Portafolios de prácticas de EC

Por Francisco Navarro Morales

Práctica 2

Ejercicio 5.1

Comprendiendo el funcionamiento de suma.

A continuación un seguimiento de cómo puedo comprobar el funcionamiento de la pila y las ordenes call y ret.

En primer lugar hay que tener en cuenta que si ejecutamos 0x80 realizamos una subrutina de servicio, que leerá de EAX el número de servicio a realizar, y utilizará como ‘‘argumentos’’ el contenido de los registros EBX, ECX, EDX...

Para ver el funcionamiento de la pila:

1. Compilamos el programa suma.s con opciones de debug: ‘‘as --32 -g suma.s o suma.o’’ + ‘‘ld -m elf_i386 suma.o -o suma’’ y ejecutamos con ddd (‘‘ddd suma’’)

2. Establecemos un breakpoint al comienzo el programa para que este se detenga al lanzarlo con run. Avanzamos instrucción a instrucción con ‘‘step’’. Las primera instruccione simplemente colocan los ‘‘argumentos’’ de la función suma en su sitio, (registros ebx, ecx...). Al llegal a la llamada al sistema (Call suma) comprobamos la dirección de la instrucción que sigue a Call suma, que en este caso es 0x08048099, con la llamada al sistema, esta dirección queda almacenada en la pila.

3. Podemos comprobar los valores de la pila con el menú ‘‘Data->memory’’ y rellenando:
   examine: 5 (por ejemplo) , hex, bytes, from $esp
   El valor que habrá en el tope de la pila será precisamente 0x08048099, cuando termine de ejecutarse la función suma, la llamada a ‘‘ret’’ devolverá la ejecución a esta dirección y la eliminará de la pila.

4. Respecto a la funcion suma:

```
suma:
        push %edx
        mov $0, %eax
        mov $0, %edx
bucle:
        add (%ebx,%edx,4), %eax
        inc       %edx
        cmp  %edx,%ecx
        jne bucle

        pop %edx
        ret
```

push %edx almacena el valor de edx en la pila, en este caso %edx no contiene nada importante, pero si tuviera algún valor no querríamos perderlo, y dado que vamos a utilizar este registro como índice del bucle, conviene guardar su valor.
eax será el registro donde se guarde el resultado de la función (ocurre así por defecto); por ello, y dado que ya se ha realizado la llamada a la subrutina y el código contenido en eax no

nos es útil, lo ponemos a 0, (al igual que edx, que como hemos dicho, va a ser el índice del bucle) para ir sumando sobre ese valor 0.

5. respecto a instrucción add(%ebx,%edx,4), %eax ; esta instrucción tiene, como la mayoría, una fuente y un destino. En este caso la fuente depende de tres ‘‘parámetros’’ (%ebx,%edx,4) estos son: el primero (%ebx) es el registro base, es decir, a partir del cual vamos a calcular la dirección que será la fuente del comando add; el segundo (%edx) será el registro índice (ya lo hemos comentado antes), indica cuantas ‘‘posiciones’’ debe incrementarse la dirección base. El tercer parámetro es un número entero y se refiere al tamaño de las posiciones que se avanza el vector. Es una constante multiplicativa que indica el tamaño del dato del vector, así en este caso vale cuatro porque se refiere a bytes, $4B = 2^3 * 4bits = 2^5 bits = 32bits$, que es el tamaño que ocupa cada número del vector. Si, por ejemplo, fuera un vector de números de 64 bits, el valor debería ser 8).

6. inc %edx solo incrementa el índice del bucle

7. las instrucciones cmp y jne hacen la comprobación del bucle, jne comprueba si los argumentos de cmp son iguales y, si no lo son, ‘‘salta’’ a la primera instrucción del bucle. En caso de que sean iguales, es decir, que el índice se corresponda con el contenido de ecx (que tenía el total de números a sumar), jne no hace nada y el programa continúa su ejecución por la siguiente instrucción.

8. finalmente pop %edx recupera el valor que edx tenía antes y ret devuelve la ejecución a la orden siguiente a ‘‘call suma’’, cuya dirección se había almacenado en la pila (también la elimina de la pila).

9. Las últimas instrucciones copian el resultado de la función en ‘‘resultado’’ (desde eax, que es donde se ha almacenado). Después resultado se añade a la pila dos veces, al igual que el formato, y se hace una llamada a la subrutina printf, que hace un pop de los tres últimos elementos de la pila, de forma que estos son ‘‘formato, resultado, resultado’’ y muestra por pantalla el resultado en ASCII decimal/hex (formato).

10. Finalmente se añaden doce posiciones (bytes) a la pila, se coloca el valor 1 en eax y el valor 0 en ebx y se llama a la subrutina _exit(0) ejecutando 0x80; es decir, el programa imprime en pantalla el resultado y devuelve un valor de retorno 0.

Sumar N enteros sin signo de 32 bits en una plataforma de 32bits sin perder precisión.

