

# Diseño Basado en Microprocesadores

## Práctica 2

### Programación en ensamblador x86 de 32 bits (II)

---

## Índice

1. Objetivos	1
2. Las vistas de memoria de eclipse	1
3. Ejercicios	4
3.1. Ejercicio 1 . . . . .	4
3.2. Ejercicio 2 . . . . .	4
3.3. Ejercicio 3 . . . . .	5

---

## 1. Objetivos

En esta práctica se pondrán en marcha nuevas instrucciones y se usarán facilidades de depuración de la plataforma eclipse adicionales a las presentadas en la práctica anterior.

## 2. Las vistas de memoria de eclipse

Cuando estamos depurando un programa en la perspectiva *Debug* Eclipse proporciona dos vistas diferentes para visualizar y modificar el contenido de la memoria: la “vista de memoria” y la “vista de explorador de memoria” (*Memory browser*). La “vista de memoria” está presente desde las primeras versiones de Eclipse CDT y es más compleja que la “vista de explorador de memoria”. La “vista de explorador de memoria” es más básica pero también más fácil de usar. En cierta medida son redundantes y quizás se unifiquen en una versión futura de Eclipse.

La vista de memoria de eclipse (ver figura 1) permite visualizar el contenido de la memoria en distintos formatos así como modificar su contenido. La vista de memoria se muestra normalmente en la parte inferior de la perspectiva *Debug*, la perspectiva que activamos cuando realizamos una sesión de depuración. En caso de que la vista de memoria no sea visible podemos activarla seleccionando *Window → Show View → Memory*.

La vista de memoria está dividida en una serie de paneles dispuestos horizontalmente. A la izquierda tenemos el panel de monitores de memoria o *Monitors*. Un monitor de memoria

nos permite visualizar una zona de memoria del proceso que se está depurando especificando la dirección de comienzo del bloque. El contenido del bloque de memoria definido por cada monitor de memoria se visualiza a la derecha en los paneles de volcado de memoria o *Renderings*. Cada *Rendering* puede visualizarse en diferentes formatos: hexadecimal, ascii, entero con signo y entero sin signo. El formato por defecto es hexadecimal.

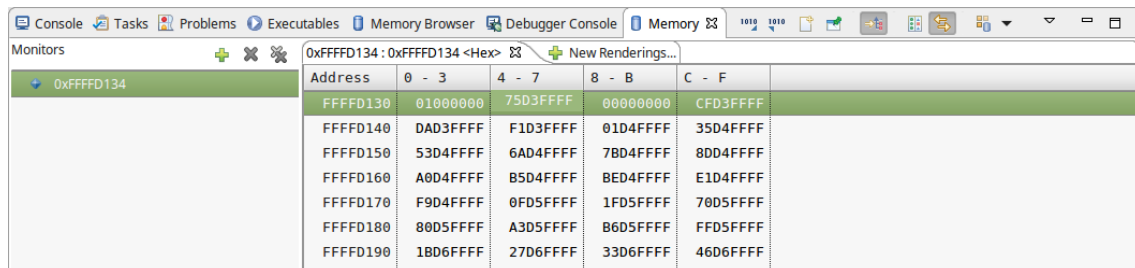


Figura 1: La vista de memoria de eclipse.

Para añadir un monitor de memoria pulsaremos el botón . Aparecerá una ventana en donde podemos introducir la dirección a partir de la cual queremos visualizar la memoria (ver figura 2). Podemos indicar la dirección introduciendo directamente el número de dirección de memoria en decimal o en hexadecimal (precediéndola de 0x), por ejemplo 0xFF9C5218, pero también podemos especificarla mediante una expresión que será evaluada para obtener la dirección. Por ejemplo, si estamos depurando dentro de una función en C que contiene una variable local llamada `var` podemos visualizar las posiciones de memoria en la que está almacenada y siguientes introduciendo la expresión `&var` que, como sabemos, significa dirección de `var`. Puede proporcionarse cualquier expresión que se evalúe a una dirección válida para el proceso que estamos depurando.

