SANTIAGO AYECHU GÁRRIZ

Práctica 2ª: Representación de los Datos

ÍNDICE

Introducción *3*

Desarrollo *3*

Módulo `datos_size.s` **3**

Módulo `datos_sufijos.s` **4**

Módulo `datos_direccionamiento.s` **6**

Módulos Fuente Comentados **8**

Comandos de Compilación **13**

Historial Comandos GDB + Salida **14**

Conclusiones *23*

Introducción

En esta práctica se llevó a cabo la depuración de tres programas, cuyo código está disponible en la sección de "Módulos Fuente Comentados".

En el primer programa, `datos_size.s`, se declaran diferentes tipos de variables locales y se introduce el concepto de MACRO. En la depuración, de dicho programa, se ven diferentes formas de analizar el contenido de la memoria principal mediante el depurador GDB.

En cuanto al segundo programa, `datos_sufijos.s`, se empieza declarando, de nuevo, variables locales que, posteriormente, serán utilizadas para realizar diferentes instrucciones como `mov`. Se muestran registros de diferentes bits y algunos casos de error (como se verá el caso de que el sufijo del mnemónico sea mayor que el tamaño del registro destino).

Por último, en el último programa, `datos_direccionamiento.s`, se ven distintos tipos de direccionamientos así como: direccionamiento inmediato, direccionamiento indexado, relativo al PC, entre otros. Para ello, se realiza la depuración de este programa para poder ir viendo la ejecución del programa instrucción por instrucción.

Desarrollo

Módulo `datos_size.s`

1. La variable men1, cuyo contenido es la string "hola", esta se almacena secuencialmente en memoria. De tal forma, que al hacer un volcado de la string queda de la siguiente manera:

```
(gdb) x /5cb &men1
0x56558007:     104 'h' 111 'o' 108 'l' 97 'a'  1 '\001'
(gdb) x /5xb &men1
0x56558007:     0x68    0x6f    0x6c    0x61    0x01
(gdb)
```

   Para ello hemos usado el comando eXamine imprimiendo el contenido en diferentes formatos. Hemos impreso, inicialmente, el contenido de la variable `men1` en formato carácter (de ahí la c de /5cb) con sus respectivos equivalentes ASCII en decimal. El carácter 'h' equivale al decimal 104 de la tabla ASCII. Hemos incluido 5 bytes ya que la string hola tiene 4 bytes, junto con el carácter NULL (0x00) son 5 bytes.

2. El carácter 'o', como se puede ver en la imagen anterior, equivale al código 111 en la tabla ASCII. O lo que es lo mismo, el carácter 'o' equivale al código `0x6f` de la tabla ASCII en formato hexadecimal. Otra forma que se puede ver el código ASCII equivalente a un carácter es la siguiente:

```
sayechu@sayechu-MacBookPro:~$ showkey -a

Pulse cualquier tecla -- o Ctrl-D para salir de este programa

o       111 0157 0x6f
```

En una terminal, escribimos el comando `showkey -a` y, a continuación, pulsamos el carácter del cual queremos obtener su código ASCII. Como se puede observar en la 4 columna se obtiene el mismo resultado: `0x6f`.

3. Para saber la dirección de memoria en donde se almacena la string “hola”, almacenada en la variable `men1`, en este caso usaremos el comando print (p), como se observa a continuación:

```
(gdb) p /a &men1
$5 = 0x56558007
(gdb)
```

Lo que hacemos es añadir el argumento `/a` para indicar que queremos obtener una ADDRESS, una dirección, que es la de la variable `men1`. En la foto adjuntada inicialmente, en el comando eXamine nos aparece la dirección de memoria junto con su contenido, también se podría usar ese comando para ver la dirección de memoria en donde se encuentra almacenada la string “hola”.

4. Al igual que en el apartado 3, para ver la dirección de memoria principal donde se encuentra almacenada la array ‘lista’ usaremos, de nuevo, el comando print (p).

```
(gdb) p /a &lista
$6 = 0x5655800b
(gdb)
```

   a. Para obtener el contenido de los 4 primeros bytes del array ‘lista’ usaremos el comando eXamine.

```
(gdb) x /4xw &lista
0x5655800b:     0x00000001      0x00000002      0x00000003      0x00000004
(gdb)
```

Lo que hemos hecho ha sido imprimir los 4 bytes en formato hexadecimal (por ello la x de /4**x**w).
Como se observa, se obtienen los 4 primeros bytes del array lista (1, 2, 3, 4).

Módulo `datos_sufijos.s`

1. Sabiendo que: La arquitectura i386/amd64 utiliza LITTLE ENDIAN, el contenido del dato da4 se almacena en memoria siguiente dicho formato. Es decir, los MSB se almacenan en las posiciones de memoria más bajas, mientras que los LSB se almacenan en las posiciones de memoria mayores. Por ello, cuando hacemos un

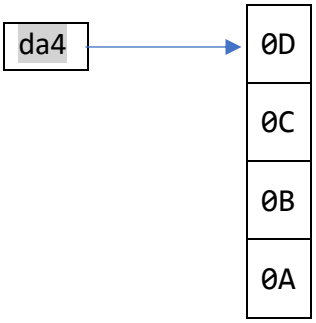
eXamine sobre da4, aparece en orden inverso al que ha sido declarado inicialmente como se puede ver en la siguiente captura de pantalla.

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

da4 fue declarado al comienzo del programa de la siguiente manera:

```
da4: .4byte     0x0A0B0C0D
```

da4 se almacena de la siguiente manera en memoria:



2. La instrucción `mov da1, %ecx` no tiene ningún sufijo. Al no tener sufijo, es el tamaño del registro quien especifica el tamaño de los operandos fuente y destino. Siendo ECX un registro de 32 bits (4 bytes), se moverán 4 bytes a partir de da1 al registro ecx.

