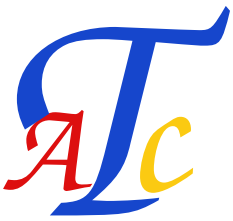

Universidad de Córdoba

Arquitectura y Tecnología de Computadores

Práctica 4

MIPSim

Arquitecturas Avanzadas de Procesadores

D. Miguel Ángel Montijano Vizcaíno (el1movim@uco.es)
D. Héctor Martínez Pérez (el2mapeh@uco.es)

Curso 2025/2026

1 Introducción

Esta práctica se centrará en el aprendizaje del diseño y uso del entorno de simulación de arquitecturas segmentadas MIPS, llamado MIPSim. Pese a ser una aplicación desarrollada hace muchos años por John L. Hennessy y David A. Patterson de la Universidad de Viena, permite llevar a cabo simulaciones de código máquina en un procesador MIPS segmentado.

Los objetivos principales de la práctica son:

- Introducción al simulador MIPSim.
- Conocer su estructura, componentes, etapas, etc.
- Crear, modificar, almacenar y ejecutar simulaciones.

2 Instalación del software

MIPSim, como ya se ha comentado, se trata de un software antiguo, por lo tanto, hay que tener en cuenta sus limitaciones. En el apartado de esta práctica 4, se puede encontrar un recurso con este simulador.

En primer lugar, descargue el recurso llamado *MIPSim32*, a continuación, descomprima su contenido y proceda a ejecutar *'MIPSIM32.EXE'*. En S.O. Windows, muy probablemente lance un mensaje de seguridad, así que será necesario permitir su ejecución. En S.O. Linux, pese a ser un ejecutable *.exe*, puede lanzarse mediante aplicaciones como *Wine*.

La ventana de la aplicación tiene un tamaño fijo; dependiendo de la resolución y el tipo de monitor, es posible que se corte/pierda una parte de esta ventana. No obstante, la parte visual debería ser suficiente para mostrar toda la información relevante del pipeline del procesador.

3 Entorno MIPSim

3.1 Descripción

MIPSim dispone de un subconjunto de instrucciones que siguen el estándar del ISA MIPS. Incorpora 32 registros de propósito general de 32 bits. Además, su pipeline está segmentado en 5 etapas; esto implica que en un solo ciclo de reloj pueden concurrir 5 instrucciones diferentes. Estas cinco etapas consisten (por orden):

- **FETCH:** Etapa de búsqueda de instrucción. *pc* apunta a la instrucción actual.
- **DECODE:** Decodificación de la instrucción y lectura de registros. Se lleva a cabo la lectura de los operandos desde el banco de registros.
- **EXECUTE:** Ejecución de la instrucción y cálculo de la dirección de memoria.
- **MEMORY:** Acceso a la memoria de los datos.

- **WRITEBACK:** Etapa de postescritura, se almacena el resultado en el banco de registros.

3.2 Interfaz de usuario

Al arrancar MIPSim se observará la interfaz principal tal y como refleja la Figura 1. En esta pantalla inicial se pueden observar 3 zonas principales:

- Un menú superior donde el usuario puede controlar la propia aplicación y las simulaciones.
- La parte izquierda donde se muestran las instrucciones máquina que van a ser ejecutadas por el procesador en el entorno de simulación.
- La parte derecha donde se muestra el esquema del procesador segmentado MIPS, con las distintas etapas y unidades que lo componen.

3.3 Configuración de simulaciones

Para la carga de una simulación almacenada en disco duro, seleccione la opción del menú superior *File* → *Open...* y para almacenarla *File* → *Save*. Tenga en cuenta, que la opción *File* → *New* crea una nueva simulación, eliminando los valores de la simulación actualmente cargada.

Existe una barra de herramientas principal en la parte superior, mostrada en la Figura 2. Mediante los botones de esta barra de herramientas, el usuario puede llevar a cabo distintas acciones de simulación. De entre todos ellos, los tres situados más a la derecha están enfocados en la ayuda de la aplicación. Los botones situados en la parte central sirven para que el usuario controle el estado de la simulación: adelantar, un paso adelante, un paso atrás, parar, reanudar, etc. Finalmente, los tres botones situados más a la izquierda permiten llevar a cabo las distintas configuraciones de la simulación. A continuación, se verá con detalle la información que muestran y dejan configurar estos tres botones.

3.3.1 Memoria de Instrucciones

Con el botón de más a la izquierda, se abrirá una nueva ventana como la mostrada en la Figura 3 llamada *'assembler'*. Por si resulta más intuitivo, esta misma venta también puede abrirse haciendo clic en el propio banco de instrucciones mostrado en el diagrama del procesador etiquetado como *'Instr.'*

Desde esta ventana emergente podremos agregar las instrucciones máquina que serán procesadas en el pipeline del procesador. En esta ventana se pueden visualizar diferentes columnas: *address* (dirección de memoria), *label* (etiqueta para posibles saltos), *opcode* (instrucción completa en lenguaje ensamblador), *word* (palabra correspondiente a dicha instrucción de memoria, y finalmente, la última columna *instruction set* (listado con las instrucciones disponibles en MIPSim).

