LECCIÓN 10: ARITMÉTICA DE PUNTO FLOTANTE

1. Introducción.

Las unidades aritméticas de las que venimos hablando están diseñadas para trabajar con números enteros. Para trabajar con números en punto flotante tenemos dos posibilidades: utilizar hardware específico de punto flotante que se presenta en forma de procesador auxiliar (coprocesadores matemáticos) y que para el programador de sistemas significa simplemente una ampliación del conjunto de instrucciones y registros de la máquina, otro enfoque posible es simular por software las operaciones en punto flotante.

2. Representación binaria de números reales

Un número real consta de parte entera y parte fraccionaria y su representación binaria es la siguiente:

$Abc.def = a\,2^2 + b\,2^1 + c\,2^0 + d\,2^{-1} + e\,2^{-2} + f\,2^{-3}$

En la práctica para representar en binario un número real trabajamos por separado con su parte entera y su parte fraccionaria:

Sea por ejemplo 23.85 La parte entera 23 = 10111 y la parte fraccionaria la pasamos a binario multiplicando por 2 y quedándonos con la parte fraccionaria:

.85 x 2 = 1.70
.70 x 2 = 1.40
.40 x 2 = 0.80
.80 x 2 = 1.60
.60 x 2 = 1.20
.20 x 2 = 0.40
.40 x 2 = 0.80

Luego 0.85 = 0.1101100 ….

Por tanto 23.85 = 10111.1101100….

La forma habitual de representación binaria de números reales es la normalizada, de modo que $23.85 = 1.01111101100\ 2^4$

- **Formato de números en punto flotante: la norma IEEE-754**

Tenemos dos posibles formatos de números reales: simple precisión y doble precisión.

En **simple precisión** la longitud de palabra es de 32 bits

$N = (-1)^S\ 1.F\ 2^{E-127}$

Vemos que la mantisa está normalizada de modo que $1 \leq F \leq 2$ y que el exponente se almacena en exceso a 127 para evitar tener que usar otro bit de signo

Signo (S) 1 bit	Exponente (E) 8 bits	Mantisa (F) 23 bits

En **doble precisión** la longitud de palabra es 64 bits

$N = (-1)^S \ 1.F \ 2^{E-1023}$

Ahora el exponente está en exceso a 1023 y la mantisa está normalizada lo mismo que en el punto anterior

Signo (S) 1 bit	Exponente (E) 11 bits	Mantisa (F)52 bits

Dado que la representación de números reales bajo estos formatos es aproximada hay dos conceptos importantes en la aritmética en punto flotante: rango y precisión

- **Rango** : Nos da el conjunto de intervalos donde existen números representables, depende del exponente
- **Precisión** : Nos da la diferencia entre dos números representables consecutivos, depende del número de bits de la mantisa.

El rango y la precisión son conceptos antagónicos pues para mejorar la precisión habría que aumentar la mantisa y por tanto reducir el exponente lo que lleva a una disminución del rango.

- **Tipos de números según la norma IEEE-754**
  - **Normalizados**: $0 < E < E_{max} \quad 1 \leq 1.F < 2 \qquad E = (-1)^S \ 1.F \ 2^{E-127}$
  - **Cero** : $E = 0 \quad F = 0 \quad (-1)^S \times 0$ existe +0 y –0
  - **Infinitos** $E = E_{max} \quad F = 0 \qquad (-1)^S \times \infty$ existe +infinito y – infinito
  - **No reales (not a number)** $E = E_{max} \quad F = 0$
  - **Denormales** $E = 0 \quad F \neq 0$

3. Operaciones aritméticas con números en punto flotante

- **Suma y resta en punto flotante**
  - Alinear mantisas : Tomar el número con menor exponente y desplazar su mantisa a la derecha hasta igualar los exponentes

- Sumar o restar mantisas
- Normalizar el resultado si fuera necesario
- Redondear la mantisa al número de bits apropiado
- Normalizar si fuera preciso

En la transparencia 2.12 tenemos un esquema de una unidad de suma/resta en punto flotante.

