

Sesión de depuración saludo.s

1. ¿Qué contiene EDX tras ejecutar `mov longsaludo, %edx`? ¿Para qué necesitamos esa instrucción, o ese valor? Responder no sólo el valor concreto (en decimal y hex) sino también el significado o del mismo (¿de dónde sale?) Comprobar que se corresponden los valores hexadecimal y decimal mostrados en la ventana Status->Registers.

```

1: $edx
28

.section .data
saludo:
    .ascii "Hola a todos!\nHello, World!\n"
longsaludo:
    .int    .-saludo

.section .text
.global _start
_start:
    mov $4, %eax
    mov $1, %ebx
    mov $saludo, %ecx
    mov longsaludo, %edx
    int $0x80
  
```

eax	0x4	4
ecx	0x80490db	134516955
edx	0x1c	28
ebx	0x1	1

Necesitamos esa instrucción para saber el tamaño de la cadena a mostrar por la salida estandar. Ese valor proviene de la definición de `longsaludo`, la cual indica que contendrá un entero que ira desde la posición de memoria actual “.” - la dirección de comienzo de `saludo`, con lo que obtenemos el tamaño de la cadena `saludo`.

2. ¿Qué contiene ECX tras ejecutar “`mov $saludo, %ecx`”? Indicar el valor en hexadecimal, y el significado del mismo. Realizar un dibujo a escala de la memoria del programa, indicando dónde empieza el programa (`_start`, `.text`), dónde empieza `saludo` (`.data`), y dónde está el tope de pila (`%esp`).

2: \$ecx
134516955

```

.section .data
saludo:
    .ascii "Hola a todos!\nHello, World!\n"
longsaludo:
    .int    .-saludo

.section .text
.global _start
_start:
    mov $4, %eax
    mov $1, %ebx
    mov $saludo, %ecx
    mov longsaludo, %edx
    int $0x80
  
```

DDD: Registers

Registers		
eax	0x4	4
ecx	0x80490db	134516955
edx	0x0	0
ebx	0x1	1
esp	0xffffd120	0xffffd120
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80480c7	0x80480c7 <_start+15>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

3. ¿Qué sucede si se elimina el símbolo de dato inmediato (\$) de la instrucción anterior? (*mov saludo, %ecx*) Realizar la modificación, indicar el contenido de ECX en hexadecimal, explicar por qué no es lo mismo en ambos casos. Concretar de dónde viene el nuevo valor (obtenido sin usar \$).

The screenshot shows a debugger window with two tabs at the top: '1: \$ecx' and '2: saludo'. Both show the value 1634496328. Below is the assembly code:

```
.section .data
saludo:
    .ascii "Hola a todos!\nHello, World!\n"

longsaludo:
    .int    .-saludo

.section .text
.global _start

_start:
    mov $4, %eax
    mov $1, %ebx
    mov saludo, %ecx
    mov longsaludo, %edx
    int $0x80
```

On the right, the 'DDD: Registers' window shows the state of the registers:

Registers	0x4	4
eax	0x4	4
ecx	0x616c6f48	1634496328
edx	0x0	0
ebx	0x1	1
esp	0xffffd120	0xffffd120
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80480c8	0x80480c8 <_start+16>
eflags	0x202	[IF 1
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

At the bottom, there are checkboxes for 'Integer registers' and 'All registers'.

Lo que sucede es que cuando ponemos \$ mueve la dirección de memoria en donde está almacenado *saludo* a *%ecx* y cuando no lo ponemos se interpreta el contenido de *saludo* como un entero y se mandan los primeros 32 bits de la frase “*Hola a todos!\nHello, World!\n*” al registro *%ecx*, que en este caso concreto almacenaría las primeras 4 letras ‘*Hola*’ y según la tabla ASCII proporcionada en el Apéndice 1 corresponde a 0x616c6f48, siendo 48 la ‘*H*’.

4. ¿Cuántas posiciones de memoria ocupa la variable *longsaludo*? ¿Y la variable *saludo*? ¿Cuántos bytes ocupa por tanto la sección de datos? Comprobar con un volcado *Data-> Memory* mayor que la zona de datos antes de hacer *Run*.