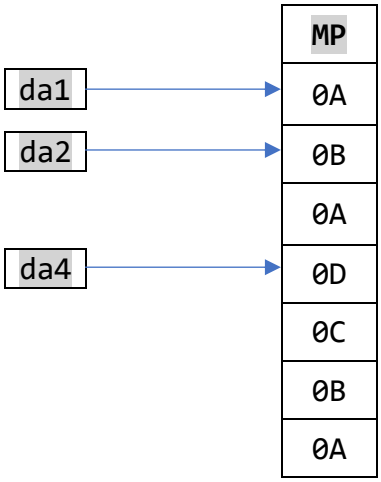
```
(gdb) p $ecx
$4 = 0
(gdb) n
(gdb) p /x $ecx
$5 = 0xd0a0b0a
(gdb)
```

Al igual que ocurre en el apartado anterior, ocurre en este. Las variables se han almacenado en memoria siguiente el formato Little Endian, es por eso que aparecen en ese orden. Se muestra las declaraciones de las variables y cómo han sido almacenadas en memoria a continuación.

```
da1: .byte      0x0A

da2: .2byte     0x0A0B

da4: .4byte     0x0A0B0C0D
```



Por tanto, al ejecutar dicha instrucción vamos a mover al registro ecx, 4 bytes a partir de da1, es decir 0A, 0B, 0A, 0D y se imprime en su correspondiente formato.

Módulo datos_direccionamiento.s

1. Habiendo compilado el módulo fuente datos_direccionamiento.s cargando la tabla de símbolos, en el depurador se ha impreso la dirección de memoria del array da2 y su contenido del primer elemento.

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

Como se puede observar el contenido del primer elemento del array es 0x0a0b y está almacenado en la dirección de memoria: 0x4008 siguiendo el formato Little Endian.

2. En este caso, en vez de imprimir solo el primer elemento, vamos a imprimir los 4 elemento de 2bytes del array (da2), usando el comando eXamine.

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

Se ha decidido imprimir en formato hexadecimal.

3. También se puede mostrar los elementos del array haciendo un casting.

```
(gdb) p /x (short[4])da2
$2 = {0xa0b, 0xf5c, 0xffeb, 0xffff}
(gdb) x /4xh (short *)&da2
0x4008: 0x0a0b  0x0f5c  0xffeb  0xffff
(gdb)
```

4. Para comprobar que se alacena siguiendo el formato Little Endian, hemos impreso el primer elemento del array da2 volcando el contenido byte por byte a partir de da2.

```
(gdb) x /1xb &da2
0x4008: 0x0b
0x4009: 0x0a
0x400a: 0x5c
0x400b: 0x0f
(gdb)
```

Como se observa, los MSB se almacenan en posición de memoria menores mientras que los LSB se almacenan en posiciones de memoria superiores. Las direcciones `0x4008` y `0x4009` son las relacionadas con el primer elemento del array da2, mientras que las posiciones `0x400a` y `0x400b` son las posiciones del segundo elemento del array.

5. Sabiendo que el elemento -21 se encuentra en la posición 3 del array (da2 + 2), imprimiremos ese elemento del array tanto con el comando print (p) como con el comando eXamine (x), realizando un casting en ambos casos.

```
(gdb) p /x *((short *)&da2+2)
$6 = 0xffeb
(gdb) p /d *((short *)&da2+2)
$7 = -21
(gdb)
```

```
(gdb) x /dw (short *)&da2+2
0x400c: -21
(gdb) x /xw (short *)&da2+2
0x400c: 0xffffffeb
(gdb)
```

Se ha impreso ambos resultados tanto en formato signed decimal (d) como en hexadecimal.

6. Por último, se ha introducido dos comandos para desensamblar. El desensamblado consiste en “transformar” del lenguaje máquina al lenguaje ensamblador, obteniendo el siguiente resultado:

```
+disas salto1
Dump of assembler code for function salto1:
   0x565561da <+0>:     mov    $0x1,%eax
   0x565561df <+5>:     mov    $0x0,%ebx
   0x565561e4 <+10>:    int    $0x80
End of assembler dump.
```

```
+disas /r salto1
Dump of assembler code for function salto1:
   0x565561da <+0>:     b8 01 00 00 00    mov    $0x1,%eax
   0x565561df <+5>:     bb 00 00 00 00    mov    $0x0,%ebx
   0x565561e4 <+10>:    cd 80     int    $0x80
End of assembler dump.
```

En este segundo caso, añadiendo el argumento `/r` al comando `disas`, se pueden observar cómo realmente están escritas, en lenguaje máquina, las instrucciones en la memoria principal.

Módulos Fuente Comentados

A continuación, se muestran los tres módulos fuente usados en esta práctica (`datos_size.s` , `datos_sufijos.s` , `datos_direccionamiento.s`) con sus respectivos comentarios en rojo.