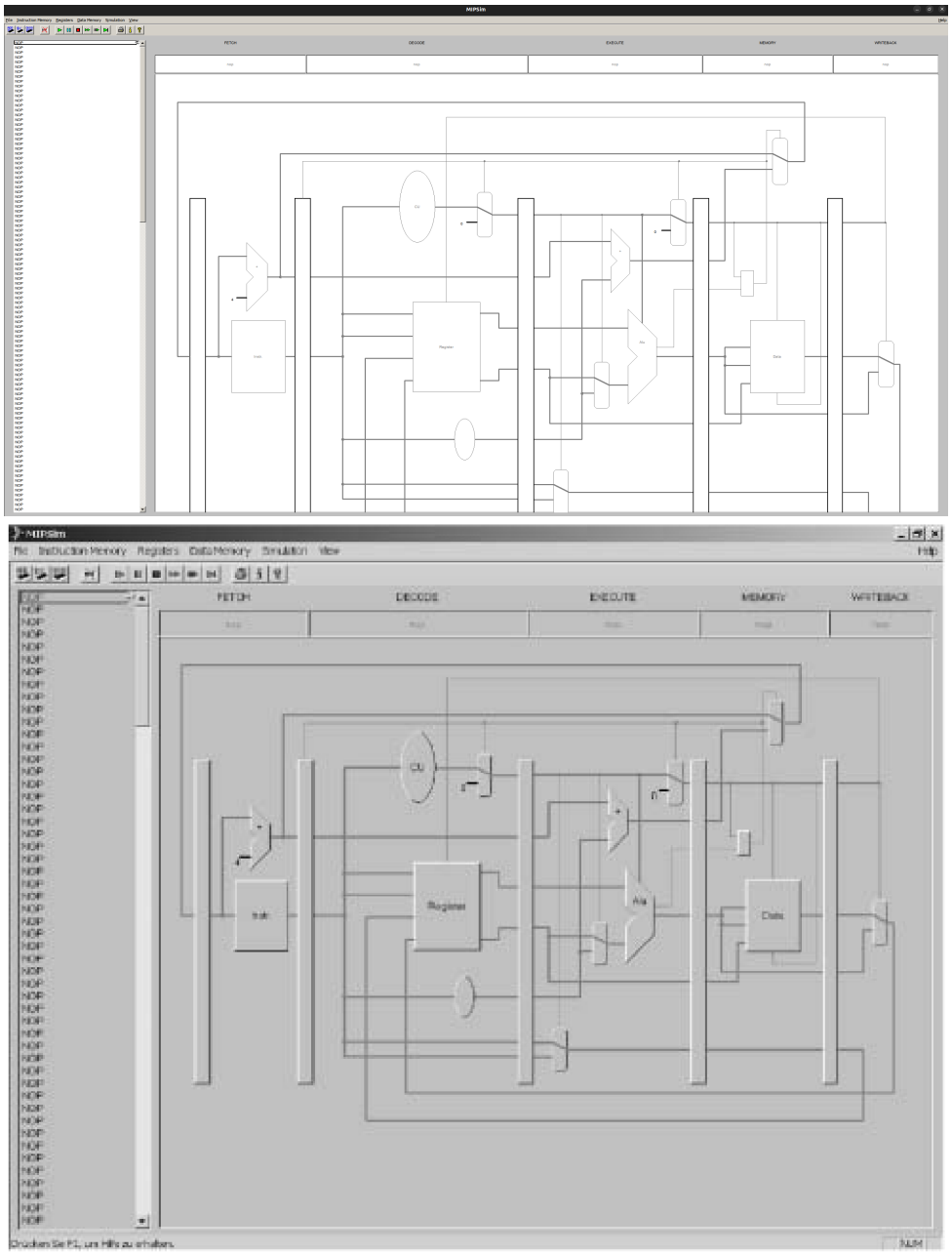

Figura 1: Interfaz principal del entorno MIPSim.

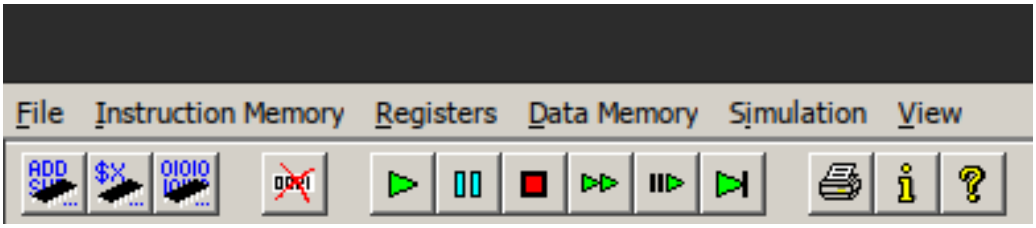

Figura 2: Barra de herramientas del simulador MIPSim.

