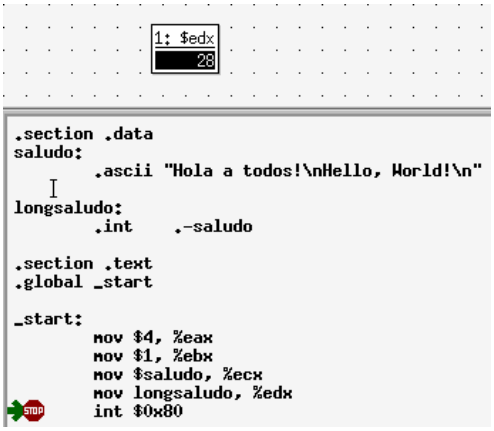
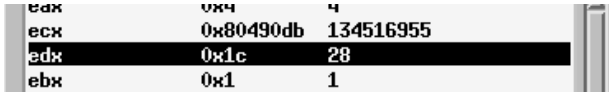
Sesión de depuración saludo.s

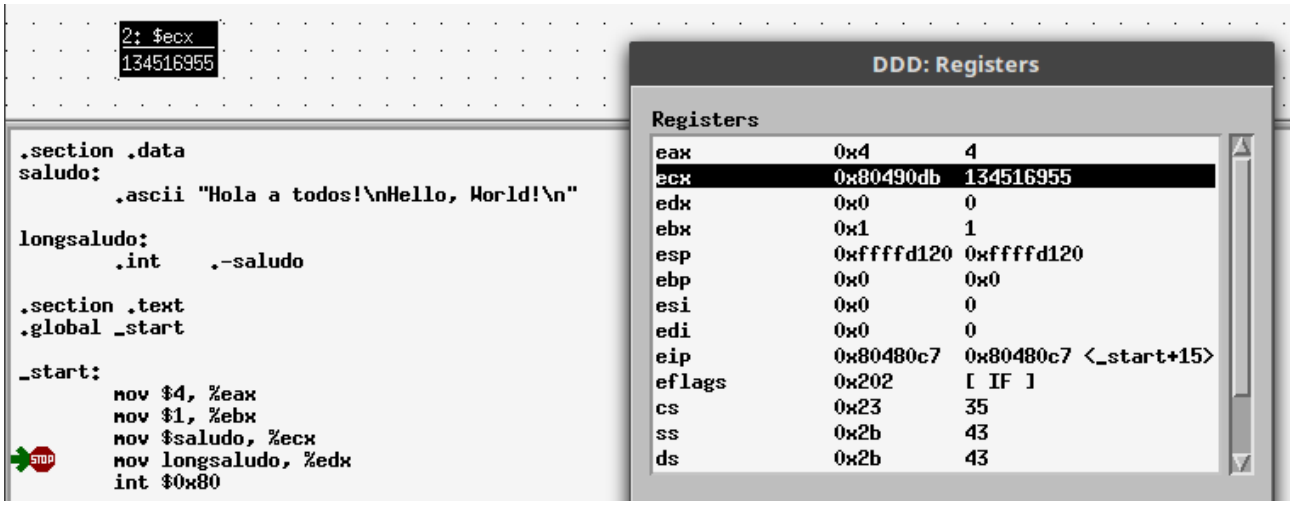
1. ¿Qué contiene EDX tras ejecutar mov longsaludo, %edx? ¿Para qué necesitamos esa instrucción, o ese valor? Responder no sólo el valor concreto (en decimal y hex) sino también el significado o del mismo (¿de dónde sale?) Comprobar que se corresponden los valores hexadecimal y decimal mostrad dos en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



