Ejercicio 1

Si inicialmente los registros se encuentran en este estado:

```
a     = $ffffffffffff
b     = $ffffffffffff
x     = $ffffffffffff
```

y se ejecuta el siguiente código:

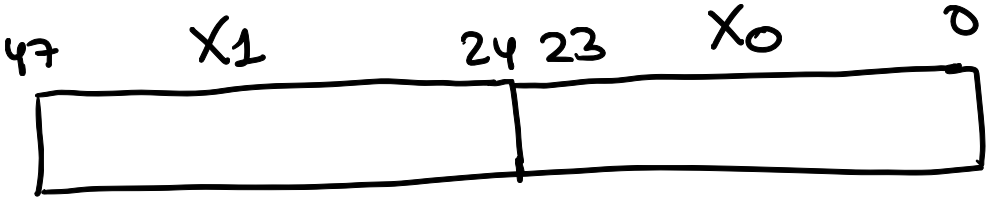
```
move  #$3d,x1
move  #$3d,a1
move  #$3d,b
```

Indicar el estado final de los registros

```
a     =
b     =
x     =
```

La instrucción move #$3d,X1 lo que hace es usar direccionamiento inmediato y cargar el número 3d (hexa) en X1 /

(ver pág 80 pdf del manual)



Es un move immediate short into 24 bit register WEPO resulta:

porque transfiero a X1

X1 X0

FFFFFF ; FFFFFF → 3d0000 ; FFFFFF

Luego tengo move #$3d,a1 y como a1 es parte del acumulador A, se transfiere a la parte menos significativa

A2 A1 A0

FF FFFFFF FFFFFF → A2 A1 A0 FF 0003d FFFFFF

(ver ejemplo A pag 80 (pdf) del manual)

La última instrucción es move #$3d,b lo cual resulta en:

B2	B1	B0		B2	B1	B0
FF	FFFFFF	FFFFFF	→	00	3d0000	000000

∴ luego, los estados finales de los registros son:

a = FF 0000 3d FFFFFF

b = 00 3d0000 000000

x = 3d 000 FFFFFF

Ejercicio 2

Si inicialmente los registros se encuentran en este estado:

```
a     = $000000000000000
b     = $000000000000000
x     = $000000000000
```

y se ejecuta el siguiente código:

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

Indicar el estado final de los registros

```
a     =
b     =
x     =
```

inicialmente tenía X1 000000 X0 000000 y ahora tenía

X1 X0
caba 00 000000 luego con el move X1, a

con a = 00 000000 000000 ahora tenía

A0 A1 A2
FF caba 00 000000 → mover de registro a acumulador

es signado fraccionario. se pone FF para mantener signo.

y con el move x1,b1 tengo B0 00 cabe00 B1 000000 B2

Finalmente, tengo

a = ff cabe00 000000

b = 00 cabe00 000000

x = cabe00 000000

Ejercicio 3

Si inicialmente los registros se encuentran en este estado:

```
a     = $00a00000000000
y     = $000000000000
ccr   = $00
```

ccr → condition code register

y se ejecuta el siguiente código:

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

Indicar el estado final de los registros

```
x     =
y     =
r7    =
ccr   =
```

Primero tenemos move a1, x1 luego

A2 A1 A0

a = 00 a00000 000000

X0 X1

x = xxxxxx xxxxxx

move a1, x1

a = 00 a00000 000000

=> x = xxxxxx a00000

X1 X0

luego move a, y1 lo que pasa es que queremos mover un número mayor a 1 (a000 → 1,25) y por eso se prende el bit de Unit del SR (ojo: es un bit "sticky" entonces si