datos_size.s

```
### Program: operando_size
### Descripción: declarar y acceder a distintos tamaños de operandos
### Compilación: gcc -nostartfiles -m32 -g -o datos_size datos_size.s

        ## MACROS       En PREPROCESAMIENTO, se sustituye MACROS por dato asociado
        .equ SYS_EXIT, 1          Se sustituye donde ponga SYS_EXIT por valor 1
        .equ SUCCESS,   0         Se sustituye donde ponga SUCCESS por valor 0

        ## VARIABLES LOCALES          DECLARACIÓN VARIABLES LOCALES
        .data

da1:    .byte   0x0A                da1 variable de 1 byte, contenido 0x0A = 0b00001010
da2:    .2byte  0x0A0B    da2 variable de 2 bytes, inicializada a 0x0A0B
da4:    .4byte  0x0A0B0C0D        da4 variable de 4 bytes, inicializada a 0x0A0B0C0D
men1:   .ascii  "hola"        men1 string inicializada a “hola”
lista:  .int    1,2,3,4,5         lista array de enteros, inicializada a 1,2,3,4,5

        ## INSTRUCCIONES
        .global _start
        .text
_start:
```

```
        mov $SYS_EXIT, %eax          código de llamada al S.O: subrutina exit
        mov $SUCCESS,  %ebx          argumento de salida al S.O, según convenio
fin:    int $0x80                    llamada al S.O para que ejecute subrutina

        .end            FIN PROGRAMA
```

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       En PREPROCESAMIENTO, se sustituye MACROS por dato asociado
        .equ SYS_EXIT, 1          Se sustituye donde ponga SYS_EXIT por valor 1
        .equ SUCCESS,   0         Se sustituye donde ponga SUCCESS por valor 0

        ## VARIABLES LOCALES          DECLARACIÓN VARIABLES LOCALES
        .data

da1:    .byte   0x0A                da1 variable de 1 byte, contenido 0x0A = 0b00001010
da2:    .2byte  0x0A0B    da2 variable de 2 bytes, inicializada a 0x0A0B
da4:    .4byte  0x0A0B0C0D        da4 variable de 4 bytes, inicializada a 0x0A0B0C0D
da44:   .4byte  0xFFFFFFFF        da44 variable de 4 bytes, inicializada a 0xFFFFFFFF
men1:   .ascii  "hola"        men1 string inicializada a “hola”
lista:  .int    1,2,3,4,5         lista array de enteros, inicializada a 1,2,3,4,5


        ## INSTRUCCIONES
        .global _start
        .text
_start:

        ## Reset de Registros      INICIALIZACIÓN A 0 DE REGISTROS
        xor  %eax,%eax  LA OPERACIÓN LÓGICA XOR VALE 0 SI LOS DOS
        xor  %ebx,%ebx  OPERANDOS SON IGUALES, POR TANTO HACIENDO
        xor  %ecx,%ecx  LA OPERACIÓN SOBRE EL MISMO REGISTRO, TODOS LOS
        xor  %edx,%edx  OPERANDOS SON IGUALES, POR TANTO SE INICIALIZAN A 0
```

```
## Carga de datos
## mov da1,da4  ERROR: NO SE PUEDE REFERENCIAR LOS DOS OP. A MEMORIA
mov  da4,%eax          REGISTRO EAX 32 BITS, 4 BYTES. VARIABLE DA4 4 BYTES
movl da4,%ebx          REGISTRO EBX 32 BITS, 4 BYTES. MOVER 4 BYTES (L)
movw da4,%cx           REGISTRO CX, 16 BITS, 2 BYTES. MOVER 2 BYTES (W)
movb da4,%dl           REGISTRO DL, 8 BITS, 1 BYTE. MOVER 1 BYTE (B)

## Reset de Registros   INICIALIZACIÓN A 0 DE REGISTROS
xor  %eax,%eax  LA OPERACIÓN LÓGICA XOR VALE 0 SI LOS DOS
xor  %ebx,%ebx  OPERANDOS SON IGUALES, POR TANTO HACIENDO
xor  %ecx,%ecx  LA OPERACIÓN SOBRE EL MISMO REGISTRO, TODOS LOS
xor  %edx,%edx  OPERANDOS SON IGUALES, POR TANTO SE INICIALIZAN A 0

## Carga de datos
## movw  da4,%al        ERROR: AL REGISTRO 1 BYTE, NO PODEMOS MOVER 2 BYTES
mov da4,%al            SE MUEVE 1 BYTE DE DA4 A REGISTRO AL (8 BITS)

#movb da4,%ebx  #AVISO: incoherencia entre registro EBX y el sufijo b
#movb %ebx,da44 #AVISO: incoherencia entre registro EBX y el sufijo b
#mov  %bx,da44  MOV 2 BYTES DE REGISTRO BX A DA44

mov  da1,%ecx          MOV DA1 (1 BYTE) A REGISTRO ECX (4 BYTES)
mov  da4,%dx           MOV DA4 (4 BYTE) A REGISTRO DX (2 BYTES)

## Reset de Registros   INICIALIZACIÓN A 0 DE REGISTROS
xor  %eax,%eax  LA OPERACIÓN LÓGICA XOR VALE 0 SI LOS DOS
xor  %ebx,%ebx  OPERANDOS SON IGUALES, POR TANTO HACIENDO
xor  %ecx,%ecx  LA OPERACIÓN SOBRE EL MISMO REGISTRO, TODOS LOS
xor  %edx,%edx  OPERANDOS SON IGUALES, POR TANTO SE INICIALIZAN A 0

## Carga de datos

mov  da1,%al        MOV DA1 (1 BYTE) A REGISTRO AL (1 BYTE)

## inc da1  #ERROR:  sin sufijo en inc por ser referencia sólo a memoria
incb da1            INCREMENTAR 1 BYTE DA1
incw da2                    INCREMENTAR 2 BYTES DA2
incl da4            INCREMENTAR 4 BYTES DA4
```

```
## salida
        mov $SYS_EXIT, %eax          código de llamada al S.O: subrutina exit
        mov $SUCCESS,  %ebx          argumento de salida al S.O, según convenio
        int $0x80                    llamada al S.O para que ejecute subrutina

        .end            FIN PROGRAMA
```

datos_direccionamiento.s

```
### Program:     datos_direccionamiento.s
### Descripción: Emplear estructuras de datos con diferentes direccionamientos

