Entrega práctica 2 Estructura Computadores

Alejandro Poyatos López 2ºB(2)

Para el ejercicio 5.1 suma de 64 bits sin signo he usado el siguiente código

```
.section .data
        .macro linea
        # .int 1,1,1,1                          //Declaración de todas las variables que vamos a proceder a sumar
        # .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 0x123456789ABCDEF                  //variable de 64 bits que contiene el valor de la operación suma
salida:  .ascii "resultado = %llu\n\0"              //variable que se va a imprimir por terminal
.section .text
main:   .global main
        mov    $lista, %ebx                         //operación que mete el puntero de la lista en el registro ebx
        mov longlista, %ecx                         //Introducir en el registro ecx el numero de elementos que es un valor
        call suma                                   //salto a suma
        mov %eax, resultado
        mov %edx, resultado+4
        push resultado+4
        push resultado
        push resultado+4
        push resultado
        push $salida
        call printf
        add $20, %esp
```

```
        mov $1, %eax
        mov $0, %ebx
        int $0x80
suma:
        mov $0, %eax                    //reiniciar acumulador
        mov $0, %edx                    //reiniciar contador
        mov $0, %esi                    //reiniciar indice
bucle:
        add (%ebx,%esi,4), %eax         //Comienzo de las operaciones que realizan la suma
        jne wthcarry                    //salto a withoutcarry si no ha habido acarreo
        inc %edx                        //si no hay salto contaremos el acarero
wthcarry:
        inc  %esi                       //aumenta el valor del contador
        cmp  %esi,%ecx                  //comprueba que no haya llegado al final
        jne bucle                       //salto a bucle si no lo ha hecho la flag
        ret                             //cuando termine retorna al main
```

Obteniendo los siguientes resultados, en base a descomentar y comentar la líneas de int para obtener los resultados:

[1, …] : 32

[2, …] : 64

[1, 2, 3, 4, 1, …] : 80

[0xFFFFFFFF, …] : 4294967264

[0x08000000, …] : 4294967296

[0x10000000, 0x20 …, 0x40…,0x80…,0x10] : 15032385536

Para el ejercicio 5.2 suma de 64 bits con signo, he usado el siguiente código:

```
.section .data
        .macro linea
        #       .int -1,-1,-1,-1                                        //Nueva tanda de ejemplos
        #       .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
        #       .int 1,-2,1,-2
        #       .int 1,2,-3,-4
        #       .int 0x7fffffff,0x7fffffff,0x7fffffff,0x7fffffff
        #       .int 0x80000000,0x80000000,0x80000000,0x80000000
        #       .int 0x04000000,0x04000000,0x04000000,0x04000000
        #       .int 0x08000000,0x08000000,0x08000000,0x08000000
        #       .int 0xfc000000,0xfc000000,0xfc000000,0xfc000000
        #       .int 0xf8000000,0xf8000000,0xf8000000,0xf8000000
                .int 0xf0000000,0xe0000000,0xe0000000,0xd0000000
        .endm
lista:  .irpc i,12345678
                linea
        .endr
longlista:  .int (.-lista)/4
resultado: .quad 0x123456789ABCDEF                  //variable que contiene el valor del resultado de la suma
formato:   .ascii "suma = %lld\n\0"                 //variable de formato printf que es llamada
.section .text
main:   .global main
        mov    $lista, %ebx                 //operación que mete el puntero de la lista en el registro ebx
        mov longlista, %ecx                 //Introducir en el registro ecx el numero de elementos que es un valor
        call suma
        mov %eax, resultado
        mov %edx, resultado+4
        push resultado+4
        push resultado
        push resultado+4
        push resultado
        push $formato
        call printf
        add $20, %esp
```

```
        mov $1, %eax
        mov $0, %ebx
        int $0x80
suma:
        mov $0, %edi                    //Reiniciamos el acumulador1
        mov $0, %ebp                    //Reiniciamos el acumulador2
        mov $0, %esi                    //Reiniciamos el indice
bucle:
        mov (%ebx,%esi,4), %eax         //Comienzo de las operaciones de suma
        cltd                            //Extiende el bit más significativo de EAX en los de EDX
        add %eax, %edi
        adc %edx, %ebp                  //Suma edx y ebp además de suma uno al resultado en caso CF
        inc  %esi                       //Incrementa el indice
        cmp  %esi,%ecx                  //comprueba que no hayamos llegado al final
        jne bucle                       //salto a bucle si no ha terminado
        mov %edi,%eax
        mov %ebp, %edx
        ret
```

