Ejercicio TC32-37p

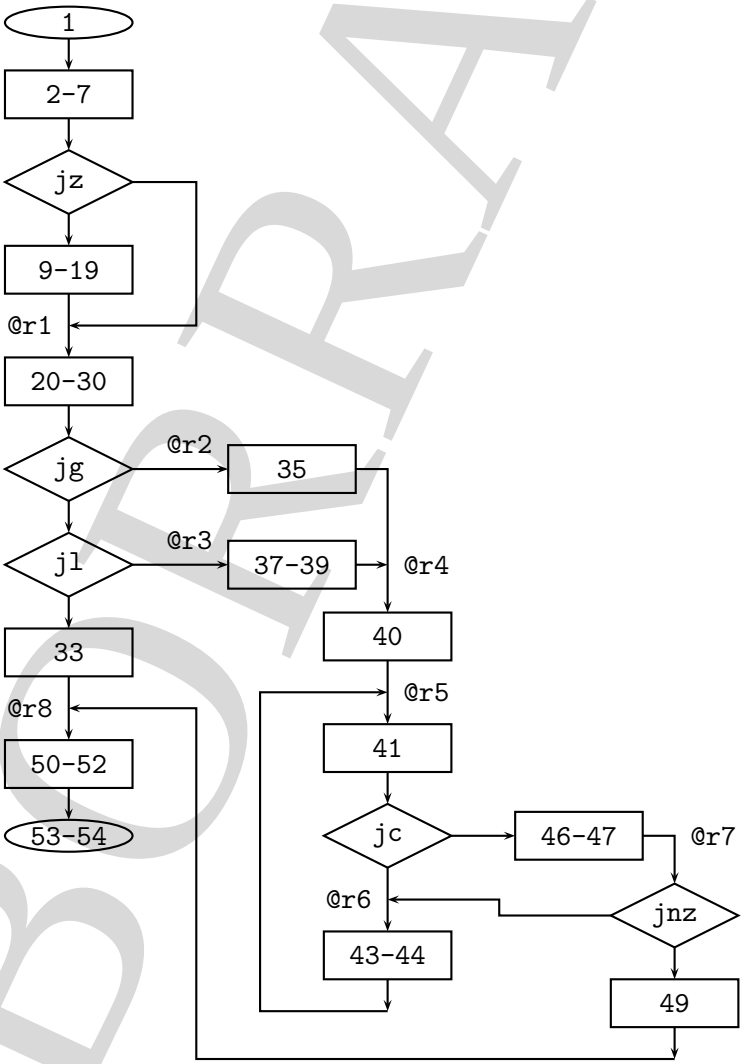
Analizar la rutina en lenguaje ensamblador del 80x86 que se propone en el recuadro adjunto. Razonar la función que realiza y la forma en que debe emplearse. Traducir a seudocódigo de alto nivel el algoritmo que sigue la rutina. ¿Funcionará bien en cualquier situación?.

```
1  Rutina        proc    near
2          push    ecx
3          fld     st(0)
4          push    word ptr 0
5          ftst
6          fstsw   ax
7          sahf
8          jz      @r1
9          fabs
10         fldlg2
11         fxch
12         fyl2x
13         push    eax
14         fstcw   word ptr [esp]
15         and     word ptr [esp],0F3FFh
16         or      word ptr [esp],007FFh
17         fldcw   word ptr [esp]
18         add     sp,4
19         fist    word ptr [esp]
20 @r1:    push    eax
21         fstcw   word ptr [esp]
22         and     word ptr [esp],0F3FFh
23         or      word ptr [esp],003FFh
24         fldcw   word ptr [esp]
25         add     sp,4
26         fstp    st(0)
27         pop     ax
28         mov     cx,ax
29         push    word ptr 10
30         cmp     cx,0
31         jg      @r2
32         jl      @r3
33         fld1
34         jmp     @r8
35 @r2:    fild    word ptr [esp]
36         jmp     @r4
37 @r3:    fld1
38         neg     cx
39         fidiv   word ptr [esp]
40 @r4:    fld1
41 @r5:    shr     cx,1
42         jc      @r7
```

```
43  @r6:    fld     st(1)
44          fmulp   st(2),st(0)
45          jmp     @r5
46  @r7:    fmul    st(0),st(1)
47          or      cx,cx
48          jnz     @r6
49          fstp    st(1)
50  @r8:    add     sp,2
51          fdiv
52          pop     ecx
53          ret
54  Rutina          endp
```

