1994. Febrero. Primera semana.

Diseñe una subrutina para el MC68000 que invierta el orden de un vector de datos en memoria. Cada componente del vector ocupa un byte. El vector invertido debe quedar almacenado en las mismas posiciones de memoria que el original. Por ejemplo, dado el vector de componentes {0A, 0B, 0C, 0D, 0E} expresado en hexadecimal; el resultado de la subrutina debe ser: {0E, 0D, 0C, 0B, 0A}. Los parámetros que se le pasan a la subrutina son la dirección de comienzo del vector en el registro A0 y el tamaño del vector en el registro A1.

Siga el proceso indicado a continuación:

1º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

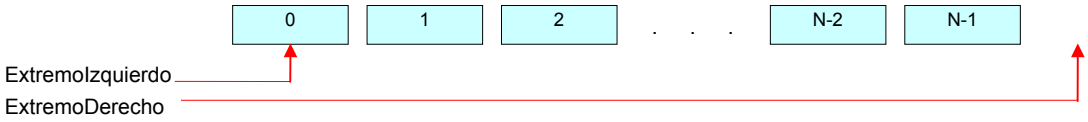
2º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

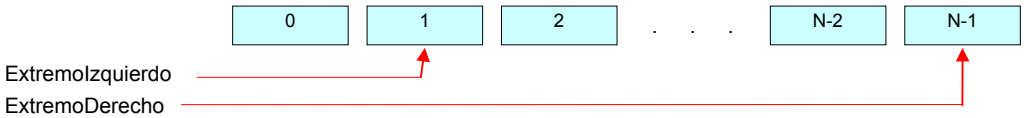
1º.- Descripción en pseudocódigo tipo lenguaje de alto nivel de la subrutina:

```
Semilongitud      :  ENTERO ;
ExtremoIzquierdo  :  PUNTERO A BYTE ;
ExtremoDerecho    :  PUNTERO A BYTE ;
Semilongitud      := ( < Longitud del vector > DIV 2) - 1 ;
ExtremoIzquierdo  := < Dirección del primer elemento > ;
ExtremoDerecho := < Dirección del último elemento > + 1 ;

REPETIR
   ExtremoDerecho    := ExtremoDerecho - 1 ;
   Intercambiar( ExtremoIzquierdo^ , ExtremoDerecho^ ) ;
   ExtremoIzquierdo  := ExtremoIzquierdo + 1 ;
   Semilongitud      := Semilongitud - 1 ;
HASTAQUE Semilongitud = -1 ;
```

Estado de la porción de vector considerada, antes de cada pasada por el bucle :



Estado de la misma porción, después de la pasada por el bucle :



En la siguiente pasada la porción considerada es la que resulta de eliminar en esta los dos elementos de los extremos. El proceso continúa hasta que la variable `Semilongitud` vale -1. En ese momento hay dos casos posibles para el estado de los punteros:
Si N es impar ambos punteros apuntan al elemento central. Este elemento se intercambia consigo mismo.
Si N es par los punteros apuntan a los dos elementos centrales. El último intercambio es entre estos dos elementos.

Los punteros más apropiados son los registros de direcciones especificados en el enunciado: A0 y A1. Cuando se necesite alguna variable intermedia lo más cómodo y eficaz (por su rapidez de acceso) es utilizar un registro de datos; y direccionamiento mediante registro. Si dentro del bucle REPETIR_HASTAQUE utilizamos los modos de direccionamiento relativo a registro con predecremento y con posincremento en las operaciones de intercambio, nos ahorraremos las sentencias de avanzar el puntero `ExtremoIzquierdo` y de retroceder el puntero `ExtremoDerecho`.

2º.- Fichero listado generado por el ensamblador cruzado:

```
MC68000 Cross Assembler
Copyright (C) Stephen Croll, 1991.  Author: Stephen Croll
Version 2.00 beta 1.02

                                     1 *
                                     2 *    Escuela de Informática de la UNED
                                     3 *    Asignatura: Estructura y Tecnología de Computadores I
                                     4 *    Tutoría del Centro Asociado de Plasencia
                                     5 *    Examen del 03-02-94
                                     6 *
                                     7 *
                                     8 * Propósito:    Crear una subrutina que invierta el orden de los elementos
                                     9 *               de un vector almacenado en memoria.
                                    10
                                    11
00001000                            12 Principal       org     $1000
00001000  4EB9 0000100C             13                 jsr     InvierteVector ; Llamada a la subrutina
00001006  1E3C 00E4                 14                 move.b  #228,D7
0000100A  4E4E                      15                 trap    #14
                                    16
0000100C  40E7                      17 InvierteVector  move.w  SR,-(SP)       ; Comienzo de la subrutina
0000100E  48E7 C000                 18                 movem.l D0-D1,-(SP)
00001012  4280                      19                 clr.l   D0
                                    20
00001014  2209                      21                 move.l  A1,D1          ; Longitud del vector
00001016  D3C8                      22                 adda.l  A0,A1          ; Dirección del posterior al último
00001018  E249                      23                 lsr     #1,D1          ; Divide por dos la longitud del
vector
0000101A  5301                      24                 subq.b  #1,D1          ; para obtener el centro del
vector.
                                    25
0000101C                            26 InicioBucle
                                    27
                                    28 *               Intercambiamos los elementos situados en (A0) y ((A1)-1)
                                    29 *               Con la instrucción EXG solo es posible el direccionamiento
                                    30 *               directo a registro, por lo cual debemos intercambiarlos
                                    31 *               con una secuencia de tres instrucciones.
                                    32
0000101C  1010                      33                 move.b  (A0),D0
0000101E  10E1                      34                 move.b  -(A1),(A0)+
00001020  1280                      35                 move.b  D0,(A1)
                                    36
00001022  51C9 FFF8                 37                 dbf     D1,InicioBucle; Los intercambios cesan cuando se
llega
                                    38                                     ; al centro del vector.
                                    39
00001026  4CDF 0003                 40                 movem.l (SP)+,D0-D1   ; Restaura los registros usados
0000102A  4E77                      41                 rtr                   ; Restaura el SR y sale de la
subrutina
                                    42
0000102C                            43                 end

No errors detected.
```

1994. Febrero. Segunda semana.

Diseñe una subrutina para el MC68000 que cuente el número de veces que aparece en una cadena de caracteres ASCII cada una de las cinco vocales del alfabeto. La dirección de comienzo de la cadena es un parámetro de la subrutina; y se le pasa en una posición de memoria etiquetada como DCADENA. La cadena finaliza con el carácter nulo en ASCII. El resultado de la subrutina será un vector de cinco palabras localizado a partir de la posición de memoria etiquetada como AEIOU, con la cuenta respectiva de cada una de las vocales.

Siga el proceso indicado a continuación:

1º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

2º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

1º.- Funcionamiento de la subrutina

A0 apuntará a la dirección DCADENA. Los elementos de esta ristra de caracteres se recorren con un bucle. A0 se incrementa de forma automática una vez en cada pasada del bucle. Como la ristra se recorre desde las posiciones bajas hasta las altas, el modo de direccionamiento más adecuado con A0 es el relativo a registro con posincremento.

A1 lo utilizamos para apuntar a varias posiciones de memoria, siempre las mismas en todas las pasadas (no lo incrementamos). El modo más adecuado de tratar A1 es el relativo a registro con desplazamiento.

En la estructura general de la subrutina podemos ver que:
- El bucle construido es del tipo WHILE, la comprobación de salida se hace al comienzo de cada pasada.
- El conjunto de parejas de instrucciones *Comparación-Salto* sintetizan una ramificación tipo CASE.