Mi código es el siguiente:

```
  # Suma.s

  .section .data
          .macro linea # (tal y como se indica en el guión)
                #.int 0xFC000000, 0xFC000000, 0xFC000000, 0xFC000000
                .int 1,1,1,1
                #.int 2,2,2,2
                #.int 1,2,3,4
                #.int -1,-1,-1,-1
          .endm

lista:    .irpc i,12345678
          linea
          .endr

longlista:    .int (.-lista)/4

resultado:    .quad 0x0123456789ABCDEF
```

```
        #  FE CD AB 89 67 45 23 01
        # 100 101 102..... (little endian)
  #asigna a los menos significativos posiciones de memoria menores

formato: .ascii "suma = %llu = %llx \n\0"


  .section .text

main: .global main

  /** ebx es el puntero al array (primera posición)
   ** ecx es una especie de "total utilizados" del array
   ** call suma llama a la función y guarda la dirección
   ** de la siguiente instrucción en stack
  */

  mov $lista, %ebx
  mov longlista, %ecx
  call suma

  /**
   ** $eax contiene la parte más significativa de resultado,
   ** %edx la menos significativa. Para concatenar %eax y %ebx
   ** en resultado movemos la parte más significativa a resultado
   ** y luego la menos significativa 4B por delante (resultado + 4 )
   ** Como no existe push de 64 bits, tenemos que hacer dos push para
   ** meter resultado en la pila, como además el formato de printf
   ** requiere que resultado esté dos veces en la pila, hacemos un
   ** total de 4 push. Push del formato  y finalmente llamamos a printf.
   ** tras ejecutar printf hay que retirar las dos instancias de resultado
   ** y el formato de la pila, como esto ocupa 12 Bytes (4B cada dato) hacemos
   ** add $12 al puntero de la pila %esp
  **/

  mov %eax, resultado
  mov %edx, resultado+4
  push resultado+4
  push resultado
  push resultado+4
  push resultado
  push $formato
  call printf
  add $12, %esp

  //  Preparamos la salida, return(0) y llamamos a la subrutina adecuada

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


  /**
    * cargamos %esi en pila para conservar su valor
    * ponemos a 0 los registros que vamos a usar: %eax y %edx para resultado
    * %esi será el índice de lectura
```

```
  **/
suma:
  push %esi
  mov $0, %edx
  mov $0, %esi
  mov $0, %eax


  /**
   * suma el elemento numero  %esi del array apuntado por %ebx en %eax
   * comprueba si hay acarreo, si no lo hay salta la siguiente instrucción.
   * En caso de que sí haya acarreo, incremente %edx
  **/
bucle:
  add (%ebx, %esi, 4), %eax
  jnc endif
  inc %edx

  /**
   * incrementa el indice de lectura y lo compara con %ecx,
   * que tiene el total de iteraciones si coincide, jne no
   * tiene efecto y se continua por la instrucción pop %esi,
   * si no coincide se repite el bucle (salta a la etiqueta bucle)
   * Pop %esi recupera el valor que tenía el registro
  **/
endif:
  inc %esi
  cmp %esi, %ecx
  jne bucle

  pop %esi
  ret
  // retorno de ejecución a la siguiente instrucción a call suma.
```

Ejercicio 5.2

He reutilizado el código anterior cambiando algunos aspectos:

- Ahora el formato devuelto en el printf es lld
- La función suma ahora necesita utilizar los registros edx, esi, edi y eax, por lo que no he podido utilizar %esi como ‘‘índice’’ del bucle. En su lugar he utilizado el propio %ecx, que tiene el máximo de iteraciones, decrementandolo una unidad al principio para evitar que acceda a la posición vector[longitud_lista] (que produciría un segmentation fault) y después decremento dicho valor en cada iteración (la suma final es igual independientemente de si se suman las posiciones de la 0 a la longitud-1 que al revés).
- He optado por la segunda opción propuesta, usando la instrucción CDQ para extender el signo de %eax en %edx. Lo que hago ahora es mover a %eax el dato a sumar, extender su signo a %edx y luego sumar en %esi y %edi, la parte más significativa con adc y la menos con add.
- También he modificado la forma de hacer la comprobación para salir del bucle, esta vez he usado je en lugar de jne. Además, he dejado el decremento después de la comprobación. Esto se debe a que sí debe iterar cuando $ecx vale 0, pero no cuando vale -1. Por eso primero se ejecuta el cuerpo del bucle, después se comprueba si %ecx vale 0, y por último se decrementa el índice, solo si %ecx no valía cero. En el momento en que la comparación sea true (%ecx == 0) no se hará el decremento y se saltará a ‘‘zero:’’
- También he tenido que pasar después %esi a %eax y %edi a %edx para que todo funcione correctament

