

Diseño Basado en Microprocesadores
Práctica 1
Entorno de desarrollo para x86
Primeros pasos en ensamblador

Índice

1. Objetivos	2
2. La plataforma Eclipse	2
3. Ejemplo de creación de un programa	2
3.1. Inicio y configuración de Eclipse	2
3.2. Creación del proyecto	3
3.3. Configuraciones de un proyecto	5
3.4. Añadir ficheros fuente al proyecto	5
3.5. Ajustar las opciones de construcción del proyecto	9
3.6. Construcción del proyecto	11
4. Ejecución de nuestro programa desde Eclipse	12
5. Depuración de programas desde Eclipse	12
5.1. Cambiar la sintaxis del desensamblado	12
5.2. Iniciar la depuración	13
5.3. La vista Debug	14
5.4. La vista Variables	14
5.5. La vista Registers	14
5.6. La vista Disassembly	15
5.7. Barra de herramientas de depuración	15
5.8. Puntos de ruptura (<i>breakpoints</i>)	16
5.9. Depuración de funciones en ensamblador	17
5.10. Terminar la sesión de depuración	18
5.11. La salida estándar y la consola durante la depuración	18
6. Ejercicios	18
6.1. Ejercicio 1	18
6.2. Ejercicio 2	18
6.3. Ejercicio 3	19

1. Objetivos

La primera práctica dedicada a la familia x86 tiene como objetivo conocer el manejo del entorno de desarrollo Eclipse y realizar los primeros programas en ensamblador.

2. La plataforma Eclipse

Eclipse es una plataforma para la creación de entornos de desarrollo integrados (IDE). Es un proyecto de código abierto y al estar desarrollado en Java funciona tanto en Windows como en Linux. Esta plataforma ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java, llamado Java Development Toolkit (JDT), y el IDE para C/C++, llamado C Development Toolkit (CDT), que usaremos en las prácticas.

3. Ejemplo de creación de un programa

En esta sección se indicarán los pasos necesarios para realizar un programa en C que llamará a una función escrita en ensamblador a través de la creación de un ejemplo concreto. Cuando el programa se ejecute pedirá al usuario la introducción de dos números enteros. A continuación los comparará, obtendrá el mayor de los dos números y lo imprimirá. El programa estará compuesto de dos ficheros fuente, uno conteniendo el código en C y otro que contendrá la función en ensamblador. El fuente en C se encargará de pedir al usuario los datos de entrada e imprimir el resultado, mientras que el fuente en ensamblador contendrá la función que comparará los números y obtendrá el mayor de los dos.

3.1. Inicio y configuración de Eclipse

Iniciar Eclipse abriendo el fichero `eclipse` situado en el directorio `eclipse` de la carpeta personal (puede ser conveniente situar un enlace en el escritorio). Al iniciarse, Eclipse nos pide que indiquemos cual es el directorio de espacio de trabajo que vamos a usar. El directorio de espacio de trabajo contendrá los diferentes proyectos que creemos así como las preferencias que fijemos para trabajar con Eclipse.

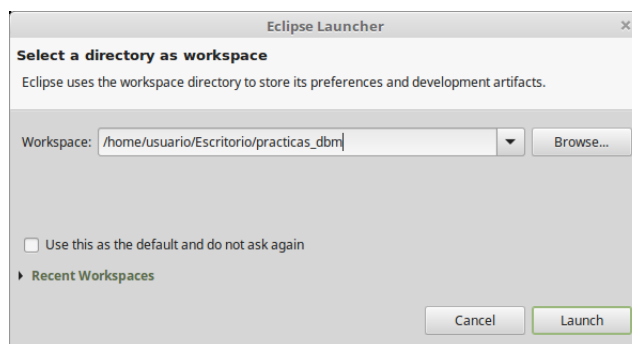


Figura 1: Ventana de selección del espacio de trabajo

Si es la primera vez que abrimos el espacio de trabajo, Eclipse nos mostrará una ventana

de bienvenida. Pulsando sobre el icono *Go to workbench* iremos directamente al entorno de desarrollo.

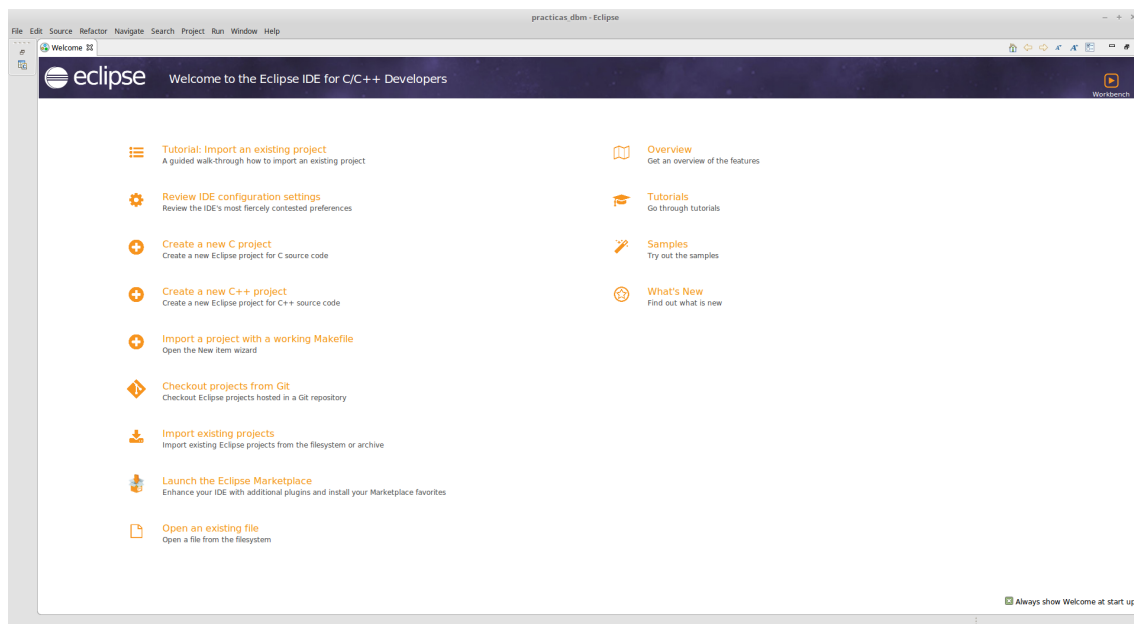


Figura 2: Si aparece la ventana de bienvenida. Pulsar el icono *Go to workbench*.

Seguidamente cambiaremos algunas opciones de configuración. Lo primero que haremos será desactivar la opción *Project → Build Automatically*. Si esta opción está activada el proyecto se recompilará automáticamente cada vez que hacemos un cambio en un fichero fuente y preferimos que esto sólo ocurra cuando nosotros lo deseemos.

