Facultad de Informática
Universidad Complutense de Madrid

# Fundamentos de Computadores II - Práctica 1
# Descubriendo el entorno de trabajo

Daniel Báscones (danibasc@ucm.es)

21 de febrero de 2023

## 1. Objetivos

En este laboratorio exploraremos el funcionamiento interno de un computador. Programaremos un procesador RISC-V, utilizando lenguaje ensamblador. Aprenderemos cómo un procesador va ejecutando las instrucciones básicas que acaban componiendo los programas más complejos que podamos imaginar. En esta primera práctica, veremos:

- Una primera aproximación al repertorio de instrucciones de RISC-V: operaciones con datos, saltos y accesos a memoria.
- Cómo utilizar el entorno de desarrollo Eclipse, con las herramientas GNU para RISC-V.
- La estructura de un programa ensamblador, y cómo acaba ejecutándose en la máquina.

## 2. Introducción a RISC-V

RISC-V define una ISA (*Instruction Set Architecture*), o conjunto de instrucciones. Para ofrecer mayor flexibilidad, se divide en una parte básica, y numerosas extensiones. Nos vamos a centrar en la ISA base **RV32I**. En este caso, el conjunto de instrucciones es de **32 bits**, y opera sobre **números enteros** (***I****nteger*). Las principales características son:

- Tiene un **banco de registros** con 32 registros diferentes, donde se guardan temporalmente los valores con que operan las instrucciones aritmético-lógicas.
- Tiene una **memoria** de hasta 4GB, a la que se accede mediante instrucciones de acceso a memoria. La memoria contiene los datos y las instrucciones del programa.
- Tiene un **contador de programa (pc)**, un registro especial (separado del banco de registros), que indica la dirección de memoria de la instrucción que se está ejecutando.
- Todas las instrucciones tienen 32 bits de anchura.

# 3. Uso del entorno en los laboratorios

**NOTA IMPORTANTE:** *El entorno de desarrollo es altamente configurable en apariencia, por tanto las imágenes aquí mostradas podrían no corresponderse con las de otro usuario. No obstante, las instrucciones se pueden aplicar independientemente de cómo se configure el entorno. Si en algún caso no apareciese una ventana en nuestro entorno, podemos buscarla mediante* `window ->show view` *seleccionando la ventana que queramos. Para colocarla, bastará con arrastrarla a una parte de nuestro entorno de desarrollo.*

En los laboratorios, el entorno ya está instalado. Para abrirlo, debemos entrar con el sistema operativo windows. Una vez iniciada sesión, hay que ejecutar eclipse. Lo podemos encontrar buscando "EclipseRV" en el buscador de windows, o en `C:/software/electronica/EclipseRV`. Al lanzarlo, se mostrará una ventana donde habrá que hacer click en `Launch` (Figura 1).

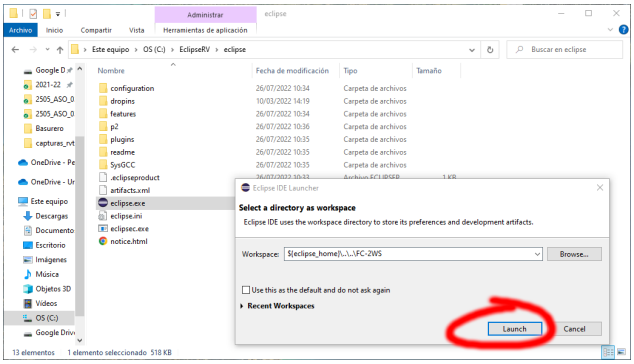


Figura 1: Ejecutando eclipse

Habrá que seleccionar un *workspace* o carpeta de trabajo donde se guardarán las prácticas. Se proporciona un workspace (carpeta) con los proyectos ya preconfigurados para mayor facilidad del alumno (ver web `http://www.fdi.ucm.es/profesor/mendias/FC2/FC2.html`). Estas carpetas son portables, y se recomienda tenerlas en un pendrive para traer el workspace de casa a los laboratorios o viceversa.

**NOTA:** Para utilizar un *workspace* concreto (o si no nos aparece el diálogo de selección de *workspace*), hay que seleccionarlo desde eclipse con `File->Switch Workspace->Other...` Una vez seleccionado, el entorno estará listo para trabajar, como muestra la Figura 2.