        ## MACROS #En PREPROCESAMIENTO, se sustituye MACROS por dato asociado
        .equ SYS_EXIT, 1        #Se sustituye donde ponga SYS_EXIT por valor 1
        .equ SUCCESS,   0       #Se sustituye donde ponga SUCCESS por valor 0

        ## VARIABLES LOCALES
        .data

        .align 4   # Alineamiento con direcciones de MP múltiplos de 4
da2:    .2byte  0x0A0B,0b0000111101011100,-21,0xFFFF #Array elementos de 2Byt
        .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          # INICIALIZACIÓN A 0 DE REGISTROS
        xor  %ebx,%ebx          # OPERACIÓN XOR, SI OPERANDOS IGUALES 0
        xor  %ecx,%ecx
        xor  %edx,%edx
```

```
        xor  %esi,%esi          # REGISTROS DE INDICES
        xor  %edi,%edi          # REGISTROS DE INDICES

        ## ALGORITMO sum1toN

        mov $4,%si              # Direccionamiento inmediato

bucle:  add lista(,%esi,2),%di  #Direccionamiento Indexado 2 * R.I (ESI)+lista
        dec %si ## Direccionamiento a registro
        jns bucle           ## Direccionamiento relativo al PC


        ## EJERCICIOS SOBRE DIRECCIONAMIENTO

        lea buffer,%eax   #inicializo el puntero EAX. Direc. Indirecto
        ## mov da2,(%eax)       #ERROR: los dos operandos hacen referencia a memoria
        mov da2,%bx             #bx registro 2 bytes, da2 2 bytes
        mov %bx, (%eax)#mov contenido registro bx a dirección que tiene eax
        incw da2                ## Direccionamiento directo
        lea  da2,%ebx           # LOAD EFFECTIVE ADDRESS (LEA). Direcc. Index.
        ## inc 2(%ebx) #ERROR: dirección efectiva a memoria y no hay sufijo
        incw 2(%ebx)            # INCREMENTO EBX EN 2

        mov $3,%esi             #Direccionamiento Inmediato. Mover 3 a ESI
        mov da2(,%esi,2),%ebx   # EA = 2*esi+da2 | ebx <- M[EA]

        ## SALTOS INCONDICIONALES

        ## Direccionamiento relativo
        jmp salto1          #salto relativo al contador de programa pc -> eip
        xor %esi,%esi           #esi, registro índice a 0
salto1:

        ## SALIDA

        mov $SYS_EXIT, %eax         # código de llamada al S.O: subrutina exit
        mov $SUCCESS,  %ebx         # argumento de salida al S.O, según convenio
        int $0x80                   # llamada al S.O para que ejecute subrutina

        .end            FIN DE PROGRAMA
```

Comandos de Compilación

A continuación, se muestran los comandos usados para la compilación de los tres programas sobre los que trata esta práctica.

- El programa de `datos_size.s` se compiló usando el Toolchain automatico usando el siguiente comando:

  ```
  gcc -m32 -g -nostartfiles -o datos_size datos_size.s
  ```

  Añadiendo los siguientes argumentos:

  -m32: módulos fuente y objeto para la arquitectura i386.
  -nostartfiles : especifica que el punto de entrada no es main sino _start.
  -g para cargar tabla de símbolos

  Este comando nos genera un módulo binario ejecutable (datos_size) listo para ser cargado en memoria.

- El programa `datos_sufijos.s` se compiló, a diferencia el anterior, usando el Toolchain manual usando los siguientes comandos:

  ```
  as --32 -gstabs -o datos_sufijos.o datos_sufijos.s
  ```

  Añadiendo los siguientes argumentos:

  *.s : módulo fuente en lenguaje ASM
  *.o : módulo objeto reubicable
  -gstabs: generación de la tabla de símbolos e inserción en el módulo ejecutable.
  --32 : módulos fuente y objeto para la ISA de 32 bits

  ```
  ld -melf_i386 -o datos_sufijos datos_sufijos.o
  ```

  Añadiendo el argumento:

  -melf_i386: módulos objeto para la ISA de 32 bits

- A diferencia de los otros dos programas, el programa `datos_direccionamiento.s` tiene la etiqueta main, y no _start como en los casos anteriores. Por ello, no es necesario añadir el argumento -nostartfiles en el comando de compilación del Toolchain automático. Como en este caso compilamos usando el Toolchain Automático, no añadimos dicho argumento y se compila con el siguiente comando:

  ```
  gcc -m32 -g -o datos_direccionamiento
        datos_direccionamiento.s
  ```

  Añadiendo los siguientes argumentos:

-m32: módulos fuente y objeto para la arquitectura i386.
-g para cargar tabla de símbolos

Historial Comandos GDB + Salida

A continuación, se muestra el `.txt` generado a partir de la depuración del módulo `datos_size.s`. Cabe destacar que también se adjunta los comandos, junto a su salida, con sus respectivos comentarios (en rojo). Cuando hacemos un examine sobre una dirección de memoria, volcamos el contenido en memoria a partir de la dirección de memoria indicada en el comando eXamine. Aparecen algunos de los comandos que se han preguntado previamente en el módulo `datos_size.s` , junto con su respuesta de ejecución.