2º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:
Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*       Escuela de Informática de la UNED
*       Asignatura: Estructura y Tecnología de Computadores I
*       Tutoría del Centro Asociado de Plasencia
*
*
* Propósito:    Crear una subrutina que cuente las apariciones en una
*               cadena ASCII de cada una de las cinco vocales

*       ZONA DE CÓDIGO

Principal       org     $1000           ; Comienzo del programa principal
                jsr     CuentaVocales
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

CuentaVocales   move.w  SR,-(SP)
                movem.l  D0/A0-A1,-(SP)
                movea   #DCADENA,A0     ; Para permitir direccionamientos con posincremento
                movea   #AEIOU,A1       ; Para permitir direccionamientos con desplazamiento
                clr.b   0(A1)
                clr.b   1(A1)
                clr.b   2(A1)
                clr.b   3(A1)
                clr.b   4(A1)

InicioBucle     tst.b   (A0)            ; Bucle del tipo WHILE, con
                beq     FindelBucle     ; comprobación de salida al principio

                move.b (A0)+,D0

                cmp.b   #97,D0
                beq     EsLetra_A
                cmp.b   #65,D0
                beq     EsLetra_A

                cmp.b   #69,D0
                beq     EsLetra_E
                cmp.b   #101,D0
                beq     EsLetra_E

                cmp.b   #73,D0
                beq     EsLetra_I
                cmp.b   #105,D0
                beq     EsLetra_I

                cmp.b   #79,D0
                beq     EsLetra_O
                cmp.b   #111,D0
                beq     EsLetra_O

                cmp.b   #85,D0
                beq     EsLetra_U
                cmp.b   #117,D0
                beq     EsLetra_U
                bra     InicioBucle

EsLetra_A       addq.b  #1,0(A1)
                bra     InicioBucle

EsLetra_E       addq.b  #1,1(A1)
                bra     InicioBucle

EsLetra_I       addq.b  #1,2(A1)
                bra     InicioBucle

EsLetra_O       addq.b  #1,3(A1)
                bra     InicioBucle

EsLetra_U       addq.b  #1,4(A1)
                bra     InicioBucle

FindelBucle     movem.l (SP)+,D0/A0-A1
                rtr

*       ZONA DE DATOS

        org     $2000
DCADENA dc.b    'Hola a todos',0
        cnop    0,2
AEIOU   ds.b    5
                end
```

1994. Septiembre.

Sea una matriz de dimensión *N* filas por *M* columnas almacenada en la memoria de un microprocesador MC68000. Cada elemento de la matriz es de un byte. La matriz se guarda en la memoria organizada por filas, es decir, primero se guardan consecutivamente los *M* elementos de la fila 1, seguidos por los *M* de la fila 2, después los *M* de la fila 3 y así sucesivamente.

Diseñe una subrutina para el MC68000 que obtenga el elemento *(i, j)* de la matriz descrita. La matriz estará almacenada a partir de la dirección indicada en el registro A0. La dimensión de la matriz se pasará en los registros D1 y D2; que contendrán los valores de *N* y *M*, respectivamente. Los valores de *i* y de *j* se pasarán en los registros D3 y D4, respectivamente. La subrutina devolverá el elemento en el registro D0.

Siga el proceso indicado a continuación:

1º.- Obtener la fórmula que calcula la ubicación relativa de un elemento cualquiera *(i, j)* respecto al elemento *(1, 1)*; a partir de los índices *i* y *j* y de las dimensiones *N* y *M*.

2º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

3º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

1º.- Obtención de la fórmula algorítmica

$$A_{N\,M} = \begin{pmatrix} A_{[1,\,1]} & A_{[1,\,2]} & \cdots & A_{[1,\,M]} \\ A_{[2,\,1]} & A_{[2,\,2]} & \cdots & A_{[2,\,M]} \\ \cdots & \cdots & \cdots & \cdots \\ A_{[N,\,1]} & A_{[N,\,2]} & \cdots & A_{[N,\,M]} \end{pmatrix} \Rightarrow \; < \textit{Dirección del elemento a } _{i,\,j} > = < \textit{Dirección del elemento a } _{1,\,1} > + (i-1) * M + (j-1)$$

*ADR(a[i, j]) = ADR(a[1, 1] + (i-1) * M + (j-1)*

2º.- Análisis de la subrutina.

Está incluido dentro de la redacción de la subrutina, en los campos de comentarios.

4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    ╔═══════════════════════════════════════════════════════════════╗
*    ║    Escuela de Informática de la UNED                          ║
*    ║    Asignatura: Estructura y Tecnología de Computadores I      ║
*    ║    Tutoría del Centro Asociado de Plasencia                   ║
*    ║    Examen del 08-09-94                                        ║
*    ╚═══════════════════════════════════════════════════════════════╝
*
* Propósito:    Crear una subrutina que lea el elemento a[i,j] de una matriz
*               de dimensiones N filas y M columnas

* Parámetros pasados a la subrutina:
*               ((A0)) = a[1, 1] Direccionamiento relativo a registro de direcciones
*               (D1)   = N       Direccionamiento mediante registro de datos
*               (D2)   = M       Direccionamiento mediante registro de datos
*               (D3)   = i       Direccionamiento mediante registro de datos
*               (D4)   = j       Direccionamiento mediante registro de datos

* Dato que debe devolver la subrutina:
*               (D0)   = a[i, j]

* El modo de direccionamiento más efectivo en el tratamiento de matrices es el
* relativo a registro con índice:  d(An, Dn)
* Pero en nuestro caso, cuando estamos redactando el código de la subrutina,
* no conocemos la constante d, por tanto debemos utilizarlo de la forma 0(An, Dn)
* Dn contendrá el valor (i-1)*M + (j-1)


Principal       org     $1000
                jsr     LeeElemento  ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14

LeeElemento     move.w  SR,-(SP)        ; Preserva el SR
                move.l  D5,-(SP)        ; Preserva el contenido anterior de D5
                clr.l   D5              ; Utilizaremos D5 como acumulador

                move.w  D3,D5
                subq.w  #1,D5
                mulu    D2,D5           ; A partir de aquí, en D5 son significativos los 32 bits
                add.l   D4,D5
                subq.l  #1,D5

                move.b  0(A0,D5),D0     ; Los elementos de la matriz son de tamaño byte
                move.l  (SP)+,D5        ; Restaura el registro D5

                rtr                     ; Restaura el SR y sale de la subrutina

                end
```

1995. Febrero. Primera semana.

Diseñe una subrutina para el MC68000 que analice si una ristra de caracteres ASCII es o no palíndroma. Una ristra es palíndroma si se lee igual de derecha a izquierda que de izquierda a derecha. Por ejemplo, las dos ristras siguientes cumplen esta propiedad:

```
RISTRA1     DC.B    'DABALEARROZALAZORRAELABAD',0
RISTRA2     DC.B    '35699653',0
```

La dirección de comienzo de la ristra es un parámetro de la subrutina que se le pasa e el registro A0. La ristra finaliza con el carácter nulo en ASCII. El resultado de la subrutina será un 1 si la cadena es palíndroma y un 0 si no lo es; y se devolverá en el registro D5.

Siga el proceso indicado a continuación:

1º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

2º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

1º.- Funcionamiento de la subrutina

El análisis de la subrutina está incluido en su redacción. Los nombres de las etiquetas y los comentarios son lo suficientemente aclaratorios.