Como puede observarse, dispone de una barra inferior para poder llevar a cabo acciones de almacenamiento, carga, etc. Haciendo clic en una determinada instrucción en la columna *opcode*, el usuario puede modificar o añadir nuevas instrucciones que se ejecu-

tarán en la simulación.

Nótese que son instrucciones máquina, los registros que el usuario introduzca deben identificarse mediante su código numérico y no por su nombre.

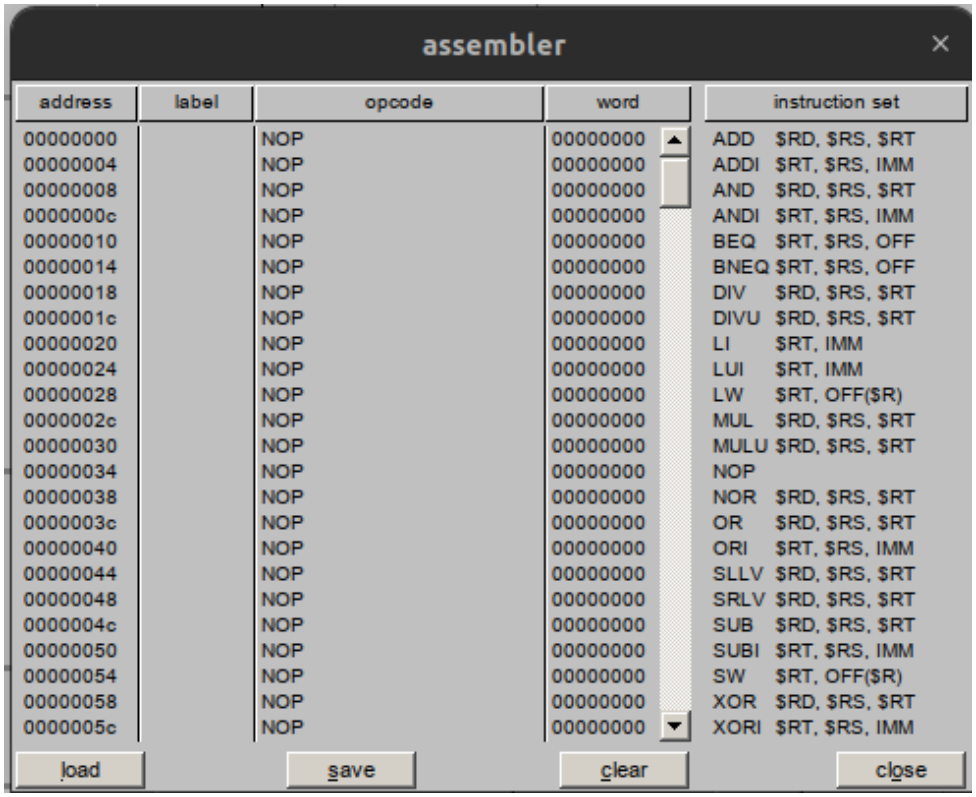

Figura 3: Ventana emergente para el manejo de instrucciones de memoria.

3.3.2 Registros

Pulsando el segundo botón (situado en el centro de los tres botones mostrados en la Figura 2) se abrirá la ventana emergente *'Register'* tal y como en la Figura 4. Al igual que la anterior ventana, también puede abrirse pulsando sobre el diagrama del procesador, en concreto en la caja etiquetada como *'Register'*.

En esta ventana el usuario podrá manejar el contenido de los registros internos del procesador. Únicamente tendrá que cambiar los valores de los registros deseados y pulsar *'enter'*.

3.3.3 Memoria de Datos

Pulsando el último de los tres botones más a la izquierda mostrados en la Figura 2 se abrirá la ventana emergente *'data memory'* tal y como muestra la Figura 5. Al igual que la anterior ventana, también puede abrirse pulsando sobre el diagrama del procesador, en concreto en la caja etiquetada como *'Data'*.

En esta ventana el usuario podrá manejar el contenido de los datos almacenados en las distintas posiciones de memoria. Simplemente seleccionando la dirección de memoria

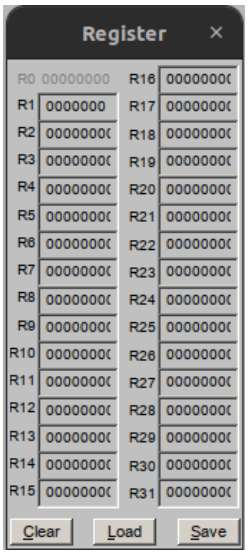

Figura 4: Ventana emergente para el manejo de los registros del procesador.

deseada y fijando su contenido con el valor deseado, después de pulsar *enter* este se verá modificado.