Para evitar la necesidad de recordar grabar todos los ficheros fuentes que hayan sufrido modificación antes de lanzar manualmente la construcción de las aplicaciones, activaremos la opción que hace que Eclipse los salve automáticamente por nosotros. Para ello seleccionamos la opción *Window → Preferences*. En la ventana de preferencias seleccionamos *General → Workspace*, marcamos la casilla *Save automatically before build* y pulsamos *Apply*.

Eclipse incluye un corrector ortográfico para los comentarios que puede adaptarse a cualquier idioma si el usuario suministra un fichero de lista de palabras adecuado. Si no tenemos instalado dicho diccionario es conveniente desactivar el corrector o la mayoría de las palabras en los comentarios serán marcadas como error. Para ello, en la ventana de preferencias seleccionamos *General → Editors → Text editors → Spelling*, desactivamos la casilla *Enable spell checking* y pulsamos el botón *Apply*.

3.2. Creación del proyecto

Para crear un nuevo proyecto seleccionamos la opción *File → New → C project*. Introduce *practica_01_ejemplo_01* como nombre para el proyecto. En *Project type* selecciona *Empty Project*. También tendremos que seleccionar el compilador de C que vamos a usar. Según estemos usando Linux o Windows este paso es ligeramente diferente.

- **Si estamos usando Linux:** en caso de que en el apartado *Toolchains* aparezcan varias opciones, asegúrate de que está seleccionada la opción *Linux GCC*.

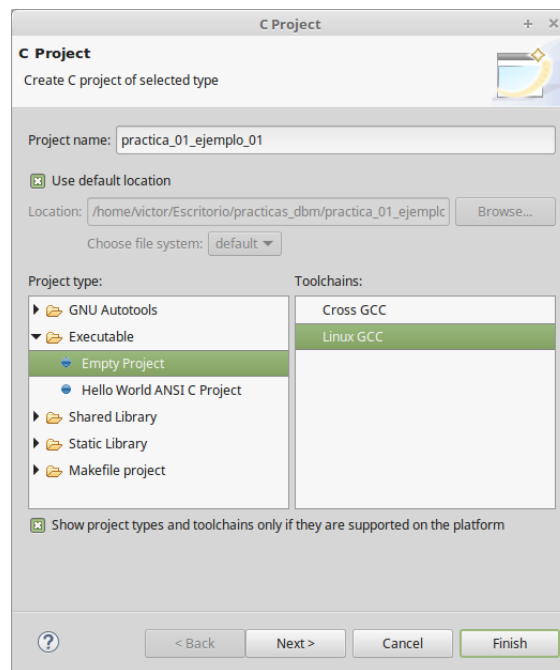


Figura 3: Ventana de creación de un nuevo proyecto en Linux.

■ Si estamos usando Windows:

desmarca la casilla “*Show project types and toolchains only if they are supported on the platform*” y, a continuación, dentro del panel “*Toolchains*” selecciona “*MinGW GCC*”.

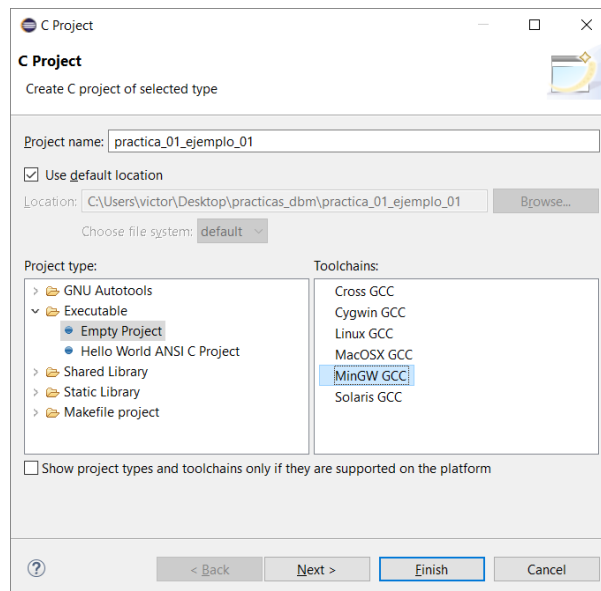


Figura 4: Ventana de creación de un nuevo proyecto en Windows.

Seguidamente, pulsa el botón *Finish*.

Al crear el proyecto Eclipse creará un directorio para él dentro del directorio de espacio

de trabajo. En la ventana del explorador del proyecto o *Project Explorer* de la izquierda podemos ver el icono de la carpeta correspondiente al proyecto recién creado.

3.3. Configuraciones de un proyecto

Como veremos más adelante, desde Eclipse podemos ajustar las opciones que se usarán al llamar al compilador, ensamblador y enlazador cuando construyamos el proyecto. Eclipse permite además que manejemos diferentes configuraciones de un mismo proyecto, pudiendo seleccionar las opciones de construcción que se usarán para cada una. Al crear un proyecto se crean por defecto dos configuraciones: la configuración *Debug* y la configuración *Release*. Los ficheros específicos de cada configuración se almacenan automáticamente en directorios que Eclipse crea dentro del directorio principal de nuestro proyecto.

La configuración *Debug* está pensada para ser utilizada durante la fase de desarrollo de la aplicación. Por ello tiene activados los flags del compilador que hacen que se añada información de depuración al ejecutable y por contra tiene desactivados los flags de optimización del código. De esta forma, resulta posible la depuración mediante un depurador simbólico. El motivo de desactivar la optimización es que, en caso contrario, el depurador no puede encontrar en muchas ocasiones la correspondencia entre el código máquina y la línea de código fuente original de la que procede debido a las supresiones y alteraciones en el orden del código generado que produce la optimización. Esto se traduce frecuentemente en un comportamiento desconcertante del depurador que conviene evitar.

Por otro lado, la configuración *Release* tiene desactivados los flags para añadir información de depuración y tiene activados los flags para optimizar el código. Como es lógico, la activación de la optimización persigue obtener un programa más pequeño y/o rápido. La eliminación de la información de depuración tiene como objetivo reducir el tamaño del ejecutable y también dificultar la tarea de los que quieran conocer el funcionamiento interno de nuestro programa con fines ilícitos.

Al construir el proyecto se usarán las opciones fijadas en la configuración activa. La configuración *Debug* está activa por defecto. Podemos cambiar la configuración activa, por ejemplo, pulsando con el botón derecho sobre el nombre del proyecto en la ventana del explorador de proyectos, seleccionando *Build Configurations* → *Set Active* y la configuración que deseemos.

3.4. Añadir ficheros fuente al proyecto

A continuación, crearemos los ficheros fuentes que formarán parte del proyecto de ejemplo. Para ello, elegiremos la opción *File* → *New* → *Source file*. Alternativamente podemos pulsar con el botón derecho del ratón sobre el nombre de nuestro proyecto en la ventana del explorador de proyecto y seleccionar *File* → *New* → *Source file*.

