

# Trabajo Práctico N°1

## Data ALU

Grupo 3

**AUTORES:**

Santiago Agustín ARRIBÉRE,  
Matías Santiago FRANCOIS,  
Joaquín Oscar GAYTAN,  
Pablo Martín SCHEINFELD

**PROFESORES:**

Ing. Daniel Andrés JACOBY

CIUDAD AUTÓNOMA DE BUENOS AIRES

Agosto 2021

## 1. Ejercicio 1

```

main          ORG      p: $e000
              EQU      *

              move     #$3d, x1
              move     #$3d, a1
              move     #$3d, b

end            main

```

Tomando los registros con los siguientes valores iniciales:

$a = \text{\$ffffffffffffffff}$

$b = \text{\$ffffffffffffffff}$

$x = \text{\$ffffffffffffffff}$

Instruccion	Cambios	Comentarios
-	$a = \text{\$ffffffffffffffff}$ $b = \text{\$ffffffffffffffff}$ $x = \text{\$ffffffffffffffff}$	Carga inicial de valores
move #\$3d,x1	$x = \text{\$3d0000ffffffff}$	Se carga el valor de 8 bit interpretado como un número signado fraccionario, por lo cual se alinea a la izquierda.
move #\$3d,a1	$a = \text{\$ff00003dffffff}$	Se carga el valor \$3d solo en el registro $a_1$ interpretado como entero no signado sin modificar $a_2$ y $a_0$
move #\$3d,b	$b = \text{\$003d00000000}$	Se guarda el valor \$3d en el acumulador b interpretandolo como numero de punto fijo, por lo cual se extiende el signo siendo $b_2$ \$00 y se completa $b_0 = \text{\$000000}$

En la tabla 1.1 se muestra el estado final de los registros.

Registro	Valor inicial	Valor final
a	$\text{\$ffffffffffffffff}$	$\text{\$ff00003dffffff}$
b	$\text{\$ffffffffffffffff}$	$\text{\$003d0000000000}$
x	$\text{\$ffffffffffffffff}$	$\text{\$3d0000ffffffff}$

TABLA 1.1: Estado inicial y final de los registros.

Se adjunta el estado final de los registros simulados.

```

next
x=      $3d0000fffff y=      $00000000000
a=      $ff00003dfffff b=      $003d00000000000
      x1=      $3d0000 x0=      $ffffff r7=      $0000 n7=      $0000 m7=      $ffff
      y1=      $000000 y0=      $000000 r6=      $0000 n6=      $0000 m6=      $ffff
a2=      $ff a1=      $00003d a0=      $ffffff r5=      $0000 n5=      $0000 m5=      $ffff
b2=      $00 b1=      $3d0000 b0=      $000000 r4=      $0000 n4=      $0000 m4=      $ffff
pc=      $e004 sr=      $0300 omr=      $02 r3=      $0000 n3=      $0000 m3=      $ffff
la=      $0000 lc=      $0000 r2=      $0000 n2=      $0000 m2=      $ffff
ssh=      $0000 ssl=      $0000 r1=      $0000 n1=      $0000 m1=      $ffff
ipr=      $0000 bcr=      $ffff sp=      $00 r0=      $0000 n0=      $0000 m0=      $ffff
cyc=      000085 ictr=      000003 cnt1=      000000 cnt2=      000000 cnt3=      000000 cnt4=      000000
p:$e003 000000 = nop

```

FIGURA 1.1: Estado final de los registros (simulación).

## 2. Ejercicio 2

En el presente ejercicio se ejecutó el siguiente programa.

```

main
    ORG      p : $e000
    EQU      *

    move     # $caba00 , x1
    move     x1 , a
    move     x1 , b1

end          main

```

Teniendo en cuenta que los registros parten desde los siguientes estados iniciales:

$a = \$0000000000000000$

$b = \$0000000000000000$

$x = \$0000000000000000$

Instrucción	Cambios	Comentarios
-	a = \$0000000000000000 y = \$0000000000000000 x = \$0000000000000000	Carga inicial de valores
move # \$caba00, x1	x = \$caba00000000	Se modifica el valor del registro x1 con el valor caba00 por lo que x resulta modificado al valor mostrado.
move x1,a	a = \$ffcaba00000000	Se mueve x1 al registro a como el valor caba empieza en 1 (base decimal) se completa con ff (base hexadecimal) hacia adelante.
move x1, b1	b = \$00caba00000000	En este caso el valor del registro x1 se guarda en el b1 por lo que b2 y b0 no se ven afectadas por este cambio (a diferencia de lo que hubiera pasado si guardábamos en el valor de x1 en b).