Solución

El análisis del código se facilita si se estudia su estructura general lo cual se consigue a través del diagrama del flujo de ejecución de las instrucciones. Se obtiene así el siguiente esquema, donde dentro de los bloques se indican los números de las instrucciones que contienen.



Hasta la instrucción 30 el código es bastante lineal y se analizará primero.

Mirando por encima el código propuesto se nota que la pila del procesador (sistema) indexada por `SP` (o por `ESP`) se usa para guardar variables locales o temporales. Para facilitar el análisis es mejor darles nombre a estas variables.

Por ejemplo, en la instrucción 4

```
push word ptr 0
```

se pone un cero en la pila; sea n el nombre de esta variable que se inicializa a cero. Esta variable se escribe, en la instrucción 19, empleando el índice `ESP` de la pila:

```
fist word ptr [esp]
```

y por fin, en la instrucción 27

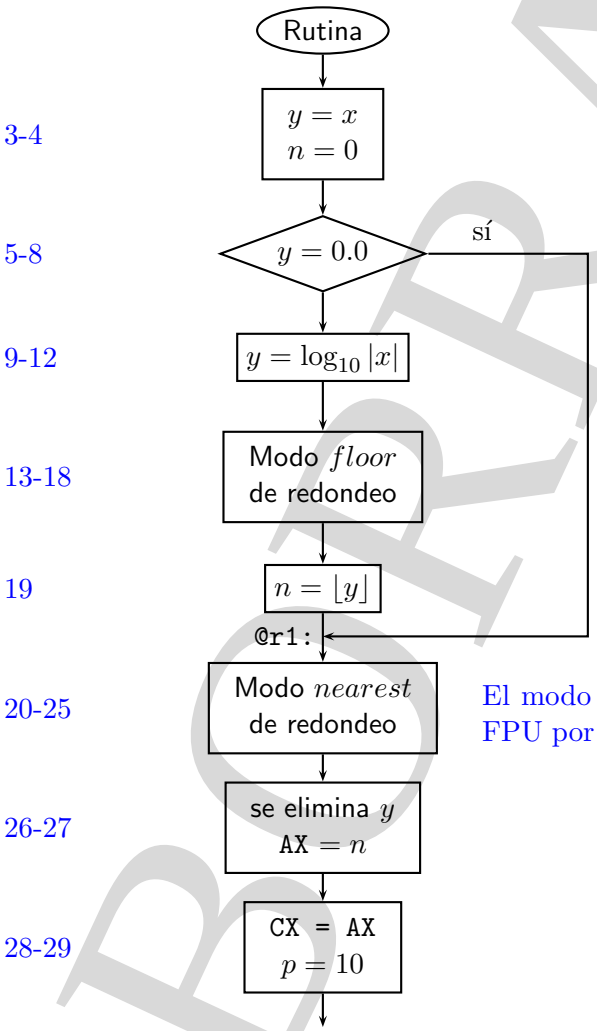
```
pop ax
```

se copia en el registro `AX` y se elimina de la pila.

También conviene, por claridad, nombrar con letras los valores guardados en los registros de la FPU, como si fuesen variables. De esta forma la instrucción 3

```
fld st(0)
```

se interpreta como $y = x$, es decir a `ST(1)` se le llama x y a `ST(0)` y, aunque justo antes era `ST(0)` a la que se le llamaba x. Por otro lado, esta instrucción 3 da a entender que a la entrada de la rutina se le pasa en `ST(0)` un valor (x). Hasta la línea 30 no es demasiado complicado llegar al siguiente esquema:



Los números a la izquierda se corresponden con los números de las instrucciones implicadas en cada bloque.