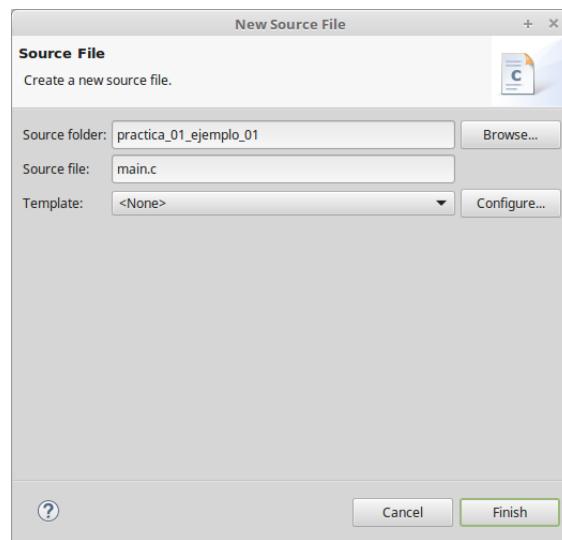


Figura 5: Ventana de creación de un nuevo fichero fuente.

En la ventana de nuevo fichero fuente rellenar el campo *Source File* con `main.c`. En la opción *Template* podemos elegir *None* o *Default C source template*, en cuyo caso se generará automáticamente un comentario de cabecera en el fichero fuente.

Introducir el siguiente programa.

Listado 1: Fichero fuente en C para el programa de ejemplo.

```
/*=====
 * Fichero: main.c
 *-----
 * Fichero en C para el programa de ejemplo.
 * =====
 */

#include <stdio.h>

int mayor(int x, int y);

int main(int argc, char *argv[])
{
    int    a, b, c;
    char   t;

    /* Desactivar el buffer de stdout para facilitar la depuración a través de
     * la consola de Eclipse.
     */
    setvbuf(stdout, NULL, _IONBF, 0);

    printf("\nIntroduzca un número entero: ");
    scanf("%d", &a);

    printf("\nIntroduzca otro número entero: ");
```

```

scanf("%d", &b);

c = mayor(a, b);

printf("\nEl mayor de los números es el %d", c);

printf("\nPulse 's' y ENTER para salir.");

do
{
    scanf("%c", &t);
}
while(t != 's' && t != 'S');

return 0;
}

```

En el listado fuente del programa en C puede extrañar el uso de la función `setvbuf`. La llamada a esta función tiene como objetivo desactivar el buffer del *stream* de salida estándar *stdout*. Este ajuste nos permitirá mejorar el comportamiento de la ventana de la consola cuando ejecutemos nuestro programa en el depurador. Esto se explica con más detalle en el apartado 5.11.

Ahora introduciremos el fichero fuente con la función en ensamblador que devolverá el mayor de los números. Crear para ello un nuevo fichero fuente. Esta vez el nombre debe ser `mayor.S`. Los ficheros fuentes en ensamblador suelen tener extensión `.S` o `.asm`. Es importante que la 'S' sea mayúscula para que Eclipse reconozca el fichero como fuente en ensamblador.

El fichero fuente en ensamblador será ligeramente diferente según estemos trabajando en Linux o en Windows. Si estamos usando Linux introduciremos el siguiente listado.

Listado 2: Función `mayor` en ensamblador si estamos usando Linux.

```

;=====
; Fichero: mayor.S
;-----
; Fichero con la función mayor en ensamblador que devuelve el mayor de dos
; números enteros pasados como parámetros.
;
; Versión para Linux.
;=====

global    mayor

section .text

mayor:    push    ebp
          mov     ebp, esp

          mov     eax, [ebp + 8]
          cmp     eax, [ebp + 12]
          jge     short el_primer_es_mayor

```

```

    mov     eax, [ebp + 12]

el_primer_es_mayor:
    pop     ebp
    ret

```

Mientras que si estamos usando Windows el fichero fuente a introducir será el siguiente.

Listado 3: Función `mayor` en ensamblador si estamos usando Windows.

```

;=====
; Fichero: mayor.S
;-----
; Fichero con la función mayor en ensamblador que devuelve el mayor de dos
; números enteros pasados como parámetros.
;
; Versión para Windows.
;=====

global     _mayor

section .text

_mayor: push    ebp
        mov     ebp, esp

        mov     eax, [ebp + 8]
        cmp     eax, [ebp + 12]
        jge     short el_primer_es_mayor
        mov     eax, [ebp + 12]

el_primer_es_mayor:
        pop     ebp
        ret

```

Como vemos, la diferencia entre uno y otro consiste en la presencia de un carácter guion bajo delante del identificador del nombre de la función, tanto en la directiva `global` como en la etiqueta que marca el punto de entrada a la función. Esto se debe a que, por defecto, los compiladores para Windows añaden automáticamente un guion bajo delante de todos los identificadores al escribirlos en los ficheros objeto, mientras que NASM no lo hará así. Por tanto, para que el enlazador pueda resolver la llamada que en la función `main` se realiza a la función `mayor` debemos poner manualmente un guion bajo delante del nombre de la función. Lo mismo tendremos que hacer con todos los símbolos que exportemos desde los fuentes en ensamblador con intención de que sean accesibles desde el código en C. Esto no será necesario en el caso de símbolos a los que no necesitemos acceder desde C. Por ejemplo, la etiqueta `el_primer_es_mayor` se ha escrito sin precederla de un guion bajo porque sólo se necesita dentro del fuente en ensamblador para marcar la instrucción destino del salto `jge` y no se precisa acceder a ella desde C.

3.5. Ajustar las opciones de construcción del proyecto

En las prácticas, en lugar del ensamblador GNU, llamado *gas*) por *GNU Assembler*, usaremos el ensamblador NASM. La razón para hacerlo así es que, como *gas* está pensado para ser llamado primordialmente por *gcc*, que siempre le suministra código correcto, la comprobación de errores que realiza es limitada. Además, el ensamblador *gas* no reconoce todas las instrucciones SIMD de punto flotante SSE, SSE2, etc.

Para que Eclipse llame a NASM en lugar de a *gas* cuando construyamos el programa hay que cambiar la correspondiente opción del proyecto. Esto se consigue seleccionando *Project* → *Properties* en el menú principal o pulsando con el botón derecho del ratón sobre el icono de nuestro proyecto en el panel del explorador de proyecto y seleccionando *Properties*. En la ventana de propiedades del proyecto seleccionamos *C/C++ Build* → *Settings* en la vista en árbol de la izquierda. Verificamos que está activa la configuración *Debug*. En la ficha *Tool Settings* seleccionamos *GCC Assembler* y en el campo *Command* sustituimos *as* por *nasm*.