```
+file datos_size                Cargamos modulo datos_size
Reading symbols from datos_size...
+focus cmd                      Enfocamos en terminal para poder usar flechas teclado
Focus set to cmd window.
+b _start              Breakpoint, punto de ruptura, en etiqueta _start
Punto de interrupción 1 at 0x1000: file datos_size.s, line 22.
+run                            ejecutamos el programa
Starting program: /home/sayechu/Escritorio/EECC/P2/Ej1/datos_size

Breakpoint 1, _start () at datos_size.s:22
+x /tb &da1          volcamos un byte del contenido de 'da1' en formato binario
0x56558000:     00001010
+x /1xb &da1         volcamos un byte del contenido de 'da1' en formato hexadecimal
0x56558000:     0x0a
+x /1xw &da4                    volcamos una word en hexadecimal a partir de 'da4'
0x56558003:     0x0a0b0c0d
+x /20xb &da1                   volcamos 20 bytes a partir de 'da1'
0x56558000:     0x0a    0x0b    0x0a    0x0d    0x0c    0x0b    0x0a    0x68
0x56558008:     0x6f    0x6c    0x61    0x01    0x00    0x00    0x00    0x02
0x56558010:     0x00    0x00    0x00    0x03
+x /1xh &da2      volcamos una media word en formato hexadecimal a partir de 'da2'
0x56558001:     0x0a0b
+x /5cb &men1                   volcamos 5 caracteres a partir de dirección de 'men1'
0x56558007:     104 'h'  111 'o'  108 'l'  97 'a'   1 '\001'
+p /s (char *)&men1            imprimir contenido de string hasta encontrar NULL (0x00)
$1 = 0x56558007 "hola\001"
+p /a &men1                     imprimir dirección de memoria de 'men1'
$2 = 0x56558007
```

```
+x /5xw &da4                volcamos 5 words en hexadecimal a partir de 'da4'
0x56558003:    0x0a0b0c0d    0x616c6f68    0x00000001    0x00000002
0x56558013:    0x00000003
+p /a &lista                imprimir dirección de memoria de etiqueta 'lista'
$3 = 0x5655800b
+p *(int *)&lista@5         imprimir "array" de 5 elementos a partir de dir. de &lista
$4 = {1, 2, 3, 4, 5}
+p (int [5])lista    imprimir contenido de cinco elementos de 'lista'
$5 = {1, 2, 3, 4, 5}
+x /i &_start               desensambla, convierte código máquina a código ensamb.
=> 0x56556000 <_start>:  mov    $0x1,%eax
+disas /r _start    desensambla, convierte código máquina a código ensamb.
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.
+x /dw (int *)&lista+1      volcar segundo elemento de 'lista' -> 2
0x5655800f:    2
+p /a (int *)&lista+1       imprimir dirección de lista+1
$6 = 0x5655800f
+quit                       salir de la depuración
```

A continuación, se muestra el `.txt` generado a partir de la depuración del módulo `datos_sufijos.s`. Cabe destacar que también se adjunta los comandos, junto a su salida, con sus respectivos comentarios (en rojo).

Cabe recalcar que, antes de ejecutar las instrucciones, volcamos el contenido con el comando eXamine indicando el número de bytes, para ver el contenido que se va a mover en las instrucciones. Por ejemplo, antes de ejecutar la instrucción

```
movb da4, %dl
```

sabiendo que movemos 1 byte a partir de da4 al registro dl (por el sufijo b), hacemos un eXamine de 1 byte a partir de da4. Así, vemos el contenido que se mueve a dicho registro. Posteriormente comprobamos con el comando print e imprimimos el contenido de dicho registro para ver que se ha realizado correctamente.

```
+file datos_sufijos              cargamos fichero a depurar datos_sufijos
Reading symbols from datos_sufijos...
+focus cmd                       hacemos focus en línea de comandos
Focus set to cmd window.
```

```
+b _start                    punto de interrupción en etiqueta _start
Punto de interrupción 1 at 0x8049000: file datos_sufijos.s, line 25.
+run                                 ejecutamos depuración
Starting program: /home/sayechu/Escritorio/EECC/P2/Ej2/datos_sufijos

Breakpoint 1, _start () at datos_sufijos.s:25
+layout regs                         abrimos ventana de registros
+x /4xb &da4                         volcamos 4 bytes en hexadecimal a partir de da4
0x804a003:      0x0d    0x0c    0x0b    0x0a
+n                                   ejecutamos siguiente instrucción (xor %eax, %eax)
+n                                   ejecutamos siguiente instrucción (xor %ebx, %ebx)
+n                                   ejecutamos siguiente instrucción (xor %ecx, %ecx)
+n                                   ejecutamos siguiente instrucción (xor %edx, %edx)
+p /x $eax                           imprimir contenido registro eax en hexadecimal
$1 = 0x0
+p /x $ebx                           imprimir contenido registro ebx en hexadecimal
$2 = 0x0
+p /x $ecx                           imprimir contenido registro ecx en hexadecimal
$3 = 0x0
+p /x $edx                           imprimir contenido registro edx en hexadecimal
$4 = 0x0
+x /4xb &da4                         volcar 4 bytes en hexadecimal a partir de da4
0x804a003:      0x0d    0x0c    0x0b    0x0a
+n                                   ejecutamos siguiente instrucción (mov da4, %eax)
+p /x $eax                           imprimir contenido registro eax formato hexadecimal
$5 = 0xa0b0c0d
+x /4xb &da4                         volcar 4 bytes en hexadecimal a partir de da4
0x804a003:      0x0d    0x0c    0x0b    0x0a
+n                                   ejecutamos siguiente instrucción (movl da4, %ebx)
+p /x $ebx                           imprimir contenido registro ebx formato hexadecimal
$6 = 0xa0b0c0d
+x /2xb &da4                         volcar 2 bytes en hexadecimal a partir de da4
0x804a003:      0x0d    0x0c
+n                                   ejecutamos siguiente instrucción (movw da4, %cx)
+p /x $cx                            imprimir contenido registro cx formato hexadecimal
$7 = 0xc0d
+x /1xb &da4                         volcar 1 byte en hexadecimal a partir de da4
0x804a003:      0x0d
+n                                   ejecutamos siguiente instrucción (movb da4, %dl)
+p /x $dl                    imprimir contenido registro dl en f. hexadecimal
```