TABLA 2.1: Paso a paso de las instrucciones ejecutadas.

En la tabla 2.2 se muestra el estado final de los registros.

Registro	Valor inicial	Valor final
x	\$0000000000000000	\$caba00000000
a	\$0000000000000000	\$ffcaba00000000
b	\$0000000000000000	\$00caba00000000

TABLA 2.2: Estado inicial y final de los registros.

Se adjunta el estado final de los registros simulados.

```

next
x=      $caba00000000 y=      $000000000000
a=      $ffcaba00000000 b=      $00caba00000000
      x1= $caba00 x0= $000000 r7= $0000 n7= $0000 m7= $ffff
      y1= $000000 y0= $000000 r6= $0000 n6= $0000 m6= $ffff
a2= $ff a1= $caba00 a0= $000000 r5= $0000 n5= $0000 m5= $ffff
b2= $00 b1= $caba00 b0= $000000 r4= $0000 n4= $0000 m4= $ffff
pc= $e005 sr= $0300 omr= $02 r3= $0000 n3= $0000 m3= $ffff
la= $0000 lc= $0000 r2= $0000 n2= $0000 m2= $ffff
ssh= $0000 ssl= $0000 r1= $0000 n1= $0000 m1= $ffff
ipr= $0000 bcr= $ffff sp= $00 r0= $0000 n0= $0000 m0= $ffff
cyc=000102 ictr= 000005 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p: $e004 000000 = nop

```

FIGURA 2.1: Estado final de los registros (simulación).

### 3. Ejercicio 3

En este ejercicio se corre el siguiente programa.

```

main      ORG      p: $e000
          EQU      *
```

```

move    a1 , x1
move    a , y1
move    a , r7
move    a1 , x0

```

```
end      main
```

Teniendo en cuenta el siguiente estado inicial en los registros.

$a = \$00a00000000000$

$y = \$0000000000000$

$ccr = \$00$

A continuación se muestra un desglose del programa, indicando aquellos registros que cambian a medida que este se ejecuta.

Instrucción	Cambios	Comentarios
-	$a = \$00a00000000000$ $x = \$xxxxxxxxxxxx$ $y = \$0000000000000$ $r7 = \$xxxx$ $ccr = \$00$	Carga inicial de valores.
move a1,x1	$x = \$a00000xxxxxx$ $ccr = \$00$	Carga de a1 (\$a00000) en x1.
move a,y1	$y = \$7fffff000000$ $ccr = \$40$	Se interpreta al registro a como punto fijo. $a = 1,25$ . Se produce overflow. Se redondea y1 al máximo número positivo representable. En el CCR, se activa el bit <i>Limit</i> (L=1) indicando lo anterior
move a,r7	$r7 = \$ffff$ $ccr = \$40$	Al mover el acumulador a a r7, se produce overflow. Como r7 es entero, se interpreta a a como entero. Se redondea r7 al máximo valor representable (\$ffff)
move a1,x0	$x = \$a00000a00000$	En este caso se realiza una operación de copia de registro. No se interpreta el número como punto fijo.

TABLA 3.1: Paso a paso de las instrucciones ejecutadas.

En la tabla 3.2 se muestra el estado final de los registros.

Registro	Valor inicial	Valor final
x	\$xxxxxxxxxxxx	\$a00000a00000
y	\$000000000000	\$7fffff000000
r7	\$xxx	\$ffff
ccr	\$00	\$40

TABLA 3.2: Estado inicial y final de los registros.