The screenshot shows a debugger window with a tab '1: longsaludo' showing the value 28. Below is the assembly code:

```
.section .data
saludo:
    .ascii "Hola a todos!\nHello, World!\n"

longsaludo:
    .int    .-saludo

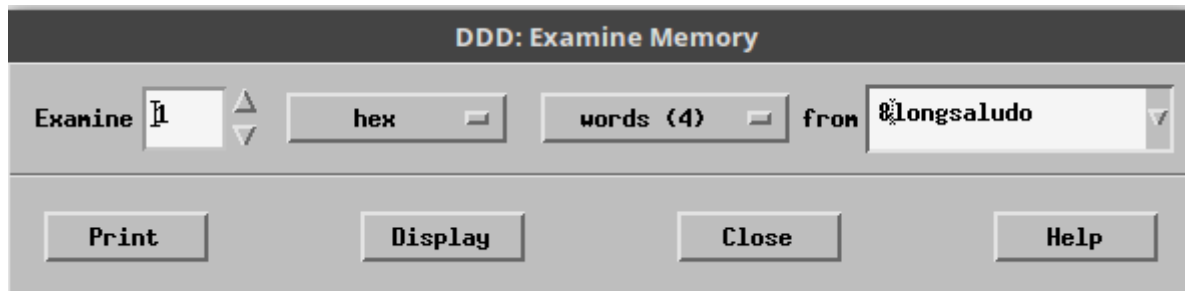
.section .text
.global _start

_start:
    mov $4, %eax
    mov $1, %ebx
    mov saludo, %ecx
    mov longsaludo, %edx
    int $0x80

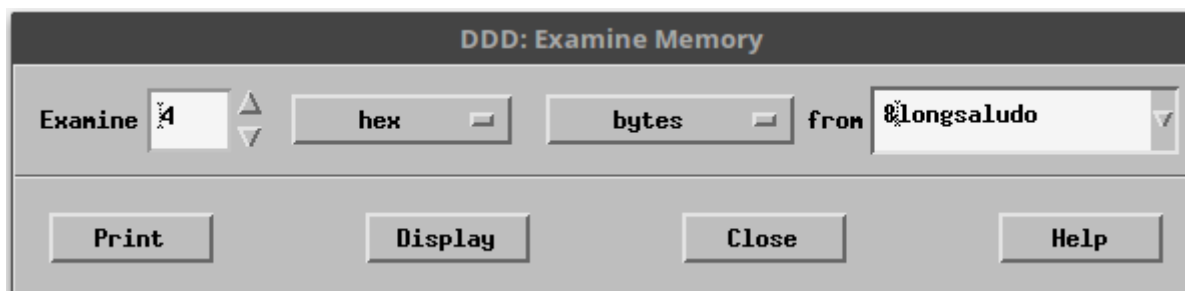
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

La variable *longsaludo* ocupa 4 bytes en memoria ya que contiene un *int* y este ocupa 32 bits = 4 bytes. Y la variable *saludo* ocupa 28 bytes (1 byte por carácter) ya que es el valor que se calcula en *longsaludo*.

5. Añadir dos volcados Data->Memory de la variable `longsaludo`, uno como entero hexadecimal, y otro como 4 bytes hex. Teniendo en cuenta lo mostrado en esos volcados... ¿Qué direcciones de memoria ocupa `longsaludo`? ¿Cuál byte está en la primera posición, el más o el menos significativo? ¿Los procesadores de la línea x86 usan el criterio del extremo mayor (big-endian) o menor (little-endian)? Razonar la respuesta.



```
(gdb) x /1xw &longsaludo
0x80490f7: 0x0000001c
```



```
(gdb) x /4xb &longsaludo
0x80490f7: 0x1c 0x00 0x00 0x00
```

`Longsaludo` ocupa la dirección de memoria `0x80490f7`. El byte más significativo es `0x00` y está en la última posición. Utilizan el criterio de *little-endian*.

6. ¿Cuántas posiciones de memoria ocupa la instrucción `mov $1, %ebx`? ¿Cómo se ha obtenido esa información? Indicar las posiciones concretas en hexadecimal.

```
0x080480bd <+5>:  mov    $0x1,%ebx
0x080480c2 <+10>:  mov    $0x80490db,%ecx
```

A través del ddd mostrando la ventana de código máquina podemos obtener que la instrucción ocupa 5 Bytes.

7. ¿Qué sucede si se elimina del programa la primera instrucción `int 0x80`? ¿Y si se elimina la segunda? Razonar las respuestas.

Si se elimina la primera instrucción `0x80` lo que ocurriría es que no se haría la llamada al sistema para mostrar el mensaje de "Hola a todos!\nHello, World!\n" por la salida estándar y directamente finalizaría el programa al ejecutarse la segunda llamada al sistema.

Y si quitasemos la segunda instrucción de 0x80 lo que ocurriría sería un *segmentation fault* debido a que no se efectúa la llamada al sistema para salir con el valor de retorno deseado.

```
juanka1995@juanka-desktop ~/practicass/ec/2 Practica 2 $ ./saludo
Segmentation fault
```

8. ¿Cuál es el número de la llamada al sistema READ (en kernel Linux 32bits)? ¿De dónde se ha obtenido esa información?

El número de la llamada al sistema READ es el 0x03. Esta información la he obtenido de la siguiente url:

<http://syscalls.kernelgrok.com/>