```
$8 = 0xd
+n                                  ejecutamos siguiente instrucción (xor %eax, %eax)
+n                                  ejecutamos siguiente instrucción (xor %ebx, %ebx)
+n                                  ejecutamos siguiente instrucción (xor %ecx, %ecx)
+n                                  ejecutamos siguiente instrucción (xor %edx, %edx)
+p /x $eax                          imprimir contenido registro eax en hexadecimal
$9 = 0x0
+p /x $ebx                          imprimir contenido registro ebx en hexadecimal
$10 = 0x0
+p /x $ecx                          imprimir contenido registro ecx en hexadecimal
$11 = 0x0
+p /x $edx                          imprimir contenido registro edx en hexadecimal
$12 = 0x0
+p /x $al                   imprimir contenido registro al en hexadecimal
$13 = 0x0
+n                                  ejecutamos siguiente instrucción (mov da4, %al)
+p /x $al                   imprimir contenido registro al en hexadecimal
$14 = 0xd
+x /4xb &da1                        volcar 4 bytes en hexadecimal a partir da1
0x804a000:      0x0a    0x0b    0x0a    0x0d
+p /x $ecx                          imprimir contenido registro ecx formato hexadecimal
$15 = 0x0
+n                                  ejecutamos siguiente instrucción (mov da1, %ecx)
+p /x $ecx                          imprimir contenido registro ecx formato hexadecimal
$16 = 0xd0a0b0a
+p /x $dx                           imprimir contenido registro dx formato hexadecimal
$17 = 0x0
+x /2xb &da4                        volcar 2 bytes en hexadecimal a partir de da4
0x804a003:      0x0d    0x0c
+n                                  ejecutamos siguiente instrucción (mov da4, %dx)
+p /x $dx                           imprimir contenido registro dx, formato hexadecimal
$18 = 0xc0d
+n                                  ejecutamos siguiente instrucción (xor %eax, %eax)
+n                                  ejecutamos siguiente instrucción (xor %ebx, %ebx)
+n                                  ejecutamos siguiente instrucción (xor %ecx, %ecx)
+n                                  ejecutamos siguiente instrucción (xor %edx, %edx)
+p /x $eax                          imprimir contenido registro eax en hexadecimal
$19 = 0x0
+p /x $ebx                          imprimir contenido registro ebx en hexadecimal
$20 = 0x0
```

```
+p /x $ecx                          imprimir contenido registro ecx en hexadecimal
$21 = 0x0
+p /x $edx                          imprimir contenido registro edx en hexadecimal
$22 = 0x0
+x /1xb &da1                        volcar 1 byte en hexadecimal a partir de da1
0x804a000:      0x0a
+n                                  ejecutamos siguiente instrucción (mov da1, ,%al)
+p /x $al                   imprimir contenido registro al, en hexadecimal
$23 = 0xa
+x /1xw &da1                        volcar 1 word a partir de da1
0x804a000:      0x0d0a0b0a
+x /1xh &da1                        volcar 1 media word a partir de da1
0x804a000:      0x0b0a
+n                                  ejecutamos siguiente instrucción (incb da1)
+x /1xh &da1                        volcar 1 media word a partir de da1
0x804a000:      0x0b0b
+n                                  ejecutamos siguiente instrucción (incw da2)
+n                                  ejecutamos siguiente instrucción (incl da4)
+p /x $eax                          imprimir contenido registro eax, en hexadecimal
$24 = 0xa
+n                          ejecutamos siguiente instrucción (mov $SYS_EXIT, %eax)
+p /x $eax                          imprimir contenido registro eax, en hexadecimal
$25 = 0x1
+p /x $ebx                          imprimir contenido registro ebx, en hexadecimal
$26 = 0x0
+n                          ejecutamos siguiente instrucción (mov $SUCCESS, %ebx)
+p /x $ebx                          imprimir contenido registro ebx, en hexadecimal
$27 = 0x0
+n                                  ejecutamos siguiente instrucción (int $0x80)
[Inferior 1 (process 5896) exited normally]
+quit                               salir de depuracion
```

A continuación, se muestra el `.txt` generado a partir de la depuración del módulo `datos_direccionamiento.s`. Cabe destacar que también se adjunta los comandos, junto a su salida, con sus respectivos comentarios (en rojo).

`datos_direccionamiento.s` se trata de un programa en el que, partiendo de unas variables declaradas en la sección de datos, se realizan diferentes tipos de direccionamientos como los comentados anteriormente (direccionamiento inmediato, indrecto, inmediato, relativo a pc..)

En la salida de la depuración, adjuntada a continuación, antes de ejecutar una instrucción se imprime el contenido inicial de los registros antes de ser ejecutada dicha instrucción e inmediatamente posterior a ser ejecutada dicha instrucción. Esto hace mas fácil se entender los distintos tipos de direccionamientos y la forma en la que trabaja la máquina.

```
+file datos_direccionamiento                cargamos modulo datos_direccionamiento
Reading symbols from datos_direccionamiento...
+b main            punto de interrupción en etiqueta main
Punto de interrupción 1 at 0x118d: file datos_direccionamiento.s, line 28.
+run                        ejecutamos la depuracion
Starting program: /home/sayechu/Escritorio/EECC/P2/Ej3/datos_direccionamiento
[Depuración de hilo usando libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".