Se adjunta el estado final de los registros simulados.

```

trace
x= $a00000a00000 y= $7fffff000000
a= $00a0000000000000 b= $0000000000000000
x1= $a00000 x0= $a00000 r7= $ffff n7= $0000 m7= $ffff
y1= $7fffff y0= $000000 r6= $0000 n6= $0000 m6= $ffff
a2= $00 a1= $a00000 a0= $000000 r5= $0000 n5= $0000 m5= $ffff
b2= $00 b1= $000000 b0= $000000 r4= $0000 n4= $0000 m4= $ffff
pc= $e005 sr= $0040 omr= $02 r3= $0000 n3= $0000 m3= $ffff
la= $0000 lc= $0000 r2= $0000 n2= $0000 m2= $ffff
ssh= $0000 ssl= $0000 sp= $00 r1= $0000 n1= $0000 m1= $ffff
ipr= $0000 bcr= $ffff r0= $0000 n0= $0000 m0= $ffff
cyc= $000102 ictr= $000004 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p: $e004 000000 = nop
0x

```

FIGURA 3.1: Estado final de los registros (simulación).

## 4. Ejercicio 4

En el presente ejercicio se ejecutó el siguiente programa.

```

ORG      p: $e000
main     EQU      *

        macr      x0 , x1 , a
        rnd       b
        mpyr      x0 , x1 , b

end       main

```

A continuación, en la tabla 4.1 se puede observar los cambios resultantes luego de ejecutar paso por paso cada una de las instrucciones del programa.

Instrucción	Cambios	Comentarios
-	a = \$00000123800000 b = \$ff000000ffffff x = \$400000400000	Carga inicial de valores
macr x0,x1,a	a = \$00200124000000	Al registro a se le suma el producto de x0 y x1 y se lo redondea. Luego, como en a0 resultaría \$800000, se redondea el resultado.
rnd b	b = \$ff000001000000	Se redondea b. Como b0 guardaba \$ffffff, en b1 resulta \$000001.
mpyr x1,x0,b	b = \$00200000000000	En el registro b se guarda el producto de x1 y x0 y se lo redondea. Como b0 resulta \$000000, el redondeo no influye en el resultado.

TABLA 4.1: Paso a paso de las instrucciones ejecutadas.

Los valores finales de los registros a y b se observan en la tabla 4.2.

Registro	Valor Inicial	Valor Final
a	\$00000123800000	\$00200124000000
b	\$ff000000ffffff	\$00200000000000

TABLA 4.2: Valores iniciales y finales de los registros a y b.

Las instrucciones del programa fueron ejecutadas en el simulador, del cual se obtuvieron los resultados mencionados anteriormente. En las figuras 4.1, 4.2, 4.3, 4.4 y 4.5 se muestran las capturas de la ejecución en el simulador.

```
display
x= $400000400000 y= $000000000000
a= $00000123800000 b= $ff000000ffffff
      x1= $400000 x0= $400000 r7= $0000 n7= $0000 m7= $ffff
      y1= $000000 y0= $000000 r6= $0000 n6= $0000 m6= $ffff
a2= $00 a1= $000123 a0= $800000 r5= $0000 n5= $0000 m5= $ffff
b2= $ff b1= $000000 b0= $ffffff r4= $0000 n4= $0000 m4= $ffff
      r3= $0000 n3= $0000 m3= $ffff
pc= $e000 sr= $0300 omr= $02 r2= $0000 n2= $0000 m2= $ffff
la= $0000 lc= $0000      r1= $0000 n1= $0000 m1= $ffff
ssh= $0000 ssl= $0000 sp= $00 r0= $0000 n0= $0000 m0= $ffff
ipr= $0000 bcr= $ffff
cyc=000000 ictr= 000000 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
```

FIGURA 4.1: Carga inicial de los registros.

```
disassemble p:e000
p:$e000 2000a3      = macr x1,x0,a
p:$e001 200019      = rnd b
p:$e002 2000a9      = mpyr x1,x0,b
p:$e003 000000      = nop
p:$e004 000000      = nop
```

