Práctica 2: Programación en ensamblador X86 Linux

Nombre: Javier Martín Gómez

Curso: 2ºC

DNI: 77143672Q

Diario de trabajo

8/10/2015: Empieza las prácticas, el profesor ha usado las 2 horas para explicarla

15/10/2015: Empiezo un poco perdido sin saber cómo comenzar, al cabo de un rato voy probando con la actividad suma sin signo

22/10/2015: Después de un poco de trabajo en casa acabo la suma sin signo y comienzo la suma con signo pero me atasco un poco, pido ayuda a los compañeros

29/10/2015: Acabo la suma con signo con ayuda de compañeros y empiezo con la media también con otros compañeros, lo conseguimos acabar.

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

```
.section .data


# Para facilitarlo

.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

#lista:                 .int 1,2,10,  1,2,0b10,  1,2,0x10 #Lista antigua suma.s
longlista:   .int (.-lista)/4 # Posición - la pos de la lista para saber el tamaño, entre 4 porque son 4 números
#resultado:             .int -1 #Antiguo suma.s

# Rusultado 64 bits
resultado: .quad 0x8877665544332211 # Formato de 64 bits, dos .int, la dirección es arbitraria pero esta viene bien
formato:   .string "i:%lli / u:%llu / 0x%llx\n"


.section .text
        .extern printf # printf es una función externa

#Main para facilitarlo
main:       .global main

        mov    $lista, %ebx  # Son para salvar, guardamos lista en ebx
        mov longlista, %ecx # Guardamos el tamaño de lista en ecx
        call suma
        mov %eax, resultado #EAX Parte menos significativa 32 bits
        mov %edx, resultado+4 #EDX Parte más significativa 32 bits

        call printf2 # Hemos guardado el printf en una subrutina que nos imprime el resultado

        # Salir del programa
        mov $1, %eax
        mov $0, %ebx
        int $0x80

printf2:
        # Para printf de 64
        # Son tres formatos, de modo que tenemos que meterlo 3 veces en la pila para imprimirlo
```

```
        # Primera vez
        pushl resultado + 4 # Primero la parte significativa = 32 bits
        pushl resultado # Luego la menos significativa = 32 bits
        # Ya tenemos los 64 bits
        # Segunda
        pushl resultado + 4
        pushl resultado
        # Tercera
        pushl resultado + 4
        pushl resultado
        # Cargamos el formato
        pushl $formato
        # Imprimimos
        call printf
        # Corregimos la pila
        add $28, %esp # Sumamos 28 al puntero de pila (4 * 7 push) para saber donde estamos

        # Volvemos al main
        ret

suma:
        # Ponemos los registros que vamos a usar a 0
        mov $0, %eax # Parte menos significativa de resultado
        mov $0, %edx # Parte más significativa

        pushl %esi

        mov $0, %esi # Para el índice, la iteración
bucle:
        adc (%ebx,%esi,4), %eax # "ebx + (esi * 4)" y lo movemos a eax, adc suma con acarreo, detecta cuando hay
        desbordamiento y suma el acarreo
        inc %esi # Incrementa pos en 1
        cmp  %esi,%ecx # Comparamos el índice con el tamaño del vector para saber si hemos llegado al final
        jne bucle # Si no hemos llegado volvemos a bucle
```

```
        # Volvemos al main
        pop %esi
        ret
```

Suma N enteros con signo de 32 bits en una plataforma de 32 bits

```
.section .data

# Para facilitarlo
.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

#lista:         .int 1,2,10,  1,2,0b10,  1,2,0x10 #Lista antigua suma.s
longlista:    .int (.-lista)/4 # Posición - la pos de la lista para saber el tamaño, entre 4 porque son 4 números
#resultado:     .int -1 #Antiguo suma.s

# Rusultado 64 bits
resultado: .quad 0x8877665544332211 # Formato de 64 bits, dos .int, la dirección es arbitraria pero esta viene bien
formato:    .string "i:%lli / u:%llu / 0x%llx\n"
```

```
.section .text
        .extern printf # printf es una función externa

#Main para facilitarlo
main:   .global main

        mov    $lista, %ebx  # Son para salvar, guardamos lista en ebx
        mov longlista, %ecx # Guardamos el tamaño de lista en ecx
        call suma
        mov %eax, resultado #EAX Parte menos significativa 32 bits
        mov %edx, resultado+4 #EDX Parte más significativa 32 bits

        call printf2 # Hemos guardado el printf en una subrutina que nos imprime el resultado

        # Salir del programa
        mov $1, %eax
        mov $0, %ebx
        int $0x80

printf2:
        # Para printf de 64
        # Son tres formatos, de modo que tenemos que meterlo 3 veces en la pila para imprimirlo
        # Primera vez
        pushl resultado + 4 # Primero la parte significativa = 32 bits
        pushl resultado # Luego la menos significativa = 32 bits
        # Ya tenemos los 64 bits
        # Segunda
        pushl resultado + 4
        pushl resultado
        # Tercera
        pushl resultado + 4
        pushl resultado
        # Cargamos el formato
```

```
        pushl $formato
        # Imprimimos
        call printf
        # Corregimos la pila
        add $28, %esp # Sumamos 28 al puntero de pila (4 * 7 push) para saber donde estamos

        # Volvemos al main
        ret

suma:
        # Ponemos los registros que vamos a usar a 0
        mov $0, %eax # Parte menos significativa de resultado
        mov $0, %edx # Parte más significativa

        pushl %esi

        mov $0, %esi # Para el índice, la iteración
bucle:
        mov (%ebx,%esi,4), %eax
        cltd # Convierte 32 bits en 64 bits, duplicando el signo

        add  %eax,%edi
        adc  %edx,%ebp

        inc %esi # Incrementar índice
        cmp %esi,%ecx # Comparar índice con tamaño
        jne bucle # Si el índice no e sigual al tamaño, saltamos al bucle

        mov %edi,%eax
        mov %ebp, %edx

        pop %esi
        ret
```

Media de N enteros con signo de 32 bits en una plataforma de 32 bits

```
.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

#lista:         .int 1,2,10,  1,2,0b10,  1,2,0x10 #Lista antigua suma.s
longlista:  .int (.-lista)/4
#resultado:     .int -1 #Antiguo suma.s
media:          .int 0x89ABCDEF
resto:          .int 0x01234567

# Rusultado 64 bits
# resultado: .quad 0x8877665544332211
formato:   #.string "i:%lli / u:%llu / 0x%llx\n"
        .ascii "media= %8d \t resto= %8d \n"
        .ascii "hexadecimal= 0x%08x \t resto= 0x%08x \n\0"
```

```
.section .text
        .extern printf
#Main para facilitarlo
main:   .global main

        mov    $lista, %ebx
        mov longlista, %ecx
        call suma
        mov %eax, media # Media
        mov %edx, resto # Resto

        call printf2

        # Todo lo siguiente es para salir
        mov $1, %eax
        mov $0, %ebx
        int $0x80

printf2:
        # Para printf de 64
        push resto
        push media
        push resto
        push media
        push $formato
        call printf
        add $20, %esp
        ret

suma:
        mov $0, %edi
        mov $0, %ebp
```

```
        mov $0, %esi


bucle:
        mov (%ebx,%esi,4), %eax
        cltd
        add  %eax,%edi
        adc  %edx,%ebp
        inc %esi
        cmp %esi,%ecx
        jne bucle
        mov %edi,%eax
        mov %ebp, %edx

        idiv %ecx # Media
        ret
```