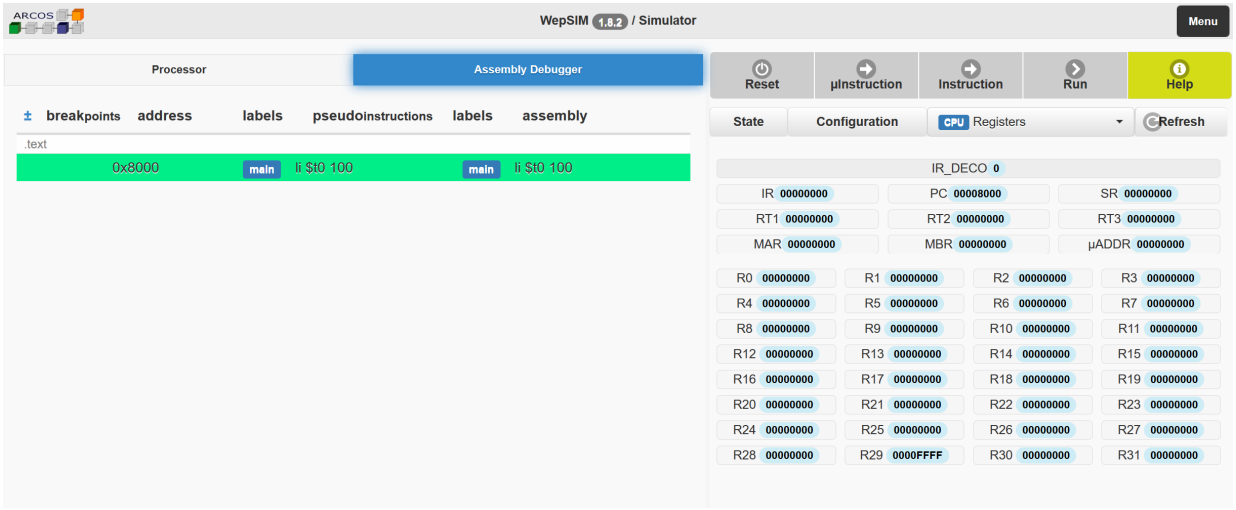
MUY IMPORTANTE:

1. **Aunque la puntuación se pueda repartir en apartados, la corrección de la práctica se realizará a nivel global, es decir, esto incluye:**
    a. **Si un ejercicio no se entrega la calificación de toda la práctica será de cero.**
    b. **Si se detecta un error de concepto grave en la práctica (en cualquier apartado de cualquier ejercicio), la valoración global de toda la práctica será de cero puntos (0 puntos).**
2. **Se realizarán comprobaciones para evitar copias totales o parciales en trabajo entregado, es decir, esto incluye:**
    a. **En caso de encontrar implementaciones comunes en dos prácticas (o contenidos similares en la memoria), ambas obtendrán una calificación de 0.**
    b. **En caso de encontrarse fragmentos de código obtenidos directamente de Internet no referenciados, la práctica tendrá una calificación de cero.**
3. **Ha de respetar el formato y nombres pedidos, esto incluye:**
    a. **Es fundamental respetar el nombre y formato de los ficheros puesto que en caso contrario supondrá una nota de 0 (cero). Ello incluye no respetar minúsculas, entregar un .rar en lugar de .zip, un .docx en lugar de pdf (no vale renombrar), etc.**
    b. **El texto del archivo con la memoria (memoria.pdf) ha de ser seleccionable y copiable. Es decir, si se genera un archivo PDF basado en una imagen por página (para evitar su tratamiento por las herramientas de control de copia) entonces la calificación será de un cero para toda la práctica.**

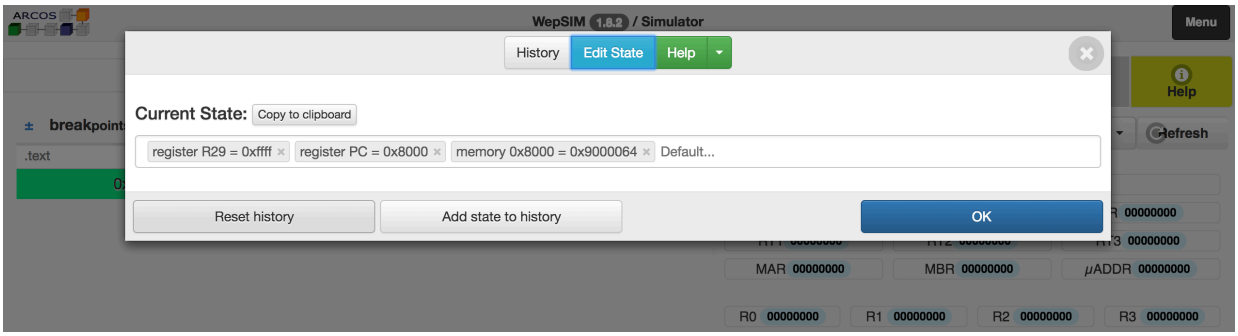
Apéndice 1. Ejemplo de validación de la instrucción li