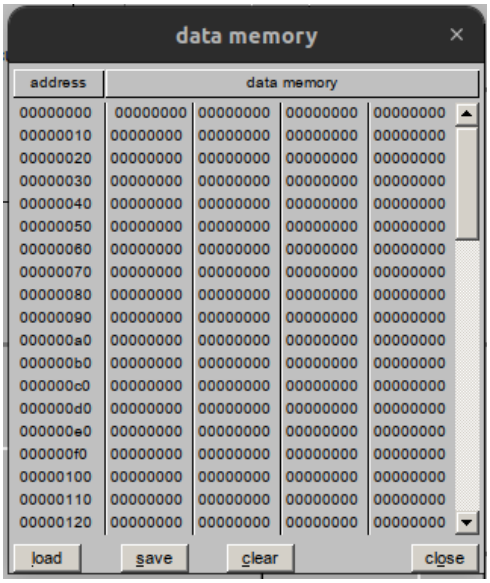

Figura 5: Ventana emergente para el manejo de los datos almacenados en memoria.

3.3.4 Control de la simulación

Del bloque central de seis botones destinados a la simulación, el usuario puede llevar a cabo distintas operaciones relacionadas con este aspecto. A continuación, se detalla una lista de las distintas acciones que se pueden llevar a cabo:

- **Start simulation:** Lanza la simulación con los valores actuales fijados por el usuario.
- **Pause simulation:** Se detiene la simulación con los valores actuales (no se lleva a cabo un reinicio).

- **Stop, reset simulation:** Se detiene y se inicializa la simulación.
- **Simulation, fast mode:** Lleva a cabo una simulación rápida.
- **Simulation, slow mode:** Lleva a cabo una simulación lenta.
- **Simulation, single step:** Lleva a cabo una simulación paso a paso a través de cada etapa para cada instrucción.

3.3.5 Lista de instrucciones

La columna de la parte más a la izquierda muestra el pipeline de instrucciones que se van a llevar a cabo durante la simulación, es decir, las instrucciones que el usuario haya introducido en el banco de instrucciones. La instrucción que entra en el actual ciclo de reloj se encuentra marcada mediante un recuadro y una flecha.

3.3.6 Camino de datos

Es la zona principal del simulador (parte derecha). Como se ha descrito ya con anterioridad, muestra las 5 etapas de un procesador MIPS segmentado, con las distintas unidades y buses. En la parte superior se puede observar el nombre de las distintas etapas junto con el nombre de la instrucción que está atravesando por cada una de las diferentes etapas en un determinado ciclo de reloj.

3.3.7 Repertorio de instrucciones

A lo que respecta al conjunto de instrucciones disponibles, es un número reducido de las vistas ya con el simulador MARS. A continuación, se presenta el listado de instrucciones sin entrar en detalles, ya que, como se acaba de comentar, se vieron con detalle en las anteriores prácticas.

Instrucción	**Descripción**
ADD $RD, $RS, $RT	Suma el contenido de RS y RT almacenándolo en RD.
ADDI $RT, $RS, IMM	Suma el contenido de RS e IMM almacenándolo en RT.
SUB $RD, $RS, $RT	Resta el contenido de RS y RT almacenándolo en RD.
SUBI $RT, $RS, IMM	Resta el contenido de RS e IMM almacenándolo en RT.
DIV $RD, $RS, $RT	Almacena el cociente de RS y RT en RD. Con desbordamiento.
DIVU $RD, $RS, $RT	Almacena el cociente de RS y RT en RD. Sin desbordamiento.
MUL $RD, $RS, $RT	Almacena el producto de RS y RT en RD. Con signo.
MULU $RD, $RS, $RT	Almacena el producto de RS y RT en RD. Sin signo.

Tabla 1: Instrucciones disponibles para aritmética.

Instrucción	Descripción
AND $RD, $RS, $RT	AND lógico de RS y RT almacenándolo en RD.
ANDI $RT, $RS, IMM	AND lógico de RS e IMM almacenándolo en RT.
OR $RD, $RS, $RT	OR lógico de RS y RT almacenándolo en RD.
ORI $RT, $RS, IMM	OR lógico de RS e IMM almacenándolo en RT.
XOR $RD, $RS, $RT	XOR lógico de RS y RT almacenándolo en RD.
XORI $RT, $RS, IMM	XOR lógico de RS e IMM almacenándolo en RT.
NOR $RD, $RS, $RT	NOR lógico de RS y RT almacenándolo en RD.
SLLV $RD, $RS, $RT	Almacena en RD, RS desplazado a la izquierda, RT bits.
SRLV $RD, $RS, $RT	Almacena en RD, RS desplazado a la derecha, RT bits.