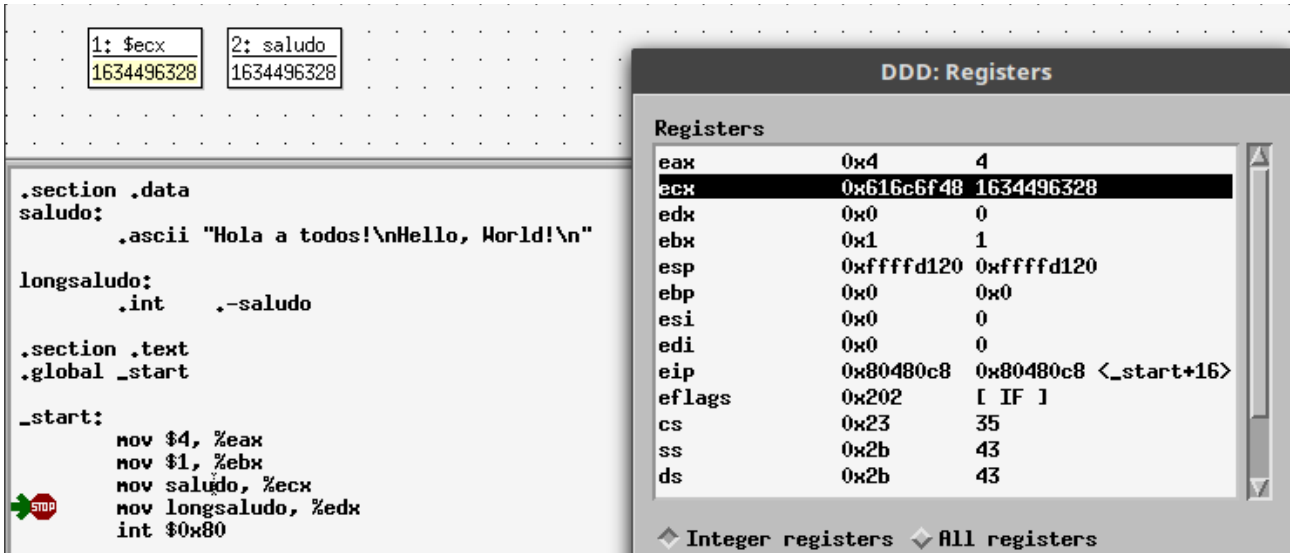


Necesitamos esa instrucción para saber el tamaño de la cadena a mostrar por la salida estandar. Ese valor proviene de la definición de longsaludo, la cual indica que contendra un entero que ira desde la posicion de memoria actual "." - la direccion de comienzo de *saludo*, con lo que obtenemos el tamaño 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).

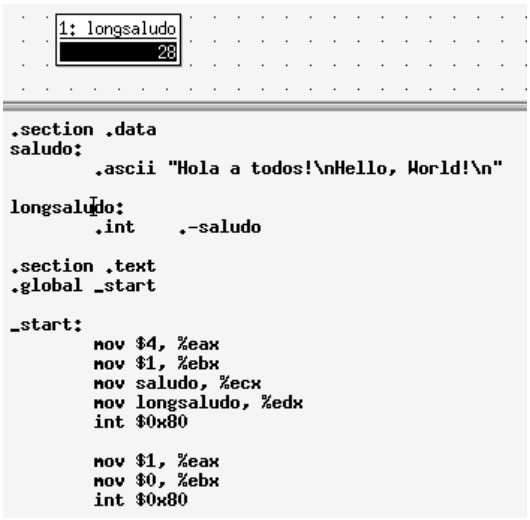


3. ¿Qué sucede si se elimina el símbolo de dato inmediato ($) de la instrucción anterior? (mov saludo, %ecx) Realizar la modificación, indicar el contenido de ECX en hexadecimal, explicar por qué no o es lo mismo en ambos casos. Concretar de dónde viene el nuevo valor (obtenido sin usar $).