En este apartado se muestra un ejemplo de cómo se puede realizar la validación de la instrucción li mediante un test simple usando el programa "`li $t0 100`" y comprobando la diferencia entre el estado del procesador antes de ejecutar la instrucción y el estado una vez ejecutada.

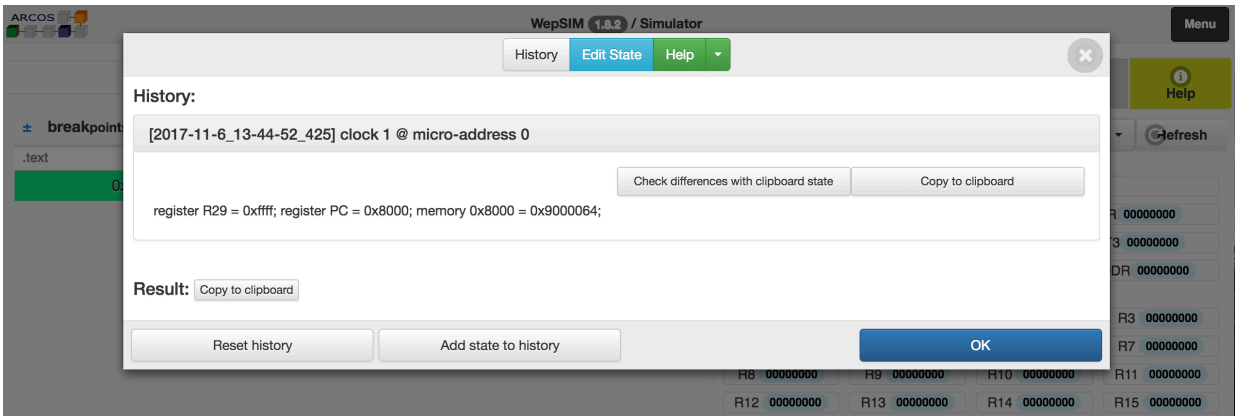
Una vez cargado el ejemplo se tiene la pantalla mostrada a continuación:



A continuación se pulsa el botón "State" y "Edit State" para mostrar el estado; Por defecto el valor de los elementos es cero y solo se muestran aquellos elementos que tienen un valor distinto de cero en el cuadro de diálogo:

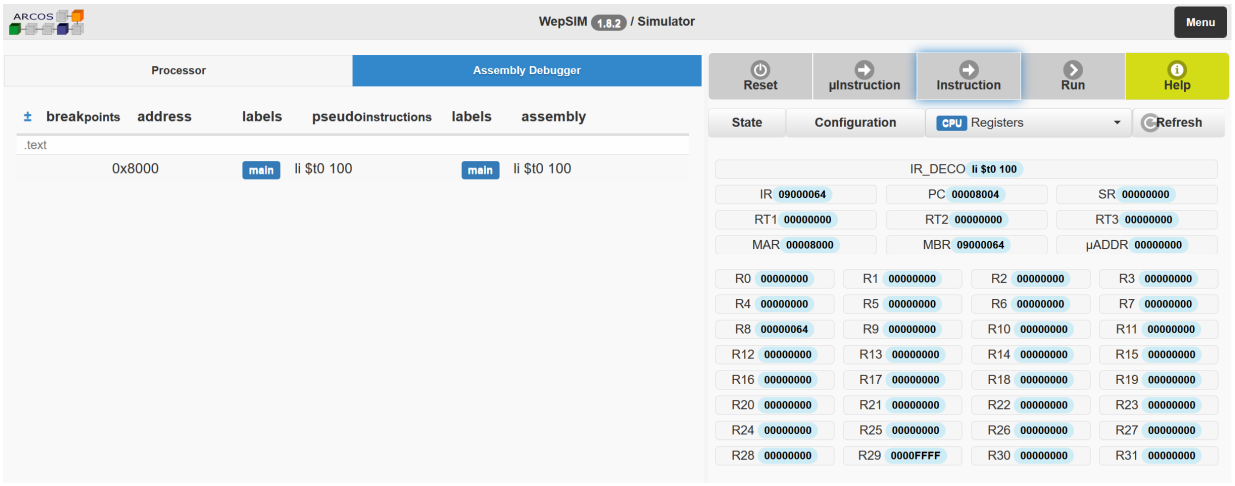


Si pulsamos "Add state to history" se añadirá este estado actual al historial de estados. Pulsando "History" se puede consultar dicho historial:



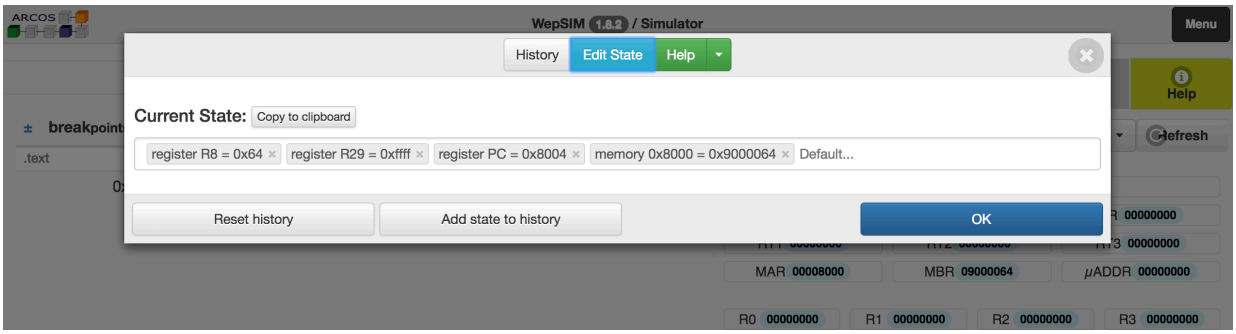
Volvemos a pulsar “OK” para volver.

De vuelta en el programa de ejemplo (`li $t0 100`) ejecutamos la instrucción (botón “Instruction”) y se mostrará la pantalla:

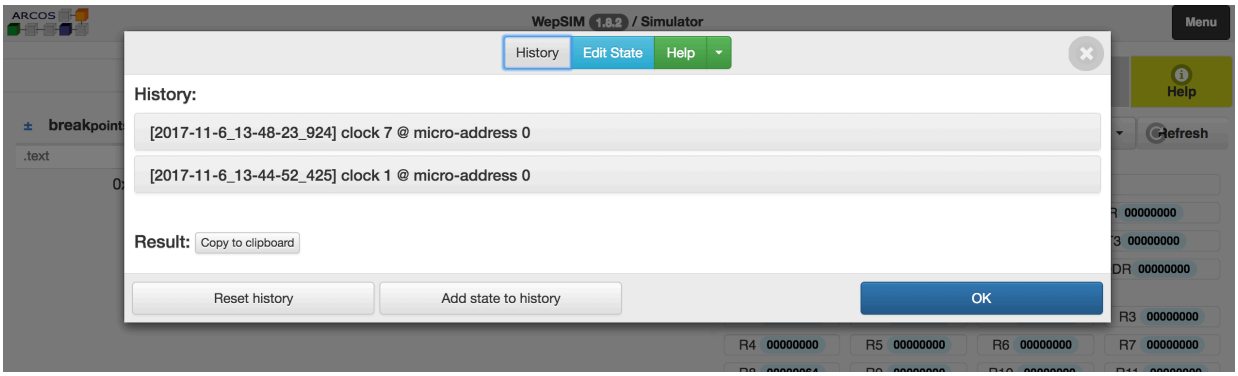


Se puede ver que los valores de los registros mostrados han cambiado. Han cambiado tanto algún registro visible al programador (R8) como algún registro no visible (IR, PC, etc.)

