Documentación
PRÁCTICA 2

Estructura de computadores

UNIVERSIDAD
DE GRANADA

Fernández Mertanen, Jonathan

Índice

Jonathan Fernandez Mertanen
Universidad de Granada 2018-2019

Este documento recoge la documentación y explicación de la realización de la práctica 2 de Estructura de Computadores

5.1- Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits usando uno de ellos como acumulador de acarreos (N≈16)

Modificaciones del programa

La modificación necesaria en este ejercicio es añadir un salto en caso de no tener acarreo para seguir con el bucle, y en el caso de tenerlo, incrementar en 1 el registro que acumula los acarreos (**%edx**)

```
suma:
      mov  $0, %esi  #contador bucle
      mov  $0, %eax  #acumulador de suma
      mov  $0, %edx  #acumulador acarreos
bucle:
      add  (%rbx,%rsi,4), %eax
      jnc continuarbucle      #condicional de acarreo
      inc %edx                #incrementamos registro de acarreo
continuarbucle:
      inc   %rsi
      cmp   %rsi,%rcx
      jne    bucle
      ret
```

suma_uns_jnc.s

Macro de Tests

```
.macro linea
      #if TEST==1
            .int 1, 1, 1, 1
      #elif TEST==2
            .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
      #elif TEST==3
            .int 0x08000000, 0x08000000, 0x08000000, 0x08000000
      #elif TEST==4
            .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
      #elif TEST==5
            .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
      #else
```

Jonathan Fernandez Mertanen
Universidad de Granada 2018-2019

```
            .error "Definir TEST entre 1..5"
      #endif
            .endm
```

suma_uns_jnc.s

Opciones de compilación

```
$> gcc -x assembler-with-cpp -D TEST=1 -no-pie -nostartfiles -g
suma_uns_jnc.s -o suma_uns_jnc
```

5.2- Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits mediante extensión con ceros (N≈16)

Modificaciones del programa

Esta solución es más sencilla a la anterior ya que con la instrucción **adc** se realiza esa condicional teniendo en cuenta el flag de acarreo y extendiendo 0's.

```
suma:
      mov  $0, %esi  #contador bucle
      mov  $0, %eax  #acumulador de suma
      mov  $0, %edx  #acumulador acarreos
bucle:
      add  (%rbx,%rsi,4), %eax
      adc  $0, %edx     #acumular acarreo en edx
      inc   %rsi
      cmp   %rsi,%rcx
      jne    bucle
      ret
```

suma_uns_adc.s

Macro de Tests

```
.macro linea
      #if TEST==1
            .int 1, 1, 1, 1
      #elif TEST==2
            .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
      #elif TEST==3
            .int 0x08000000, 0x08000000, 0x08000000, 0x08000000
```

```
      #elif TEST==4
            .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
      #elif TEST==5
            .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
      #else
            .error "Definir TEST entre 1..5"
      #endif
            .endm
```

suma_uns_adc.s

Opciones de compilación

```
$> gcc -x assembler-with-cpp -D TEST=1 -no-pie -nostartfiles -g
suma_uns_adc.s -o suma_uns_adc
```

5.3- Sumar N enteros con signo de 32 bits sobre dos registros de 32 bits (mediante extensión de signo, naturalmente) (N≈16)

Modificaciones del programa

En este ejercicio, el detalle está en comprobar el signo del número que vamos a sumar, y considerar acarreos diferentes en cada caso, si los hubiera, extendiendo 0's o 1's.

```
suma:
      mov  $0, %esi  #contador bucle
      mov  $0, %eax  #acumulador de suma
      mov  $0, %edx  #acumulador acarreos
bucle:
      mov  (%rbx,%rsi,4), %ebp
      or     %ebp, %ebp
      jns nosigno       #si es positivo, saltar a nosigno
      add  (%rbx,%rsi,4), %eax
      adc $0xffffffff, %edx   #acarreo negativo
      jmp continuarbucle
nosigno:
      add  (%rbx,%rsi,4), %eax
      adc  $0, %edx                 #acarreo positivo
continuarbucle:
      inc   %rsi
      cmp   %rsi,%rcx
      jne    bucle
```

```
	ret
```

suma_sig.s

Macro de Tests

```
.macro linea
	#if TEST==1
		.int -1, -1, -1, -1
	#elif TEST==2
		.int 0x04000000, 0x04000000, 0x04000000, 0x04000000
	#elif TEST==3
		.int 0x08000000, 0x08000000, 0x08000000, 0x08000000
	#elif TEST==4
		.int 0x10000000, 0x10000000, 0x10000000, 0x10000000
	#elif TEST==5
		.int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
	#elif TEST==6
		.int 0x80000000, 0x80000000, 0x80000000, 0x80000000
	#elif TEST==7
		.int 0xf0000000, 0xf0000000, 0xf0000000, 0xf0000000
	#elif TEST==8
		.int 0xf8000000, 0xf8000000, 0xf8000000, 0xf8000000
	#elif TEST==9
		.int 0xf7ffffff, 0xf7ffffff, 0xf7ffffff, 0xf7ffffff
	#else
		.error "Definir TEST entre 1..9"
	#endif
		.endm
```

suma_sig.s

Opciones de compilación

```
$> gcc -x assembler-with-cpp -D TEST=1 -no-pie -nostartfiles -g
suma_sig.s -o suma_sig
```

5.4- Media y resto de N enteros con signo de 32 bits calculada usando registros de 32 bits (N≈16)

Modificaciones del programa

En este ejercicio, la única modificación necesaria es guardar el resultado de la suma para poder realizar la división. Como IDIV utiliza **%eax** y **%edx** para almacenar el resultado de la

división, debemos primero salvar la información de estos, que en nuestro caso, es el resultado de la sumatoria.

En resultados negativos, la división (media truncada) devuelve valores extraños y no he sido capaz de solucionarlo.

```
suma:
      mov  $0, %esi  #contador bucle
      mov  $0, %eax  #acumulador de suma
      mov  $0, %edx  #acumulador acarreos
bucle:
      mov  (%rbx,%rsi,4), %ebp
      or     %ebp, %ebp
      jns nosigno
      add  (%rbx,%rsi,4), %eax
      adc $0xffffffff, %edx
      jmp continuarbucle
nosigno:
      add  (%rbx,%rsi,4), %eax
      adc  $0, %edx
continuarbucle:
      inc   %rsi
      cmp   %rsi,%rcx
      jne    bucle

      mov %eax, %esi    #libero eax y edx para la division
      mov %edx, %ebp
      xor %edx, %edx    #si edx no era 0, damage floating point exception
      idiv %rcx         #divido entre el tamaño de lista
      ret
```

media_sig.s

Macro de Tests

```
.macro linea
      #if TEST==1
            .int 1, 2, 1, 2
      #elif TEST==2
            .int -1, -2, -1, -2
      #elif TEST==3
            .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
      #elif TEST==4
            .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
      #elif TEST==5
            .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
```

```
      #elif TEST==6
            .int 2000000000, 2000000000, 2000000000, 2000000000
      #elif TEST==7
            .int -2000000000, -2000000000, -2000000000, -2000000000
      #else
            .error "Definir TEST entre 1..7"
      #endif
            .endm
```

media_sig.s

Opciones de compilación

```
$> gcc -x assembler-with-cpp -D TEST=1 -no-pie -nostartfiles -g
media_sig.s -o media_sig
```