# 4. Instalación del entorno en casa

En esta primera práctica, vamos a probar a hacer un código sencillo, comprobando la funcionalidad del entorno de desarrollo eclipse. Para ello, lo primero es realizar la instalación del entorno:

1. Crea una carpeta para la instalación. Es importante que la ruta no contenga espacios ni caracteres extraños (acentos, símbolos...) y tampoco sea muy profunda. Recomendamos `C:/EclipseRV`.

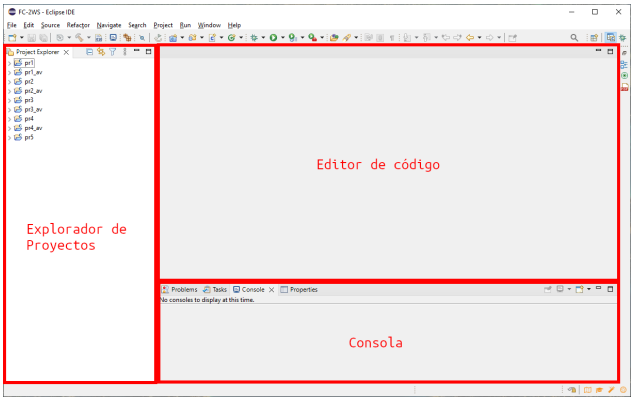


Figura 2: Perspectiva de desarrollo en eclipse

2. Descarga y descomprime el archivo `EclipseRV.zip` en dicha carpeta desde la web `http://www.fdi.ucm.es/profesor/mendias/FC2/FC2.html`. Puedes borrar el archivo comprimido tras este paso.

3. Para hacer las prácticas, ejecuta `C:/EclipseRV/eclipse/eclipse.exe`. (Figura 1).

4. Una vez lanzado el eclipse, debemos seleccionar un *workspace*.

5. Desde la misma web, podemos descargar dos workspaces diferentes: `FC2practicasWS` para las prácticas, y `FC2ejerciciosWS` con los ejercicios preparados para su ejecución.

Otra opción es utilizar la máquina virtual (NOTA: no funciona en los MAC-M1 y M2):

1. Descarga la máquina virtual (enlazada en el campus, o proporcionada por los profesores) y descomprímela en tu disco duro.

2. Descarga VirtualBox desde `https://www.virtualbox.org/wiki/Downloads` para tu sistema operativo.

3. Importa la máquina descomprimida, y tendrás en el escritorio de la máquina virtual tanto un acceso directo a Eclipse (ya configurado) como la carpeta del workspace.

# 5. Código de ejemplo

Una vez instalado el entorno, vamos a realizar la primera práctica, para aprender a manejarnos con Eclipse, y con el lenguaje ensamblador RISC-V.

En primer lugar, veamos el aspecto que tiene un programa en ensamblador. Para ello, utilizamos este código sencillo que calcula el mayor de dos números:

```
int a = 5, b = 8;
int mayor;
if (a > b)
    mayor = a;
else
    mayor = b;
```

El código traducido a ensamblador RISC-V es el siguiente:

```
.data                   //sección de datos con valor inicial
    A:      .word 5     //dato 'A' con valor 5
    B:      .word 8     //dato 'B' con valor 8

.bss                    //sección de datos sin valor inicial
    MAYOR:  .space 4    //dato 'MAYOR' ocupa 4 bytes

.text                   //sección de código
.global main            //''main'' se puede llamar desde fuera
main:                   //punto de entrada del código
    la t2, A            //cargamos en t2 la DIRECCIÓN de A
    lw t0, 0(t2)        //usamos la dirección para cargar el VALOR de A
    la t3, B            //cargamos en t3 la DIRECCIÓN de B
    lw t1, 0(t3)        //usamos la dirección para cargar el VALOR de B
    ble t0, t1, mayb    //Si t0 (A) <= t1 (B), salto (condición inversa)
                        //en este caso no hemos saltado (a es mayor)
    la t4, MAYOR          //cargamos en t4 la DIRECCIÓN de MAYOR
    sw t0, 0(t4)          //guardamos t0 (A) en MAYOR
    j fin                 //saltamos al final del programa
mayb:                   //en este caso, b es mayor (o igual)
    la t4, MAYOR          //cargamos en t4 la DIRECCIÓN de MAYOR
    sw t1, 0(t4)          //guardamos t1 (B) en MAYOR
fin:                    //fin del programa
    j fin               //salto sobre sí mismo, parando el programa
```