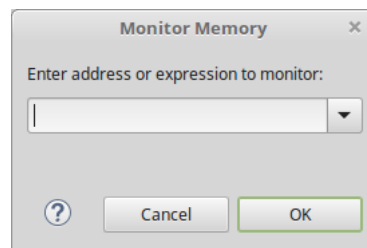


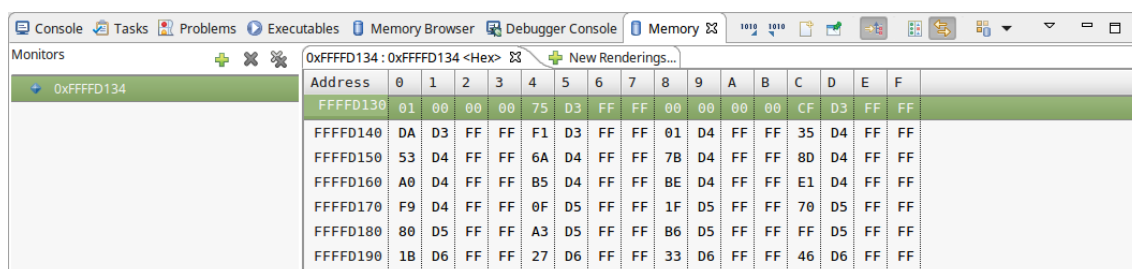
Figura 2: Al añadir un monitor de memoria debemos dar la dirección de comienzo.

En la expresión también pueden aparecer uno o más registros del microprocesador, de forma que sus valores determinen la dirección a mostrar. Esto resulta muy útil cuando en el programa un registro se está usando como puntero y queremos visualizar la memoria a partir de la dirección señalada por el registro. En caso de querer usar el nombre de un registro para especificar la dirección de memoria hay que tener en cuenta que el depurador GDB, que es el que internamente va a procesar la expresión, necesita que los nombres de los registros se escriban precedidos de un símbolo dólar '\$'. Por ejemplo, si en un momento dado el registro `esi` se está usando como puntero, podemos introducir '\$esi' como expresión para que se realice un volcado de memoria a partir de la dirección de memoria apuntada por `esi`.

Hay que tener en cuenta que la expresión se evalúa sólo en el momento de introducirla para determinar la dirección de comienzo del volcado de memoria. Si el valor de la expresión

cambia después de ese momento, el volcado de memoria no se adaptará a los cambios, sino que seguirá mostrando la memoria a partir de la dirección determinada en un primer momento (salvo los desplazamientos que el usuario realice con las barras de desplazamiento del panel *Rendering*, lógicamente).


Cuando añadimos un monitor de memoria el volcado se realiza inicialmente en hexadecimal y organizado en cuatro columnas de cuatro bytes cada una (16 bytes en total en cada fila), tal como se mostró en la figura 1. Un detalle muy importante del volcado hexadecimal es que, a pesar de este agrupamiento de cuatro en cuatro, lo que vemos es una imagen bruta de la memoria y no valores de 32 bits obtenidos teniendo en cuenta la ordenación *little endian* de la familia x86. Podemos cambiar el formato del volcado hexadecimal pulsando con el botón derecho del ratón sobre el panel de *Rendering* y seleccionando *Format...*. La configuración seleccionada puede ser salvada para ser usada por defecto. La figura 3 muestra el resultado de organizar un panel de *Rendering* en 16 columnas de 1 byte.





Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FFFFD130	01	00	00	00	75	D3	FF	FF	00	00	00	00	CF	D3	FF	FF
FFFFD140	DA	D3	FF	FF	F1	D3	FF	FF	01	D4	FF	FF	35	D4	FF	FF
FFFFD150	53	D4	FF	FF	6A	D4	FF	FF	7B	D4	FF	FF	8D	D4	FF	FF
FFFFD160	A0	D4	FF	FF	B5	D4	FF	FF	BE	D4	FF	FF	E1	D4	FF	FF
FFFFD170	F9	D4	FF	FF	0F	D5	FF	FF	1F	D5	FF	FF	70	D5	FF	FF
FFFFD180	80	D5	FF	FF	A3	D5	FF	FF	B6	D5	FF	FF	FF	D5	FF	FF
FFFFD190	1B	D6	FF	FF	27	D6	FF	FF	33	D6	FF	FF	46	D6	FF	FF

Figura 3: El volcado hexadecimal organizado en 16 columnas de un byte.

Una vez que hemos desplegado un volcado o *Rendering* de memoria en hexadecimal podemos añadir nuevos *Renderings* para visualizar los datos de otra forma. Para ello pulsamos sobre la ficha **+ New Rendering...** y a continuación seleccionamos el tipo de volcado que queremos añadir, que puede ser un nuevo volcado hexadecimal, un volcado ascii, un volcado como enteros con signo o como enteros sin signo. Si seleccionamos el volcado ascii, el contenido de la memoria será interpretado como códigos ascii y se mostrarán los caracteres correspondientes. Si elegimos un volcado como enteros con signo el contenido de la memoria se tomará en grupos de cuatro bytes que se interpretarán como datos de 32 bits con signo almacenados en *little endian*. Análogamente, si elegimos un volcado como enteros sin signo el contenido de la memoria se tomará en grupos de cuatro bytes que se interpretarán como datos de 32 bits sin signo almacenados en orden *little endian*. Por tanto, al hacer el volcado como enteros con o sin signo sí se tiene en cuenta la ordenación *little endian* en memoria de los múltiples bytes de los datos.