no lo volvemos a poner en 0 manualmente no cambia)
con move a, y1 tenemos

x = a00000 000000

a = 00 a00000 000000

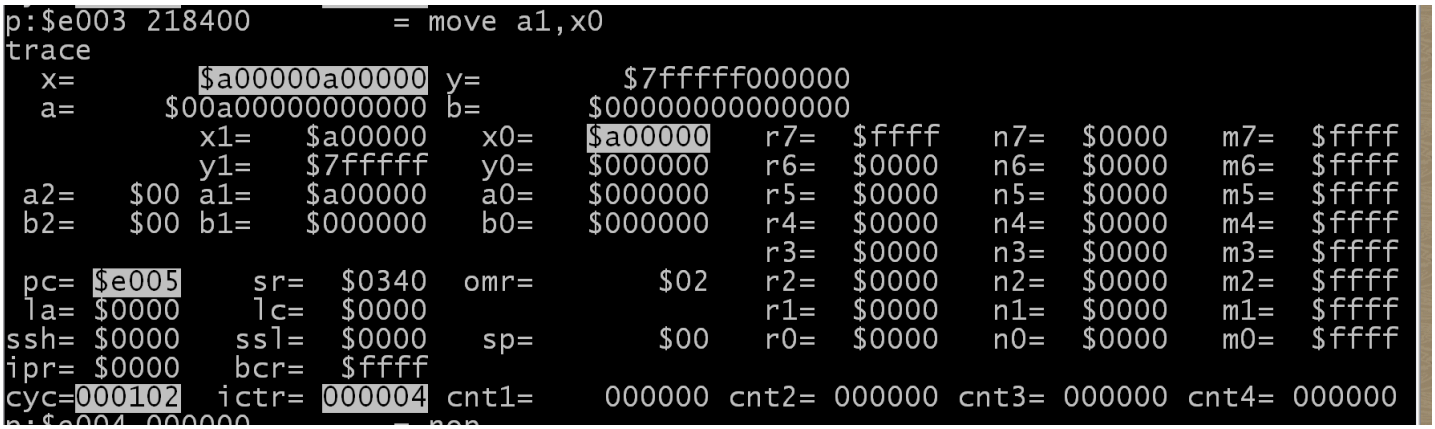
y = 7 fffff 00000

↳ completa con el número positivo más grande posible para avisar que nos pasamos.

Luego está move a, r7 y pasa algo parecido a lo de antes, y nos deja r7 = ffff para avisar que nos pasamos (a, x, y no se modifican). Se ve la simulación a continuación:

```
p:$e001 21c700          = move a,y1
trace
  x=       $a00000000000 y=       $7fffff000000
  a=     $00a00000000000 b=     $000000000000000
           x1=   $a00000   x0=   $000000   r7=  $0000   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
                                           r3=  $0000   n3=  $0000   m3=  $ffff
 pc= $e003    sr=  $0340  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=  $ffff
cyc=000068  ictr= 000002 cnt1=    000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p:$e002 21d700          = move a,r7
trace
  x=       $a00000000000 y=       $7fffff000000
  a=     $00a00000000000 b=     $000000000000000
           x1=   $a00000   x0=   $000000   r7=  $ffff   n7=  $0000   m7=  $ffff
           y1=   $7fffff   y0=   $000000   r6=  $0000   n6=  $0000   m6=  $ffff
0>
```

Finalmente tenemos el move a1, x0 que deja x = a00000 a00000 (a e y quedan igual) como no cambiamos el SR queda como estaba antes (SR = 0340). La simulación es:



Entonces el estado final es:

X = a 00000 a00000

a = 00 a00000 000000

Y = 7fffff 000000

r7 = ffff

CCR = $40 (SR = 0340)

Ejercicio 4

```
Si inicialmente los registros se encuentran en este estado:
a     = $00000123800000
b     = $ff000000ffffff
x     = $400000400000

y se ejecuta el siguiente código:
macr  x0,x1,a
rnd   b
mpyr  x0,x1,b

Indicar el estado final de los registros
a     =
b     =
```

valores iniciales

```
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=  $ffff
cyc=000000  ictr= 000000 cnt1=    000000 cnt2= 000000 cnt3= 000000 cnt4= 000000
p:$e000 000000        = nop
```