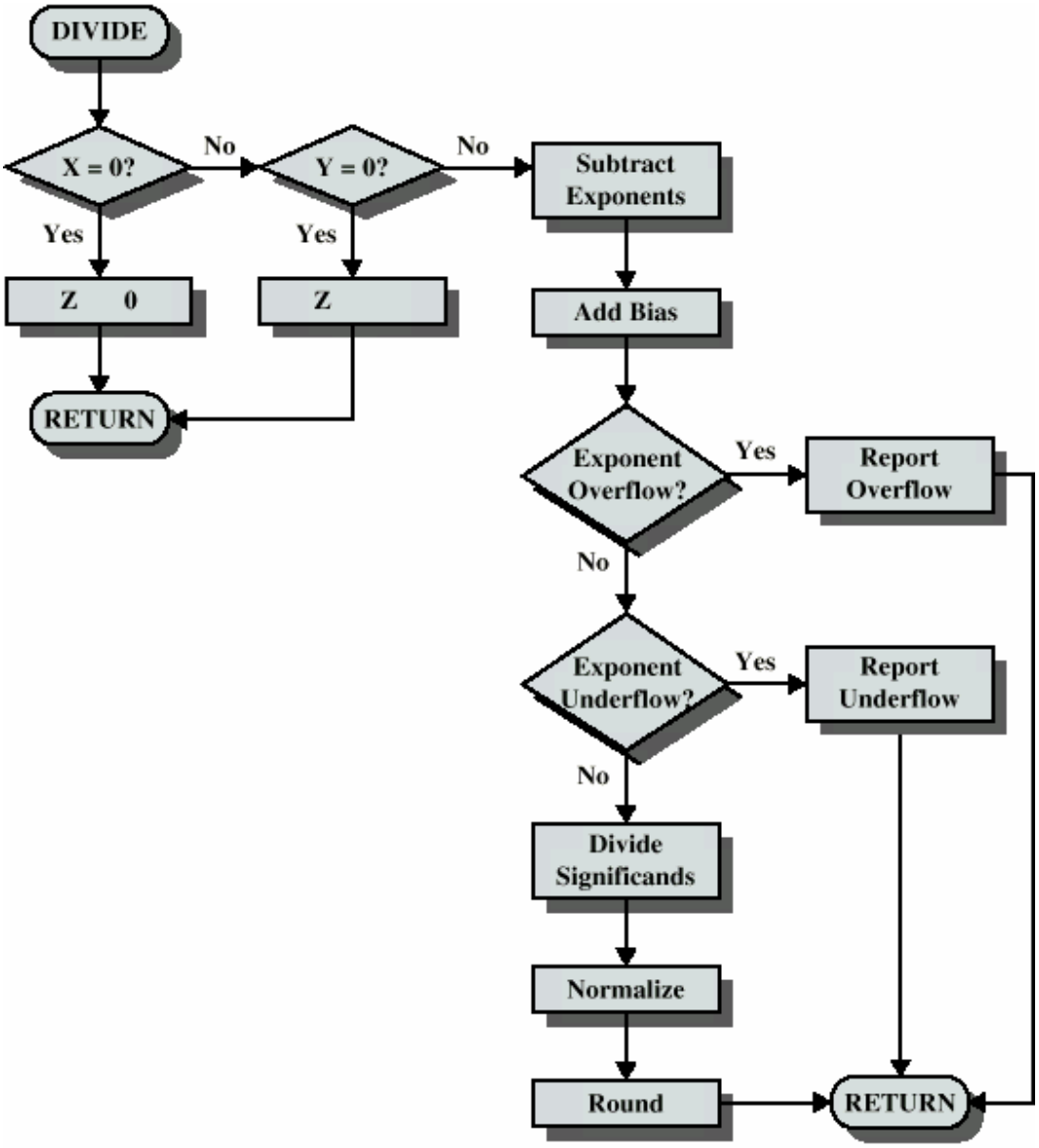
- **Multiplicación y división en punto flotante**