4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 02-02-95
*
* Propósito:    Crear una subrutina que analice si una
*               ristra de caracteres es un palíndromo

*       ZONA DE CÓDIGO

Principal       org     $1000
                jsr     Analiza         ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14

Analiza         move.w  SR,-(SP)        ; Comienzo de la subrutina
                movem.l D0-D1/A1,-(SP)
                movea.l A0,A1

* Este bucle calcula la longitud de la ristra de caracteres.
BuscaUltimo     tst.b   (A1)+           ; Examina el actual;
                bne     BuscaUltimo     ; Si no es igual al ASCII nulo,
                                        ; vuelve para examinar el siguiente

* En este momento A1 apunta al byte siguiente al nulo.
* Para que apunte al byte siguiente al último caracter no nulo:
                suba.l   #1,A1

* Este fragmento calcula: [Número máximo de pasadas] = ( (A1) - (A0) ) DIV 2 - 1
                move.l  A1,D0
                sub.l   A0,D0           ; resta
                lsr     #1,D0           ; divide por dos, semilongitud de la ristra
                subq.l  #1,D0           ; resta
* Ahora D0 contiene el número máximo de pasadas por el bucle, pues como máximo
* llegará hasta (D0) = -1 ( una de las condiciones de salto de DBcc ).

* Este bucle va examinando elementos simétricos respecto del centro.
* La salida se produce cuando se cumple alguna de estas dos condiciones:
*    1ª  Dos elementos comparados son diferentes
*    2ª  (D0) = -1

BuscaDiferentes move.b  -(A1),D1          ; Este paso intermedio es necesario pues
                cmp.b   (A0)+,D1          ; el destino en CMP debe ser un registro de datos.
                bne     NoPalindromo      ; Encontró dos diferentes. Sale del bucle
                dbf     D0,BuscaDiferentes ; Sigue buscando si (D0) > -1
TodosSonIguales move.b  #SI,D5
                bra     Fin
NoPalindromo    move.b  #NO,D5

Fin             movem.l (SP)+,D0-D1/A1
                rtr                     ; Sale de la subrutina

*       ZONA DE DATOS
        org     $2000
RISTRA1 dc.b    'DABALEARROZALAZORRAELABAD',0

*       La siguiente directiva ORG tiene dos finalidades:
* 1ª Dejar un espacio de 128 bytes entre estas dos ristras, por si cambiamos la primera por otra más
larga.
* 2ª Colocar ambas ristras en posiciones conocidas de memoria.

* Cuando queramos analizar la primera ristra, le damos al contenido de A0 el valor $2000
* Cuando queramos analizar la segunda ristra, le damos al contenido de A0 el valor $2080

        org     $2080
RISTRA2 dc.b    'No es palíndromo',0
SI      equ     1
NO      equ     0

                end
```

1995. Febrero. Segunda semana.

Sea un vector de datos almacenado en la memoria de un computador con un microprocesador MC68000. Cada elemento del vector ocupa un byte; y contiene un dato numérico entero representado en complemento a dos.

Diseñe una subrutina en el ensamblador del antedicho microprocesador que recorra el vector dado y calcule cuántos de sus elementos contienen un valor negativo par. El vector estará almacenado a partir de la dirección indicada en el registro A0; y el tamaño del vector se pasará en el registro D0. La subrutina devolverá el número de elementos negativos pares en el registro D1.

Siga el proceso indicado a continuación:

1º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

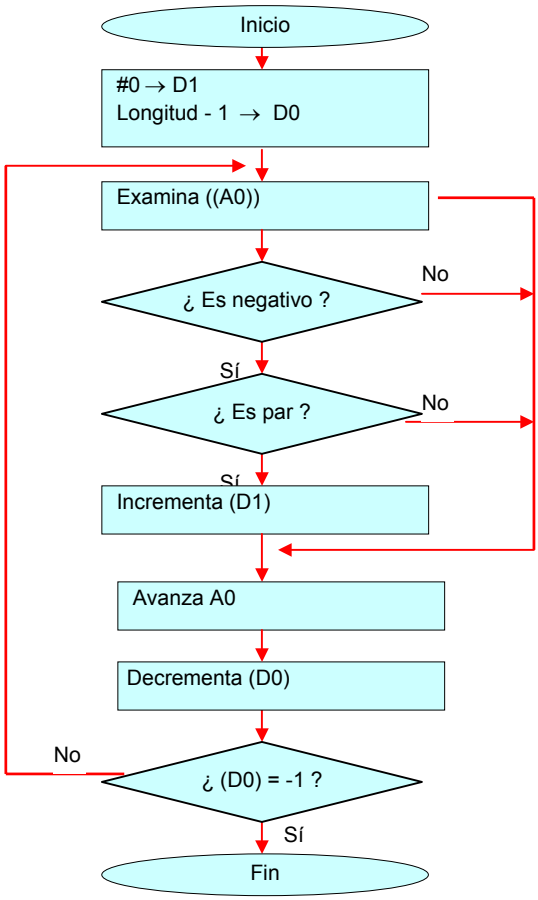
2º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

1º.- Funcionamiento de la subrutina.

En la representación de complemento a dos, todos los números pares (tanto positivos como negativos) tienen su bit menos significativo puesto a cero. Utilizaremos esta propiedad para comprobar la paridad de cada número.

Diagrama de flujo de la subrutina:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 16-02-95
*
* Propósito:    Crear una subrutina que cuente cuántos números negativos pares hay
*               en un vector almacenado en memoria


Principal       org     $1000
                jsr     Cuenta          ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14

Cuenta          move.w  SR,-(SP)        ; Comienzo de la subrutina
                movem.l D0/A0,-(SP)
                clr.l   D1              ; Contador de números negativos pares

                subi.l  #1,D0           ; En la última pasada (D0) debe valer -1

InicioBucle     tst.b   (A0)
                bge     FindelBucle     ; Salta si es mayor o igual que cero
MiraParidad     btst    #0,(A0)
                bne     FindelBucle     ; Si no es par
                addi.l  #1,D1           ; incrementa el contador de números negativos pares
FindelBucle     adda.l  #1,A0           ; Para procesar el siguiente número en la próxima pasada
                dbf     D0,InicioBucle  ; Si (D0) > -1, vuelve para examinar el siguiente

                movem.l (SP)+,D0/A0     ; Restaura los registros usados
                rtr                     ; Sale de la subrutina

                end
```

1995. Septiembre.

Diseñe una subrutina para el MC68000 que cuente el número de veces que aparece un carácter dado en una cadena de caracteres ASCII de longitud desconocida. Se sabe que el último carácter es el nulo ASCII. Los parámetros que recibe la subrutina son:

A0: Registro de direcciones que contiene la dirección del primer carácter de la ristra.
D0: Código ASCII del carácter buscado.

El dato que debe devolver, número de apariciones, estará en el registro D1.

Siga el proceso indicado a continuación:

1º.- Realizar un análisis del funcionamiento de la subrutina, indicando los tipos de instrucciones y los modos de direccionamiento más adecuados para el problema planteado, así como los recursos necesarios (registros, memoria, etc.).

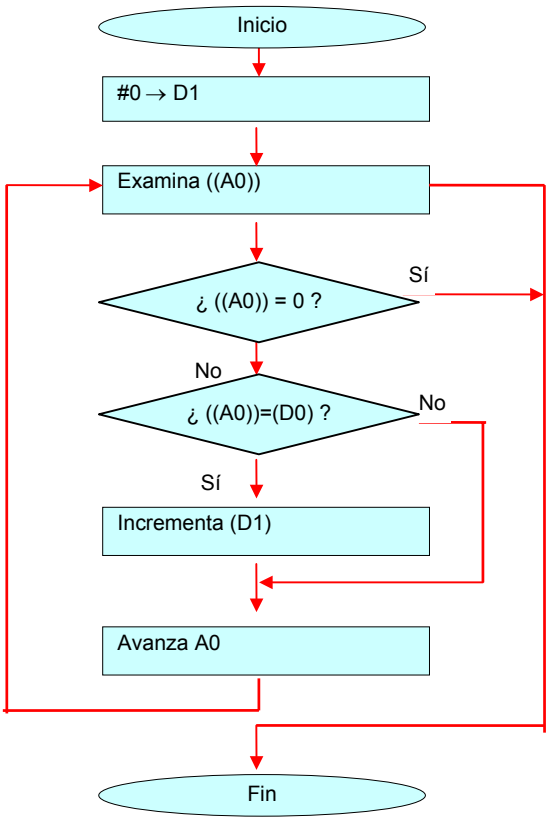
2º.- Redactar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas.

