Vicente Martínez Rodríguez

5.1 Suma de N enteros sin signo de 32bits.

```
.section .data

    .macro linea
                .int 1,1,1,1
        #       .int 2,2,2,2
        #       .int 1,2,3,4
        #       .int -1,-1,-1,-1
        #       .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
        #       .int 0x08000000,0x08000000,0x08000000,0x08000000
        #       .int 0x10000000,0x20000000,0x40000000,0x80000000
     .endm
lista: .irpc i,12345678
                linea
        .endr
longlista:      .int (.-lista)/4        # Numero de enteros que se suman

resultado:      .quad-1                 # Obtiene los registros que tienen el resultado en direcciones
consecutivas


.section .text
_start:  .global _start

        mov $lista, %ebx                # Almacenamos en ebx el contenido lista
        mov longlista, %ecx             # Almacenamos en ecx longlista
        call suma                       # Llamamos a la función suma
        mov %eax, resultado             # Movemos la primera parte de la suma a resultado
     mov %edx, resultado+4              # Movemos el resto de la suma a resultado a la parte alta ya que es
little-endian

        mov $1, %eax                    # Le introducimos al registro eax el valor 1
        mov $0, %ebx                    # Le introducimos al registro ebx el valor 0
        int $0x80                       # Llamamos a exit

suma:
        push %esi                       # Guardamos el contenido anterior de esi
        mov $0, %eax                    # Y le asignamos a los tres registros el valor 0
        mov $0, %edx                    # Acumula el acarreo
        mov $0, %esi                    # El registro contador
bucle:
        add (%ebx,%esi,4), %eax         # Vamos sumando el registro que toque a eax
  jnc no_hay_acarreo                    # Si no hay acarreo llamamos a no_hay_acarreo
        inc   %edx                      # Incrementamos el registro edx si hubiera acarreo
no_hay_acarreo:
  inc   %esi                            # Incrementamos el registro contador esi
        cmp  %esi,%ecx                  # Comprobamos si el contado ha llegado al final
        jne bucle                       # Si esi y ecx no son iguales significa que no hemos llegado al
                                        # final de bucle asi que volvemos a llamar a bucle
        pop %esi                        # Finalmente si no llama al bucle significa que ha terminado el
                                        # el bucle por lo que se restaura esi
        ret
```

5.2 Suma de N enteros con signo de 32bits.

```
.section .data

    .macro linea
            .int 1,-2,1,-2
        #   .int 2,2,2,2
        #   .int 1,2,3,4
        #   .int -1,-1,-1,-1
        #   .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
        #   .int 0x08000000,0x08000000,0x08000000,0x08000000
        #   .int 0x10000000,0x20000000,0x40000000,0x80000000
     .endm
lista: .irpc i,12345678
            linea
       .endr

longlista:      .int (.-lista)/4

resultado:      .quad-1


.section .text
_start:  .global _start

        mov $lista, %ebx            # Almacenamos en %ebx el contenido lista
        mov longlista, %ecx         # Almacenamos en %ecx longlista
        call suma                   # Llamamos a la función suma
        mov %edi, resultado         # Movemos la primera parte de la suma a resultado
        mov %ebp, resultado+4       # Movemos el resto de la suma a resultado en la parte alta

        mov $1, %eax                # Le introducimos al registro %eax el valor 1
        mov $0, %ebx                # Le introducimos al registro %ebx el valor 0
        int $0x80                   # Llamamos a exit(0)

suma:
        push %esi                   # Guardamos el posible contenido anterior de %esi
        mov $0, %esi                # Y le asignamos a los cuatro registros el valor 0
        mov $0, %eax
        mov $0, %edx                # El registro %edx se encargar de acumular los acarreos
        mov $0, %edi                # Ponemos a 0 el registro en el que almacenamos los primeros 32 bits
bucle:
        add (%ebx,%esi,4), %eax     # Vamos sumando el registro que toque a %eax
         cdq                        # Extension de signo para %eax, EDX:EAX m ExtSigno(EAX)
        add %eax, %edi              # Sumamos en %edi los 32 primeros bits
        adc %edx, %ebp              # Sumamos con acarreo, si hay, de los siguientes 32 bits en la pila

        inc  %esi                   # Incrementamos el registro contado %esi
        cmp  %esi,%ecx              # Comparamos para ver si ha llegado al final el bucle
        jne bucle                   # Si no ha llegado al final saltamos de nuevo al inicio del bucle

        pop %esi                    # En caso contrario restauramos %esi y hacemos un return
        ret
```