Como vemos, es algo más largo, porque las operaciones que realiza ocurren a un nivel muy elemental, y son más sencillas que las de alto nivel en código c. A continuación ejecutaremos este código en el entorno, para ver cómo funcionan los diferentes elementos del mismo. Pero antes de nada, debemos decir al procesador cómo se colocan estas instrucciones en la memoria. Para ello se emplea el *linker script*, que define la estructura del programa. En nuestro caso, es el siguiente:

```
ENTRY( main )                  //ejecuta comenzando en ''main''
SECTIONS {                     //distribución del código en memoria
     . = 0x0;                  //escribe desde la dirección 0
     .text    : {* (.text)}    //coloca .text (desde la dirección 0)
     . = 0x10000;              //escribe desde la dirección 0x10000
     .data    : {* (.data)}    //coloca .data (desde la dir 0x10000)
     .bss     : {* (.bss)}     //coloca .bss (después de .data)
     .rodata  : {* (.rodata)}  //coloca .rodata (después de .bss)
}
```

En este caso, el código comenzará en la dirección `0x0`, y las diferentes secciones de datos se situarán en la dirección `0x10000`. Esto permite separar los diferentes componentes del programa a diferentes secciones de memoria o, incluso, a diferentes memorias.

## 5.1. Arrancando el entorno

Lo primero que haremos, tras ejecutar eclipse, es explorar los archivos fuente que utilizaremos a nuestro proyecto. Normalmente estos ficheros estarán vacíos, o a medio completar, y la práctica consistirá en terminarlos. En este ejemplo utilizaremos el proyecto `pr1`. (Ver Figura 3).