Solución:

1º.- Funcionamiento de la subrutina

Los tipos de instrucciones, modos de direccionamiento y recursos empleados están explicados en el fichero fuente.

Diagrama de flujo de la subrutina:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 07-09-95
*
* Propósito:    Crear una subrutina que cuente el número de veces que aparece
*               un carácter dado en una ristra de caracteres ASCII.
*               El código del carácter buscado lo recibe en el registro D0
*               El número de apariciones lo devuelve en D1.
*               La ristra está almacenada a partir de la posición $2000; y su
*               longitud es desconocida, pero se sabe que su último carácter es 00
*               Para probar el programa, antes de empezar la ejecución,
*               se introduce en D0 es código ASCII ( compactado en hexadecimal )
*               del carácter buscado.

*               ZONA DE CÓDIGO

Principal       org     $1000
                movea.l #DireccionRistra,A0
                jsr     BuscayCuenta    ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14

BuscayCuenta    move.w  SR,-(SP)        ; Comienzo de la subrutina
                movem.l D0/A0,-(SP)
                clr.l   D1              ; Contador de apariciones

* Este bucle es tipo LOOP. La condición de salida está dentro del cuerpo.

InicioBucle     tst.b   (A0)
                beq     FueradelBucle   ; Condición de salida del bucle ( tipo LOOP )
                cmp.b   (A0),D0         ; Compara el carácter actual con el buscado
                bne     BuscaOtro       ; Salta si no son iguales
                addi.l  #1,D1           ; incrementa el contador de apariciones
BuscaOtro       adda.l  #1,A0           ; Para procesar el siguiente carácter en la próxima pasada
                bra     InicioBucle     ; Fin del bucle. La salida no está aquí

FueradelBucle   movem.l (SP)+,D0/A0     ; Restaura los registros usados
                rtr                     ; Sale de la subrutina

*               ZONA DE DATOS
                org     $2000
DireccionRistra dc.b    'Esto es una ristra. Aquí se buscan los caracteres',0

                end
```

1996. Febrero. Primera semana.

Dados el centro (x_c, y_c) y el radio r de una circunferencia, puede conocerse la posición de un punto $P(x, y)$ respecto la misma mediante alguno de los siguientes criterios:

Si $(x - x_c)^2 + (y - y_c)^2 < r^2$, entonces el punto es interior a la circunferencia.
Si $(x - x_c)^2 + (y - y_c)^2 = r^2$, entonces el punto está en la circunferencia.
Si $(x - x_c)^2 + (y - y_c)^2 > r^2$, entonces el punto es exterior a la circunferencia.

Diseñe una subrutina en el lenguaje ensamblador del MC68000 que reciba las coordenadas del centro y el radio de una circunferencia en los registros D0, D1 y D2, respectivamente; e indique la posición relativa respecto dicha circunferencia de un punto *P*, cuyas coordenadas se reciben en los registros D3 y D4. Todos los datos son números enteros de 16 bits positivos o negativos y almacenados en complemento a dos. Suponga que en ningún caso se producirá desbordamiento en las operaciones aritméticas realizadas. La subrutina devolverá en el byte menos significativo del registro D5 el carácter *'I'* en ASCII si el punto *P* es interior; el carácter *'E'* si es exterior; o el carácter *'P'* si pertenece a la circunferencia.

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[15:0] recibe x_c.
En D1[15:0] recibe y_c.
En D2[15:0] recibe r.
En D3[15:0] recibe x.
En D4[15:0] recibe x.

b) Resultado devuelto por la subrutina al programa principal:
En D5[7:0] devuelve el código ASCII de *'I'*, *'E'* ó *'P'*; según el punto sea interior, exterior o frontera, respectivamente.

2º.- Descripción textual del algoritmo.

Se realiza el cálculo de la expresión $(x - x_c)^2 + (y - y_c)^2$; y según sea menor, mayor o igual que r^2, se ejecutará un bloque de instrucciones u otro. Las operaciones de $(x_1 - x_2)^2$ las aislaremos en otra subrutina, llamada *QDistancia*$(x_1 - x_2)$.

3º.- Descripción detallada del algoritmo.

a) En pseudocódigo tipo lenguaje de alto nivel:

```
QSuma    := QDistancia(x - xc) + QDistancia(y - yc)  ;
QRadio   := r * r ;

IF QSuma < QRadio
   THEN
      Situacion := Interior ;
   ELSE
      IF QSuma = QRadio
         THEN
            Situacion := Frontera ;
         ELSE
            Situacion := Exterior ;
      END
END
```

b) En un pseudocódigo más adaptado a las instrucciones de salto de los lenguajes ensambladores, el fragmento de los IF anidados tiene esta forma:

```
            [ Salto a E1 si se cumple la condición  QSuma ≥ QRadio ]
            [ Situacion := Interior ]
            [ Salto incondicional a FIN ]
      E1    [ Salto a E2 si se cumple la condición QSuma > QRadio ]
            [ Situación := Frontera ]
            [ Salto incondicional a FIN ]
      E2    [ Situación := Exterior ]
      FIN   [ Indicar la situación ]
```

Descripción de la subrutina *QDistancia*$(x_1 - x_2)$

En todas las subrutinas que se piden en los ejercicios de los exámenes se especifica que deben recibir los argumentos y devolver los datos de retorno en sendos registros. En una situación más realista (véase las UUDD, página 438), los argumentos se pasan a la subrutina almacenándolos previamente en la pila. Vamos a aprovechar que la subrutina auxiliar*QDistancia*$(x_1 - x_2)$ la especificamos totalmente nosotros, su interfaz no nos viene impuesta, para construir una subrutina que pasa sus argumentos y resultados utilizando la pila.

En esta subrutina no tiene mucho interés explicar su algoritmo, pues se limita a realizar las operaciones $(x_1 - x_2)^2$. Pero sí es ilustrativo ver cómo evoluciona la pila, desde antes de preparar los argumentos, hasta después de extraer los datos de retorno.

Codificación de la subrutina; y ejemplo de utilización desde un programa principal:

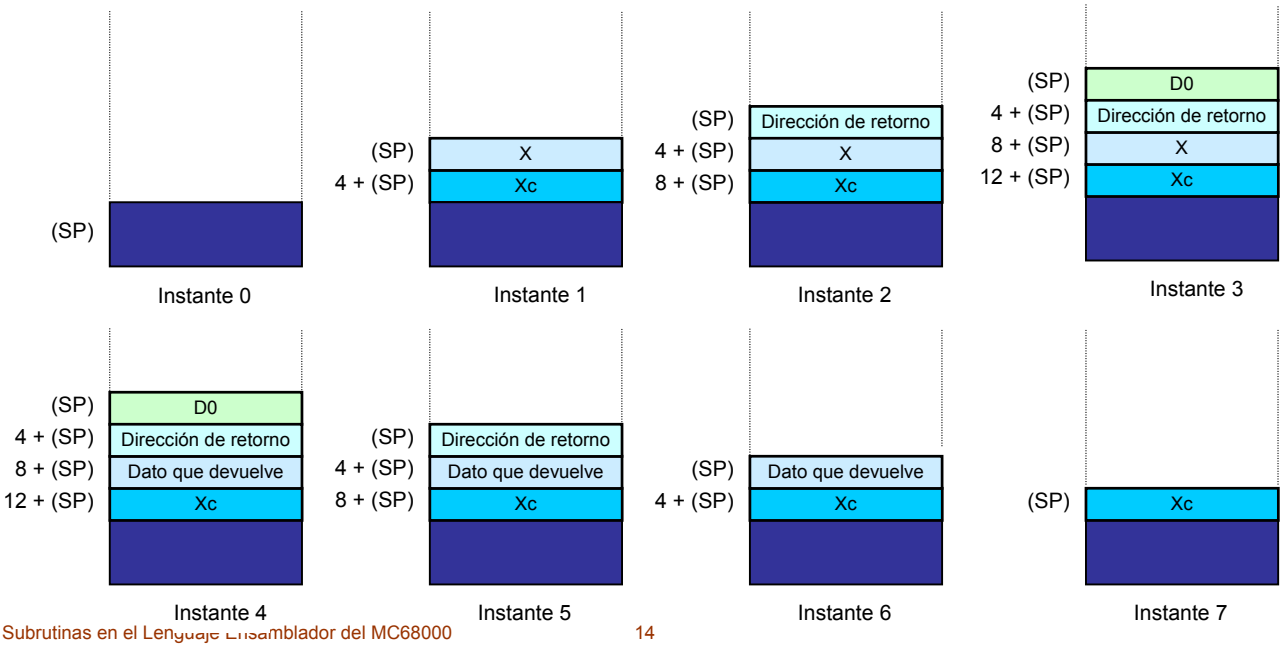
```
Principal  org     $1000       ; Instante 0. Antes de preparar la entrada en la subrutina
           movem.l D0-D1,-(SP) ; Instante 1. Envía los argumentos a la pila
           jsr     QDistancia  ; Instante 2. Llamada a la subrutina
           move.l  (SP)+,D3    ; Instante 7. Extraemos de la pila el dato devuelto por la subrutina
           move.l  (SP)+,D1    ; Instante 8. (Igual al instante 0) Liberamos el lugar ocupado por Xc

           move.b  #228,D7
           trap    #14         ; Fin del programa principal

QDistancia move.l  D0,-(SP)    ; Instante 3. Preserva los 32 bits de D0
           move.l  8(SP),D0    ; Copia X ( sin extraerlo de la pila ) en D0
           sub.l   12(SP),D0   ; Realiza la operación (X-Xc)
           muls    D0,D0       ; Realiza la operación (X-Xc) * (X-Xc)
           move.l  D0,8(SP)    ; Instante 4. Coloca en la pila el dato que debe devolver
           move.l  (SP)+,D0    ; Instante 5. Restaura el contenido de D0
           rts                 ; Instante 6.

           end
```

Notas:

1ª.- En la instrucción `movem.l D0-D1,-(SP)` , el procesador almacena siempre todos los cuatro bytes de cada registro, independientemente del sufijo empleado.

2ª.- Las unidades del desplazamiento en el direccionamiento relativo a registro no se ven influenciadas por el sufijo del nemotécnico. Por ejemplo, en la instrucción `move.l 8(SP),D0` , aunque el dato X tiene encima dos palabras largas y el sufijo del nemotécnico es L, el desplazamiento se mide en bytes, no en palabras largas.

Evolución de la pila:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*       Escuela de Informática de la UNED
*       Asignatura: Estructura y Tecnología de Computadores I
*       Tutoría del Centro Asociado de Plasencia
*       Examen del 30-01-96
*
* Autor:        José Garzía
* Propósito:    Analizar la posición relativa de un punto respecto una circunferencia

*       ZONA DE CÓDIGO

Principal       org     $1000           ; Comienzo del programa principal
                jsr     Analiza         ; Llamada a la subrutina primaria
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

*               Comienzo de la subrutina secundaria

* Esta subrutina recibe dos argumentos ( X y Xc ) almacenados en la pila;
* realiza la operación (X-Xc) * (X-Xc) ; y el resultado lo devuelve en la pila

QDistancia      move.l  D0,-(SP)        ; Preserva los 32 bits de D0
                move.l  8(SP),D0        ; Copia X ( sin extraerlo de la pila ) en D0
                sub.l   12(SP),D0       ; Realiza la operación (X-Xc)
                muls    D0,D0           ; Realiza la operación (X-Xc) * (X-Xc)
                move.l  D0,8(SP)        ; Coloca en la pila el dato que debe devolver
                move.l  (SP)+,D0        ; Restaura el contenido de D0
                rts                     ; Fin de la subrutina secundaria


*               Comienzo de la subrutina primaria

Analiza         move.w  SR,-(SP)
                movem.l D0-D4,-(SP)     ; Preserva los registros que va a usar

                movem.l D0/D3,-(SP)     ; Prepara la 1ª llamada a la secundaria
                jsr     QDistancia      ; 1ª llamada
                move.l  (SP)+,D0        ; Dato devuelto en la 1ª llamada
                move.l  (SP)+,D3        ; Vacía el lugar ocupado en la pila por D3

                movem.l D1/D4,-(SP)     ; Prepara la 2ª llamada a la secundaria
                jsr     QDistancia      ; 2ª llamada
                move.l  (SP)+,D1        ; Dato devuelto en la 2ª llamada
                move.l  (SP)+,D4        ; Vacía el lugar ocupado en la pila por D4

                add.l   D1,D0           ; QSuma
                mulu    D2,D2           ; QRadio
                cmp     D2,D0

*               Fragmento de código que transcribe los IF ELSE anidados del algoritmo

                bcc MayorOigual
                move.b  #Interior,D5
                bra Fin
MayorOigual     bhi     Mayor
                move.b  #Frontera,D5
                bra Fin
Mayor           move.b  #Exterior,D5

Fin             movem.l (SP)+,D0-D4     ; Restaura los registros que ha usado
                rtr                     ; Fin de la subrutina primaria


*       ZONA DE DATOS

                org     $2000
Interior        equ     'I'             ; 'I' en ASCII es $49 = 73
                cnop    0,2
Frontera        equ     'P'             ; 'P' en ASCII es $50 = 80
                cnop    0,2
Exterior        equ     'E'             ; 'E' en ASCII es $45 = 69

                end
```

Ejemplos de ejecuciones con diferentes conjuntos de datos de entrada:

DATOS DE ENTRADA					RESULTADO	INTERPRETACIÓN
D0	D1	D2	D3	D4	D5[7:0]	
0	0	3	2	0	49	Interior
0	0	3	3	0	50	Frontera
0	0	1	2	0	45	Exterior

1996. Febrero. Segunda semana.

El logaritmo por defecto en base n de un número X es el mayor entero p tal que $n^p \geq X$. Por ejemplo, el logaritmo por defecto en base 10 del número 9 es 0; el de 85 es 1; el de 277 es 2; etc.

Diseñe una subrutina en el ensamblador del MC68000 que reciba como parámetros sendos números n y X; y calcule el logaritmo por defecto en base n del número X. Los números n y X son enteros positivos de 16 bits representados en binario sin signo; y se encuentran almacenados en los registros D0 y D1, respectivamente. La subrutina devolverá el valor calculado en el registro D2. Por supuesto, ni está definido el logaritmo de un número $X<0$; ni tiene sentido el logaritmo en base $n=1$. Suponga que $X>0$; y que siempre se cumplirá que $n^p < 2^{16}$, es decir, que n^p siempre cabe en un número de 16 bits.