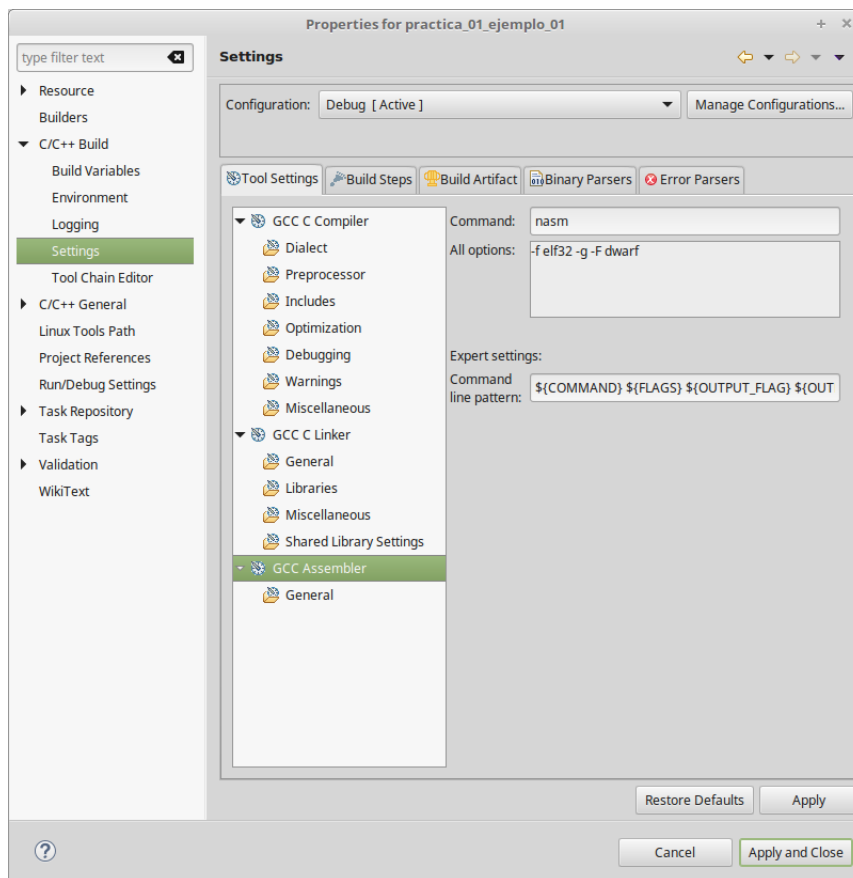


Figura 6: Tanto en Linux como en Windows, cambiar el comando de llamada al ensamblador por *nasm*.

Ahora seleccionamos *General* debajo de *GCC Assembler* y en el campo *Assembler flags* añadimos los flags que de indican según estemos trabajando en Linux o en Windows

- **Si estamos usando Linux:**

en *Assembler flags* añadimos los siguientes flags:

```
-f elf32 -g -F dwarf
```

El flag `-f` seguido de `elf32` indicará a NASM que genere los ficheros objeto en el formato ELF32 que es el utilizado por los módulos objeto y ejecutables de 32 bits en Linux. El flag `-g` le dice a NASM que incorpore información de depuración en los ficheros objeto, la cual nos resultará muy útil a la hora de depurar nuestros programas. La opción `-F dwarf` hará que la información de depuración esté en formato DWARF, que es el mismo formato de información de depuración que usará compilador GCC.

Además si nuestro Linux es de 64 bits, que es el caso del Linux instalado en los ordenadores del laboratorio, y puesto que los primeros programas que realizaremos en prácticas serán aplicaciones de 32 bits, en este momento tendremos que añadir flags para el compilador y el enlazador de forma que generen código de 32 bits. Para ello, en las *Tool Settings* de la ventana de propiedades del proyecto, seleccionamos *GCC C Compiler* → *Miscellaneous* y en el campo *Other flags* añadimos la opción

```
-m32
```

a las que ya existan. Seleccionamos ahora *GCC C Linker* → *Miscellaneous* y añadimos la misma opción

```
-m32
```

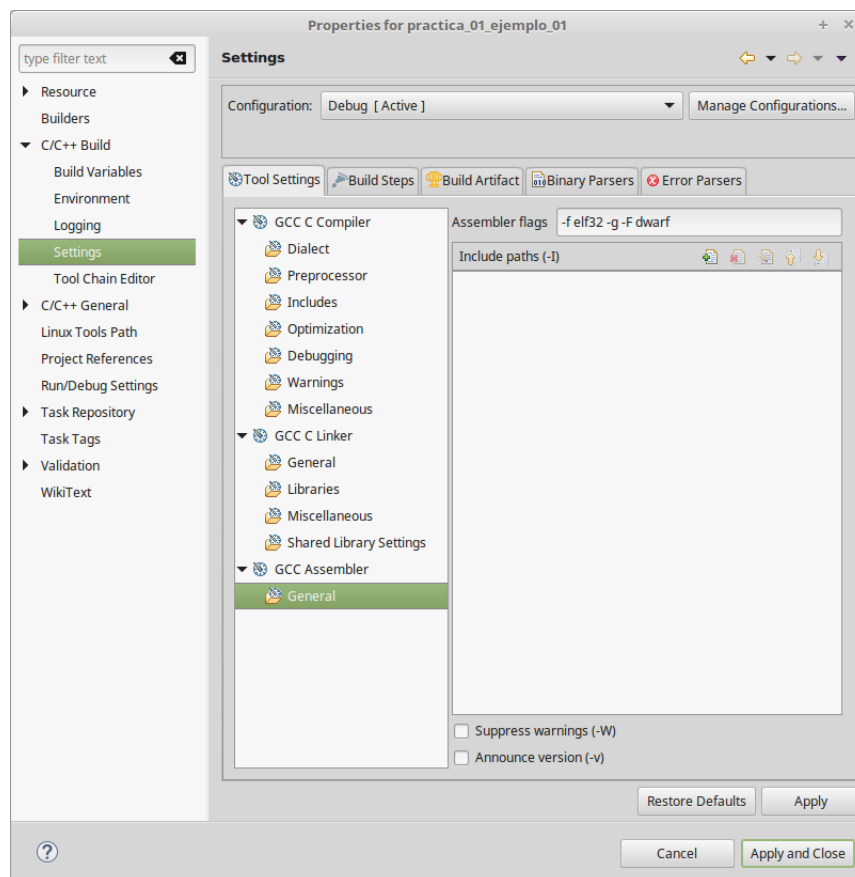


Figura 7: Ajustes de flags para `nasm` en Linux.