Se vuelve a consultar el estado usando el botón “State” y “Edit State” y aparece la siguiente pantalla:



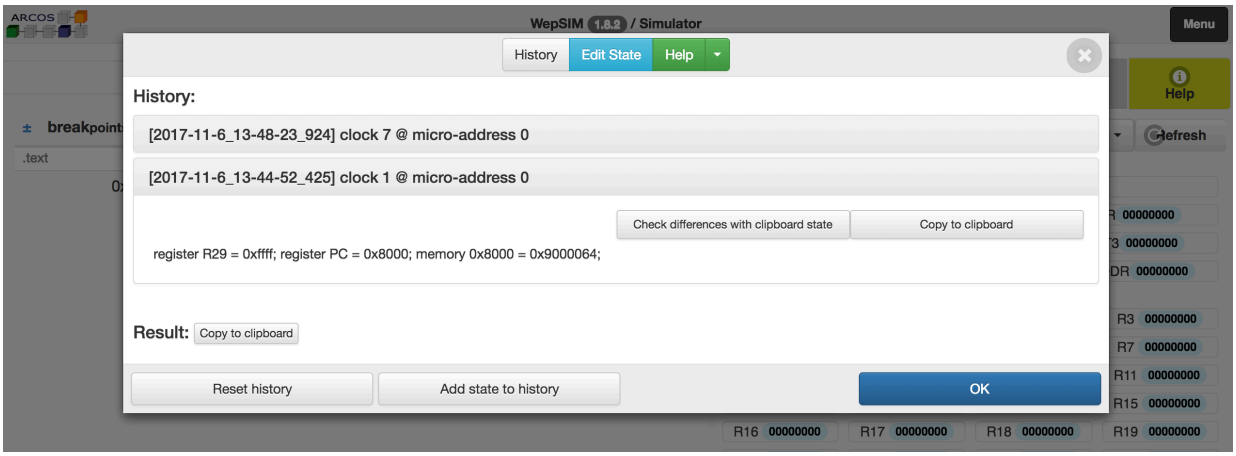
Se volverá a añadir al historial usando el botón “Add state to history”.
Usando el botón “History” se puede consultar dicho historial, aparece la pantalla:



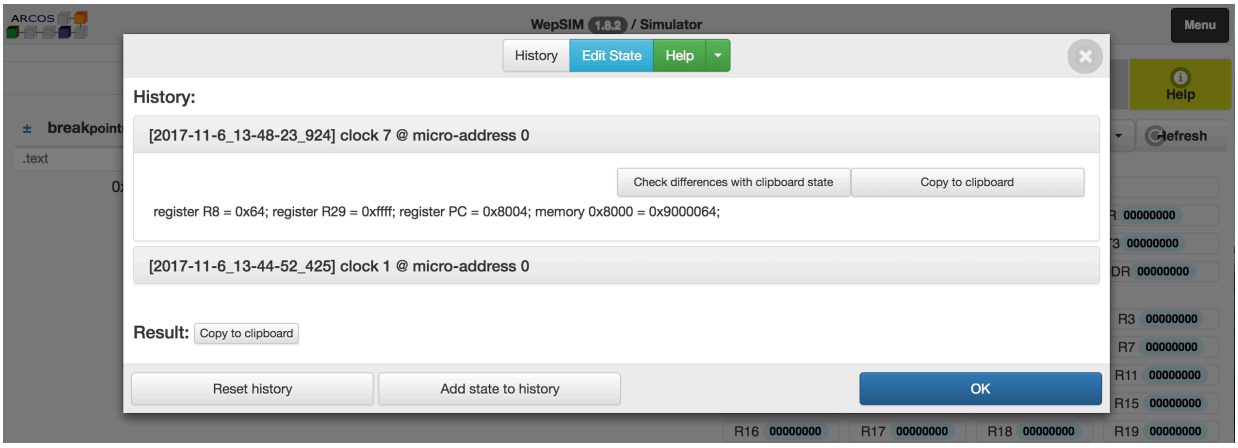
Donde se muestra los dos estados guardados, entre corchetes se indica el instante del tiempo en que se solicitó guardar el estado en el historial (según el reloj del dispositivo que ejecuta WepSIM). También se muestra a continuación el instante de tiempo respecto al programa que ejecuta indicando el ciclo de reloj en que ese estado fue guardado. Por último, se indica la

microdirección de la memoria de control con las señales generadas en el ciclo de reloj asociado. Destacar que el último estado guardado está arriba y el más antiguo abajo.