FIGURA 4.2: Carga de las instrucciones a ejecutar.

```

p:$e001 200019          = rnd b
trace
x=      $400000400000 y=      $000000000000
a=      $00200124000000 b=      $ff000000ffffff
      x1= $400000    x0= $400000    r7= $0000    n7= $0000    m7= $ffff
      y1= $000000    y0= $000000    r6= $0000    n6= $0000    m6= $ffff
a2= $00 a1= $200124  a0= $000000    r5= $0000    n5= $0000    m5= $ffff
b2= $ff b1= $000000  b0= $ffffff    r4= $0000    n4= $0000    m4= $ffff
      r3= $0000    n3= $0000    m3= $ffff
pc= $e002    sr= $0310    omr= $02    r2= $0000    n2= $0000    m2= $ffff
la= $0000    lc= $0000    r1= $0000    n1= $0000    m1= $ffff
ssh= $0000    ssl= $0000    sp= $00    r0= $0000    n0= $0000    m0= $ffff
ipr= $0000    bcr= $ffff
cyc= $000051 ictr= $000001 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000

```

FIGURA 4.3: Primera instrucción ejecutada (macr x0,x1,a).

```

p:$e001 200019          = rnd b
trace
x=      $400000400000 y=      $000000000000
a=      $00200124000000 b=      $ff000001000000
      x1= $400000    x0= $400000    r7= $0000    n7= $0000    m7= $ffff
      y1= $000000    y0= $000000    r6= $0000    n6= $0000    m6= $ffff
a2= $00 a1= $200124  a0= $000000    r5= $0000    n5= $0000    m5= $ffff
b2= $ff b1= $000001  b0= $000000    r4= $0000    n4= $0000    m4= $ffff
      r3= $0000    n3= $0000    m3= $ffff
pc= $e003    sr= $0338    omr= $02    r2= $0000    n2= $0000    m2= $ffff
la= $0000    lc= $0000    r1= $0000    n1= $0000    m1= $ffff
ssh= $0000    ssl= $0000    sp= $00    r0= $0000    n0= $0000    m0= $ffff
ipr= $0000    bcr= $ffff
cyc= $000068 ictr= $000002 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000

```

FIGURA 4.4: Segunda instrucción ejecutada (rnd b).

```

p:$e002 2000a9          = mpyr x1,x0,b
trace
x=      $400000400000 y=      $000000000000
a=      $00200124000000 b=      $00200000000000
      x1= $400000    x0= $400000    r7= $0000    n7= $0000    m7= $ffff
      y1= $000000    y0= $000000    r6= $0000    n6= $0000    m6= $ffff
a2= $00 a1= $200124  a0= $000000    r5= $0000    n5= $0000    m5= $ffff
b2= $00 b1= $200000  b0= $000000    r4= $0000    n4= $0000    m4= $ffff
      r3= $0000    n3= $0000    m3= $ffff
pc= $e004    sr= $0310    omr= $02    r2= $0000    n2= $0000    m2= $ffff
la= $0000    lc= $0000    r1= $0000    n1= $0000    m1= $ffff
ssh= $0000    ssl= $0000    sp= $00    r0= $0000    n0= $0000    m0= $ffff
ipr= $0000    bcr= $ffff
cyc= $000085 ictr= $000003 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000

```

FIGURA 4.5: Última instrucción ejecutada (mpyr x0,x1,b).

## 5. Ejercicio 5

En el presente ejercicio se analiza la ejecución del siguiente código.

```
ORG      p:$e000
```



```

main          EQU      *

               move     #$400000 , x0
               add      x0 , a
               add      x0 , a

end            main

```

### 5.1. Status Register inicializado en \$0300

Se imponen las siguientes condiciones iniciales a los registros.

$$a = \$0000000000000000$$

$$sr = \$0300$$

Dentro de estas condiciones se destaca que el valor inicial del status register considera ambos bits de escala (S1, S0) en cero, por lo que no se produce escalamiento en el valor del acumulador.

En la tabla 5.1 se observa como se desarrolla la ejecución del programa en cuestión.

Instrucción	Cambios	Comentarios
-	$a = \$0000000000000000$ $sr = \$0300$	Carga inicial de valores.
move #\$400000,x0	$x0 = \$400000$	Carga el valor inmediato en x0. No hay cambios en el sr.
add x0,a	$a = \$0040000000000000$ $sr = \$0300$	Suma el valor de x0 en a. $a = 0,5$ . El acumulador estaba inicializado en 0.
add x0,a	$a = \$0080000000000000$ $sr = \$0320$	Suma el valor de x0 en a. $a = 1$ . Se activa el bit Extension (E=1).