El código es:

```
   # Suma.s

  .section .data
          .macro linea # (tal y como se indica en el guión)
                #.int 0xFC000000, 0xFC000000, 0xFC000000, 0xFC000000
                #.int 1,1,1,1
                #.int 2,2,2,2
                #.int 1,2,3,4
                 .int -1,-1,-1,-1
          .endm

lista:    .irpc i,12345678
          linea
          .endr

longlista:    .int (.-lista)/4

resultado:    .quad 0x0123456789ABCDEF

        #  FE CD AB 89 67 45 23 01
        # 100 101 102..... (little endian)
  #asigna a los menos significativos posiciones de memoria menores

formato: .ascii "suma = %lld = %llx \n\0"


  .section .text

main: .global main

  /** ebx es el puntero al array (primera posición)
   ** ecx es una especie de "total utilizados" del array
```

```
   ** call suma llama a la función y guarda la dirección
   ** de la siguiente instrucción en stack
  */

  mov $lista, %ebx
  mov longlista, %ecx
  call suma


  mov %eax, resultado
  mov %edx, resultado+4
  push resultado+4
  push resultado
  push resultado+4
  push resultado
  push $formato
  call printf
  add $12, %esp
  //  Preparamos la salida, return(0) y llamamos a la subrutina adecuada

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


  /**
    * ponemos a 0 los registros que vamos a usar: %eax y %edx para resultado
    * %esi será el índice de lectura
  **/
suma:
  mov $0, %edx
  mov $0, %esi
  mov $0, %edi
  mov $0, %eax
  dec %ecx


  /**
   * (se suma de la posición %ecx -1 a la 0, restando)
   * pone el elemento numero  %ecx del array apuntado por %ebx
   * en %eax, extiende el signo de %eax en %edx:%eax y después
   * mueve %eax a %esi y %edx a %edi
   * resultado = %edi:%esi
  **/
bucle:
  mov (%ebx,%ecx,4), %eax
  cdq
  // extensión de signo de %eax en %edx:%eax
  add %eax, %esi
  adc %edx, %edi
  cmp $0, %ecx
  je zero
  dec %ecx
  jmp bucle

zero:
```

```
  mov %esi, %eax
  mov %edi, %edx
  ret
  // retorno de ejecución a la siguiente instrucción a call suma.
```

Ejercicio 5.3

Para calcular la media basta con dividir el resultado obtenido de la suma entre longlist. Dado que se trata de números con signo, utilizaré la instrucción DIV longlista,
que realiza la división EDX:EAX/longlista y almacena el cociente de la división en EAX y el cociente en EDX, que precisamente son los registros que la llamada a suma "devuelve"
y que el programa lee al finalizar.
En principio parece que no hay que hacer más cambios. El programa final se queda así:

```
  # Suma.s

  .section .data
          .macro linea # (tal y como se indica en el guión)
                #.int 0xFC000000, 0xFC000000, 0xFC000000, 0xFC000000
                 .int 1,-2,1,-2
                #.int 1,1,1,1
                #.int 2,2,2,2
                #.int 1,2,3,4
                #.int -1,-1,-1,-1
          .endm

lista:    .irpc i,12345678
          linea
          .endr

longlista:    .int (.-lista)/4

resultado:    .quad 0x0123456789ABCDEF

        #  FE CD AB 89 67 45 23 01
        # 100 101 102..... (little endian)
  #asigna a los menos significativos posiciones de memoria menores

formato: .ascii "media = \n\tcociente: %8d = %0x%08x \n\tresto: %11d = %0x%08x\n\0 "


  .section .text

main: .global main

  /** ebx es el puntero al array (primera posición)
   ** ecx es una especie de "total utilizados" del array
   ** call suma llama a la función y guarda la dirección
   ** de la siguiente instrucción en stack
  */

  mov $lista, %ebx
  mov longlista, %ecx
  call suma
```

```
  mov %eax, resultado
  mov %edx, resultado+4
  push resultado+4
  push resultado+4
  push resultado
  push resultado
  push $formato
  call printf
  add $12, %esp
  //  Preparamos la salida, return(0) y llamamos a la subrutina adecuada

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


  /**
    * ponemos a 0 los registros que vamos a usar: %eax y %edx para resultado
    * %esi será el índice de lectura
  **/
suma:
  mov $0, %edx
  mov $0, %esi
  mov $0, %edi
  mov $0, %eax
  dec %ecx


  /**
   * (se suma de la posición %ecx -1 a la 0, restando)
   * pone el elemento numero  %ecx del array apuntado por %ebx
   * en %eax, extiende el signo de %eax en %edx:%eax y después
   * mueve %eax a %esi y %edx a %edi
   * resultado = %edi:%esi
  **/
bucle:
  mov (%ebx,%ecx,4), %eax
  cdq
  // extensión de signo de %eax en %edx:%eax
  add %eax, %esi
  adc %edx, %edi
  cmp $0, %ecx
  je zero
  dec %ecx
  jmp bucle

zero:
  mov %esi, %eax
  mov %edi, %edx
  idivl longlista
  ret
  // retorno de ejecución a la siguiente instrucción a call suma.
```