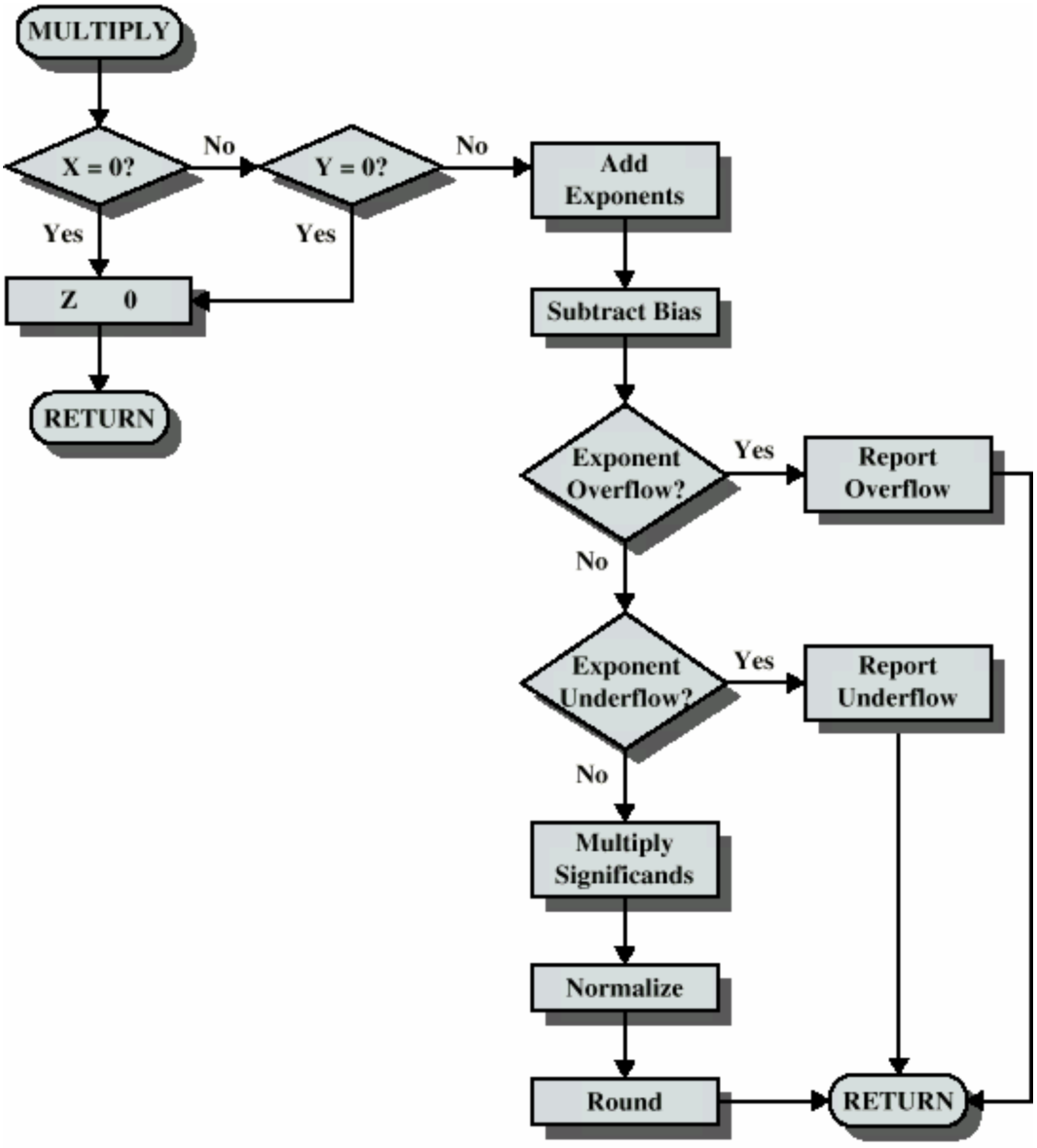
  - Sumar o restar los exponentes (y restar o sumar el exceso)
  - Multiplicar o dividir las mantisas
  - Normalizar el resultado
  - Redondear la mantisa al número apropiado de bits
  - Normalizar si es preciso
  - Determinar el signo del resultado

  En la transparencia 2.12 tenemos un esquema de la multiplicación en punto flotante.

