Estructura de computadores

Práctica 2: Nivel de máquina

Federico García Garrido.
fedeggj@correo.ugr.es
Grupo A2
Octubre 2018.

Universidad de Granada.
Ingeniería Informática.
Estructura de computadores

1.Suma sin signo:

La suma de dos signos en 32 bits que conlleve acarreamiento necesita de dos registros para ellos hemos utilizado dos registros de 32 bits para controlar el acarreamiento **%eax** y **%edx**.

En esta práctica hemos usado dos formas de realizar esta suma con desbordamiento, una con un salto y otra con instrucción adc que suma si el desbordamiento está activo. A parte de estos dos registros de 32 bits hemos usado:

-%rbx: para direccionar la lista

-%rcx: para la longitud de la lista

-%rsi: para el índice del bucle

-%rdi y **%rsi**: para pasar los parámetros a la función printf

Los registros **%eax** y **%edx** se mueven a resultado, al ser dos registros se guardan en resultado y resultado+4 (4*8 = 32 bits, se guarda el resultado en 64 bits).

Aparate de estos registros hemos usado unas variables lista (donde se almacenaban los datos de la lista de .int), longlista (donde se almacenaban el numero de datos en la lista), resultado (el cual es .quad para almacenar los resultados de **%eax** y **%edx**) y formato (donde almacenamos el formato de impresion que luego pasaremos como parámetro a printf)

Para la primera forma con un salto jnc el código sería:

```
21  main: .global  main
22
23      mov     $lista, %rbx # direccion de array lista
24      mov  longlista, %rcx # longitud de la lista
25      call suma       # == suma(&lista, longlista);
26      mov  %eax, resultado # salvamos el resultado
27      mov  %edx, resultado+4 # salvamos el resultado
28
29      # llamada a la función printf en c
30      mov   $formato, %rdi    # metemos los parametros de la funcion printf en sus corrrespondientes registros
31      mov   resultado,%rsi
32      mov   resultado,%rdx
33      mov   resultado,%ecx
34      mov   resultado,%r8d
35      mov          $0,%eax    # varargin sin xmm
36      call  printf        # == printf(formato, res, res);
37
38      # acabar
39      mov  resultado, %edi
40      call _exit      # ==  exit(resultado)
41
42  suma:
43
44      mov  $0, %eax # ponemos los valores a 0 antes del bucle
45      mov  $0, %rsi
46      mov  $0, %edx
47
48  bucle:
49
50      add  (%rbx,%rsi,4), %eax # vamos acumulando en eax las sumas
51      inc %rsi            # incrementamos indice
52      jnc salto           # salta si CF=0
53      inc   %edx          # incrementamos edx si hay acarreo
54      ret
55
56  salto:
57
58      cmp   %rsi,%rcx     # compara que el indice es diferente que la longitud
59      jne    bucle        # si la comparacion es diferente salta de nuevo al bucle
60
61      ret
62
```

Ilustración 1: versión con salto jnc

Vemos del código en la línea 52 como nos comemos el incremento del acarreo si no ha habido acarreo para asi controlar **%edx**. (JNC solo salta cuando el acarreo no existe CF=0)

Un código más sencillo gracias a la instrucción adc nos quedaría asi:

```
20  main: .global  main
21
22      mov     $lista, %rbx # direccion de array lista
23      mov  longlista, %rcx # longitud de la lista
24      call suma       # == suma(&lista, longlista);
25      mov  %eax, resultado # salvamos el resultado
26      mov  %edx, resultado+4 # salvamos el resultado
27
28      # llamada a la funcion printf en c
29      mov   $formato, %rdi
30      mov   resultado,%rsi
31      mov   resultado,%rdx
32      mov   resultado,%ecx
33      mov   resultado,%r8d
34      mov          $0,%eax    # varargin sin xmm
35      call  printf        # == printf(formato, res, res);
36
37      # acabar
38      mov  resultado, %edi
39      call _exit      # ==  exit(resultado)
40
41
42
43  suma:
44
45      mov  $0, %eax # ponemos los valores a 0 antes del bucle
46      mov  $0, %rsi
47      mov  $0, %edx
48
49
50  bucle:
51
52      add  (%rbx,%rsi,4), %eax # vamos acumulando en eax las sumas
53      adc $0,%edx
54      inc %rsi            # incrementamos indice
55
56
57      cmp   %rsi,%rcx     # compara que el indice es igual que la longitud
58      jne    bucle
59
60
61      ret
62
63
```

Ilustración 2: versión con suma de acarreo adc

Simplemente con la instrucción adc acumulamos el acarreo en **%edx** cuando ocurre desbordamiento en **%eax**.

Los resultados de prueba para este código son:

Lista (x4 veces cada dato)	Resultado
1,1,1,1	16 = 0x10hex
2,2,2,2	32 = 0x20 hex
1,2,3,4	40 = 0x28 hex
-1,-1,-1,-1	Aquí da fallo al estar sin signo (0xffffffff0 hex)
0x08000000, 0x08000000, 0x08000000, 0x08000000	6442450959 = 0x18000000f hex
0x10000000, 0x20000000, 0x40000000, 0x80000000	4294967296 = 0x100000000 hex

2.Suma con signo:

Para la suma con signo también hemos usado los registros **%eax** y **%edx** pero hemos usado dos registros adicionales para ir acumulando la suma (yo he usado **%r8d** y **%edi** que eran dos registros de 32 bits que no se ustaban utilizando en el momento del bucle). Para trabajar con signo lo primero que tenemos que hacer es extender el signo del número pasado a **%eax** (esto lo haremos con la instrucción cdq que nos lo hace directamente en **%eax**) y luego lo sumamos al acumulador **%r8d** y **%edx** también se lo sumamos al otro acumulador pero teniendo en cuenta el acarreo con la instrucción adc.