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[07:00] recibe la base n.
En D1[15:00] recibe el número X.

b) Resultado devuelto por la subrutina al programa principal:
En D2[07:00] devuelve el logaritmo por defecto.

2º.- Descripción textual del algoritmo.

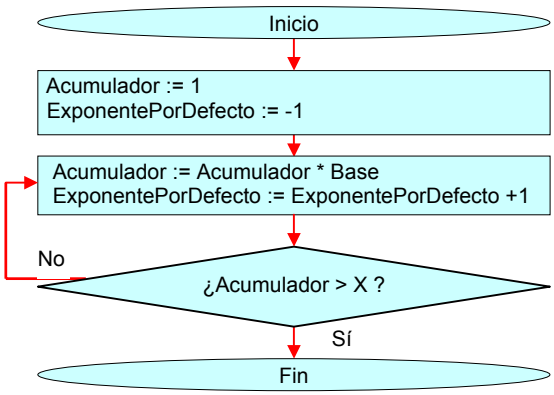
El algoritmo es iterativo. Usaremos un acumulador para ir almacenando las potencias parciales; y un contador que nos indique el exponente por defecto en curso.

3º.- Descripción detallada del algoritmo.

a) En pseudocódigo tipo lenguaje de alto nivel:

```
ExponentePorDefecto  := -1 ;
REPETIR
   INC( ExponentePorDefecto ) ;

HASTAQUE  (Base)^( ExponentePorDefecto  + 1)  > X
```

b) Con diagrama de flujo:



Constantes:

Base n. Almacenada en D0[07:00].
Número X. Almacenado en D1[15:00].

Variable intermedia:

`Acumulador`. Es el resultado intermedio $(Base)^{(Exponentepordefecto+1)}$, almacenado en D3[15:0].

Variable devuelta:

`ExponentePorDefecto`. Es el contador, almacenado en D2[7:0].

El contador va retrasado una unidad respecto del exponente auténtico al que se refiere el acumulador. Es decir, en cada pasada por el bucle, *`ExponentePorDefecto`* es una unidad menor que el exponente de la variable intermedia $(Base)^{(Exponentepordefecto+1)}$.

La finalidad de este desfase es que a la salida del bucle (provocada cuando $(Base)^{(Exponentepordefecto+1)} > X$) la variable *`ExponentePorDefecto`* nos dé el entero inmediatamente anterior (logaritmo por defecto) al exponente que provocó la salida.

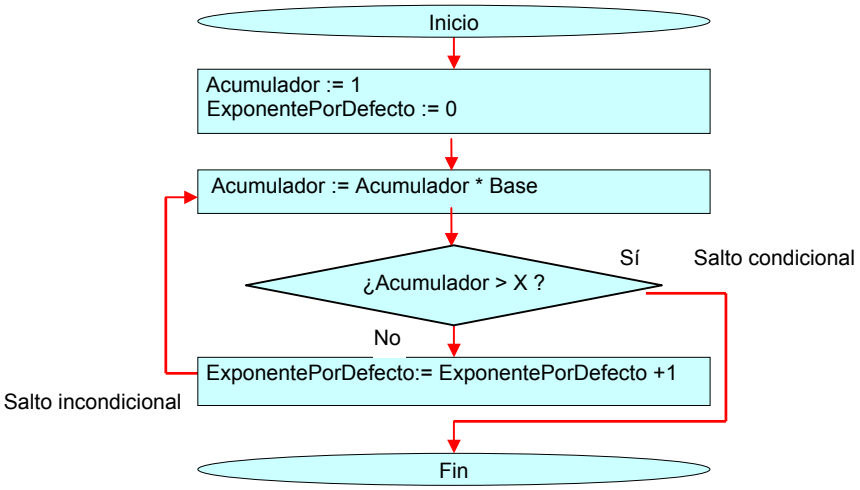
4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*       Escuela de Informática de la UNED
*       Asignatura: Estructura y Tecnología de Computadores I
*       Tutoría del Centro Asociado de Plasencia
*       Examen del 13-02-96
*
*
* Propósito:    Crear una subrutina que calcule logaritmos por defecto

                org     $1000
Principal       jsr     LogPorDefecto   ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

LogPorDefecto01 move.w  SR,-(SP)
                move.l  D3,-(SP)        ; Comienzo de la subrutina
                move.w  #1,D3           ; Inicializa el Acumulador
                move.b  #-1,D2          ; Inicializa el Contador

InicioBucle01   mulu    D0,D3           ; Actualiza el Acumulador
                addq.b  #1,D2           ; Incrementa el Contador
                cmp     D1,D3
                bls     InicioBucle01   ; Salta si D3 menor o igual que D1
                movem.l (SP)+,D3        ; Restaura el registro usado
                rtr                     ; Sale de la subrutina

                end                     ; Fin del ensamblado
```

Aquí concluye la resolución de todos los requerimientos del problema. Si realizamos una ejecución en modo traza; o si establecemos un punto de ruptura en la instrucción `move.b #-1,D2` , veremos que tras su ejecución, el byte menos significativo de D2 contiene FF, es decir, -1 en complemento a dos.

Es interesante analizar la situación en que nos encontraríamos si en el microprocesador utilizado los circuitos aritmético-lógicos de suma no pudieran operar con números negativos. El contador (la variable *`ExponentePorDefecto`*) no podría ser inicializado a -1. Con los recursos disponibles en esta nueva situación, el desfase no podría ser introducido en la inicialización de las variables. Podemos producir un desfase de otra forma: realizando la comparación después del cálculo de la variable intermedia; y antes de incrementar el contador, como refleja este diagrama de flujo:

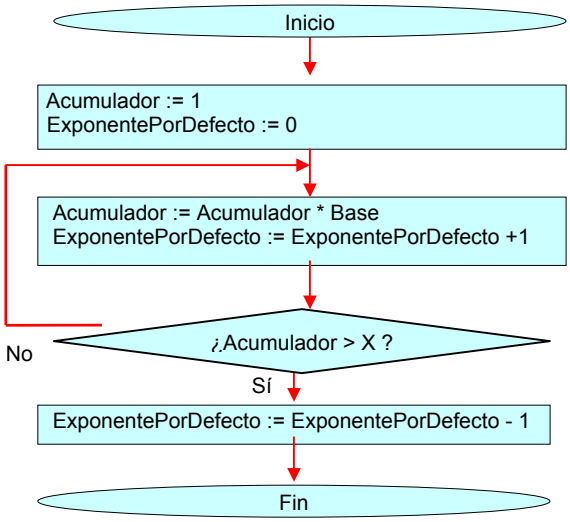


Haciendo un estudio del número de saltos del algoritmo, vemos que hay un salto condicional y un salto incondicional. Un algoritmo eficiente debe tener el menor número posible de saltos, para lograr estos dos objetivos:

1º.- *Mayor facilidad de redacción y de mantenimiento*. Un programa con muchos saltos tiene una lógica muy confusa.

2º.- *Mayor velocidad de ejecución*. Los saltos ralentizan mucho los programas. El Contador de Programa (PC) siempre se incrementa automáticamente tras la decodificación de la instrucción. En caso de haber un salto, ese incremento también lo hace, pero no sirve para nada. Debe buscar la dirección de la próxima instrucción de forma no automática, lo cual requiere más tiempo que si no hubiera salto.

En el algoritmo del siguiente diagrama de flujo sólo hay un salto condicional. Como contrapartida, estamos obligados a hacer una restauración del contador después de salir del bucle. Pero es sólo una instrucción aritmética, el retardo es siempre el mismo independientemente de cuáles sean los números procesados. En cambio, en el anterior, si el número es muy grande (números procesados muy grandes), el incremento del retardo (un salto más en cada pasada) aumenta linealmente con dicho número de pasadas.



Para poder comparar las tres subrutinas, las hemos codificado juntas en un mismo programa principal. El fichero fuente que las contiene es este:

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 13-02-96
*
*
* Propósito:    Crear subrutinas que calculen logaritmos por defecto

                org     $1000
Principal       jsr     LogPorDefecto01 ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

* Primera Subrutina

LogPorDefecto01 move.w  SR,-(SP)
                move.l  D3,-(SP)        ; Comienzo de la subrutina
                move.w  #1,D3           ; Inicializa el Acumulador
                move.b  #-1,D2          ; Inicializa el Contador

InicioBucle01   mulu    D0,D3           ; Actualiza el Acumulador
                addq.b  #1,D2           ; Incrementa el Contador
                cmp     D1,D3
                bls     InicioBucle01   ; Salta si D3 menor o igual que D1
                movem.l (SP)+,D3        ; Restaura el registro usado
                rtr                     ; Sale de la subrutina

* Segunda Subrutina

LogPorDefecto02 move.w  SR,-(SP)
                move.l  D3,-(SP)        ; Comienzo de la subrutina
                move.w  #1,D3           ; Inicializa el Acumulador
                move.b  #0,D2           ; Inicializa el Contador

Intermedia      mulu    D0,D3           ; Actualiza el Acumulador
                cmp     D1,D3
                bhi     Supera          ; Salta si D3 es mayor que D1
SeQuedaCorto    addq.b  #1,D2           ; Incrementa el Contador
                bra     Intermedia      ; Salto incondicional
Supera          movem.l (SP)+,D3        ; Restaura el registro usado
                rtr                     ; Sale de la subrutina

* Tercera Subrutina

LogPorDefecto03 move.w  SR,-(SP)
                move.l  D3,-(SP)        ; Comienzo de la subrutina
                move.w  #1,D3           ; Inicializa el Acumulador
                move.b  #0,D2           ; Inicializa el Contador

InicioBucle03   mulu    D0,D3           ; Actualiza el Acumulador
                addq.b  #1,D2           ; Incrementa el Contador
                cmp     D1,D3
                bls     InicioBucle03   ; Salta si D3 menor o igual que D1
                subq.b  #1,D2           ; Restaura el exponente
                movem.l (SP)+,D3        ; Restaura el registro usado
                rtr                     ; Sale de la subrutina

                end                     ; Fin del ensamblado
```