Tabla 2: Instrucciones disponibles para operaciones lógicas.

Instrucción	Descripción
BEQ $RT, $RS, OFF	Si el contenido RS y RT son iguales, salta a OFF.
BNEQ $RT, $RS, OFF	Si el contenido RS y RT no son iguales, salta a OFF.

Tabla 3: Instrucciones disponibles para operaciones de salto.

Instrucción	Descripción
LI $RT, IMM	Carga en RT el valor inmediato IMM
LUI $RT, IMM	Carga en RT los 16 bits menos significativos de IMM.
LW $RT, OFF($R)	Carga en RT el contenido de la dirección de memoria de R desplazada OFF (siendo siempre múltiplo de 4 bytes).
SW $RT, OFF($R)	Almacena el contenido de RT en la dirección de memoria de R desplazada OFF (siendo siempre múltiplo de 4 bytes).

Tabla 4: Instrucciones disponibles para la carga/almacenamiento de datos.

Instrucción	Descripción
NOP	Esta instrucción no realiza nada.

Tabla 5: Instrucción de NO-Operación.

3.4 Simulación en MIPSim

En este punto, el usuario ya es conocedor del entorno de usuario que ofrece el simulador MIPSim a la hora de llevar a cabo simulaciones. A continuación, se llevará paso a paso la configuración de una simulación para posteriormente ver los resultados de la ejecución.

En primer lugar, se insertarán las instrucciones que se llevarán a cabo en la simulación, para ello, como se ha comentado anteriormente, se hará clic en el propio banco de instrucciones o en el botón correspondiente de la barra de herramientas (el posicionado más a la izquierda).

Al abrirse la ventana emergente, haremos clic en el primer *opcode* correspondiente a la posición de memoria (*address*) 0x00000000. Aquí insertaremos la instrucción:

```
LW $1, 0($0)
```

Nada más se terminen de introducir los caracteres que forman parte de la instrucción, se pulsará la tecla *'enter'* del teclado para que esta instrucción quede almacenada en el banco de instrucciones. MIPSim, antes de almacenar esta instrucción, llevará a cabo un análisis sintáctico de la misma, en caso de contener errores, saltará un mensaje de error y el usuario deberá corregir la sintaxis.

La siguientes dos instrucciones que insertaremos serán 2 instrucciones `NOP` consecutivas. En este caso ya están por defecto escritas, así que se hará clic en la instrucción correspondiente a la dirección 0x00000004 y 0x00000008 se pulsará *'Enter'*. Finalmente, en la posición de memoria 0x0000000c insertaremos la instrucción:

```
ADD $3, $1, $2
```

En este punto podemos almacenar el contenido del banco de instrucciones pulsando el botón *'save'*. En este caso, MIPSim almacenará el estado actual del banco de instrucciones en un fichero con el nombre que el usuario elija con la extensión *.mp*. No obstante, si no se desea almacenar, únicamente cerrando la ventana emergente, nuestro banco de instrucciones habrá quedado ya actualizado para la simulación.

De nuevo en la pantalla principal, se podrá observar como las nuevas instrucciones introducidas aparecen en el pipeline de instrucciones, marcando con una flecha la primera de ellas, tal y como se muestra en la Figura 6

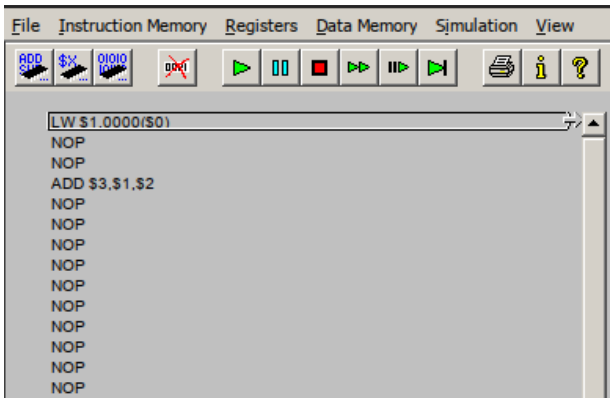

Figura 6: Pipeline de las instrucciones actualmente creadas.

El programa que hemos creado, como el usuario ya es sabedor, carga el contenido de la dirección de memoria 0x00000000 en el registro $1. A continuación, lleva a cabo una NO-Operación para, finalmente, sumar los contenidos de los registros: $1 y $2, almacenando el contenido en $3.

Dicho esto, el siguiente paso será configurar el banco de memoria. Para ello, se hará clic en el banco de datos, abriéndose la ventana emergente *'data memory'*. En la primera palabra referente a la dirección de memoria 0x00000000 almacenaremos el valor 2. De

igual manera que en la anterior configuración, la estado actual del banco de memoria puede almacenarse pulsando el botón *'save'*, con el nombre que el usuario decida junto con la extensión *.md*.