En n se pone el "orden de magnitud" de x, entendiendo por ello que

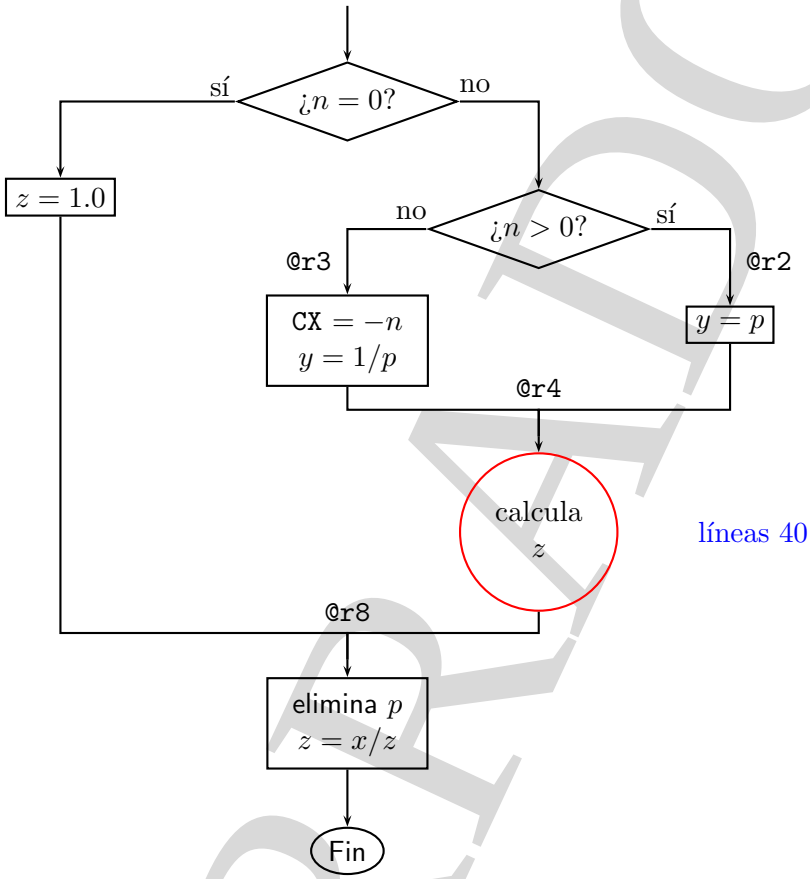
$$10^n \leq |x| < 10^{n+1}$$

El modo *nearest* de redondeo es el que tiene la FPU por defecto, y se vuelve a él si se modifica.

El valor original de x sigue preservado en `ST(0)`, su orden de magnitud $n = \lfloor \log_{10} |x| \rfloor$ está en `AX`. Las instrucciones 28 y 29 copian n en `CX` y se crea una nueva variable local en la pila, p, que se inicializa a 10.

A continuación se analiza el resto del código, desde la instrucción número 30 hasta el final.

Se comprueba el valor de n (la copia que está en `CX`) y se salta a `@r2` si $n > 0$, a `@r3` si $n < 0$ o sigue por la instrucción 33 si $n = 0$. Esto se corresponde con el siguiente esquema, aunque la primera comparación no está explícita en el código:



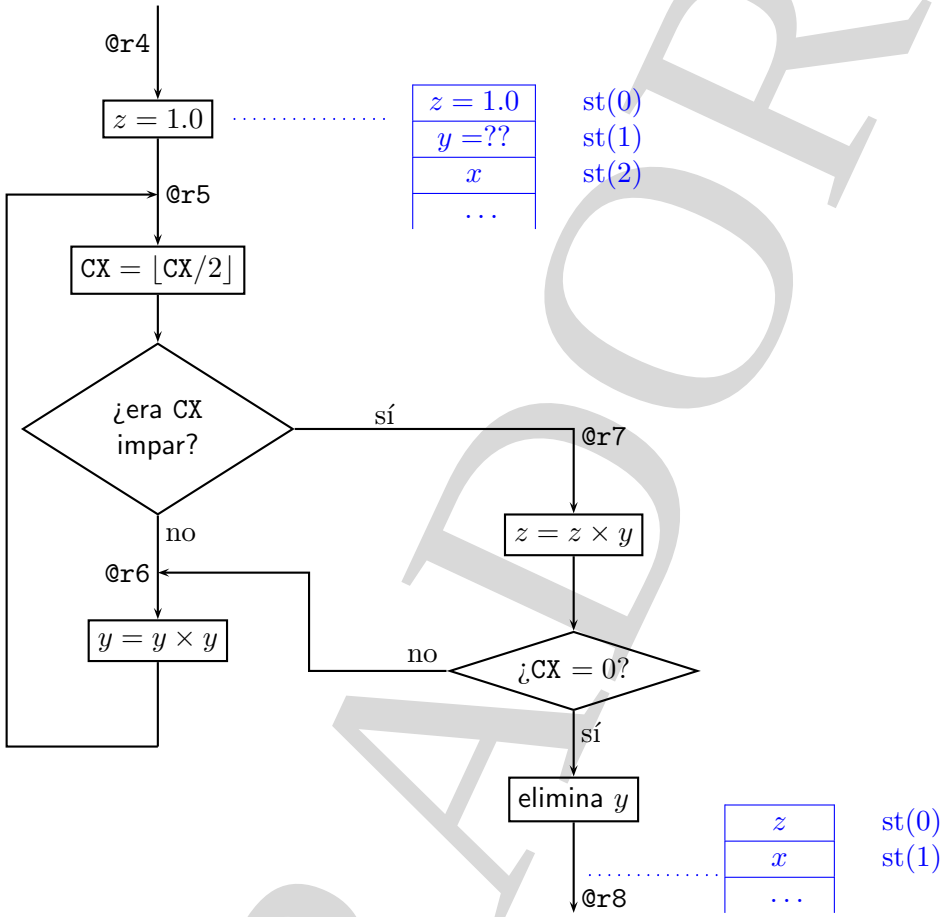
Justo antes de la línea 40 en la pila de registros de la FPU está y en `ST(0)` y x en `ST(1)`, valiendo $y = 10$ si $n > 0$ o bien $y = 0.1$ si $n < 0$. En `AX` está n y en `CX` está $|n|$.

A continuación se analizan las líneas de código 40 a 49.

En la instrucción 41 se saca el bit menos significativo de `CX` y se pone en el indicador de acarreo (`CF`), para preguntar por él en la siguiente instrucción; el efecto es doble, se ha dividido `CX` por 2 (división entera) y la paridad que tenía `CX` antes queda recogida en `CF`. Según el valor del bit se cuadra (se eleva al cuadrado) la variable y o bien se multiplica por el producto parcial z. Esto se corresponde con el algoritmo de potenciación entera. Este trozo de código calcula

$$z = y^{\texttt{CX}}$$

El siguiente esquema pone de forma gráfica al código analizado.



A la entrada de esta rama de la rutina es:

- Si $n > 0$, entonces $y = 10$.
- Si $n < 0$, entonces $y = 1/10 = 0.1$.

Y calcula

$$z = \begin{cases} 10^n & \text{si} \quad n > 0 \\ \left(\frac{1}{10}\right)^{-n} & \text{si} \quad n < 0 \end{cases}$$

O sea, que se calcula 10^n en cualquier caso, cualquiera que sea el signo de n.

Por la otra rama, si $n = 0$, se pone directamente $z = 1.0$. Como

$$10^n \leq |x| < 10^{n+1}$$

y en la instrucción 51

`fdiv`

hace $z = x/10^n$, luego devuelve un valor en `ST(0)`, z, tal que

$$1.0 \leq |z| < 10.0$$

y en `AX` su orden de magnitud, n.

La solución, abreviadamente, es:

Normaliza un número, devolviendo su orden de magnitud, n, en `AX` y en `ST(0)` $x/10^n$. Funciona bien en todos los casos excepto si a la entrada hubiese un NaN.