Normalmente la precisión con que se realizan los cálculos es mayor que la que finalmente se representa. Así en el en 8087 la precisión interna es de 80 bits frente a los 64 de doble precisión. Esto obliga a que una vez finalizados los cálculos sea preciso eliminar los bits sobrantes en la mantisa, a esta operación se la conoce como truncamiento.
Tenemos varias formas de llevarlo a cabo:

- Truncamiento por corte: consiste en eliminar los bits sobrantes. Es el método mas simple pero presenta el problema de que el error es siempre del mismo signo y por tanto se acumula en el proceso de cálculo.
- Redondeo de Von Neumann: consiste en poner a 1


DIVIDE
X = 0?
No
Y = 0?
No
Subtract Exponents
Yes
Yes
Z 0
Z
Add Bias
RETURN
Exponent Overflow?
Yes
Report Overflow
No
Exponent Underflow?
Yes
Report Underflow
No
Divide Significands
Normalize
Round
RETURN


MULTIPLY
X = 0?
No
Y = 0?
No
Add Exponents
Yes
Yes
Z 0
Subtract Bias
RETURN
Exponent Overflow?
Yes
Report Overflow
No
Exponent Underflow?
Yes
Report Underflow
No
Multiply Significands
Normalize
Round
RETURN

4. El coprocesador matemático

Para implementar las operaciones en punto flotante es habitual disponer de un procesador especializado que ejecuta las instrucciones en que los operandos son números reales. En la familia IA-32 se usaba el coprocesador matemático 80x87 que a partir del 80486 está integrado en el procesador principal.

Desde el punto de vista del programador de sistemas consiste en una ampliación de los registros y del conjunto de instrucciones.

- **Registros del coprocesador**

  - La pila de registros de datos R0-R7: Dispone de 8 registros de 80 bits (1bit de signo. 15 de exponente y 64 de mantisa) con estructura de pila. El registro que está en la cabecera de la pila se almacena en un campo (TOP) del registro de estado. Desde el punto de vista del programador la nomenclatura de los registros es st0-st7, siendo la cabecera de la pila st0. En la figura 8.3 tenemos un ejemplo de funcionamiento de la pila.
  - El registro de control FPU. Codifica las distintas formas de comportamiento del procesador de punto flotante. Consta de 16 bits organizados como sigue:
    - Bits 11-10: Control del redondeo

Bits 11-10	Función
00	Al mas próximo (por defecto)
01	Al menor
10	Al mayor
11	Truncar

    - Bits 9-8: Control de precisión:

Bits 9-8	Precisión de la mantisa
00	24 bits
01	Reservado
10	53 bits
11	64 bit

    - Bits 5-0 : Máscaras de excepciones:
      1. Bit 0: IE (Operación inválida)
      2. Bit 1: DE (Operando denormalizado)
      3. Bit 2: ZE (División por cero)
      4. Bit 3: OE (Overflow)
      5. Bit 4: UE (Underflow)
      6. Bit 5: PE (Precisión)

Inicialmente tiene el valor 037FH (Redondeo al mas próximo, precisión 64 bits y máscara a 1 para todas las excepciones.