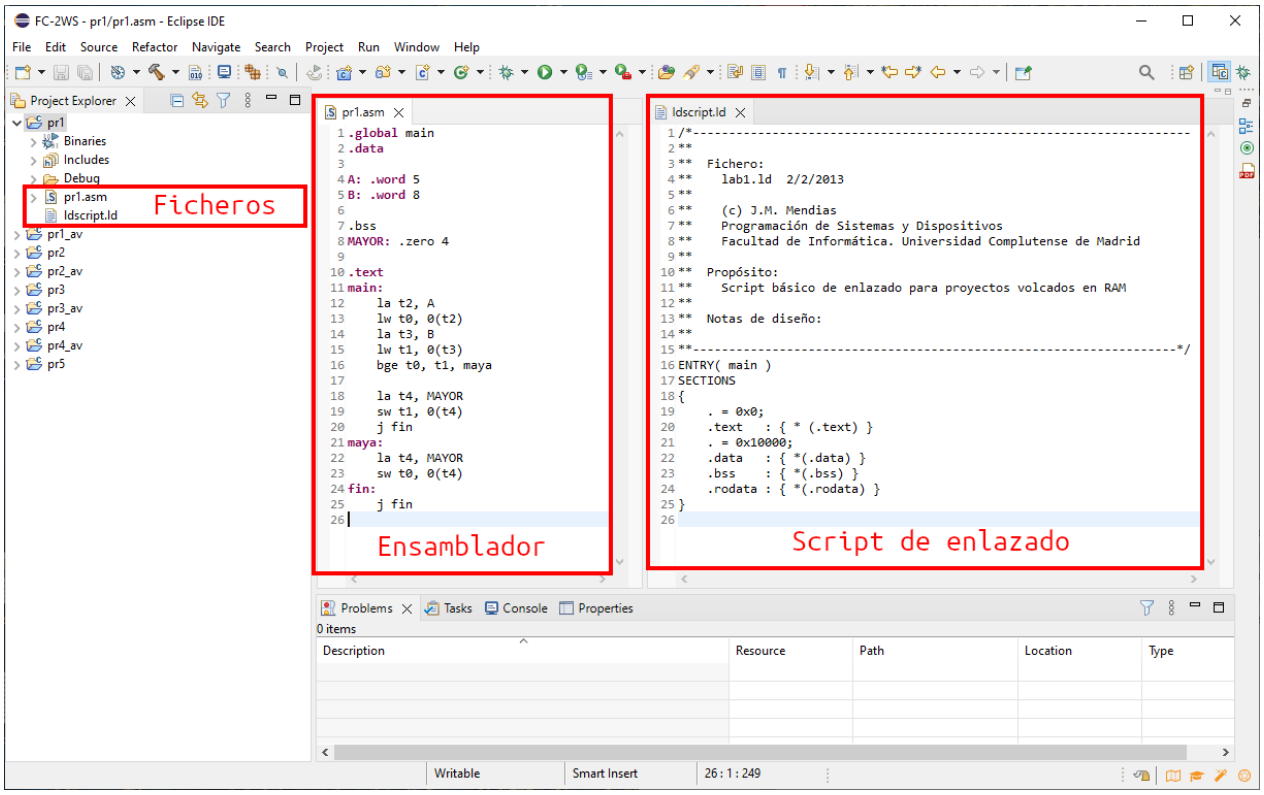


Figura 3: Vista inicial del proyecto con el código

A continuación, será necesario compilar el proyecto. Para ello, con el proyecto seleccionado

en el explorador (basta haber hecho click en la carpeta), pulsamos sobre el martillo de la barra de herramientas (*build*). Deberemos abrir la vista de consola para ver el resultado (Figura 4). Si no hay errores, podemos pasar al siguiente paso.