1996. Septiembre.

Diseñe una subrutina en el ensamblador del MC68000 que, dado un dato de tamaño palabra (16 bits), inspeccione los bits que lo componen y contabilice cuántos de entre ellos se encuentran a 1. El dato de entrada se proporciona en la palabra menos significativa del registro D0. La subrutina devolverá el número de bits a 1 en el byte menos significativo del registro D1.

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[15:00] recibe la palabra de entrada

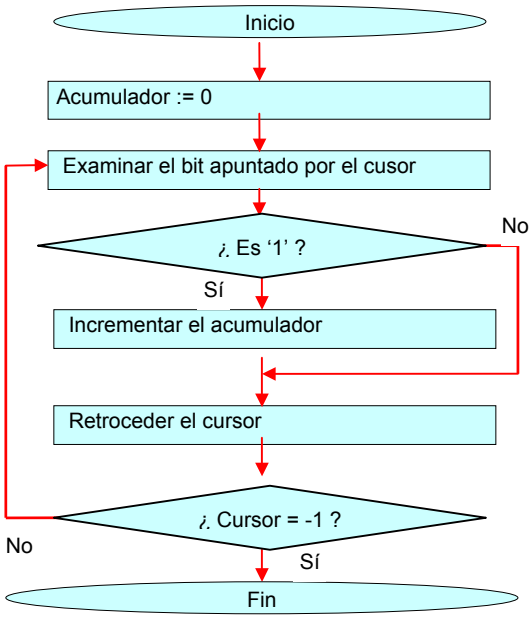
b) Resultado devuelto por la subrutina al programa principal:
En D1[07:00] devuelve el número de bits puestos a 1.

2º.- Descripción textual del algoritmo.

Se realiza un recorrido por la palabra, desde el bit 15 hasta el bit 0, utilizando una variable tipo cursor. El número de 1's se va acumulando en una variable tipo acumulador.

3º.- Descripción detallada del algoritmo.

Diagrama de flujo:



Constante:
Palabra analizada. Almacenada en D0[15:00].

Variable intermedia:
Cursor. Almacenado en D2[31:00]. Se inicializa a 15.

Variable devuelta:
Acumulador. Almacenado en D1[07:00]. Se inicializa a 0.

4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

```
*
*    ╔═══════════════════════════════════════════════════════════╗
*    ║    Escuela de Informática de la UNED                      ║
*    ║    Asignatura: Estructura y Tecnología de Computadores I  ║
*    ║    Tutoría del Centro Asociado de Plasencia               ║
*    ║    Examen del 03-09-96                                    ║
*    ╚═══════════════════════════════════════════════════════════╝
*
* Propósito:    Crear una subrutina que recibe una palabra de 16 bits en D0[15:0]
*               y devuelve en D1[7:0] el número de 1's de dicha palabra

                org     $1000
Principal       jsr     CuentaUnos      ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14

CuentaUnos      move.w  SR,-(SP)        ; Comienzo de la subrutina
                movem.l D2,-(SP)

                clr.b   D1              ; Contador de 1's
                move.l  #15,D2          ; Cursor que apunta al bit examinado

InicioBucle     btst    D2,D0           ; Examinamos un bit
                beq     EsCero
EsUno           addq.b  #1,D1
EsCero          dbf     D2,InicioBucle  ; Decrementa D2 y si procede, salta
                movem.l (SP)+,D2        ; Restaura el registro usado
                rtr                     ; Restaura el SR y sale de la subrutina

                end
```

1996. Septiembre (reserva).

Diseñar una subrutina en ensamblador del M68000 que convierta un número decimal representado de forma alfanumérica a su representación en complemento a 2. El número de entrada se encontrará almacenado en memoria como una secuencia de caracteres ASCII, correspondientes a los dígitos numéricos, precedida por el carácter "-" si el número es negativo. El final de la secuencia de caracteres estará señalado por la aparición del carácter ASCII nulo. La subrutina recibirá la dirección del primer carácter ASCII de la secuencia en el registro *A*0. La subrutina devolverá la representación en complemento a 2 en el registro *D*0. Suponer que el resultado se puede devolver en 16 bits, y que en ningún caso se producirá desbordamiento en las operaciones aritméticas realizadas.