- El registro de estado . Es un registro de 16 bits que contiene distintas informaciones sobre el estado de la máquina:
  - Banderas de excepción: Informan sobre la aparición de una excepción
    7. Bit 0: IE (Operación inválida)
    8. Bit 1: DE (Operando denormalizado)
    9. Bit 2: ZE (División por cero)
    10. Bit 3: OE (Overflow)
    11. Bit 4: UE (Underflow)
    12. Bit 5: PE (Precisión)
  - Códigos de condición: Son los bits 14 (C_3), 10 (C_2), 9 (C_1), 8 (C_0) que codifican los resultados de operaciones aritméticas y de comparación en la FPU.

Instrucciones	C_3	C_2	C_1	C_0	Condición
FCOM , FCOMP,FCOMPP, FICOM,FICOMP	0	0	X	0	ST > Fuente
	0	0	X	1	ST < Fuente
	1	0	X	0	ST = Fuente
	1	1	X	1	ST ó Fuente indefinidos

Instrucciones	C_3	C_2	C_1	C_0	Condición
FTST	0	0	X	0	ST > 0
	0	0	X	1	ST < 0
	1	0	X	0	ST = 0
	1	1	X	1	ST indefinidos

Instrucciones	C_3	C_2	C_1	C_0	Condición
FXAM	0	0	0	0	+ No normalizado
	0	0	0	1	+ NaN
	0	0	1	0	- No normalizado
	0	0	1	1	- NaN
	0	1	0	0	+ Normalizado
	0	1	0	1	+ Infinito
	0	1	1	0	- Normalizado
	0	1	1	1	- Infinito
	1	0	0	0	+ 0
	1	0	0	1	Registro vacío
	1	0	1	0	- 0
	1	0	1	1	Registro vacío
	1	1	0	0	+ Denormalizado
	1	1	0	1	Registro vacío

	1	1	1	0	- Denormalizado
	1	1	1	1	Registro vacío

- Posición del registro st0 TOP : Son los bits 13-12-11

- Registro de TAG FPU: Indica el estado de cada uno de los registros de la pila FPU. Codifica en dos bits el estado de cada uno según la tabla siguiente:

Código	Valor
00	Válido
01	Cero
10	Especial
11	vacío

- Punteros a instrucción y a datos (de la última instrucción ejecutada)
- Registro con el código de operación de la última instrucción

2. Tipos de datos en el FPU

- Enteros de 16 bits en complemento a dos
- Enteros de 32 bits en complemento a dos
- Enteros de 64 bits en complemento a dos
- Reales de 32 bits (precisión simple 1-8-23)
- Reales de 64 bits (precisión doble 1-11-52)
- Reales de 80 bits (precisión extendida 1-15-64)
- BCD de 80 bits (18 dígitos en 4 bits cada uno)

3. Instrucciones FPU

- Transferencia de datos:
  - **FLD** :Carga un operando real en la pila FPU. El operando puede estar en memoria, definido como real de 32,64 ó 80 bits) o puede ser un registro sti que se cargará en la cabecera de la pila. En todo caso el destino es siempre st0 con formato real extendido.
  - **FST** : Copia el valor de st0 en memoria con real de 32 ò 64 bits o en otro registro de la pila.
  - **FSTP** : Es similar a FST pero además hace un POP en la pila y se pierde el operando que está en la cabecera.
  - **FXCH** : intercambia el contenido de st0 y otro registro que se indique. (si no tiene operandos entonces el otro registro es st1)
  - **FCMOVcc:** Son instrucciones de movimiento condicional (Sti a ST0 si se cumple una condición dada). La condición se refiere a valores del registro de estado del Pentium.

- **FILD, FIST, FISTP**: son versiones de las instrucciones anteriores que trabajan con enteros definidos en memoria y que se convierten en reales en la pila FPU (ó viceversa).
- **FBLD, FBSTP**: en este caso trabajan con números en formato BCD empaquetado.