- Si estamos usando Windows:

en *Assembler flags* añadimos los siguientes flags:

```
-f win32 -g -F cv8
```

El flag `-f` seguido de `win32` indicará a NASM que genere los ficheros objeto en el formato PE-i386 que es el utilizado por los módulos objeto y ejecutables de 32 bits en Windows. El flag `-g` le dice a NASM que incorpore información de depuración en los ficheros objeto, la cual nos resultará muy útil a la hora de depurar nuestros programas. La opción `-F cv8` hará que la información de depuración esté en formato CodeView 8, que es el único disponible para los módulos `win32`.

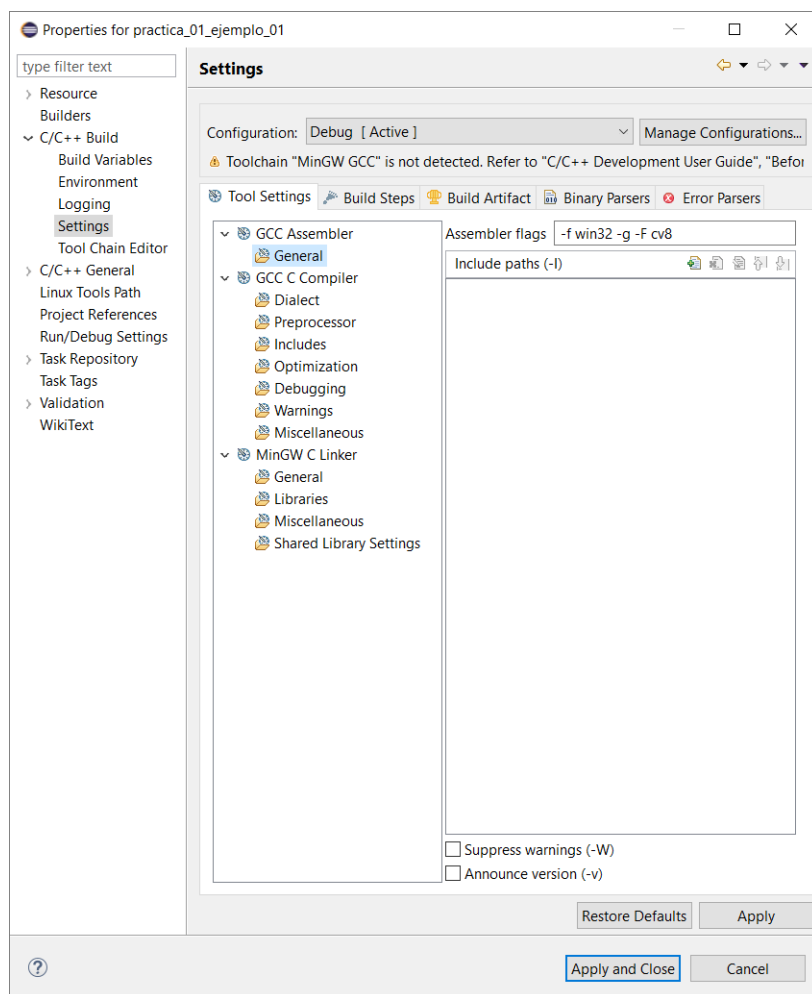



Figura 8: Ajustes de flags para `nasm` en Windows.

Como los primeros programas que desarrollaremos en las prácticas serán aplicaciones de 32 bits, la versión del compilador compatible con GCC instalado en Windows (MinGW o TDM-GCC) deberá ser una que genere código de 32 bits.


3.6. Construcción del proyecto


Para construir el proyecto podemos seleccionar la opción *Project* → *Build All* (atajo CTRL+B) o el correspondiente botón de la barra de herramientas . Si hemos cometido algún error al introducir los ficheros fuentes se indicarán en la ventana *Problems* de

la parte inferior y la línea donde se encuentra el error se marcará con una marca roja ✖. Los avisos (*warnings*) se marcan con un triángulo amarillo ⚠. Haciendo doble clic en un mensaje de error a aviso de la ventana *Problems* el cursor se desplazará a la línea correspondiente del fichero fuente. Si nos detenemos con el puntero del ratón sobre una de las marcas de error o aviso del margen izquierdo, aparecerá una ventana flotante que nos mostrará el mensaje o mensajes de error y/o aviso que hay en esa línea.

4. Ejecución de nuestro programa desde Eclipse

Una vez que hemos construido el proyecto, el ejecutable se encontrará en la carpeta correspondiente a la configuración activa que, en nuestro caso, debe ser *Debug*. Podemos ejecutar nuestro programa llamándolo desde una ventana de terminal pero también podemos ejecutarlo desde el propio Eclipse.

Para ejecutar nuestro programa desde Eclipse debemos seleccionar el proyecto correspondiente en la ventana del explorador de proyectos. A continuación podemos seleccionar la opción *Run* → *Run* (atajo CTRL+F11) o pulsar sobre el icono  de la barra de herramientas. La entrada/salida que el usuario realizaría normalmente en una ventana de terminal del sistema operativo se lleva a cabo mediante la ventana *Console* de la parte inferior. El texto que el programa envíe a *stdout* se mostrará en negro, el que corresponda a la salida a través de *stderr* en rojo mientras que la entrada que el usuario realice a través de *stdin* se mostrará en verde.

Mientras el programa se está ejecutando podemos terminar su ejecución en cualquier momento pulsando el botón *Terminate*  de la ventana de la consola.

Ejecutar el programa y comprobar que funcione correctamente.

5. Depuración de programas desde Eclipse

Una de las características más valiosas del entorno de Eclipse es la integración con el depurador GDB de las herramientas GNU. Gracias a esto tenemos acceso a la mayor parte de las posibilidades de depuración que nos ofrece GDB desde el entorno gráfico de Eclipse. En esta práctica veremos el uso básico de sólo algunas de las muchas herramientas de depuración que tenemos a nuestra disposición. En prácticas posteriores describiremos algunas más.


5.1. Cambiar la sintaxis del desensamblado

Como veremos, durante la depuración Eclipse mostrará un panel en el que aparecerán las instrucciones en ensamblador procedentes del desensamblado del programa en ejecución. Por defecto, este panel muestra las instrucciones usando la sintaxis llamada AT&T, ya que este es también el modo de desensamblado por defecto del depurador GDB. Sin embargo, en la asignatura hemos preferido usar la sintaxis Intel, puesto que coincide con la mayor parte de los manuales y libros y es más fácil de leer. Para cambiar la sintaxis del panel de desensamblado, debemos suministrar a GDB la orden adecuada en un fichero de inicialización, llamado normalmente *.gdbinit*. Usando un editor de texto plano creamos un fichero conteniendo la línea

set disassembly-flavor intel

y lo guardamos con el nombre `.gdbinit` en la carpeta del proyecto (`practica_01_ejemplo_01`). Este fichero será leído automáticamente por GDB cada vez que se inicie la depuración.