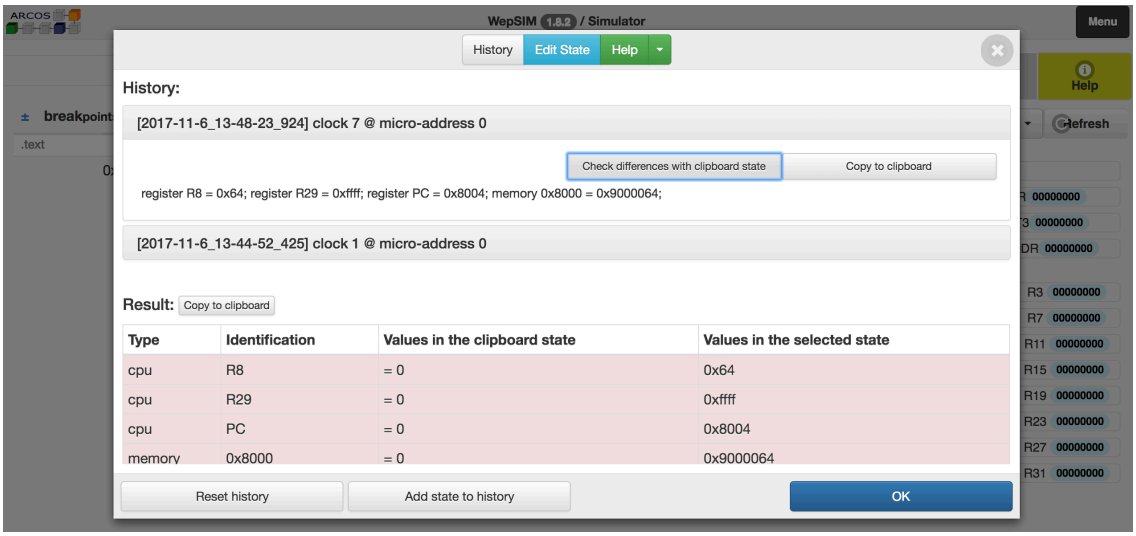
Es importante poder comparar las diferencias entre dos estados. Para ello se selecciona uno de los estados y se mostrará la siguiente pantalla:



Se pulsa el botón “Copy to clipboard” para copiar el estado al portapapeles.
A continua se selecciona el otro estado con el que se quiere comparar, como se muestra en la siguiente imagen:



Y se pulsa el botón “Check differences with clipboard state” para que se compare el estado mostrado con el se encuentra en el portapapeles. A continuación, WepSIM mostrará las diferencias, como se indica en la siguiente imagen:



Para cada elemento diferente se indica el tipo (Columna "Type"), el elemento (Columna "Id."), el valor que se encuentra en el copiado en el portapapeles (Columna "Values in the clipboard state") y el valor del estado actual (Columna "Values in the selected state").

Es importante revisar que las especificaciones (dadas por la fila de la tabla de instrucciones) se cumplen, revisando las diferencias entre los dos estados: antes de ejecutar la instrucción y después de ejecutar la instrucción. Puede ser necesario varios ejemplos para poder realizar esta verificación.

Por ejemplo, para la instrucción "li" anteriormente analizada, en sus especificaciones se indica:

Instrucción en ensamblador	**Formato de la Instrucción**	**Descripción**	***Flags* de la ALU**
…	…	…	…
`li  R val`	`000010 R  00000 val(16 bits)`	`R = val`	NO
`lui R val`	`000011 R  00000 val(16 bits)`	`R = (val << 16) | (000..0)`	NO
…	…	…	…

Y por tanto, solo el registro visible "R" indicado en la instrucción puede ser modificado. Como se indica en la tabla, el registro de estado SR no ha de ser modificado (*Flags* de la ALU: no).

Como resultado de la prueba de li hay que añadir una fila a la tabla de pruebas realizadas:

Instrucción probada	**Código de prueba**	**Caso de prueba**	**Diferencias de estados**
li	li $t0 100	Se modifica $t10 con 100 y no se modifica SR.	Type Id. Clipboard Selected cpu PC = 0x8000 0x8004 cpu R8 = 0 0x64
	…		…

Todas las instrucciones son probadas con, al menos, una prueba de forma que pueda comprobarse que se cumplen sus especificaciones: hacen lo que tienen que hacer y no hacen lo que no tienen que hacer. Por ello, no solo hay que probar casos correctos, también casos incorrectos de forma que el comportamiento para estos últimos también sea el esperado/deseado.