TABLA 5.1: Paso a paso de las instrucciones ejecutadas con  $sr = \$0300$

La tabla 5.2 refleja el estado final de los registros.

Registro	Valor inicial	Valor final
a	$\$0000000000000000$	$\$0080000000000000$
sr	$\$0300$	$\$0320$

TABLA 5.2: Estado inicial y final de los registros con  $sr = \$0300$

Se observa que luego de la ejecución del segundo *add*, el acumulador toma el valor  $a = \$0080000000000000$ . Esto surge de que se suma dos veces el valor 0,5, contenido en x0. De esta forma, el valor final del

acumulador a es 1. Luego, el bit de extensión (E) del CCR se activa, indicando que se está usando la parte entera del acumulador. Esto sucede dado que en un número de punto fijo de 24 bits no es posible representar la unidad.

```

step
x= $000000400000 y= $000000000000
a= $00800000000000 b= $00000000000000
x1= $000000 y1= $000000 x0= $400000 r7= $0000 n7= $0000 m7= $ffff
a2= $00 a1= $800000 a0= $000000 r6= $0000 n6= $0000 m6= $ffff
b2= $00 b1= $000000 b0= $000000 r5= $0000 n5= $0000 m5= $ffff
pc= $e005 sr= $0320 omr= $02 r4= $0000 n4= $0000 m4= $ffff
la= $0000 lc= $0000 r3= $0000 n3= $0000 m3= $ffff
ssh= $0000 ssl= $0000 sp= $00 r2= $0000 n2= $0000 m2= $ffff
ipr= $0000 bcr= $ffff r1= $0000 n1= $0000 m1= $ffff
cyc= 000102 ictr= 000005 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p: $e004 000000 = nop

```

FIGURA 5.1: Estado final de los registros con sr=\$0300 (simulación).

En la figura 5.1 se observa el resultado de la simulación.

## 5.2. Status Register inicializado en \$0700

Se repite el análisis anterior, pero esta vez con las siguientes condiciones iniciales:

$$a = \$00000000000000$$

$$sr = \$0700$$

Esta vez se comienza con el bit de escalamiento S0 activo. Esta condición implica que se produce un shift aritmético hacia la izquierda de los acumuladores. De esta forma, se desplaza el punto fraccionario un lugar hacia la izquierda. Esto cambia la forma en la que el DSP computa los bits *Unnormalized* y *Extension* (U y E, respectivamente) del CCR.

En la tablas 5.3 y 5.4 se muestra el paso a paso en la ejecución de las instrucciones y los resultados, respectivamente.

Instrucción	Cambios	Comentarios
-	a = \$00000000000000 sr = \$0700	Carga inicial de valores.
move #\$400000,x0	x0 = \$400000	Carga el valor inmediato en x0. No hay cambios en el sr.
add x0,a	a = \$00400000000000 sr = \$0710	Suma el valor de x0 en a. a=0,5. Se activa el bit Unnormalized del ccr (U=1).
add x0,a	a = \$00800000000000 sr = \$0700	Suma el valor de x0 en a. Se desactiva el bit Unnormalized del ccr (U=0).

TABLA 5.3: Paso a paso de las instrucciones ejecutadas con sr=\$0700

Registro	Valor inicial	Valor final
a	\$0000000000000000	\$0080000000000000
sr	\$0700	\$0700

TABLA 5.4: Estado inicial y final de los registros con sr=\$0700

De las tablas anteriores se concluye que el resultado en el acumulador a no cambió respecto del caso expuesto en la tabla 5.2. En contraposición, se aprecia que el comportamiento del CCR durante la ejecución es distinto que el de la tabla 5.1. Como se advirtió anteriormente, esto se debe al cambio en las condiciones de escalamiento impuestas por el nuevo valor inicial del status register, y en cómo estas afectan al cálculo de ciertos bits del mismo.