5.2. Iniciar la depuración

Para ejecutar nuestro programa en el depurador seleccionamos la opción *Run* → *Debug* (atajo F11) o pulsamos el icono  de la barra de herramientas. Al iniciar una sesión de depuración Eclipse nos pregunta si queremos cambiar a la perspectiva de depuración. Responderemos afirmativamente.

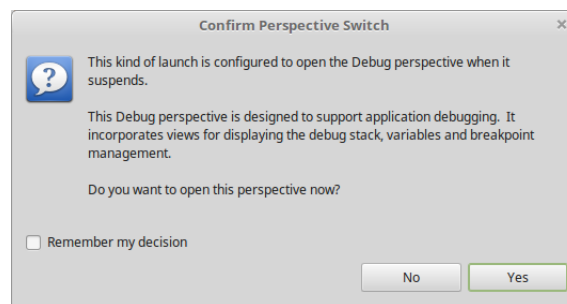


Figura 9: Al iniciar el depurador se nos invita a cambiar a la perspectiva de depuración.

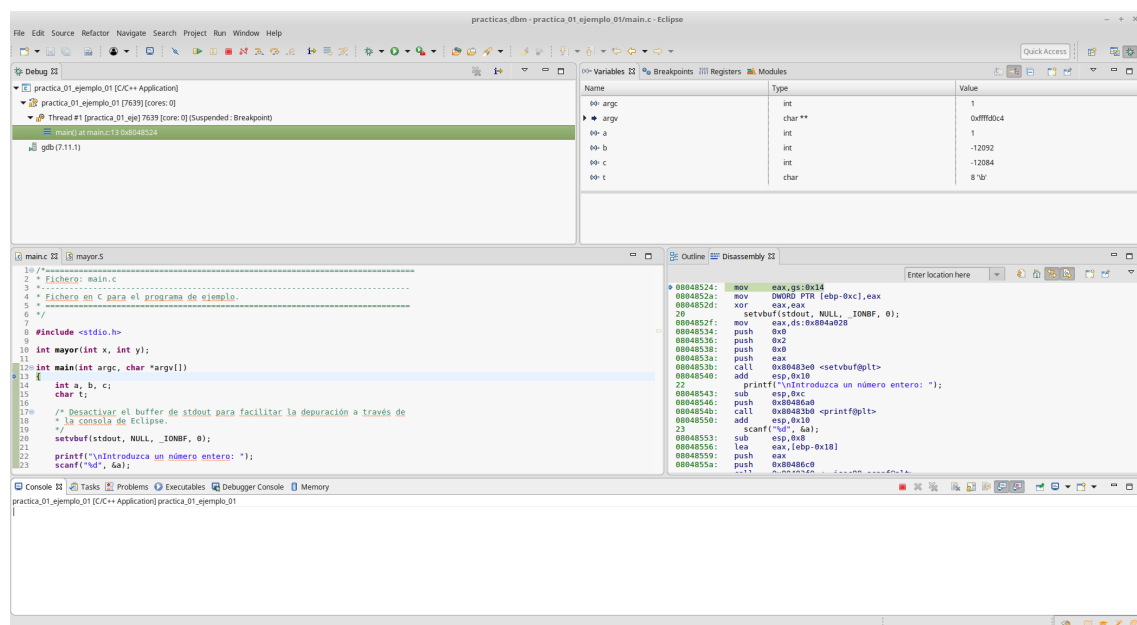


Figura 10: Perspectiva *Debug*.

En la terminología de Eclipse, una perspectiva es un conjunto de ventanas destinadas a permitir al usuario llevar a cabo una serie de acciones relativas a un aspecto particular de la fase de desarrollo. Hasta este momento había estado activa la perspectiva *C/C++* en la que se mostraban las ventanas necesarias para las operaciones de gestión y edición de los proyectos y ficheros fuentes. Por otro lado, la perspectiva de depuración muestra las ventanas que usaremos para llevar a cabo las operaciones de depuración.

5.3. La vista Debug

En la ventana o vista (en la terminología de Eclipse) *Debug* se nos muestran los procesos e hilos lanzados y los marcos de pila en una estructura jerárquica.

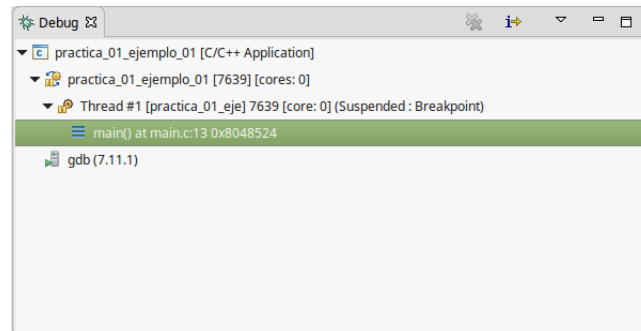


Figura 11: Vista *Debug*.

5.4. La vista Variables

En la vista *Variables* podemos visualizar las variables locales de la función en la que nos encontramos. También podemos cambiar el contenido de cualquier variable pulsando sobre su valor. Podemos añadir variables globales a esta vista pulsando con el botón derecho del ratón sobre ella y seleccionando *Add Global Variables* o pulsando el botón correspondiente de la barra de herramientas de esta vista. También podemos cambiar la base de numeración en la que se muestra el valor de una variable pulsando con el botón derecho del ratón sobre ella y seleccionando *Format*.

Probar a modificar las variables y a visualizarlas en distintos formatos. Explorar otras opciones de la vista de variables consultando la ayuda de Eclipse si es necesario.

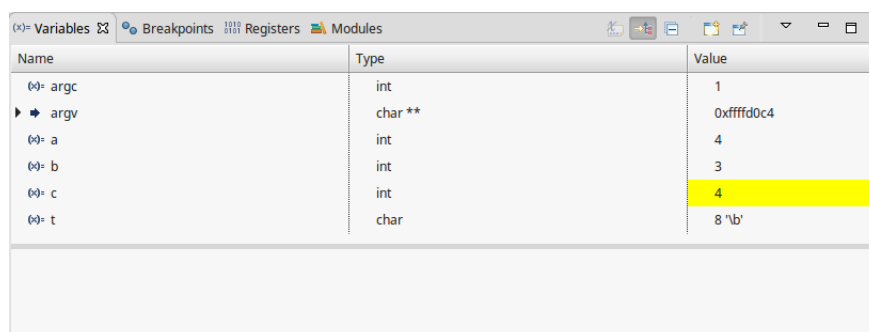


Figura 12: Vista *Variable*.

5.5. La vista Registers

En la vista *Registers* se muestran los registros del microprocesador. En caso de que no se muestre, podemos desplegarla seleccionando *Window → Show View → Registers* en el menú principal. En esta vista también cambiar el valor de cualquiera de ellos pulsando con el botón izquierdo del ratón sobre el valor que queremos cambiar. La base de numeración en

la que se muestra el valor de un registro concreto puede cambiarse también pulsando con el botón derecho y seleccionando *Number Format*.