Después de esto pasamos los resultados de los acumuladores otra vez a **%eax** y **%edx** y de nuevo los pasamos a la variable resultado.

El código de nuestro programa:

```
28  main: .global  main
29
30
31      mov     $lista, %rbx # direccion de array lista
32      mov  longlista, %rcx # longitud de la lista
33      call suma       # == suma(&lista, longlista);
34      mov  %eax, resultado # salvamos el resultado
35      mov  %edx, resultado+4 # salvamos el resultado
36
37      # llamada a printf
38      mov   $formato, %rdi # pasamos los parámetros a printf
39      mov   resultado,%rsi
40      mov   resultado,%rdx
41      mov   resultado,%r8d
42      mov          $0,%eax    # varargin sin xmm
43      call  printf        # == printf(formato, res, res);
44
45      # acabar
46      mov  resultado, %eax
47      call _exit      # ==  exit(resultado)
48
49
50
51  suma:
52
53      mov  $0, %rsi   # inicializamos indice
54      mov  $0, %r8d # acum1
55      mov  $0, %edi # acum2
56
57
58  bucle:
59
60      mov  (%rbx,%rsi,4), %eax # vamos movien en eax los datos (eax=Lista[i])
61
62      cdq                     # extension de signo edx:eax
63      add %eax,%r8d           # r8d += eax
64      adc %edx,%edi           # edi += edx
65
66      inc %rsi            # incrementamos indice
67
68
69      cmp   %rsi,%rcx     # compara que el indice es igual que la longitud
70      jne    bucle
71
72      mov %r8d,%eax       # una vez acabado el bucle retornamos los acumuladores a %eax y %edx
73      mov %edi,%edx
74
75      ret                 # retornamos estos valores %eax y %edx
76
77
```

Ilustración 3: versión con extensión de signo cdq

Los resultados de prueba para este código son:

Lista (x4 veces cada dato)	Resultado
1,1,1,1	16 = 0x10hex
1,-2,1,-2	-8 = 0xfffffffffffffff8 hex
1,2,-3,-4	-16 = 0xfffffffffffffff0 hex
0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff	34359738352 = 0x7ffffff0
0x80000000, 0x80000000, 0x80000000, 0x80000000	-34359738368 = 0xffffffff800000000 hex
0xFC000000, 0xFC000000, 0xFC000000, 0xFC000000	-1073741824 = 0xffffffffc0000000 hex
0xF8000000, 0xF8000000, 0xF8000000, 0xF8000000	-2147483648 = 0xffffffff80000000 hex

3.Media:

Para esta modificación lo que se nos pide es obtener la media la lista de enteros. Para ello, después de devolver el resultado en **%eax** y **%edx** realizamos la división del número de datos en la lista concatenándolo con **%eax** y **%edx**, después guardamos la media y el resto en dos variables de tipo .int .

En este código he cambiado todos los registros a 32 bits ya que no nos hacían falta los de 64 bits para la división ni para guardar el resultado de la media ni el resto, por lo tanto el uso de los registros es el mismo que en los demás apartados solo que cambiando su longitud a una de 32 bits.

El único registro que he usado de 64 bits ha sido **%rdi** para meter el formato que es una variable .ascii. Esta vez en printf lo que he utilizado como parámetro han sido **%esi** para la media y directamente **%edx** para el resto de la división

El código del programa sería:

```
.section .text

main: .global  main

    mov     $lista, %ebx # direccion de array lista
    mov  longlista, %ecx # longitud de la lista
    call suma       # == suma(&lista, longlista);

    # media

    idiv %ecx           # dividimos concantenando %eax y &edx
    mov  %eax,media     # guardamos la media
    mov  %edx,resto     # guardamos el resto

    # imprim
    mov   $formato, %rdi
    mov   media,%esi
    mov   resto,%edx
    mov          $0,%eax    # varargin sin xmm
    call  printf        # == printf(formato, res, res);

    # acabar
    mov  media, %eax
    call _exit      # ==  exit(resultado)



suma:

    mov  $0, %esi   # inicializamos indice
    mov  $0, %r8d # acum1
    mov  $0, %edi # acum2


bucle:

    mov  (%ebx,%esi,4), %eax # vamos moviendo en eax los datos (eax=Lista[i])

    cdq                      # extension de signo edx:eax
    add %eax,%r8d            # r8d += eax
    adc %edx,%edi            # edi += edx

    inc %esi            # incrementamos indice


    cmp   %esi,%ecx     # compara que el indice es igual que la longitud
    jne    bucle

    mov %r8d,%eax       # guardamos los datos de nuevo en %eax y %edx
    mov %edi,%edx

    ret
```

Es

Ilustración 4: media

Los resultados de prueba para este código son:

Lista (x4 veces cada dato)	Media y resto	
1,1,1,1	Media = 1	resto = 0
-1,-1,-1,-1	Media = -1	resto = 0
0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff	Media = 2147483647	resto = 0
1,2,3,4	Media = 2	resto = 8
1000000,2000000,3000000,4000000	Media = 250000	resto = 0
5,6,7,8	Media= 6	resto = 8