Pulsando el botón *Toggle Split Pane*  podemos dividir el panel derecho en dos *Renderings* diferentes, lo que permite ver simultáneamente un mismo monitor de memoria bajo dos formatos distintos.

Además de visualizar la memoria, la vista de memoria también permite alterar el contenido de la misma. Para ello, basta con hacer doble clic sobre el dato que queremos cambiar e introducir el nuevo valor (precedido de 0x si optamos por expresarlo en hexadecimal).

Mediante el botón  podemos quitar el monitor de memoria seleccionado mientras que con el botón  se eliminan todos los monitores de memoria.

### 3. Ejercicios

#### 3.1. Ejercicio 1

Crea una función en ensamblador que a partir de un dato de tipo `unsigned int` permita obtener una cadena de caracteres que lo represente en binario. El prototipo de la función será:

```
char * uint_a_cadena_binario(unsigned int dato, char * buffer);
```

donde `dato` es el valor a convertir y `buffer` señala a la dirección de memoria a partir de la cual debe almacenarse la cadena. La función devuelve a través de su nombre el propio puntero `buffer` que se recibe como argumento.

Genera siempre una cadena de 32 caracteres correspondientes a los 32 bits del dato de entrada.

No olvides situar el terminador nulo al final de la cadena.

Crea un pequeño programa en C para poner a prueba la función.

Con la ayuda del depurador, realiza la ejecución instrucción a instrucción de la función `uint_a_cadena_binario`. Después de que se haya ejecutado el código de prólogo de la función, usa la vista de memoria para mostrar las posiciones de memoria a partir de la cima de la pila y localiza la ubicación y valores de los argumentos de la función, la dirección de retorno y el valor de EBP salvado. En otra pestaña de “monitor de memoria” muestra la zona de memoria en la que se creará la cadena de caracteres y observa el proceso de creación de la misma a medida que ejecutas la función paso a paso.

#### 3.2. Ejercicio 2

Aprende el funcionamiento de la instrucción de división con signo `IDIV`. Usa esta instrucción junto con las demás que ya conoces para escribir una función en ensamblador que permita realizar la conversión de grados Celsius a Fahrenheit o viceversa. Las expresiones de conversión son:

$$T_{(^{\circ}F)} = \frac{9T_{(^{\circ}C)}}{5} + 32 \quad ; \quad T_{(^{\circ}C)} = \frac{5(T_{(^{\circ}F)} - 32)}{9}$$

El prototipo de la función es

```
int convertir_celcius_fahrenheit(int temperatura_entrada,  
                                int sentido_conversion,  
                                int *temperatura_salida);
```

donde

`temperatura_entrada`: es la temperatura de partida, que puede estar en grados Celsius o Fahrenheit.

`sentido_conversion`: si vale 0 indica que la temperatura de entrada está en grados Celsius. En caso contrario, la temperatura de entrada está en grados Fahrenheit.

`temperatura_salida`: apunta a la variable donde almacenar el resultado de la conversión. En caso de que el resultado de aplicar las formulas de conversión tenga una parte fraccionaria, basta **truncar** el resultado hacia cero.

Prueba la función con temperaturas de entrada tanto positivas como negativas y que generen temperaturas de salida tanto positivas como negativas.

Mejora la función para que la temperatura resultante esté **redondeada** al número entero de grados más próximo al verdadero resultado.

### 3.3. Ejercicio 3

Aprende el funcionamiento de las instrucciones MOVSB/MOVSX/MOVSQ y del prefijo de repetición REP.

Escribe una función en ensamblador que sirva para copiar un bloque de memoria desde su lugar de origen en otro lugar en la memoria (similar a la función `memcpy` de la biblioteca estándar de C). El prototipo de la función será:

```
void * copiar_bloque_memoria(void * destino,  
                             void * origen,  
                             unsigned int longitud);
```

donde

`destino` es un puntero al comienzo de la zona de memoria en donde hay que copiar los datos.

`origen` es un puntero al comienzo de la zona de memoria que contiene los datos a copiar.

`longitud` es el tamaño en bytes del bloque a copiar.

La función retorna el puntero al destino.

La copia debe realizarse correctamente aunque exista solapamiento parcial entre los bloques origen y destino y sin importar si el bloque origen está antes o después que el destino en la memoria.

Escribe un programa en C para comprobar que la función realiza correctamente su trabajo. Usa la vista de memoria del depurador para comprobar el funcionamiento. Comprueba que la función trabaja correctamente en todos los casos.