Breakpoint 1, main () at datos_direccionamiento.s:28
+x /xh &da2                 volcar media word en formato hexadecimal a partir de da2
0x56559008:     0x0a0b
+x /4xh &da2                volcar 4 medias word en formato hex. a partir de da2
0x56559008:     0x0a0b  0x0f5c  0xffeb  0xffff
+p /x (short[4])da2                 imprimir contenido de 4 elementos de array da2
$1 = {0xa0b, 0xf5c, 0xffeb, 0xffff}
+x /4xh (short *)&da2       volcar 4 elementos de 2 bytes del array da2
0x56559008:     0x0a0b  0x0f5c  0xffeb  0xffff
+x /1xb &da2                        volcar 1 byte en formato hex. a partir de da2
0x56559008:     0x0b
+x                                  volcar otro byte -> SE OBSERVA LITTLE ENDIAN
0x56559009:     0x0a
+x                                  volcar otro byte -> SE OBSERVA LITTLE ENDIAN
0x5655900a:     0x5c
+x                                  volcar otro byte -> SE OBSERVA LITTLE ENDIAN
0x5655900b:     0x0f
+x /dw (short *)&da2+2     tercer elemento de da2 (-21) en formato signed dec.
0x5655900c:     -21
+disas salto1                       desensamblar etiqueta salto1
Dump of assembler code for function salto1:
```

```
   0x565561da <+0>:     mov    $0x1,%eax
   0x565561df <+5>:     mov    $0x0,%ebx
   0x565561e4 <+10>:    int    $0x80
End of assembler dump.
+disas /r salto1              desensamblar instrucciones etiqueta salto1
Dump of assembler code for function salto1:
   0x565561da <+0>:     b8 01 00 00 00    mov    $0x1,%eax
   0x565561df <+5>:     bb 00 00 00 00    mov    $0x0,%ebx
   0x565561e4 <+10>:    cd 80     int    $0x80
End of assembler dump.
+n                            ejecutar siguiente instruccion (xor %eax, %eax)
+n                            ejecutar siguiente instruccion (xor %ebx, %ebx)
+n                            ejecutar siguiente instruccion (xor %ecx, %ecx)
+n                            ejecutar siguiente instruccion (xor %edx, %edx)
+n                            ejecutar siguiente instruccion (xor %esi, %esi)
+n                            ejecutar siguiente instruccion (xor %edi, %edi)
+p /x $eax                    imprimir contenido registro eax en formato hexadecimal
$2 = 0x0
+p /x $ebx                    imprimir contenido registro ebx en formato hexadecimal
$3 = 0x0
+p /x $ecx                    imprimir contenido registro ecx en formato hexadecimal
$4 = 0x0
+p /x $edx                    imprimir contenido registro edx en formato hexadecimal
$5 = 0x0
+p /x $esi                    imprimir contenido registro esi en formato hexadecimal
$6 = 0x0
+p /x $edi                    imprimir contenido registro edi en formato hexadecimal
$7 = 0x0
+p /x $si          imprimir contenido registro si en formato hexadecimal
$8 = 0x0
+n                            ejecutar siguiente instrucción (mov $4, %si)
bucle () at datos_direccionamiento.s:40
+p /x $si          imprimir contenido registro si en formato hexadecimal
$9 = 0x4
+n                            ejecutar siguiente instrucción (add lista(,%esi,2),%di)
+p /x $di          imprimir contenido registro di, en hexadecimal
$10 = 0x5                     SE LE HA PASADO EL ELEMENTO DE LA ARRAY DE ULTIMA POSICION
+p /x $si          imprimir contenido registro si en formato hexadecimal
$11 = 0x4
+n                            ejecutar siguiente instrucción (dec %si)
```

```
+n                          ejecutar siguiente instrucción (jns bucle)
+n                          ejecutar siguiente instrucción (add lista(,%esi,2),%di)
+p /x $di          imprimir contenido registro di, en hexadecimal
$12 = 0x9                   SE LE HA PASADO EL ELEMENTO DE LA ARRAY DE PENULTIMA POS.
+p /x $si          imprimir contenido registro si en formato hexadecimal
$13 = 0x3
+n                          ejecutar siguiente instrucción (dec %si)
+p /x $si          imprimir contenido registro si, en hexadecimal
$14 = 0x2
+n                          ejecutar siguiente instrucción (jns bucle)
+n                          ejecutar siguiente instrucción (add lista(,%esi,2),%di)
+p /x $di          imprimir contenido registro di, en hexadecimal
$15 = 0xc                   SE LE HA PASADO EL ELEMENTO DEI ARRAY CUYA POS ES 3. ADD.
+p /x $si          imprimir contenido registro si, en hexadecimal
$16 = 0x2
+n                          ejecutar siguiente instrucción (dec %si)
+p /x $si          imprimir contenido registro si, en hexadecimal
$17 = 0x1                   como contenido de registro si, not signed entramos bucle
+n                          ejecutar siguiente instrucción (jns bucle)
+n                          ejecutar siguiente instrucción (add lista(,%esi,2),%di)
+p /x $di          imprimir contenido registro di, en hexadecimal
$18 = 0xe                   SE LE HA PASADO EL ELEMENTO DEI ARRAY CUYA POS ES 2. ADD.
+p /x $si          imprimir contenido registro si, en hexadecimal
$19 = 0x1
+n                          ejecutar siguiente instrucción (dec %si)
+p /x $si          imprimir contenido registro si, en hexadecimal
$20 = 0x0                   SIGUE SIENDO NOT SIGNED, ENTRAR BUCLE
+n                          ejecutar siguiente instrucción (jns bucle)
+n                          ejecutar siguiente instrucción (add lista(,%esi,2),%di)
+p /x $di          imprimir contenido registro di, en hexadecimal
$21 = 0xf                   SE LE HA PASADO EL ELEMENTO DEI ARRAY CUYA POS ES 1. ADD.
+p /x $si          imprimir contenido de registro si antes de decremento
$22 = 0x0
+n                          ejecutar siguiente instrucción (dec %si)
+p /x $si          imprimir valor registro si, (-1 en decimal)
$23 = 0xffff
+n        ejecutar siguiente instrucción (jns bucle) como es signed, no salta

