**Representación de los Datos**

Índice:

* datos\_size.s:
  + compilación
  + ejecución
  + comandos de GDB
* datos\_sufijos.s:
  + compilación
  + ejecución
  + comandos de GDB
* datos\_direccionamiento.s:
  + compilación
  + ejecución
  + comandos de GDB
* Autoevaluación:
  + datos\_size.s
  + datos\_sufijos.s
  + datos\_direccionamiento.s

datos size.s:

### Programa: datos\_size

### Descripción: declarar y acceder a distintos tamaños de operandos

### Compilación: gcc -nostartfiles -m32 -g -o datos\_size datos\_size.s

## MACROS

.equ SYS\_EXIT, 1 #declara las constantes con la instrucción .equ

.equ SUCCESS, 0

## VARIABLES LOCALES

.data

da1: .byte 0x0A #a la variable da1 le reserva un byte y le da valor 0x0A

da2: .2byte 0x0A0B

da4: .4byte 0x0A0B0C0D

men1: .ascii "hola" #crea una string y le da valor hola

lista: .int 1,2,3,4,5 #crea un array de enteros y lo inicializa

## INSTRUCCIONES

.global \_start #declara la etiqueta \_start como el punto de inicio del programa

.text #le avisa al assembler que a partir de aqui empiezan las instrucciones

\_start:

mov $SYS\_EXIT, %eax #guarda en el registro eax el valor 1 para pedirle al sistema operativo que ejecute exit

mov $SUCCESS, %ebx #guarda en el registro ebx el argumento de la función exit en este caso 0

fin: int $0x80 #llama al sistema operativo

.end #termina el programa

## *Compilar:*

gcc -nostartfiles -m32 -g -o datos\_size datos\_size.s

## *Ejecutar:*

Al ejecutar el programa y hacer echo $? nos devuelve un cero, que es el valor que le hemos pasado a exit como argumento, ya que es el valor que le hemos dado al registro ebx.

## *GDB:*

//Alineamiento de los bytes de un dato

x /tb &da1 -> nos devuelve un byte del contenido de da1 en binario

x /xh &da2 -> nos devuelve el contenido de da2 en hexadecimal

x /xw &da4 -> nos devuelve un elemento (solo hay uno) de da4 en hexadecimal

x /5xb &da4 -> devuelve 5 bytes de da4 en hexadecimal

-> alineamiento little endian

//Alineamiento de los bytes de un string

x /5cb &men1 -> nos devuelve 5 bytes de men1 con le valor del caracter

-> Alineamiento en secuencia

x /5xb &men1 -> lo mismo que el de arriba pero en hexadecimal

//Volcado de un string

p /s (char \*)&men1 -> imprime una cadena de caracteres desde la primera dirección

hasta encontrar el caracter NULL.

//Volcado de un array

x /5xw &lista -> contenido de 5 elementos de lista

p /a &lista -> dirección del array lista

p /a &lista+1 -> el depurador informa que es necesario realizar algún tipo de

casting (declaración dinámica)

p /a (void \*)&lista+1 -> se incrementa en 1 byte

p /a (int \*)&lista+1 -> se incrementa en 1\*4 bytes apuntando al segundo elemento del

array

p lista -> el depurador informa que es necesario realizar un casting

p (int)lista -> primer elemento del array

p (int \*)&lista -> dirección del array lista

p (int [5])lista -> contenido de cinco elementos de lista

p \*((int \*)&lista+1) -> segundo elemento de lista

x /dw (int \*)&lista+1 -> segundo elemento de lista

p \*(int \*)&lista@5 -> array artificial de 5 elementos de tipo int a partir de la

dirección &lista.

//volcado de una instrucción

p &\_start

x /i &\_start -> desensambla: convierte el código máquina en código ensamblador.

//Desensamblar: Conversión del código máquina en ensamblador

disas /r \_start

Las instrucciones que hemos utilizado son las siguientes (historial de gdb):

+shell ls -l datos\_size\_gdb\_asm.txt

+b \_start

Punto de interrupción 1 at 0x1000: file datos\_size.s, line 22.