En este caso, al ejecutar el primer *add*, se carga el valor \$0040000000000000 en el acumulador a. Dado que ahora se trabaja con el punto fraccionario desplazado una posición hacia la izquierda, no se cumple la condición de normalización, activando el bit correspondiente en el CCR que indica que el acumulador no se encuentra normalizado. Al sumar nuevamente el valor de x0, pasa a cumplirse la condición de normalización y el bit U cambia su valor a 0.

Otra diferencia respecto al caso anterior es que luego de ejecutadas las instrucciones, no se activa el bit de extensión del CCR. Nuevamente, esto se debe al desplazamiento hacia la izquierda del punto fraccionario. En este caso, esta modificación resulta en que el valor final del acumulador sea 0,5 en lugar de 1, lo cual implica que no se está haciendo uso de la parte entera del mismo.

```

trace
x= $000000400000 y= $000000000000
a= $00800000000000 b= $00000000000000
      x1= $000000 x0= $400000 r7= $0000 n7= $0000 m7= $ffff
      y1= $000000 y0= $000000 r6= $0000 n6= $0000 m6= $ffff
a2= $00 a1= $800000 a0= $000000 r5= $0000 n5= $0000 m5= $ffff
b2= $00 b1= $000000 b0= $000000 r4= $0000 n4= $0000 m4= $ffff
      pc= $e005 sr= $0700 ovr= $02 r3= $0000 n3= $0000 m3= $ffff
      la= $0000 lc= $0000 r2= $0000 n2= $0000 m2= $ffff
ssh= $0000 ssl= $0000 r1= $0000 n1= $0000 m1= $ffff
ipr= $0000 bcr= $ffff sp= $00 r0= $0000 n0= $0000 m0= $ffff
cyc= 000102 ictr= 000005 cnt1= 000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p: $e004 000000 = nop

```

FIGURA 5.2: Estado final de los registros con sr=\$0700 (simulación).

La figura 5.2 muestra el resultado de la simulación realizada.

## 6. Ejercicio 6

En el presente ejercicio se ejecutó el siguiente programa.

```

ORG      p: $e000
main     EQU      *

        add x1, a
        rep #$a
        norm r0, a
        add x0, a

```

end                      main

Teniendo en cuenta que los registros parten desde los siguientes estados iniciales:

$a = \$0000000000000000$

$x = \$0c0000600000$

$r0 = \$0000$

Instrucción	Cambios	Comentarios
-	$a = \$0000000000000000$ $x = \$0c0000600000$ $r0 = \$0000$	Carga inicial de valores
add x1,a	$a = \$000c0000000000$	se suma a con el valor almacenado en x1 y se guarda el resultado de la suma en a.
rep # \$a norm r0,a	$a = \$00600000000000$	En esta instrucción se realiza en un loop 10 veces la siguiente instrucción en este caso la instrucción norm para ver qué ocurre dentro del loop referirse a la tabla 6.2
add x0,a	$a = \$00c00000000000$	se suma x0 al registro a, de esta manera lo que termina sucediendo es un shifteo a derecha debido a que se terminan sumando 2 números iguales. Luego de la suma los bits E, U y Z valen como se indica en la tabla 6.3.

TABLA 6.1: Paso a paso de las instrucciones ejecutadas.

Iteración	E	U	Z	Acción	Valor del acumulador a
1	0	1	0	ASL	\$00 180000 000000
2	0	1	0	ASL	\$00 300000 000000
3	0	1	0	ASL	\$00 600000 000000
4	0	0	0	NOP	\$00 600000 000000
5	0	0	0	NOP	\$00 600000 000000

TABLA 6.2: Pasos internos del Loop, luego del paso 5 se repiten las filas hasta la iteración número 10

E	U	Z
1	1	0

TABLA 6.3: Valores de los bits E, U y Z al finalizar la ejecución.