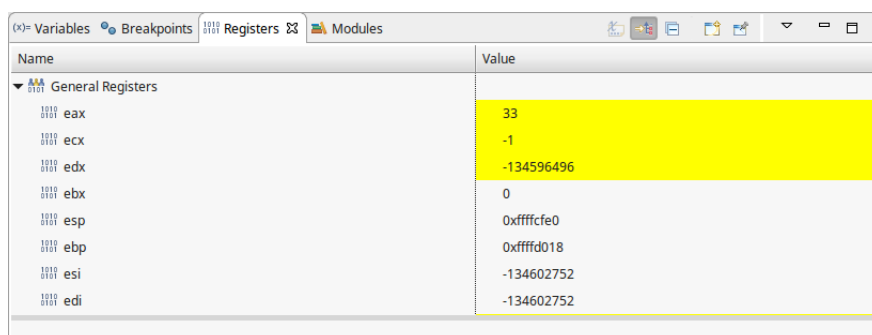


Figura 13: Vista *Registers*.

5.6. La vista Disassembly

La vista *Disassembly* muestra las instrucciones en ensamblador que resultan de desensamblar el código del programa. En caso de que no se muestre, podemos desplegarla seleccionando *Window → Show View → Disassembly* en el menú principal.

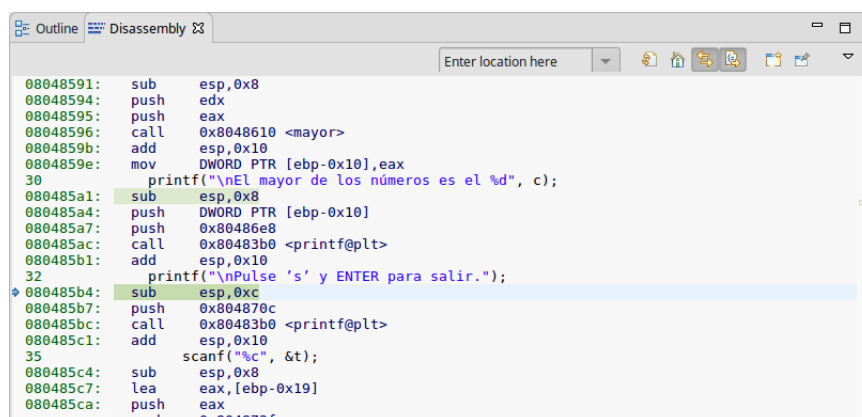


Figura 14: Vista *Disassembly*.

5.7. Barra de herramientas de depuración

En la barra de herramientas tenemos una serie de botones para controlar la ejecución del programa. En el cuadro 1 se indica brevemente la utilidad de cada uno.

Cuadro 1: Botones de la vista *Debug*

Icono	Acción	Descripción
	Remove All Terminated Launches	Limpia todos los procesos terminados de la vista debug.
	Restart	Comienza una nueva sesión de depuración para el proceso seleccionado.
	Suspend	Detiene la ejecución del hilo seleccionado.
	Terminate	Finaliza la sesión de depuración o el proceso seleccionado. La acción concreta depende del elemento seleccionado en la vista debug.
	Resume	Ejecuta el programa normalmente desde la posición actual.
	Disconnect	Desconecta el depurador del proceso seleccionado. Útil cuando el depurador ha sido conectado a un proceso previamente activo.
	Step Into	Ejecuta la línea de programa actual. Si la línea incluye la llamada a una función o subrutina el depurador entra en ella y se detiene.
	Step Over	Ejecuta la línea de programa actual. Si la línea incluye la llamada a una función o subrutina el depurador la ejecuta al completo y nos devuelve el control cuando se retorna a la función llamadora.
	Step Return	Ejecuta hasta retornar de la función o subrutina actual. El depurador se detiene cuando se sale a la función llamadora.
	Drop To Frame	Reintroduce el marco de pila seleccionado en la vista debug.
	Instruction Stepping Mode	Si se activa, el depurador realiza paso a paso a través de las instrucciones individuales en la ventana de desensamblado cuando se usan las órdenes step into o step over.
	Use Step Filters	Se activa para utilizar filtros al hacer paso a paso.

Los botones más útiles son **Suspend**, **Terminate**, **Resume**, **Step Into**, **Step Over** y **Step Return**.

5.8. Puntos de ruptura (*breakpoints*)

Un punto de ruptura o *breakpoint* es una *marca* que colocada en una localización de un programa nos permite pararlo cuando la ejecución alcanza dicha localización. De esta forma podemos ejecutar un programa a su velocidad normal y aún así detenerlo en el lugar o lugares que nos interese. Cuando se alcanza un punto de ruptura la ejecución del programa se detiene y el control vuelve al depurador con lo que podemos visualizar el estado de las variables del programa, los registros o la memoria en ese momento. A partir de ahí podemos ejecutar el programa paso a paso para analizarlo mejor en ese punto o volver a relanzar su ejecución a velocidad normal.

Para colocar un punto de ruptura basta hacer doble clic en la banda de color celeste que hay

en el margen izquierdo de la ventana de fichero fuente en la línea donde deseemos ponerlo. En el margen celeste aparecerá un icono formado por un pequeño círculo azul acompañado de una marca de verificación que nos indicarán la presencia del punto de ruptura en la línea. Para quitar el punto de ruptura podemos hacer doble clic sobre el icono.

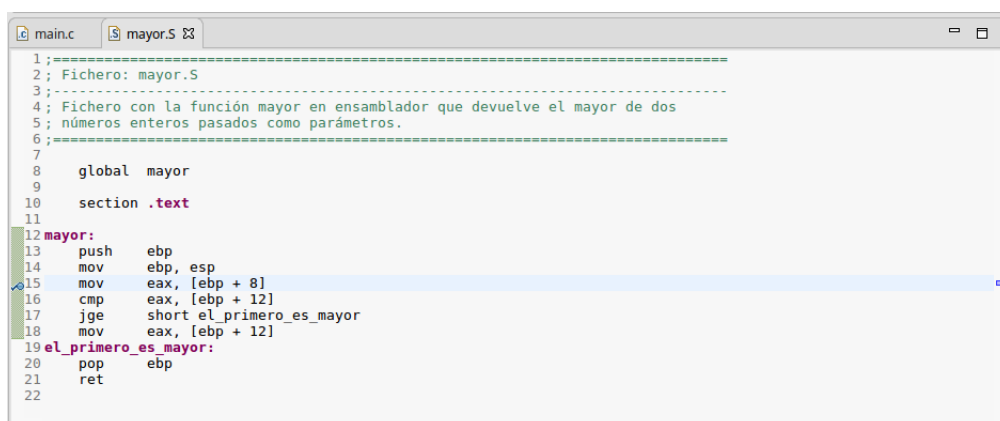


Figura 15: Una marca de *breakpoint* en una línea del un fichero fuente.