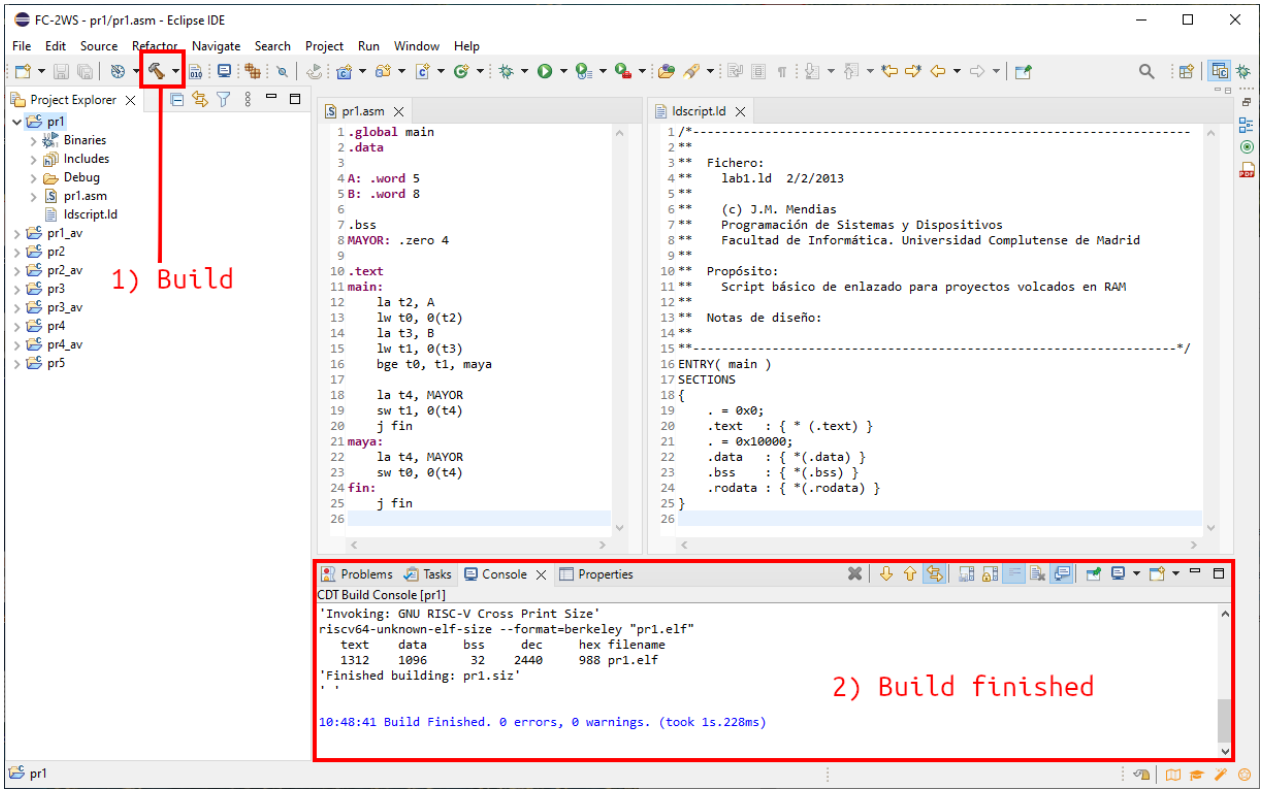


Figura 4: Compilación (*build*) del proyecto.

El siguiente paso es algo más delicado, y consiste en la depuración del proyecto (Ver Figura 5).. Debemos pulsar en las opciones de depuración (la flecha a la derecha del escarabajo de la barra de tareas), y seleccionar la depuración de nuestro proyecto. Normalmente, aparecerá un diálogo diciendo que si queremos pasar a la *perspectiva debug*. Esto nos permitirá un análisis exhaustivo de los elementos internos del procesador: registros, memoria, instrucciones... Debemos aceptar. En caso de no hacerlo o no encontrar la opción, siempre podremos cambiar desde `window->perspective->open perspective->other->debug`.

Al cambiar, nos encontraremos una perspectiva algo vacía, como se ve en la Figura 6. Deberemos ajustar algunos parámetros para poder ver toda la información que nos interesa: Registros, memoria, y desensamblado. Esta operación se ve en la Figura 7.

Finalmente, podremos ejecutar el código. Para ello disponemos de los controles de flujo de la barra superior de herramientas, en forma de flechas amarillas (Figura 8). Con ellas podremos saltar (siempre hacia adelante) en el código, tanto a nivel de instrucciones como de funciones. Si queremos ejecutar el código completo, podemos dar a la flecha verde. La ejecución podría quedarse atascada en un salto infinito, para ello deberemos pausarla con el icono de pausa (dos barras amarillas). Finalmente, para cancelar la ejecución, disponemos del botón de stop (cuadrado rojo). Esto cerrará el programa y borrará los datos de ejecución.