+p /a &buffer               imprimir dirección efectiva etiqueta buffer
$26 = 0x56559020
```

```
+p /x $eax                  imprimir contenido registro eax
$27 = 0x0
+n                          ejecutar siguiente instruccion (lea buffer, %eax)
+p /x $eax                  imprimir contenido registro eax tras ultima instruccion
$28 = 0x56559020            el valor del reg. Eax es la dirección efectiva de buffer
+p /x $bx                   imprimir contenido registro bx en formato hexadecimal
$29 = 0x0
+n                          ejecutar siguiente instrucción (mov da2, %bx)
+p /x $bx                   imprimir contenido registro bx tras ultima instrucción
$30 = 0xa0b                 registro bx 2 bytes, se han movido 2 bytes de da2 a bx
+x /2xb &da2                volcar 2 bytes en hexadecimal a partir de da2
0x56559008:     0x0b    0x0a    son los 2 bytes que se han movido a registro bx
+x /1xh &da2                volcar 1 media word a partir de da2
0x56559008:     0x0a0b          igual que el nuevo contenido registro bx
+x /1xh &buffer    imprimir 1 media word de buffer
0x56559020:     0x0000
+n      siguiente instruccion (mov %bx, (%eax)), Mueve contenido de registro bx a la dirección
almacenada en registro eax. El contenido del registro eax es la dirección efectiva de buffer por lo que se
ha movido a buffer.
+x /1xh &buffer    veamos lo que se ha movido en la ultima instrucción con direccionamiento indirecto
0x56559020:     0x0a0b  se ha movido indirectamente el contenido del registro bx
+x /1xh &da2                volcar 1 media word a partir de da2, en formato hex
0x56559008:     0x0a0b
+n                 siguiente instrucción (incw da2)
+x /1xh &da2                volcamos de nuevo 1 media word para ver resultado instruc.
0x56559008:     0x0a0c          se ha incrementado la MSB en 1
+p /x $ebx                  imprimir contenido registro ebx antes de ejecutar instruc
$31 = 0xa0b
+p /a &da2                  imprimir ADDRESS de da2
$32 = 0x56559008
+n                          ejecutar siguiente instrucción (lea da2, %ebx)
+p /x $ebx                  imprimir nuevo contenido registro ebx
$33 = 0x56559008            ebx contiene la dirección efectiva de da2
+n                          ejecutar siguiente instrucción (incw 2(%ebx))
+p /x $ebx                  imprimir contenido registro ebx en hexadecimal
$36 = 0x56559008
+p /x $esi                  imprimir contenido registro esi en hexadecimal
$37 = 0xffff
+n                          ejecutar siguiente instrucción (mov $3, %esi)
+p /x $esi                  imprimir nuevo valor de registro esi tras ultima instr.
```

```
$38 = 0x3
+p /x $ebx                imprimir contenido registro ebx en hex.
$39 = 0x56559008
+n                        ejecutar siguiente instrucción (mov da2(,%esi,2),%ebx)
+p /x $ebx                imprimir nuevo contenido registro ebx tras ultima instr.
$40 = 0x1ffff
+n                        ejecutar siguiente instrucción (jmp salto1)
salto1 () at datos_direccionamiento.s:75
+p /x $eax                imprimir contenido registro eax en hex.
$42 = 0x56559020
+n                        ejecutar siguiente instrucción (mov $SYS_EXIT, %eax)
+p /x $eax                imprimir nuevo contenido registro eax tras ultima instruc.
$43 = 0x1                         SYS_EXIT es una MACRO con valor 1
+p /x $ebx                imprimir contenido registro ebx en hex.
$44 = 0x1ffff
+n                        ejecutar siguiente instrucción (mov $SUCCESS, %ebx)
+p /x $ebx                imprimir nuevo valor registro ebx
$45 = 0x0                         SUCCESS es una MACRO con valor 0
+n                        ejecutar siguiente instrucción (int $0x80) -> llamada sist
[Inferior 1 (process 3950) exited normally]
+quit                     salir de depuración del programa datos_direccionamiento
```

Conclusiones

Se comenzó viendo diferentes definiciones de variables locales como strings, arrays, etc en el lenguaje ASM. También se ha introducido el concepto de MACRO. A lo largo de esta práctica se han utilizado dos macros (SYS_EXIT Y SUCCESS), y, como ya se ha comentado a lo largo de la práctica, las MACROS en el preprocesamiento del programa son sustituidas en las instrucciones por su correspondiente valor asociado en la sección de datos.

A continuación, en el segundo programa de datos_sufijos.s se ha visto la importancia de la utilización de sufijos en los mnemónicos ya que en algunos casos nos pueden dar un error de ensamblaje. Como ya se ha dicho en la práctica, en caso de que el mnemónico no tenga un sufijo, es el tamaño del registro quien especifica el tamaño de los operandos (tanto el operando fuente como el operando destino). Y, por el contrario, en caso de tener sufijo, el mnemónico, es dicho sufijo quien especifica el tamaño de los operandos fuente y destino.

Por último, en el módulo datos_direccionamiento.s se han mezclado los conceptos vistos hasta este punto, es decir, se han ejecutado diferentes instrucciones con sufijos y con distintos tipos de estructuras como con array. También se introducen nuevos

conceptos como los diferentes tipos de direccionamiento, entre los que cabe destacar por su uso el inmediato, el indirecto y el relativo al PC.