PRÁCTICA 2: REPRESENTACIÓN DE LOS DATOS

DATOS_SIZEGDB.TXT:

```
+x /t &da1  mostrar dirección de da1 en binario
0x56557000: 00001101000010100000101100001010
+x /x &da1 mostrar dirección de da1 en hexadecimal
0x56557000: 0x0d0a0b0a
+x /1xb &da1 mostrar dirección primer byte de da1 en hexadecimal (Little Endian)
0x56557000: 0x0a
+focus src mostrar pantalla de arriba
Focus set to src window.
+focus cmd mostrar pantalla de abajo
Focus set to cmd window.
+x /1xw &da4   mostrar dirección en 4 bytes (word) de da4 en hexadecimal
0x56557003: 0x0a0b0c0d
+x /20xb &da1 mostrar dirección en 20 bytes de da1 en hexadecimal
0x56557000: 0x0a   0x0b   0x0a   0x0d   0x0c   0x0b   0x0a   0x68
0x56557008: 0x6f   0x6c   0x61   0x01   0x00   0x00   0x00   0x02
0x56557010: 0x00   0x00   0x00   0x03
+x /1xh &da2 mostrar dirección en 2 bytes (half word) de da2 en hexadecimal
0x56557001: 0x0a0b
+x /5cb &men1 mostrar dirección de 5 bytes de men1 en char
0x56557007: 104 'h' 111 'o' 108 'l'  97 'a'   1 '\001'
+p /s (char *) &men1 mostrar contenido de la string de men1 en char
$1 = 0x56557007 "hola\001"
+x /5xb &da4 mostrar dirección de 5 bytes de da4 en hexadecimal
0x56557003: 0x0d   0x0c   0x0b   0x0a   0x68
+x /5xw &da4 mostrar dirección de 5 palabras (4 bytes) de da4
0x56557003: 0x0a0b0c0d   0x616c6f68   0x00000001   0x00000002
0x56557013: 0x00000003
+p /a &lista imprimir adress en hexadecimal de lista
$2 = 0x5655700b
+p &_start imprimir dirección etiqueta _start
$3 = (<text variable, no debug info> *) 0x56555175 <_start>
+x /i &_start mostrar instrucción de la etiqueta _start
=> 0x56555175 <_start>:  mov    $0x1,%eax
+disas /r _start mostrar intruscciones de la etiqueta _start y sus contenidos
Dump of assembler code for function _start:
=> 0x56555175 <+0>:     b8 01 00 00 00       mov    $0x1,%eax
   0x5655517a <+5>:     bb 00 00 00 00       mov    $0x0,%ebx
End of assembler dump.
+x /x &lista + 1
Cannot perform pointer math on incomplete type "<data variable, no debug info>", try casting
to a known type, or void *.
+x /x &lista+1
Cannot perform pointer math on incomplete type "<data variable, no debug info>", try casting
to a known type, or void *.
+p /x (&lista+1) imprimir contenido segundo elemento de lista en hexadecimal
Cannot perform pointer math on incomplete type "<data variable, no debug info>", try casting
to a known type, or void *.
+p /a (void *)&lista+1  imprimir dirección segundo elemento de lista
$4 = 0x5655700c
+p /a (int*)&lista+1 imprimir dirección del segundo elemento de lista en hexadecimal
$5 = 0x5655700f
+p (int[5])lista imprimir 5 string de enteros de lista
$6 = {1, 2, 3, 4, 5}
+p /x *(&lista+1) imprimir en hexadecimal el segundo elemento de lista
```

Eduardo Ezponda

Cannot perform pointer math on incomplete type "<data variable, no debug info>", try casting to a known type, or void *.
+p *((int*)&lista+1) imprimir en int el contenido del segundo elemento de lista
$7 = 2
+x /dw (int *) &lista+1 mostrar contenido en signed decimal del segundo elemento de lista
0x5655700f: 2
+p * (int*)&lista@5 imprimir contenido (5 enteros) en int de lista
$8 = {1, 2, 3, 4, 5}
+exit

DATOS_DIRECCIONAMIENTOGDB.TXT:

+layout regs mostrar el contenido de los registros
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
bucle () at datos_direccionamiento.s:40
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+x /a &buffer mostrar dirección de buffer
0x56559020: 0xa0b
+focus cmd mostrar pantalla abajo
Focus set to cmd window.
+x /2xw &buffer mostrar dirección de 2 words (4 bytes) de buffer en hexadecimal
0x56559020: 0x00000a0b 0x00000000
+x /2xh &buffer mostrar dirección de 2 half words (2 bytes) de buffer en hexadecimal
0x56559020: 0x0a0b 0x0000
+x /xh &buffer mostrar direccion de half word (2 bytes) de buffer en hexadecimal
0x56559020: 0x0a0b
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
salto1 () at datos_direccionamiento.s:75
+n avanzar una posición (next)
+n avanzar una posición (next)
+n avanzar una posición (next)
[Inferior 1 (process 6818) exited normally] el programa ha finalizado

+exit salir del gdb

DATOS_SUFIJOSGDB.TXT:

+b _start poner break point en la etiqueta _start
Punto de interrupción 1 at 0x8049000: file datos_sufijos.s, line 25.
+run ejecutar programa
Starting program: /home/eduardo/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P2/datos_sufijos

Breakpoint 1, _start () at datos_sufijos.s:25
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+layout split mostrar instrucciones con sus respectivas direcciones
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+layout regs mostrar contenido de los registros
+x /x &da1 mostrar dirección de da1 en hexadecimal
0x804a000: 0x0d0a0b0a
+x /b &da1 mostrar dirección de 1 byte de da1 en hexadecimal
0x804a000: 0x0a
+x /h &da2 mostrar 2 bytes (half word) de la dirección de da2
0x804a001: 0x0a0b
+x /w &da4 mostrar 4 bytes (word) de la dirección de da4
0x804a003: 0x0a0b0c0d
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
+n avanzar siguiente instrucción
[Inferior 1 (process 6137) exited normally]
+f mostrar pila
No hay pila.
+r volver a ejecutar el programa (run)
Starting program: /home/eduardo/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P2/datos_sufijos

Breakpoint 1, _start () at datos_sufijos.s:25
+exit salir del gdb

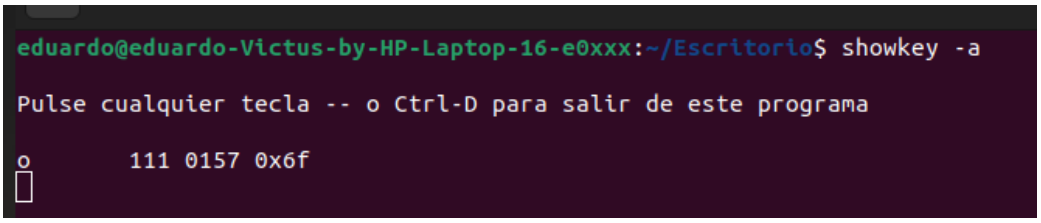
MÓDULO DATOS_SIZEGDB.TXT:

• ¿En qué orden se guardan los caracteres del string "hola"?

La variable men1 que contiene la string almacena los caracteres de forma secuencial, siendo el primer carácter “h” y último carácter “a” (en realidad es el carácter nulo, al ser .string).

• ¿Cuál es el código ASCII del carácter o?

A través de la instrucción showkey -a, la terminal nos mostrará un espacio para introducir el carácter y conocer su código ASCII.

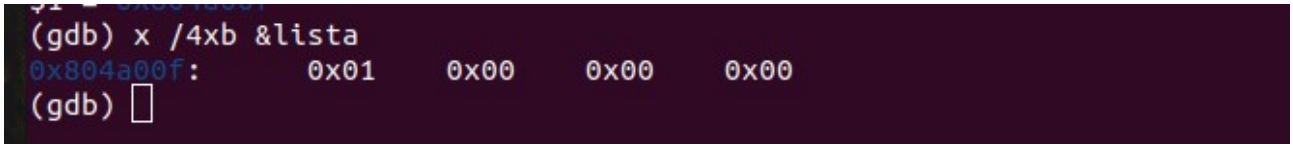


• ¿Cuál es la dirección de memoria principal donde se almacena el string "hola"?

• ¿Cuál es la dirección memoria principal donde se almacena el array lista?