+run

Starting program: /home/ubuntu20/Escritorio/datos\_size

Breakpoint 1, \_start () at datos\_size.s:22

+x /tb &da1

0x56558000: 00001010

+x /xh &da2

0x56558001: 0x0a0b

+x /xw &da4

0x56558003: 0x0a0b0c0d

+x /5xb &da4

0x56558003: 0x0d 0x0c 0x0b 0x0a 0x68

+x /5cb &men1

0x56558007: 104 'h' 111 'o' 108 'l' 97 'a' 1 '\001'

+x /5xb &men1

0x56558007: 0x68 0x6f 0x6c 0x61 0x01

+p /s (char \*)&men1

$1 = 0x56558007 "hola\001"

+x /5xw &lista

0x5655800b: 0x00000001 0x00000002 0x00000003 0x00000004

0x5655801b: 0x00000005

+p /a &lista

$2 = 0x5655800b

+p /a &lista+1

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

$3 = 0x5655800c

+p /a (int \*)&lista+1

$4 = 0x5655800f

+p lista

'lista' has unknown type; cast it to its declared type

+p (int)lista

$5 = 1

+p (int \*)&lista

$6 = (int \*) 0x5655800b

+p (int [5])lista

$7 = {1, 2, 3, 4, 5}

+p \*((int \*)&lista+1)

$8 = 2

+x /dw (int \*)&lista+1

0x5655800f: 2

+p \*(int \*)&lista@5

$9 = {1, 2, 3, 4, 5}

+p \*(int \*)&lista@4

$10 = {1, 2, 3, 4}

+p &\_start

$11 = (<text variable, no debug info> \*) 0x56556000 <\_start>

+x /i &\_start

=> 0x56556000 <\_start>: mov $0x1,%eax

+disas /r \_start

Dump of assembler code for function \_start:

=> 0x56556000 <+0>: b8 01 00 00 00 mov $0x1,%eax

0x56556005 <+5>: bb 00 00 00 00 mov $0x0,%ebx

End of assembler dump.

+disas /r fin

Dump of assembler code for function fin:

0x5655600a <+0>: cd 80 int $0x80

End of assembler dump.

+help x

Examine memory: x/FMT ADDRESS.

ADDRESS is an expression for the memory address to examine.

FMT is a repeat count followed by a format letter and a size letter.

Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),

t(binary), f(float), a(address), i(instruction), c(char), s(string)

and z(hex, zero padded on the left).

Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).

The specified number of objects of the specified size are printed

according to the format. If a negative number is specified, memory is

examined backward from the address.

Defaults for format and size letters are those previously used.

Default count is 1. Default address is following last thing printed

with this command or "print".

+q

datos sufijos.s:

### Programa: datos\_sufijos.s

### Descripción: utilizar distintos sufijos para los mnemónicos indicado distintos tamaños de operandos

### Compilación: gcc -m32 -g -o datos\_sufijos datos\_sufijos.s

## MACROS

.equ SYS\_EXIT, 1

.equ SUCCESS, 0

## VARIABLES LOCALES

.data

da1: .byte 0x0A

da2: .2byte 0x0A0B

da4: .4byte 0x0A0B0C0D

da44: .4byte 0xFFFFFFFF

men1: .ascii "hola"

lista: .int 1,2,3,4,5

## INSTRUCCIONES

.global \_start

.text

\_start:

## Reset de Registros

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

xor %edx,%edx

## Carga de datos

## mov da1,da4 ERROR: por referenciar los dos operandos a memoria

mov da4,%eax

movl da4,%ebx

movw da4,%cx

movb da4,%dl

## Reset de Registros

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

xor %edx,%edx

## Carga de datos

## movw da4,%al ERROR: incoherencia entre -w y AL

mov da4,%al #aplica el tamaño de DL

movb da4,%ebx #AVISO, NO error: incoherencia entre el registro BL y el sufijo b

movb %ebx,da44 #AVISO, NO error: incoherencia entre el registro BL y el sufijo b

mov %bx,da44 #

mov da1,%ecx

mov da4,%dx

## Reset de Registros

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

xor %edx,%edx

## Carga de datos

mov da1,%al

## inc da1 ERROR: sin el sufijo en inc por ser referencia sólo a memoria

incb da1

incw da2

incl da4

## salida

mov $SYS\_EXIT, %eax

mov $SUCCESS, %ebx

int $0x80

.end

## *Compilar:*

gcc -nostartfiles -m32 -g -o datos\_sufijos datos\_sufijos.s

Nos dice que estamos empleando `%bl' en lugar de `%ebx' debido `b' como sufijo. Esto es por la sintaxis de la instrucción: movb da4,%ebx