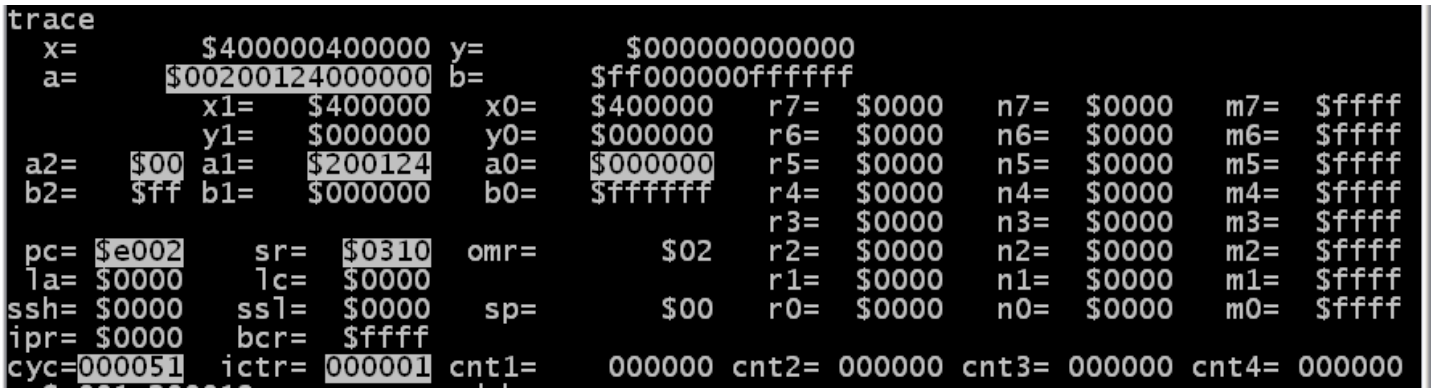
Verificamos que pusimos bien el codigo

```
disassemble
p:$e000 2000a3          = macr x1,x0,a
p:$e001 200019          = rnd b
p:$e002 2000a9          = mpyr x1,x0,b
p:$e003 000000          = nop
p:$e004 000000          = nop
p:$e005 000000          = nop
```

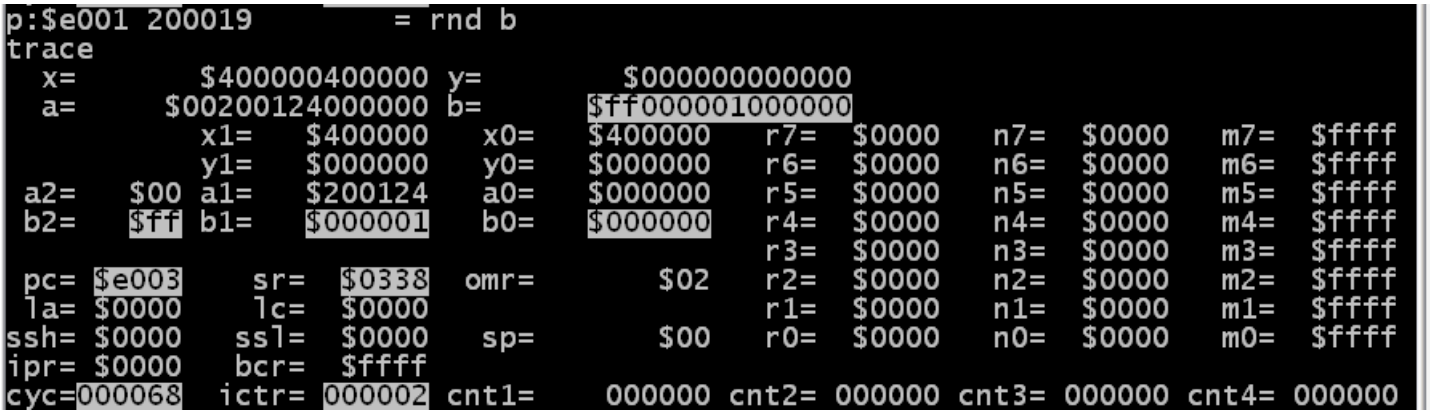
Chequeo codigo, NOSE PORQUE LO PONE AL REVES EL macr !!!

```
disassemble
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
```