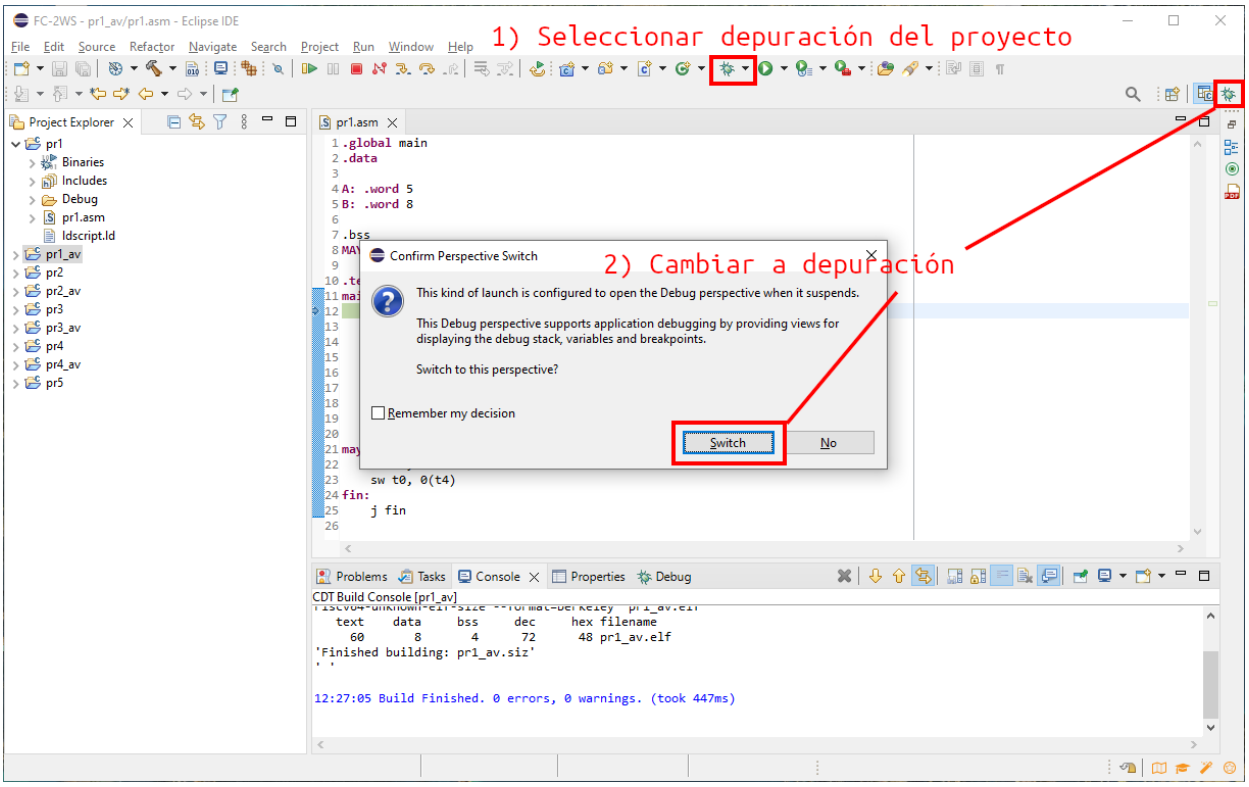


Figura 5: Cambio de perspectiva

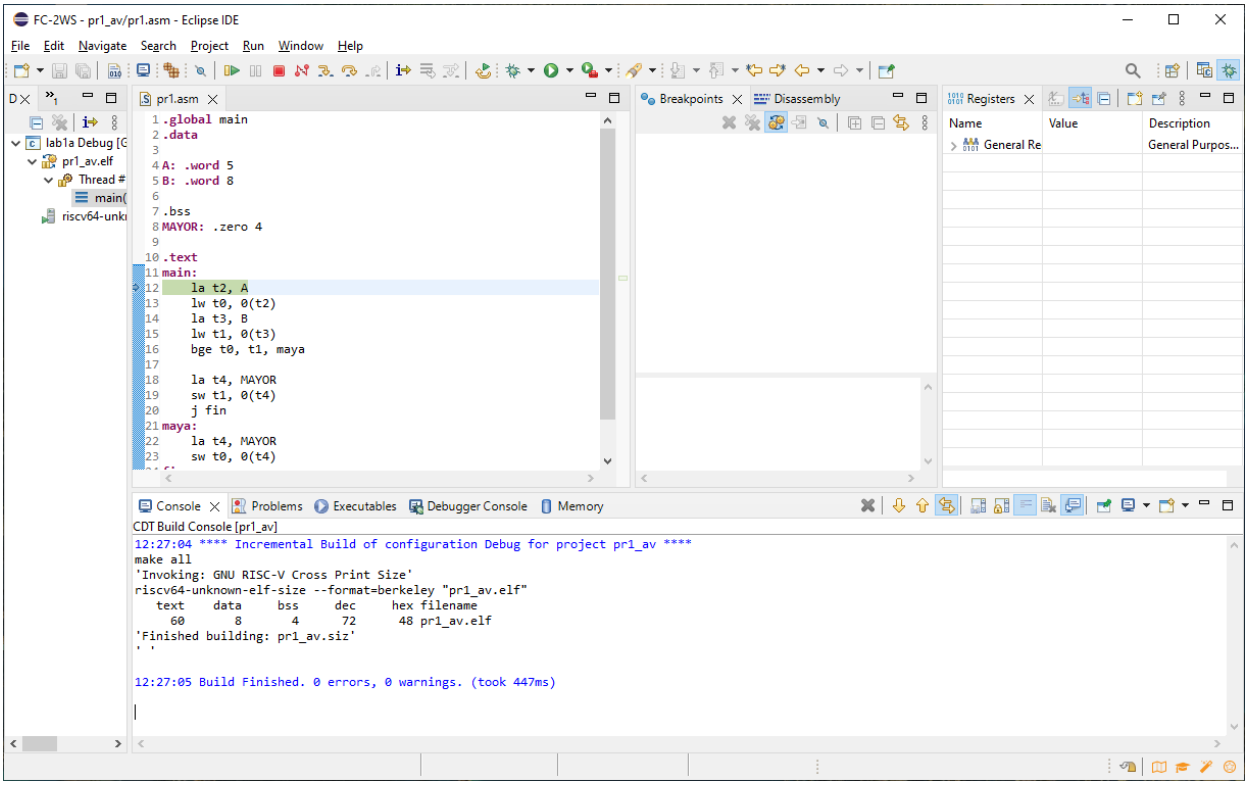

Figura 6: Perspectiva cambiada a depuración.