## *Ejecutar:*

Al ejecutar el programa y hacer un echo $? nos devuelve un 0

## *GDB:*

Al hacer layout regs se nos abren tres ventanas, en la de arriba hay tres columnas en las que nos aparecen los registros ,el valor que guardan en hexadecimal y en binario. En la ventana del medio nos aparecen las direcciones de las instrucciones que se ejecutan y las instrucciones que se ejecutan en lenguaje ensamblador. En la ventana de abajo nos aparece la interfaz del GDB con los comandos que estamos ejecutando, la línea en la que se está ejecutando el programa y el valor del contador de programa.

Las instrucciones que hemos utilizado son las siguientes (historial de gdb):

+shell ls -l datos\_sufijo\_gdb\_asm.txt

+b \_start

Punto de interrupción 1 at 0x1000: file datos\_sufijos.s, line 25.

+run

Starting program: /home/ubuntu20/Escritorio/datos\_sufijos

Breakpoint 1, \_start () at datos\_sufijos.s:25

25 xor %eax,%eax

+layout regs

datos direccionamiento.s:

### Program: datos\_direccionamiento.s

### Descripción: Emplear estructuras de datos con diferentes direccionamientos

## MACROS

.equ SYS\_EXIT, 1

.equ SUCCESS, 0

## VARIABLES LOCALES

.data

.align 4 # Alineamiento con direcciones de MP múltiplos de 4

da2: .2byte 0x0A0B,0b0000111101011100,-21,0xFFFF # Array da2 de elementos de 2 bytes

.align 4

lista: .word 1,2,3,4,5 # Array lista de elementos de 2 bytes

.align 8

buffer: .space 100 # Array buffer de 100 bytes

.align 2

saludo:

.string "Hola" # Array saludo de elementos de 1 byte por ser caracteres

## INSTRUCCIONES

.global main

.text

main:

## RESET

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

xor %edx,%edx

xor %esi,%esi

xor %edi,%edi

## ALGORITMO sum1toN

## Direccionamiento inmediato

mov $4,%si

## Direccionamiento indexado

bucle: add lista(,%esi,2),%di

## Direccionamiento a registro

dec %si

## Direccionamiento relativo al PC

jns bucle

## EJERCICIOS SOBRE DIRECCIONAMIENTO

## Direccionamiento indirecto

lea buffer,%eax #inicializo el puntero EAX

## mov da2,(%eax) ERROR: la dirección efectiva de los dos operandos hacen referencia a la memoria principal

mov da2,%bx

mov %bx, (%eax)

## Direccionamiento directo

incw da2

## Direccionamiento indexado

lea da2,%ebx

## inc 2(%ebx) ERROR: dirección efectiva a memoria y no hay sufijo

incw 2(%ebx)

mov $3,%esi

mov da2(,%esi,2),%ebx

## SALTOS INCONDICIONALES

## Direccionamiento relativo

jmp salto1 #salto relativo al contador de programa pc -> eip

xor %esi,%esi

salto1:

## SALIDA

mov $SYS\_EXIT, %eax

mov $SUCCESS, %ebx

int $0x80

.end

## *Compilar:*

gcc -m32 -g -o datos\_direccionamiento datos\_direccionamiento.s

## *Ejecutar:*

Al ejecutar el programa y hacer un echo $? nos devuelve un 0

## *GDB:*

Las instrucciones que hemos utilizado son las siguientes (historial de gdb):

+shell ls -l datos\_direccionamiento.txt

+b main

Punto de interrupción 1 at 0x119d: file datos\_direccionamiento.s, line 28.

+run

Starting program: /home/ubuntu20/Escritorio/datos\_direccionamiento

Breakpoint 1, main () at datos\_direccionamiento.s:28

28 xor %eax,%eax

+run

Programa no reiniciado.