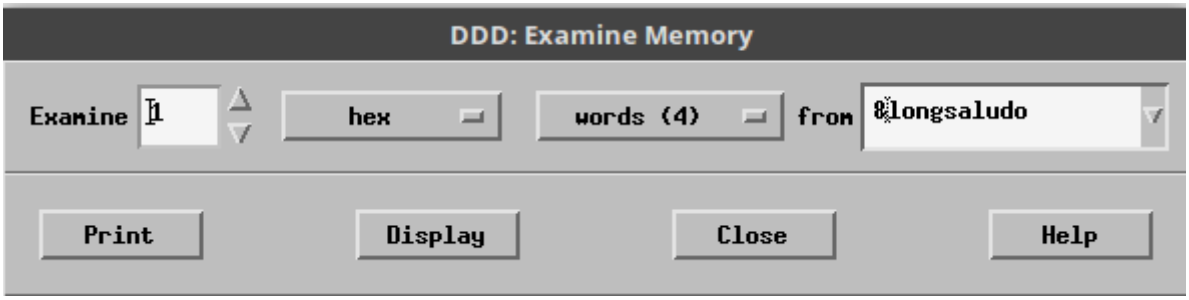
Lo que sucede es que cuando ponemos $ mueve la direccion de memoria en donde esta almacenado *saludo* a %ecx y cuando no lo ponemos se interpreta el contenido de *saludo* como un entero y se mandan *los primeros 32 bits de la frase "Hola a todos!\nHello, World!\n" al registro %ecx, que este caso concreto almacenaría las primeras 4 letras 'Hola' y según la tabla ASCII proporcionada en el Apendice 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.

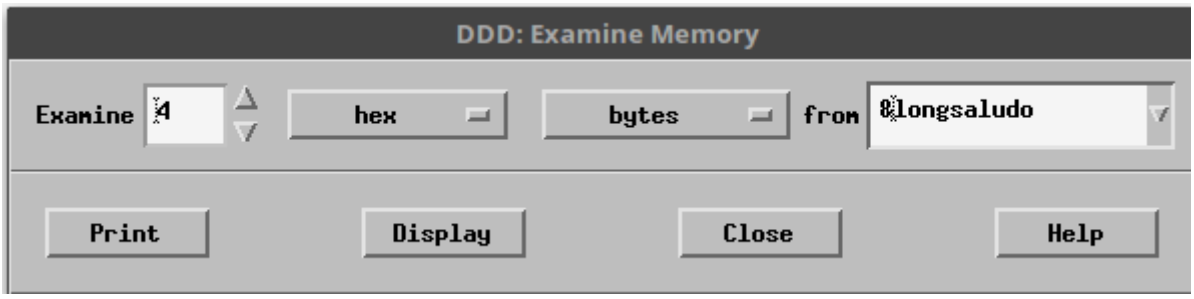


La variable *longsaludo* ocupa 4 bytes en memoria ya ya que contiene un *int* y este ocupa 32 bits = 4 bytes. Y la variable *saludo* ocupa 28 bytes (1 byte por carácter) ya que es el valor que se calcula en *longsaludo*.

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 direccion de memoria 0x80490f7. El byte mas significativo es 0x00 y esta en la ultima 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.

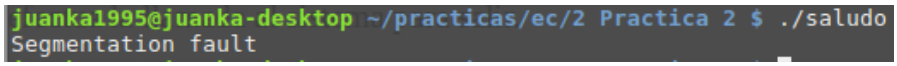
```
0x080480bd <+5>:     mov    $0x1,%ebx
0x080480c2 <+10>:    mov    $0x80490db,%ecx
```

A traves del ddd mostrando la ventana de código maquina 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 ocurrira es que no se haría la llamada al sistema para mostrar el mensaje de "Hola a todos!\nHello, World!\n" por la salida estandar y directamente finalizaria el programa al ejecutarse la segunda llamada al sistema.

Y si quitasemos la segunda instrucción de 0x80 lo que ocurrira seria un *segmetation fault* debido a que no se efectua la llamada al sistema para salir con el valor de retorno deseado.



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

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

http://syscalls.kernelgrok.com/