5.3 Media de N enteros con signo de 32bits.

```
.section .data
        .macro linea
                .int 1,1,1,1
         #      .int 2,2,2,2
         #      .int 1,2,3,4
         #      .int -1,-1,-1,-1
         #      .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
         #      .int 0x08000000,0x08000000,0x08000000,0x08000000
         #      .int 0x10000000,0x20000000,0x40000000,0x80000000
        .endm
lista:  .irpc i,12345678
                linea
        .endr

longlista:      .int (.-lista)/4    # Numero de enteros que se suman
resultado:      .int -1             # El resultado es un int porque al hacer la media no hace falta más bits


.section .text
_start: .global _start

        mov $lista, %ebx                    # Almacenamos en ebx el contenido lista
        mov longlista, %ecx                 # Almacenamos en ecx longlista
        call suma                                   # Llamamos a la función suma

        mov %eax, resultado                 # Movemos la primera parte de la suma a resultado
        mov %edx, resultado+4           # Movemos el resto de la suma a resultado a la parte alta (little-endian)

        mov $1, %eax                        # Le introducimos al registro eax el valor 1
        mov $0, %ebx                        # Le introducimos al registro ebx el valor 0
        int $0x80                                   # Llamamos a exit(0)

suma:
        push %esi                           # Guardamos el posible contenido anterior de %esi, %ebp y
%edi
        push %ebp
        push %edi
        mov $0, %eax                        #Iniciamos a 0 todos los registros
        mov $0, %edx
        mov $0, %esi
        mov $0, %ebp
        mov $0, %edi
bucle:
        add (%ebx,%esi,4), %eax         # De la lista vamos añadiendo en eax el número correspondiente
        cdq                                 # Extension de signo
        add %eax, %ebp                      # Sumamos la parte baja del número en %ebp
        adc %edx, %edi                      # Sumamos la parte alta del número y el acarreo en %edi
        mov $0, %eax                        # Limpiamos %eax y %edx para una posible siguiente
iteracion
        mov $0, %edx
        inc %esi                            # Incrementamos el contador

        cmp %esi,%ecx                       # Comparamos para ver si ha llegado al final el bucle
        jne bucle                           # Si no ha llegado al final saltamos de nuevo al inicio del bucle

        mov %ebp, %eax                      # Si se terminase el bucle, movemos el resultado %ebp a %eax
        mov %edi, %edx                      # y el %edi a %edx
        cdq                                 # Extensión de signo
```

```
idiv %ecx                    # Hacemos idiv de %ecx, que es la longitud de la lista
pop %ebp                     # Restauramos el valor de %ebp
pop %edi                     # Restauramos el valor de %edi
pop %esi                     # Restauramos el valor del registro contador
ret
```

Preguntas de Autocomprobación.

→ Sesión de depuración saludo.s

1.

EDX contiene 0x1C = 28, se utiliza para realizar la interrupción de llamada al sistema. Presenta el tamaño que ocupa la etiqueta saludo.

2.

ECX contiene 0x8049098 = 134516888, contiene la dirección de inicio de la cadena de caracteres de saludo.

El programa comienza en 0x8048074, saludo comienza en 0x8049098 y el tope de la pila está en 0xffffd250.

3.

Copia el primer elemento del array de caracteres en ecx, interpreta el char como entero; ecx contiene 0x616c6f48.

4.

Saludo ocupa 2 words y longsaludo ocupa 1 word.

6.

Haciendo obdjump -d saludo

mov $1, %ebx → ocupa 5 posiciones

```
08048074 <_start>:
8048074:    b8 04 00 00 00          mov    $0x4,%eax
8048079:    bb 01 00 00 00          mov    $0x1,%ebx
804807e:    b9 98 90 04 08          mov    $0x8049098,%ecx
```

80848079 - 80848074 = 5

7.

Si se elimina la primera instrucción int 0x80 no se mostraría por pantalla el la cadena de caracteres.

Si se elimina la segunda instrucción int 0x80 no se haría la llamada a la función exit(0).

8.

El número es 3, esto lo podemos mediante:

cat /usr/include/asm/unistd_32.h

→ #define __NR_read 3

→ Sesión de depuración suma.s

1.

%eax vale 37 es la suma de 1+2+10+1+0b10(que vale 2)+1+2+0x10(que vale 16)

(.-lista)/4 es el numero de elementos(9) entre 4 ya que cada elemento ocupa cuatro posiciones de memoria.

2.

Se obtiene 0xfffffffd, que es diferente porque se producen acarreos.

Valores que toma eax:

-Hexadecimal: 0xffffffff, 0xfffffffe, 0xfffffffd

-Decimal: -1 , -2 , -3

3.

La etiqueta suma tiene la dirección 0x8048095 y la etiqueta bucle tiene la dirección 0x80480a0.

Esto se obtiene desensamblando el programa suma con objdump -t suma

4.

%esp es el puntero a pila y %eip es el contador del programa.

5.

Antes de ejecutarse call %eip vale 0xffffd2a0

Antes de ejecutarse ret %eip vale 0xffffd29c

Hay un diferencia de cuatro que corresponde a la siguiente instrucción después de ejecutarse call, es decir mov %eax, resultado

6.

Se modifican %esp y %eip, se modifican porque call llama a una etiqueta que está en otra dirección por lo que se tiene que cambiar el %eip de la siguiente instrucción, el %esp se tiene que modificar porque se necesita un nuevo marco de pila.

7.

Se modifican los mismos, %esp y %eip porque se necesita volver a la dirección que había después de la llamada y hay que volver al marco de pila anterior a la llamada de call.

10.

Se terminaría el programa, al no haber una instrucción de retorno no se realizarían las instrucciones que hay después de la call