+x /xh &da2

0x56559008: 0x0a0b

+x /4xh &da2

0x56559008: 0x0a0b 0x0f5c 0xffeb 0xffff

+p /x (short[4])da2

$1 = {0xa0b, 0xf5c, 0xffeb, 0xffff}

+ptype lista

type = <data variable, no debug info>

+p (short[5])lista

$2 = {1, 2, 3, 4, 5}

+ptype buffer

type = <data variable, no debug info>

+p &buffer

$3 = (<data variable, no debug info> \*) 0x56559020

+ptype saludo

type = <data variable, no debug info>

+p /c (char[5])saludo

$4 = {72 'H', 111 'o', 108 'l', 97 'a', 0 '\000'}

+x /5c (char \*)&saludo

0x56559084: 72 'H' 111 'o' 108 'l' 97 'a' 0 '\000'

+p /c \*(char \*)&saludo

$5 = 72 'H'

+p /s (char \*)&saludo

$6 = 0x56559084 "Hola"

+q

Autoevaluación:

# *Módulo datos\_size.s*:

1. **¿En qué orden se guardan los caracteres del string "hola"?:** Se guardan en el orden de lectura habitual: ´h´ en la posición de memoria donde comienza el array, la ´o´ en la posición de h + 1…
2. **¿Cuál es el código ASCII del carácter o?:** el código ASCII de la ´o´ es 111
3. **¿Cuál es la dirección de memoria principal donde se almacena el string "hola"?:** en la dirección de memoria 0x56558007
4. **¿Cuál es la dirección memoria principal donde se almacena el array lista?:** en la dirección de memoria 0x5655800b
5. **¿Cuál es el contenido de los primeros 4 bytes a partir de la dirección anterior en sentido ascendente?:** las letras ´o´,´l´, ´a´ y ´\0´

# *Módulo datos\_sufijos.s*:

1. **¿En qué orden se guardan los bytes del dato da4?:** primero 0D, luego 0C, luego 0B y por último 0A.
2. **¿Cuál es el resultado de ejecutar mov da1,%ecx?:** mueve el byte de que está en da1 al registro ECX

# *Módulo datos\_direccionamiento.s*:

Este es el historial del gdb:

+shell ls -l datos\_direccionamiento\_gdb\_asm.txt

+b main

Punto de interrupción 1 at 0x119d: file datos\_direccionamiento.s, line 28.

+run

Starting program: /home/ubuntu20/Escritorio/datos\_direccionamiento

Breakpoint 1, main () at datos\_direccionamiento.s:28

28 xor %eax,%eax

+x /xh &da2

0x56559008: 0x0a0b

+x /4xh &da2

0x56559008: 0x0a0b 0x0f5c 0xffeb 0xffff

+ptype da2

type = <data variable, no debug info>

+p /x (short[4])da2

$1 = {0xa0b, 0xf5c, 0xffeb, 0xffff}

+x /4xh (short

A syntax error in expression, near `'.

+\*)&da2

orden indefinida: «». Intente con «help»

+x /4xh (short

A syntax error in expression, near `'.

+x /4xh (short\*)&da2

0x56559008: 0x0a0b 0x0f5c 0xffeb 0xffff

+x /8xb (short\*)&da2

0x56559008: 0x0b 0x0a 0x5c 0x0f 0xeb 0xff 0xff 0xff

+x /xh (short\*)&da2+2

0x5655900c: 0xffeb

+disas salto1

Dump of assembler code for function salto1:

0x565561ea <+0>: mov $0x1,%eax

0x565561ef <+5>: mov $0x0,%ebx

0x565561f4 <+10>: int $0x80

0x565561f6 <+12>: xchg %ax,%ax

0x565561f8 <+14>: xchg %ax,%ax

0x565561fa <+16>: xchg %ax,%ax

0x565561fc <+18>: xchg %ax,%ax

0x565561fe <+20>: xchg %ax,%ax

End of assembler dump.

+disas /r salto1

Dump of assembler code for function salto1:

0x565561ea <+0>: b8 01 00 00 00 mov $0x1,%eax

0x565561ef <+5>: bb 00 00 00 00 mov $0x0,%ebx

0x565561f4 <+10>: cd 80 int $0x80

0x565561f6 <+12>: 66 90 xchg %ax,%ax

0x565561f8 <+14>: 66 90 xchg %ax,%ax

0x565561fa <+16>: 66 90 xchg %ax,%ax

0x565561fc <+18>: 66 90 xchg %ax,%ax

0x565561fe <+20>: 66 90 xchg %ax,%ax

End of assembler dump.

+q