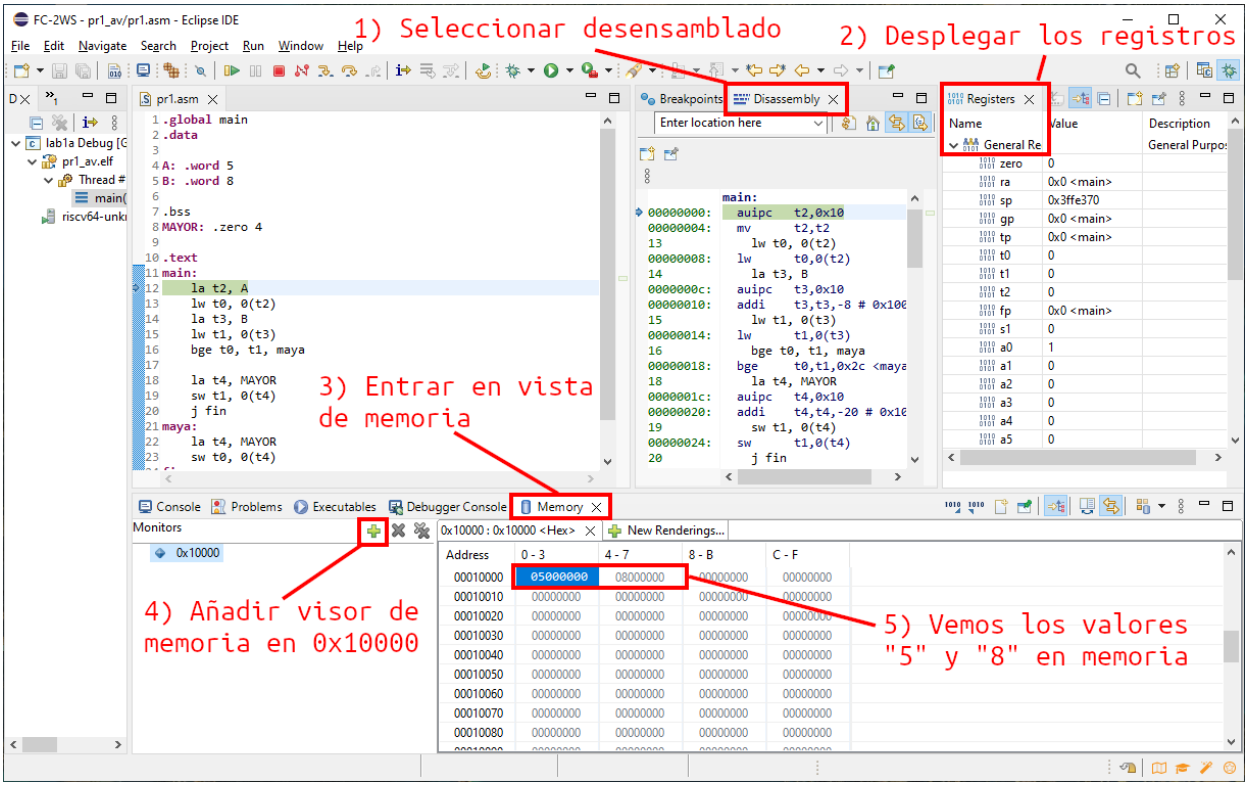


Figura 7: Perspectiva de depuración (*debug*) con registros, desemsamblado y memoria visibles.