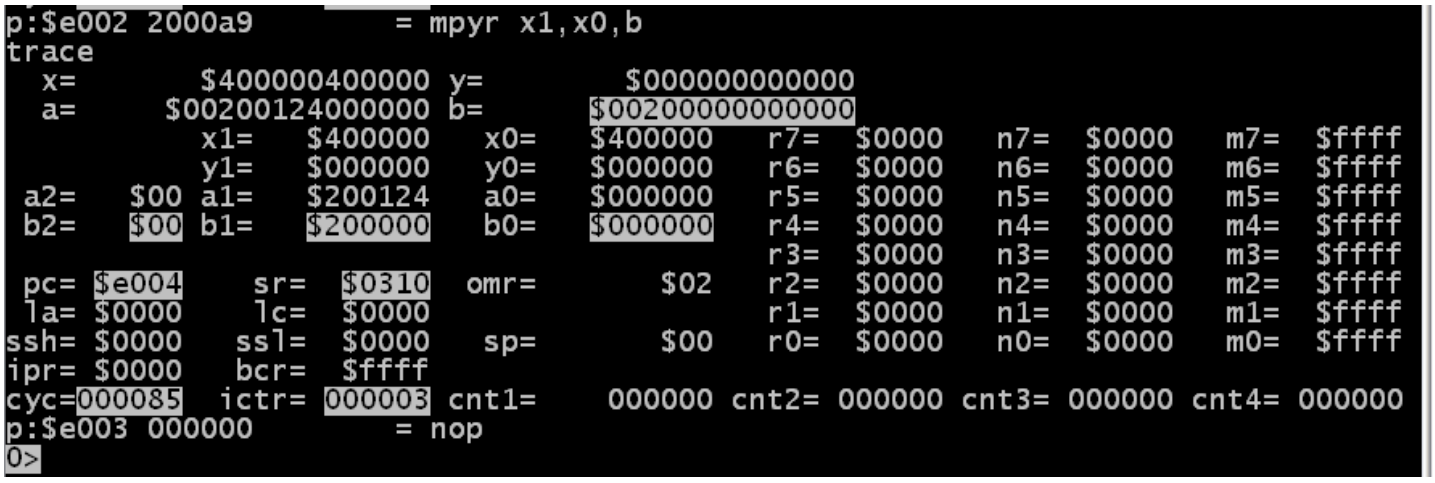
Ejecutando el macr obtenemos:



Luego el rnd



Ahora el mpyr



Ahi se ve el estado final de las variables

24 BITS

X0 = $400000 = 0100 000...
X1 = $400000 = 0100 000...
0010000 00
||
2000...

⇒ a = 00000 1238000 — 24+24 BITS = 48 BITS

⇒ a0 = 00
a1 = 000123 = 000000000000 000100100011
a2 = 800000 = 1000000...

MACR LO Q' HACE ES MULTIPLICAR X0 CON X1, EL RESULTADO SUMARLO A "A", ESA SUMA REDONDEARLA, Y POR ÚLTIMO GUARDARLA EN "A".

Como: X0 x X1 = $20000... → 48 BITS

LUEGO: $002000... + A = $00200... + $00000 1238000 =

AGREGO ESTOS DOS CEROS PORQ' X0 x X1 DA 48 BITS Y "A" TIENE 56 BITS (TENGO Q' AGREGAR 8 PARA HACER LA SUMA)

⇒ LA SUMA DA: $00200 1238000...

COMO ADEMÁS REDONDEA, Y POR (*)' SE PUEDE VER QUE A1 TERMINA EN "1" Y A0 = $800000 (1/2)

⇒ EL REDONDEO ES HACIA ARRIBA.

HOJA N°

FECHA

⇒ A1 = 00...0001.00100011 ①

A0 = 10...

⇒ A1 = 00...0001001001100 = $000124

⇒ A = $002001240000000

LA SIGUIENTE INSTRUCCION ES "rnd b". LO QUE HACE ES REDONDEAR A "b"

B2 B1 B0

B: FF 000000 FFFFFF

COMO B0 ES MAYOR A $800000 (1/2), ENTONCES EL REDONDEO ES PARA ARRIBA

⇒ B: FF 000001 000000

B2 B1 B0

POR ULTIMO, LA INSTRUCCION "mpyr X0,X1,b" LO QUE HACE ES MULTIPLICAR X0 CON X1, EL RESULTADO REDONDEARLO Y GUARDARLO EN "B". O SEA, QUE HACE ALGO SIMILAR A "MACR" PERO SIN SUMAL EL CONTENIDO DEL REGISTRO "B"

DADO QUE EL PRODUCTO ENTRE "X0" Y "X1" DABA:

$200000000000 → 48 BITS

NO HACE FALTA REDONDEAR.

A2 A1 A0

∴ A = $00 200124 000000

B = $00 200000 000000

B2 B1 B0

NOTA