Finalmente, queda configurar el banco de registros. De nuevo, haciendo clic en el propio banco, se abrirá la ventana emergente *'Register'*. Como se ha detallado, el programa creado suma los contenidos de los registros $1 y $2, donde $1 se inicializa al principio del programa; por lo tanto, inicializaremos el contenido del registro *R2* a 4. Como siempre, para almacenar el estado actual del banco de registros pulsando el botón *'save'*, el usuario podrá elegir un nombre para el fichero de almacenamiento, creándose un fichero con dicho nombre junto con la extensión *.mr*.

En este punto, la simulación ya está preparada para ser ejecutada. Para ello, se irá avanzando paso a paso en la ejecución, viendo los diferentes valores que van tomando los buses, las instrucciones que se ejecutan en las distintas etapas segmentadas, etc. Dicho esto, presionaremos una vez sobre el botón de la barra de herramientas *'Simulation single step'*. Veremos cómo ha cambiado el estado del diagrama principal, tal y como se muestra en la Figura 7. En esta misma figura puede observarse cómo la instrucción `LW` entra en la primera etapa *FETCH*, al igual que el valor que contiene cada bus en un determinado instante. El usuario puede mostrar y ocultar los valores de un determinado bus haciendo clic izquierdo sobre este. En modo de ejemplo, el valor del bus de salida del banco de instrucciones toma el mismo valor que el valor de la palabra asignada a la primera instrucción (esto puede comprobarse abriendo el banco de instrucciones).

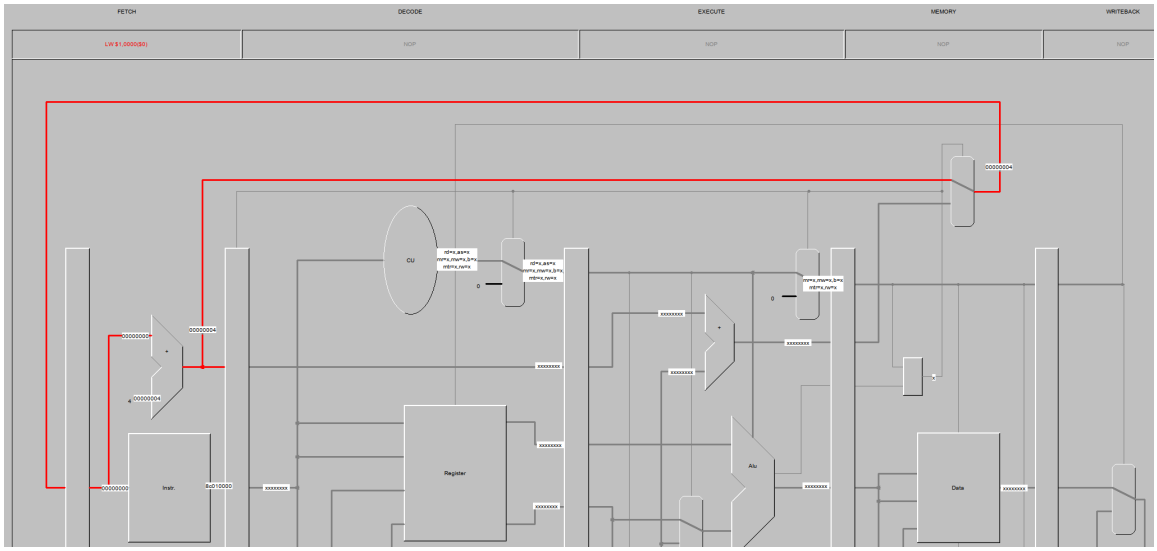

Figura 7: Estado del diagrama de la arquitectura MIPS para el primer paso de la ejecución del ejemplo.

Avanzando un paso más, veremos cómo las etiquetas de los buses mostradas se actualizan, al igual que el color de los mismos. En este caso, la instrucción `LW` ha pasado a la siguiente etapa (*DECODE*), y la instrucción `NOP` ha entrado en la primera etapa de la segmentación *FETCH*. En la etapa *DECODE*, el usuario puede observar los valores de las distintas señales emitidas por la Unidad de Control (*CU*) para esta instrucción pulsando en el bus de salida. Entre las señales, podemos encontrar:

- **rd:** Indica qué registro se usará como destino. En caso de estar activa la señal, el registro de destino será el rd, si no, el rt.
- **as:** Indica cuál es el origen del segundo operando de la ALU. Si está activa la señal, el segundo operando de la ALU será el dato inmediato, en caso contrario, provendrá del banco de registros.
- **mr:** Se activa si hay que llevar a cabo una lectura de memoria.
- **mw:** Se activa si hay que llevar a cabo una escritura a memoria.
- **b:** Se activa si se trata de una instrucción de salto para activar la lógica para el cambio del valor de *pc*.
- **mtr:** Selecciona la fuente del dato que va al registro de destino. Si está activo, el valor proviene de registro, si no, de memoria. MIPSim genera una señal con lógica inversa a la forma comúnmente usada.
- **rw:** Se activa para habilitar la escritura en el banco de registros.