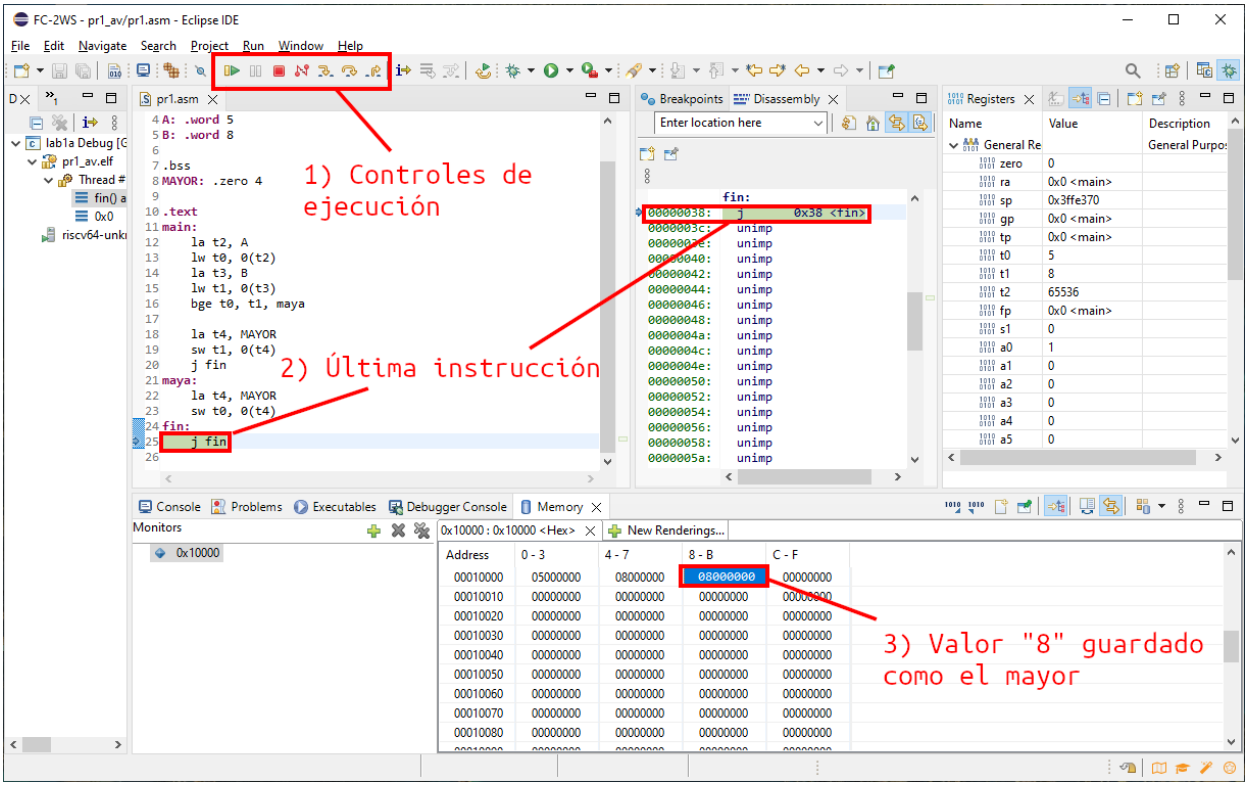


Figura 8: Perspectiva de depuración tras la ejecución.

En la Figura 8 podemos ver un ejemplo de ejecución del programa. Observamos que en la dirección de memoria `0x10008` se ha guardado un `8`, que es el valor mayor de los dos presentes en el programa.

¡Con esto ya estamos listos para experimentar con las funcionalidades del entorno!

## 5.2. Desarrollo de la práctica

Utilizando el material de clase, así como los códigos de ejemplo proporcionados, codificar en lenguaje ensamblador RISC-V el siguiente pseudocódigo de alto nivel:

```
#define N 10
int res = 0;
for (int i = 0; i < N; i++) {
    res += i;
}
```

Declarar las variables en las secciones de programa adecuadas. Hay que tener especial cuidado a la hora de elegir los registros utilizados. Comprobar, tras la ejecución, que la dirección de memoria de *res* contiene su valor. ¿Cuál es esta dirección?