Los puntos de ruptura también pueden colocarse en la vista de desensamblado también haciendo doble clic en el margen izquierdo.

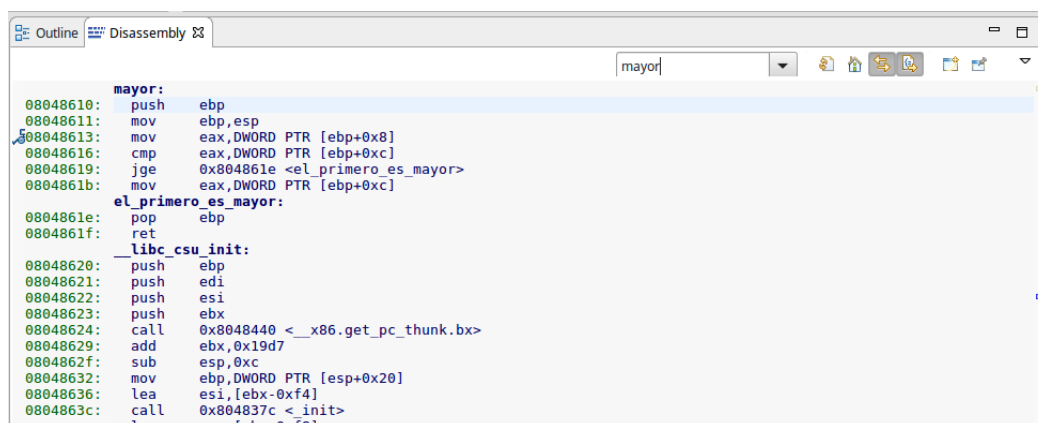


Figura 16: Una marca de *breakpoint* en la vista de desensamblado.

Si deseamos desactivar un punto de ruptura pero pensamos que podemos necesitarlo posteriormente podemos deshabilitarlo. Esto se consigue pulsado con el botón derecho sobre el icono del punto de ruptura y seleccionando la opción *Disable Breakpoint*. De esta forma la marca del punto de ruptura cambia a color blanco indicando que el punto de ruptura está inactivo. Si más tarde queremos volver a habilitarlo pulsaremos con el botón derecho sobre el icono del punto de ruptura deshabilitado y seleccionaremos *Enable Breakpoint*.

5.9. Depuración de funciones en ensamblador

Par poder analizar paso a paso la ejecución de una función en ensamblador colocaremos un punto de ruptura en la misma. Si estamos usando Linux podemos colocar el punto de



ruptura en el fichero .S original que contiene la función. Sin embargo, si estamos usando Windows, el punto de ruptura tiene que colocarse en la ventana de desensamblado.

Inicia la depuración y coloca un punto de ruptura en la instrucción

```
mov     eax, [ebp + 8]
```

Si estás usando Linux, puedes colocar el punto de ruptura en el fichero `mayor.S`.

Sin embargo, si estás en Windows, debes colocar el punto de ruptura en la ventana de desensamblado. Para localizar el código de la función `mayor` en la ventana de desensamblado, introduce `mayor` en el cuadro de búsqueda que tiene en su parte superior.

Ejecuta el programa mediante la opción *Resume*  (F8). La ejecución debería detenerse en la instrucción marcada con el punto de ruptura. A partir de ahí, podemos ejecutar las instrucciones una a una mediante la opción *Step Over* o *Step Into*. En caso de usar Windows, debes asegurarte de que el botón *Instruction Stepping Mode*  está pulsado al hacer ejecución paso a paso de las instrucciones de la función en ensamblador.

5.10. Terminar la sesión de depuración

Para terminar la sesión en cualquier momento pulsaremos el botón *Terminate* de la barra de herramientas (Ctrl+F2). Podemos volver a la perspectiva *C/C++* pulsando el botón con ese nombre que se encuentra en la esquina superior derecha. No olvidar terminar la sesión de depuración con *Terminate* antes de dejar la perspectiva *Debug* porque en caso contrario el proceso de nuestro programa quedará activo y, cuando iniciemos otra sesión de depuración, se lanzará otro que puede entrar en conflicto con el primero.

5.11. La salida estándar y la consola durante la depuración

Cuando se depura en Eclipse un programa que usa la consola para interactuar con el usuario a veces pueden surgir problemas debido a los buffers que utilizan los *streams* estándar *stdin* y *stdout*. El problema afecta sobre todo a *stdout* que tiene asociado un buffer de línea. Esto quiere decir que el buffer de salida de *stdout* se envía a la consola y se vacía sólo cuando imprimimos un carácter nueva línea '\n'. El resultado es que, cuando ejecutamos nuestro programa paso a paso y una función tal como `printf` envía caracteres a la salida estándar, la ventana de consola de Eclipse no nos muestra el resultado inmediatamente a menos que la cadena que imprimimos termine con un carácter nueva línea. Esto puede ser un inconveniente para seguir con comodidad la ejecución del programa en el depurador. Para mejorar la situación hay varias alternativas. La primera consistiría en añadir llamadas a la función `fflush` para vaciar el buffer de *stdout* (`fflush(stdout)`) cuando deseemos que la salida se refleje inmediatamente en la consola pero esto resultaría tedioso. Una solución mejor es desactivar el buffer de salida de *stdout* desde el principio tal como se hace en el programa de ejemplo mediante la función `setvbuf`:

```
setvbuf(stdout, NULL, _IONBF, 0);
```

6. Ejercicios

6.1. Ejercicio 1

Usa la instrucción CMOV para eliminar el salto condicional de la función `mayor`. Comprueba el programa en distintas condiciones para verificar que funciona correctamente.

6.2. Ejercicio 2

Consulta el manual de instrucciones para aprender el funcionamiento de la instrucción de multiplicación con signo IMUL.

Escribe una función en ensamblador que sirva para calcular el determinante de una matriz 2×2 cuyos elementos son datos de tipo `int`. Respecto al compilador de C el prototipo de la función será

```
int det(int a11, int a12, int a21, int a22);
```

Escribe un pequeño programa en C que llame a la función `det` para ponerla a prueba.

6.3. Ejercicio 3

Aprende el funcionamiento de las instrucciones lógicas AND, OR, XOR y NOT.

Crea una función en ensamblador que reciba dos argumentos de tipo `unsigned int` y devuelva como resultado un dato también de tipo `unsigned int`. Los 8 bits más significativos del valor devuelto deben ser iguales a los 8 bits más significativos del primer argumento, mientras que los 24 bits menos significativos del valor devuelto deben ser iguales a los 24 bits menos significativos del segundo argumento. Escribe un pequeño programa en C para probar la función en ensamblador y comprobar que funciona correctamente.