En el el momento actual del pipeline, la instrucción `LW` genera las siguientes señales: *rd=0*, es decir, campo destino *rt*, *as=1* la ALU usa el dato inmediato para el cálculo del desplazamiento, *mr=1* hay que leer datos de memoria, *mw=0* no hay escritura en memoria, *b=0* no se activa la lógica de salto, *mtr=0* el dato proviene de memoria, *rw=1* habilita la escritura en el banco de registros.

Avanzamos hasta que la instrucción `ADD` alcance la etapa *DECODE*, tal y como muestra la Figura 8. En este punto, si abrimos el banco de registros, vemos cómo la carga de datos se ha hecho efectiva, y el valor del registro *R1* se ha actualizado con el de la posición de memoria 0x00000000 que inicializamos previamente. Por otro lado, la salida de los valores del banco de registros corresponden a los deseados (que intervienen en la operación suma), tal y como se muestra en la Figura 9.

Avanzando un único paso en la simulación, vemos cómo la instrucción `ADD` se efectúa en la etapa *EXECUTE*, tomando el valor de $(4 + 2 = 6)$ el bus de salida de la ALU.

Finalmente, avanzando con la instrucción `ADD` hasta la etapa *WRITEBACK*, vemos cómo se actualiza el banco de registro por el color de los buses, como asimismo, abriendo la ventana emergente del banco de registros para comprobar que el valor del registro *R3=6*.

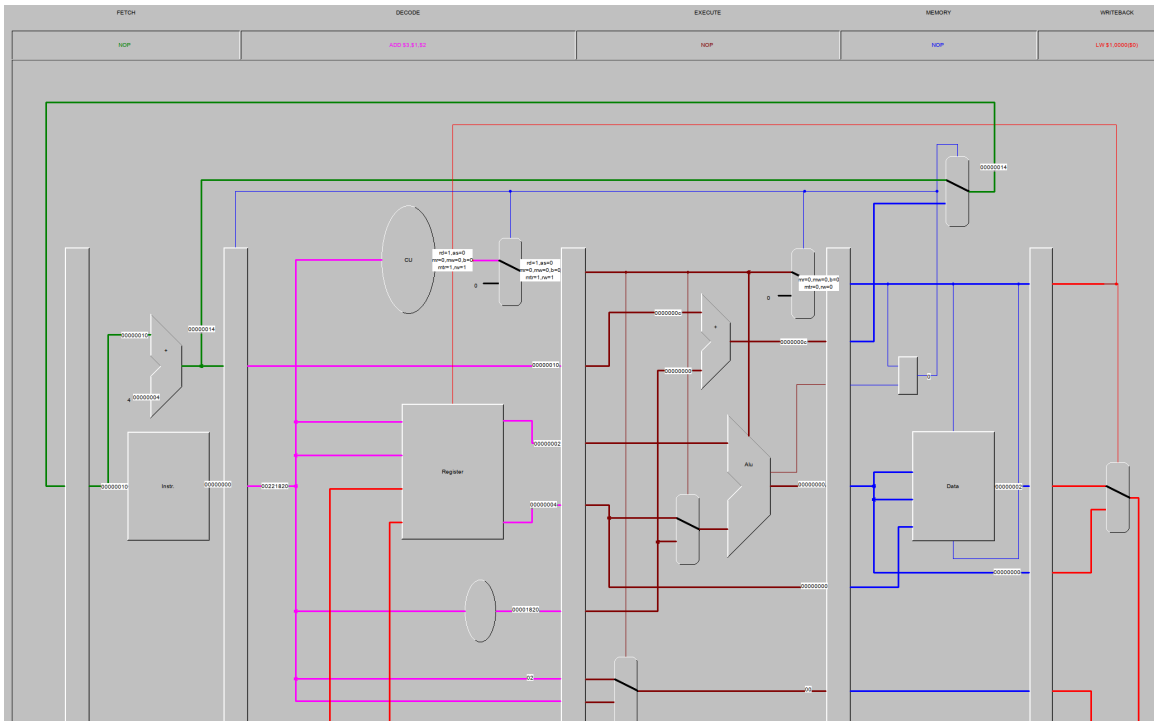

Figura 8: Estado del diagrama de la arquitectura MIPS avanzando por las instrucciones del ejemplo.