Obteniendo los siguientes resultados, en base a descomentar y comentar la líneas de int para obtener los resultados:

[-1, …] : -32

[1, -2, 1 ,-2, …] : -16

[1, 2, -3, -4 …] : -32

[0x7FFFFFFF, …] : 68719476704 en hex: fffffffe0

[0x80000000, …] : -68719476736 en hex: fffffff000000000

[0x04000000, …]: 2147483648 en hex: 80000000

[0x08000000, …] : 4294967296 en hex 100000000

[0xFC000000, …] : -2147483648 en hex ffffffff80000000

[0xF8000000, …]: -4294967296 en hex ffffffff00000000

[0xF0000000, 0xE0…, 0xE0…, 0xD0…, 0xF0…]: -17179869184 en hex fffffffc00000000

Para el ejercicio 5.3 media de n enteros con signo he usado el siguiente código:

```
.section .data
        .macro línea
        #       .int 1,-2,1,-2                          //Primera lista de valores de donde se realizan las operaciones
        #       .int 1,2,-3,-4
        #       .int 0x7fffffff,0x7fffffff,0x7fffffff,0x7fffffff
        #       .int 0x80000000,0x80000000,0x80000000,0x80000000
        #       .int 0xf0000000,0xe0000000,0xe0000000,0xd0000000
                .int -1,-1,-1,-1
        .endm
        .macro linea0
        #       .int 0,-1,-1,-1                         //Segunda línea de valores que se toma para las operaciones
        #       .int 0,-2,-1,-1
        #       .int 1,-2,-1,-1
        #       .int 19,-2,-1,-1
        #       .int 32,-2,-1,-1
        #       .int 50,-2,-1,-1
        #       .int 63,-2,-1,-1
        #       .int 64,-2,-1,-1
        #       .int 70,-2,-1,-1
        #       .int 95,-2,-1,-1
        #       .int -31,-2,-1,-1
        #       .int -10,-2,-1,-1
                .int 0,-2,-1,-1
        .endm
lista:
                linea0
                .irpc i,1234567#8
                linea
                .endr
longlista:  .int (.-lista)/4
media:      .int 0x89ABCDEF
resto:      .int 0x01234567
formato:    .ascii "media =   %8d \n resto =   %8d \n"
.section .text
main:       .global main
```

```
        mov    $lista, %ebx
        mov longlista, %ecx
        call suma
        mov %eax, media                 //Saca los valores de los registros para prepararlos para la salida
        mov %edx, resto
        push resto                      //Los prepara para la orden printf
        push media
        push resto
        push media
        push $formato
        call printf
        add $20, %esp
        mov $1, %eax
        mov $0, %ebx
        int $0x80
suma:
        mov $0, %edi        //Reiniciamos el acumulador1
        mov $0, %ebp        //Reiniciamos el acumulador2
        mov $0, %esi        //Reiniciamos el índice
bucle:
        mov (%ebx,%esi,4), %eax         //Comenzamos a operar
        cltd                            //Extiende el bit más significativo de EAX en los de EDX
        add %eax, %edi
        adc %edx, %ebp                  //Suma edx y ebp además de suma uno al resultado en caso CF
        inc  %esi                       //Aumenta de valor el índice
        cmp  %esi,%ecx                  //Comprueba que no haya llegado al final
        jne bucle                       //salto a bucle si no ha finalizado
        mov %edi,%eax
        mov %ebp, %edx
        idiv %ecx                       //Operación que divide con signo
        ret
```

Para esta tanda de operaciones obtengo los siguientes resultados:

Ejemplo	Media	Resto
[1,-2, 1, -2…]:	0	-16
[1, 2, -3, -4, 1, …]:	-1	0
[0x7FFFFFFF, …]:	0x7FFFFFFF(hex)	0
[0x80000000, …]:	0x80000000(hex)	0
[0xF0…, 0xE0…, 0xE0…, 0xD0…, …]:	0xE0000000(hex)	0
[-1, …]:	-1	0
[0, -1, -1 , -1, …]:	0	-31
[1, -2, -1, -1]:	0	-31
[19, -2, -1 ,-1…]:	0	-13
[32, -2, -1, -1…]:	0	0
[50, -2, -1, -1, …]:	0	18
[63, -2, -1, -1, …]:	0	31
[64, -2, -1, -1,…]:	1	0
[70, -2, -1, -1,…]:	1	6
[92, -2, -1, -1,…]:	1	31
[-31, -2, -1, -1,…]:	-1	-31
[-10, -2, -1, -1,…]:	-1	-10
[0, -2, -1, -1, …]:	-1	0