- Carga de constantes:
  - **FLDZ** : Carga el número +0.0 en la pila
  - **FLD1** : Carga el número +1.0 en la pila
  - **FLDPI** : Carga el número π
  - **FLDL2T**: Carga el $\log_2 10$
  - **FLDL2E** : Carga el $\log_2 e$
  - **FLDLG2**: Carga el $\log_{10} 2$
  - **FLDLGE**: Carga el $\log_e 2$

- Aritméticas
  - **FADD / FADDP**: Si el operando es un real en memoria entonces almacena en st0 la suma de ambos. Si tenemos dos operandos que son registros FPU entonces hace la operación de suma almacenando el resultado en destino. Si FADDP no tiene operandos entonces suma st0 y st1, almacena el resultado en st1 y hace pop.
  - **FIADD**: Suma un entero con st0 y guarda el resultado en st0
  - **FSUB / FSUBP** : Similar a la suma.
  - **FISUB** : Resta un entero de un real
  - **FSUBR/FSUBRP** : Resta cambiando destino y fuente
  - **FISUBR** : Resta entero de real invertida.
  - **FMUL / FMULP** : Multiplicar reales
  - **FIMUL**: Multiplica entero y real
  - **FDIV / FDIVP** : Divide reales
  - **FIDIV** : Divide real entre entero.
  - **FDIVR / FDIVRP** : División inversa
  - **FIDIVR**: Divide real entre entero inversa
  - **FABS** : Calcula el valor absoluto de ST0
  - **FCHS**: Cambia de signo a ST0
  - **FSQRT:** Calcula la raíz cuadrada de ST0
  - **FPREM / FPREM1**: Calcula el resto parcial de la división entera. Difieren en la forma de redondear el cociente, el primero redondea al menor con lo que el resto es siempre positivo (forma 387) el segundo redondea al mas próximo por lo que el resto puede ser de distinto signo (forma IEEE)
  - **FRNDINT**: Redondea a entero según indique el campo de redondeo del registro de control.
  - **FXTRACT** : Separa el exponente y la mantisa en dos registros distintos (mantisa en st0 y exponente en st1)

- Comparación y clasificación:
  - **FCOM / FCOMP / FCOMPP:** Compara dos operandos reales (st0 y otro) y posiciona los flags.
  - **FUCOM /FUCOMP / FUCOMPP** : Son iguales a las anteriores. La única diferencia es que si alguno de los números es del tipo NaN no se produce la excepción correspondiente.
  - **FICOM /FICOMP**: Compara un entero y un real.
  - **FCOMI /FCOMIP** : Son las mismas que FCOM pero se posicionan directamente los indicadores del registro de estado según la siguiente tabla:

Resultado de la comparación	ZF	PF	CF
ST > Sti	0	0	0
ST < Sti	0	0	1
ST = Sti	1	0	0
Indefinido	1	1	1

  - **FUCOMI/FUCOMIP**
  - **FTST**
  - **FXAM**

En este punto es interesante comentar cómo se puede pasar de los códigos de condición FPU (C_3 , C_2 , C_0) a los bits de registro de estado ZF, PF y CF. Mediante la instrucción FSTSW AX pasamos el contenido del registro de estado FPU al registro AX y mediante la instrucción SAHF pasamos el byte alto de AX al registro EFLAGS, con lo que por la posición que ocupan se copian los valores FPU en los correspondientes de la CPU.

- Trigonométricas : Los operandos están siempre en la cabecera de la pila expresados en radianes
  - **FSIN**
  - **FCOS**
  - **FSINCOS**
  - **FPTAN**
  - **FPATAN**

- Logarítmicas, exponenciales y de escalado:
  - **FYL2X** : ST0 = ST1 * $\log_2$ (ST0)
  - **FYL2XP1** : ST0 = ST1 * $\log_2$ (ST0 + 1.0)
  - **F2XM1** : ST0 = $2^{ST0} - 1$
  - **FSCALE**: ST0 = STO * 2^{ST1}

- Control : Permiten inicializar el FPU y modificar las palabras de control y de estado.
  - **FINIT/FNINIT**: Inicializan el FPU . Carga la palabra 037F en el registro de control (redondeo al mas próximo, 64 bits de precisión y todas las banderas de excepción enmascaradas). (La N en que se diferencian las dos versiones de la

instrucción indica que antes de ejecutar la instrucción verifica si hay alguna excepción pendiente).