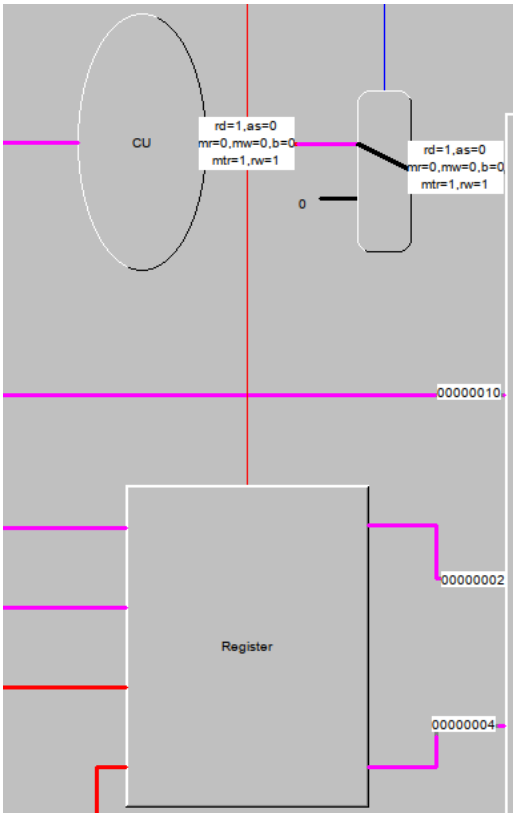

Figura 9: Estado de los buses de salida del banco de registros y señales generadas por la Unidad de Control.

Ejercicio 1

Partiendo del código que se ha usado en el guion de prácticas, responda las siguientes cuestiones.

1.1. Vuelva a lanzar la simulación del ejemplo mostrado en el guion de prácticas e indique las señales que genera la Unidad de Control con el paso de la instrucción `ADD`.

1.2. ¿Cuántos ciclos de reloj han sido necesarios hasta que se almacene el contenido de la operación `ADD` en *R3*?

1.3. ¿En qué ciclo de reloj la instrucción `LW` actualiza el banco de registros con el valor que corresponde? ¿En qué etapa del pipeline se lleva a cabo?

Ejercicio 2

Reinicie por completo el simulador, limpiando los distintos bancos (registros, datos e instrucciones). A continuación, inicialice a 1 los registros *R1, R3, R5 y R6*, a 4 la posición de memoria 0x00000008 e inserte en el banco de instrucciones las mostradas en el Código 1:

Código 1: Código para simulación en MIPSim.

```
1 LW   $2, 8($0)
2 ADD  $4, $1, $3
3 SUB  $1, $5, $6
4 SW   $4, 4($0)
```

2.1. Avance 5 ciclos de reloj en la simulación. ¿Qué valor tiene almacenado en este preciso momento el registro *R2*? ¿Qué valores contienen los buses de entrada a la ALU en la etapa EXECUTE y qué operación se está llevando a cabo en esta misma etapa?

2.2. Avance tantos ciclos de reloj como sea necesario para finalizar por completo todas las instrucciones. ¿Qué valor acaba almacenándose en la posición de memoria 0x00000004? ¿Y en el registro *R4*? ¿Por qué ocurre esto?

2.3. ¿Cuál debería ser el valor realmente de *R4*? ¿Y el de la posición de memoria 0x00000004?

2.3. Compruebe las señales que genera la Unidad de Control para cada una de las instrucciones insertadas en el banco de instrucciones.

Ejercicio 3

En este ejercicio se estudiará el comportamiento de las instrucciones de salto. Para ello, reinicie el entorno MIPSim, e introduzca en el banco de instrucciones el Código 2

Código 2: Código MIPSim para el estudio de las instrucciones de salto.

```
1         LI   $1, 0
2         LI   $2, 0
3         LI   $3, 2
4 LOOP:   ADDI $1, $1, 1
5         LW   $4, 0($2)
6         ADDI $2, $2, 4
7         BNEQ $1, $3, LOOP
```

3.1. Ejecute el código paso a paso e indique las señales emitidas por la Unidad de Control cuando la instrucción `BNEQ` atraviesa la etapa *DECODE*. ¿Cómo se efectúa el salto? Indique el comportamiento del simulador cuando el salto se activa.

3.2. ¿Qué señales emite la Unidad de Control durante el paso de la instrucción *ADDI* (la primera de ellas) por la etapa *DECODE*? ¿Cuáles son los valores de los buses de entrada y el de salida de la ALU para esta misma instrucción?

3.3. ¿Cuántas iteraciones del bucle efectúa el simulador? ¿Es el número de iteraciones esperado?

3.4. ¿Qué valor toma el bus de entrada al banco de instrucciones cuando la instrucción `BNEQ` atraviesa por primera vez la etapa *WRITEBACK*? ¿A qué instrucción hace referencia este valor?

3.5. ¿Cuántos ciclos de reloj son necesarios para finalizar por completo la ejecución del código?