## 7. Ejercicio 7

En el presente ejercicio se analiza la ejecución del siguiente código.

```

                ORG      X: $0000
                dc       $10fedc
                dc       $210fed
                dc       $4210fe
                dc       $84210f
                dc       $d84210
                dc       $fb8421

main          ORG      P: $E000
                EQU      *
                move     #$0000, r0
                move     #$0000, r4
                move     #$ffff, m0
                move     #$ffff, m4
                move     #$0800, sr

                move     x:(r0)+, a
                rep      #6
                move     a,y:(r4)+      x:(r0)+, a
                jlc      OK
                bset     #0,y:$100
OK            bclr     #6,sr

                end      main

```

Del código se destaca que se inicializa al *status register* con el valor \$0800. Del manual del DSP 56000 de Motorola se extrae que se activa el bit S1, lo que configura el *scaling mode* en *Scale Up*, lo cual produce un corrimiento del punto fraccionario hacia la derecha, aumentando la parte entera en 1 bit.

Dirección	Mapa X (origen)	Mapa Y (destino)
\$0000	\$10fedc	\$21fdb8
\$0001	\$210fed	\$421fda
\$0002	\$4210fe	\$7fffff
\$0003	\$84210f	\$800000
\$0004	\$d84210	\$b08420
\$0005	\$fb8421	\$f70842

TABLA 7.1: Estado de la memoria luego de correr el programa.

En la tabla 7.1 se muestra el resultado de la ejecución del programa, donde se puede observar el efecto del *Scale Up* ya que las posiciones de memoria en Y surgen de realizar un shift aritmético a izquierda sobre las posiciones de memoria en X, exceptuando las posiciones \$0002 y \$0003 ya que en las mismas al cargarse sobre el acumulador A e interpretarse como número de punto fijo corresponden a números, en modulo, mayor a 1, por lo cual al intentar realizar el movimiento del acumulador al espacio de memoria actuará el limitador.

```
display y:$0000..$0006
y:$0000=      $21fdb8      $421fda      $7ffffff      $800000
y:$0004=      $b08420      $f70842      $000000
```

FIGURA 7.1: Estado final de la memoria (simulación).

```
display y:$0100
y:$0100=      $000001
```

FIGURA 7.2: Estado final de la memoria Y:\$0100 (simulación).

En la figura 7.2 se aprecia que el valor final de la posición de memoria Y:\$0100 es 1. Observando el programa, se concluye que esta posición de memoria puede estar siendo empleada como flag para indicar si en algún momento de la ejecución existió un redondeo.

## 8. Ejercicio 8

En este ejercicio se implementó la subrutina *vect\_max*, la cual recibe por registros punteros a dos vectores *A* y *B*, los compara y devuelve en *B* aquellos valores de mayor valor absoluto.

Se reciben los punteros a los vectores *A* y *B* en los registros  $r_0$  y  $r_4$ , respectivamente, correspondiendo  $r_0$  a una región de memoria en X, y  $r_4$  a una región de memoria en Y. En el registro  $n_0$  se recibe el tamaño de los vectores a comparar.

Se propone el siguiente *main* de prueba.

```
ORG      x:$0000
dc       0.125
dc       0.0625
dc       -0.5
dc       0.75

ORG      y:$1000
dc       0.3125
dc       -0.0125
dc       0.5
dc       0.625
```

```

                                ORG      p : $e000

vect_max                       EQU      *

                                do        n0 , endloop
                                move      x : ( r0 )+ , x0          y : ( r4 ) , b
                                cmpm     x0 , b
                                tlt      x0 , b
                                move      b , y : ( r4 )+

endloop                         EQU      *

                                rts

main                           EQU      *

                                move      #$0000 , A
                                move      #$1000 , B
                                move      A , r0
                                move      B , r4

                                move      #$4 , n0

                                jsr       vect_max

                                end       main

```

En la figura 8.1 se muestra el resultado de simular el código propuesto.

```

evaluate y:$1000
Hex:280000 Uns:2621440 Fract:    0.3125 Bin:001010000000000000000000
evaluate y:$1001
Hex:080000 Uns:524288 Fract:    0.0625 Bin:000010000000000000000000
evaluate y:$1002
Hex:400000 Uns:4194304 Fract:    0.5 Bin:010000000000000000000000
evaluate y:$1003
Hex:600000 Uns:6291456 Fract:    0.75 Bin:011000000000000000000000

```

FIGURA 8.1: Estado final del vector B (Simulación)