- **FLDCW /FSTCW / FNSTCW** : Carga y almacena el registro de control en/desde una posición de memoria de 16 bits.
- **FSTSW/FNSTSW** : Almacena el registro de estado en una palabra de memoria de 16 bits.
- **FCLEX / FNCLEX** : Borra los bits de excepción del registro de estado.
- **FLDENV** : Carga el entorno FPU (registros de estado y de control) desde memoria
- **FSTENV/FNSTENV** : Almacena el entorno FPU
- **FSAVE/FNSAVE/FRSTOR** : Salva y recupera el estado del FPU (comprende el entorno y los demás registros)
- **FWAIT /FNWAIT** : Instrucción de sincronización.
- **FNOP** : No operación
- **FFREE** : Marca en el registro TAG con 11 (registro vacío) al registro operando.
- **FINCSTP/ FDECSTP** : Incrementa ó decrementa el valor del campo TOP del registro de estado(y por tanto incrementa ó decrementa en 1 el valor del puntero a cabecera de pila)

4.- Dos ejemplos de programación FPU.

Un primer ejemplo calcula la raiz cuadrada de una serie de números reales.

```
segment pila stack
resb 0x100

segment datos
num DD 2.0,3.0,4.0,5.0,6.0
result resd  5

segment codigo
..start:
        mov ax, datos
        mov ds,ax
        mov cx, 5
        mov si,0000
        finit
otro:   fld [num + si]
        fsqrt
        fstp [result + si]
        add si,0x0004
        loop otro
        mov ax, 0x4c00
        int 21h
```

En un segundo ejemplo queremos calcular el valor de la función 2^x siendo x un número real.

Repasando el conjunto de instrucciones FPU vemos que existen las funciones F2XM1 y FSCALE pero la primera sirve para valores de x comprendidos entre -1 y 1 y la segunda para valores enteros.

Por eso tendremos que utilizar la expresión $2^x = 2^{int(x)} * 2^{frac(x)}$ de forma que cada una de las instrucciones calcula un operando y luego se multiplican las dos.

```
; rutina que calcula 2**x con x en st0
; 2**x = 2**int(x)*2**frac(x)

segment pila stack
resb64

segment datos
x dd 6.5
result dd 0.
rcontrol dw 0
temp dw 0

segment codigo
..start:
      mov ax,datos
      mov ds,ax
      fld dword [x]                 ; x en st0
      fstcw [rcontrol]              ; guardo el reg.control actual
      fstcw [temp]                  ; modifico el reg. de control
      or  word [temp],0x0c00        ; pongo el campo de redondeo en 11 = truncar
      fldcw [temp]
      fld st0                       ; x en st0 y x en st1
      frndint                       ; ent(x) en st0 , x en st1
      fxch                          ; x en st0 y ent(x) en st1
      fsub st0,st1                  ; frac(x) en st0 y ent(x) en st1
      f2xm1                         ; 2**frac(x)-1 en st0 y ent(x) en st1
      fld1
      fadd st0,st1                  ; 2**frac(x) en st0 y ent(x) en st2
      fstp st1
      fxch                          ;ent(x) en st0 y 2**frac(x) en st1
      fld1
      fscale                        ;  1*2**ent(x) en st0,1 en st1 y 2**frac(x) en st2
      fstp st1
      fmul st0,st1                  ; 2**x en st0
      fldcw [rcontrol]
      fst dword [result]
      ; final
      mov ax,0x4c00
```

int 0x21

Fernando de Prada. (fernando@infor.uva.es) Dpto. Informática. E.T.S.Ingenieros Informáticos Valladolid Edificio Tecnologías de la Información. Campus Miguel Delibes. Tfno: 983 423000. Ext. 25639 . Fax 983 423671