La dirección del array lista en hexadecimal es: 0x804a00f

◦ ¿Cuál es el contenido de los primeros 4 bytes a partir de la dirección anterior en sentido ascendente?



MÓDULO DATOS_SUFIJOSGDB.TXT:

• ¿En qué orden se guardan los bytes del dato da4?

Sabiendo que la arquitectura i386/amd64 utiliza LITTLE ENDIAN, los MSB se muestran en las posiciones más bajas de memoria y los LSB en las más altas.

```
(gdb) x /4xb &da4
0x804a003:      0x0d    0x0c    0x0b    0x0a
(gdb)
```

• ¿Cuál es el resultado de ejecutar mov da1,%ecx?

Al ser el registro ecx un registro de 32 bits, y da1 una variable de tan solo 8 bits, el registro determinará el tamaño. Al tener una estructura little endian, se cargará también en el registro ecx el contenido de da2 “0x0a0b” y el primer byte de da4 “0x0d”, siguiendo la estructura mencionada anteriormente.

```
(gdb) p /x $ecx
$6 = 0xd0a0b0a
(gdb)
```

MÓDULO DATOS_DIRECCIONAMIENTOGDB.TXT:

• Con el depurador ejecutar el programa en modo paso a paso realizando las siguientes operaciones.

Después de compilar el archivo y añadir la tabla de símbolos...

• Array da2

◦ Imprimir la dirección de memoria del array da2 y el contenido del primer elemento:

La dirección es 0x4008 y el contenido del primer elemento (primeros dos bytes, half word) es 0x0a0b.

```
(gdb) x /xh &da2
0x4008: 0x0a0b
(gdb)
```

◦ 4 elementos de 2bytes del array da2: x /4xh &da2

Los 4 elementos del array se muestran eh hexadecimal.

```
(gdb) x /4xh &da2
0x4008: 0x0a0b  0x0f5c  0xffeb  0xffff
(gdb)
```

◦ ptype da2: no debug info: al no tener información el debuger del tamaño de los elementos es necesario indicarlos explicitamente en los comandos posteriores.

Al no tener información sobre el tamaño, necesitas castear el contenido del array.

```
(gdb) ptype da2
type = <data variable, no debug info>
(gdb)
```

■ Es necesario realizar un casting : Array de 4 elementos de tamaño 2bytes: p /x (short[4])da2

A continuación se imprime el contenido en hexadecimal de los 4 elementos del array en short.

```
(gdb) p /x (short[4]) da2
$1 = {0xa0b, 0xf5c, 0xffeb, 0xffff}
(gdb)
```

■ Fijarse con el comando eXaminar el resultado es independiente de sí hacemos un casting (short*): x /4xh (short *)&da2

Al utilizar dicho casting, el resultado es el mismo al mostrar el contenido de da2.

```
(gdb) x /4xh (short*) &da2
0x4008: 0x0a0b  0x0f5c  0xffeb  0xffff
(gdb)
```

■ El tamaño y tipo de dato lo fija el argumento del comando: /4xh

◦ Comprobar la norma de almacenamiento little endian identificando cada dirección de memoria a un byte con su contenido.

```
(gdb) x /4xb &da2
0x4008: 0x0b    0x0a    0x5c    0x0f
(gdb)
```

◦ Acceder a la dirección de memoria del elemento de valor -21 del array da2:

A continuación se muestra la dirección de memoria del tercer elemento (-21) del array.

```
(gdb) p /ax * ((short*)&da2+2)
$2 = 0xffeb
(gdb)
```

■ el argumento elemento de array en p da2[2] no es válido ya que el debugger carece de información

• Desensamblar
◦ disas salto1

Se muestran las instrucciones de la etiqueta salto1

```
Dump of assembler code for function salto1:
   0x000011da <+0>:     mov    $0x1,%eax
   0x000011df <+5>:     mov    $0x0,%ebx
   0x000011e4 <+10>:    int    $0x80
```

◦ disas /r salto1


(gdb) disas /r salto1
Dump of assembler code for function salto1:
0x000011da <+0>: b8 01 00 00 00 mov $0x1,%eax
0x000011df <+5>: bb 00 00 00 00 mov $0x0,%ebx
0x000011e4 <+10>: cd 80 int $0x80
End of assembler dump.
(gdb)