Seguir el procedimiento indicado a continuación:
1.- Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2.- Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3.- Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4.- Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

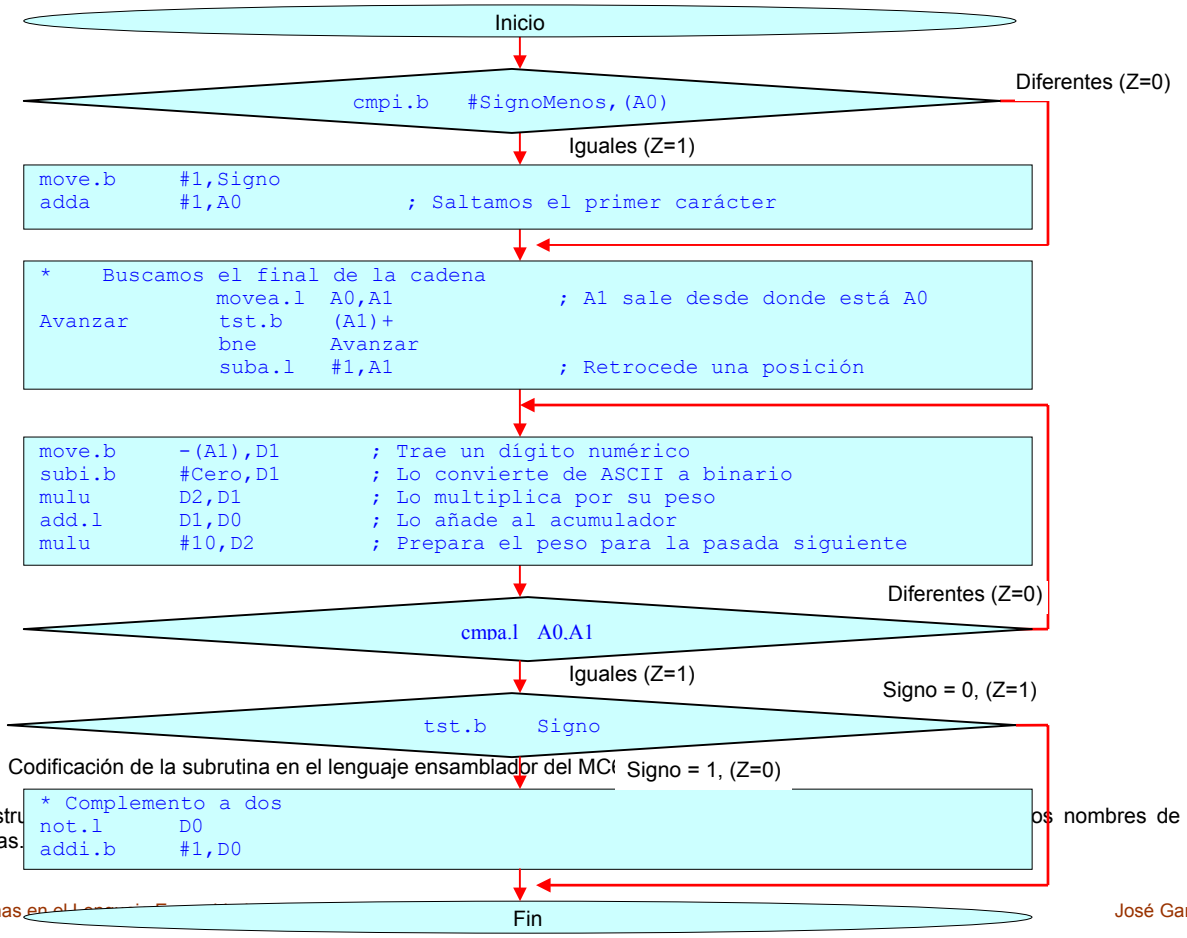
a) Parámetros pasados a la subrutina por el programa principal:
En A0[31:00] recibe la dirección del comienzo de la secuencia de caracteres.

b) Resultado devuelto por la subrutina al programa principal:
En D0[31:00] devuelve el resultado, la ristra de caracteres convertida a entero, en binario.

2º.- Descripción textual del algoritmo.

Explora el final de la secuencia. Va leyendo dígitos de la secuencia, empezando por los menos significativos. Los va convirtiendo a enteros. Y realiza la operación $|X| = \sum_{(A0)}^{(A1)} x_i \cdot 10^i$ en un acumulador aditivo.

3º.- Descripción detallada del algoritmo. Diagrama de flujo
:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC

Las instru os nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 07-09-96
*
*
* Propósito: Crear una subrutina de conversión de decimal (en ASCII) a binario
* Autor:     José Garzía

*               ZONA DE CÓDIGO

Principal         org      $1000
                  movea.l  #Almacen,A0
                  jsr      Convertir         ; Llamada a la subrutina
                  move.b   #228,D7
                  trap     #14

Convertir         move.w   SR,-(SP)          ; Comienzo de la subrutina
                  movem.l  A1/D1-D2,-(SP)

*                 En primer lugar comprobamos el signo
                  cmpi.b   #SignoMenos,(A0)
                  bne      Siga
                  move.b   #1,Signo
                  adda     #1,A0             ; Saltamos el primer carácter
Siga              nop

*                 En segundo lugar buscamos el final de la cadena
                  movea.l  A0,A1             ; A1 sale desde donde está A0
Avanzar           tst.b    (A1)+
                  bne      Avanzar
                  suba.l   #1,A1             ; Retrocede una posición

*                 Inicializaciones previas a la entrada en el bucle
                  move.l   #0,D0             ; inicializa el acumulador
                  move.l   #1,D2             ; Peso del bit menos significativo


*                 Bucle de la subrutina
InicioBucle       andi.l   #$000F,D1         ; Borra la parte más significativa
                  move.b   -(A1),D1          ; Trae un dígito numérico
                  subi.b   #Cero,D1          ; Lo convierte de ASCII a binario
                  mulu     D2,D1             ; Lo multiplica por su peso correspondiente
                  add.l    D1,D0             ; Lo añade al acumulador
                  mulu     #10,D2            ; Prepara el peso para la pasada siguiente
                  cmpa.l   A0,A1             ; Si A1 todavía no ha llegado hasta A0
                  bne      InicioBucle       ; va a buscar el siguiente dígito

FueradelBucle     tst.b    Signo
                  beq      NoComplementar
Complementar      not.l    D0
                  addi.b   #1,D0

NoComplementar    nop

                  movem.l  (SP)+,A1/D1-D2    ; Restaura los registros usados
                  rtr                        ; Sale de la subrutina

*                 ZONA DE DATOS
                  org      $2000
Cero              equ      '0'               ; '0' es lo mismo que $30, es decir, 48
SignoMenos        equ      '-'
Signo             dc.b     0
Almacen           ds.b     10                ; Reservamos sitio para diez dígitos decimales
                  end                        ; pero si hay más de diez, también podrán almacenarse
                                             ; pues detrás de esta línea no se almacena nada
```

1997. Febrero. Primera semana.

Diseñe una subrutina en el ensamblador del MC68000 para conversión de formato binario a decimal de números enteros sin signo de 16 bits. La subrutina recibirá el número binario en la palabra menos significativa del registro D0; y deberá generar una cadena de caracteres ASCII correspondiente a la secuencia de cifras decimales que represente al número pedido. Los sucesivos caracteres de la representación en ASCII se almacenarán en memoria de forma consecutiva y *ordenada de mayor a menor peso*. Detrás del último carácter de la cadena, se añadirá un carácter nulo. La subrutina devolverá en el registro A0 la dirección de comienzo de la cadena generada.

Obsérvese que las cifras decimales de un número se pueden conseguir, *ordenadas de menor a mayor peso*, como la secuencia de restos de división que van apareciendo al dividir por 10 consecutivamente el número dado, desde su valor inicial hasta que el cociente es 0. Cada cifra así obtenida se puede pasar a ASCII sumándole el código ASCII correspondiente al '0' (el 48).

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[15:00] recibe el número binario.

b) Resultado devuelto por la subrutina al programa principal:
En A0[15:00] devuelve la dirección de comienzo de la cadena generada.

2º.- Descripción textual del algoritmo.

. Calculamos la longitud del número decimal.
. Almacenamos el carácter nulo en la posición apuntada por A0 más dicha longitud.
. Vamos convirtiendo los dígitos decimales a ASCII, de uno en uno; y almacenándolos en la posición apuntada por A0 más la longitud actual, la cual se decrementa siempre en una unidad.

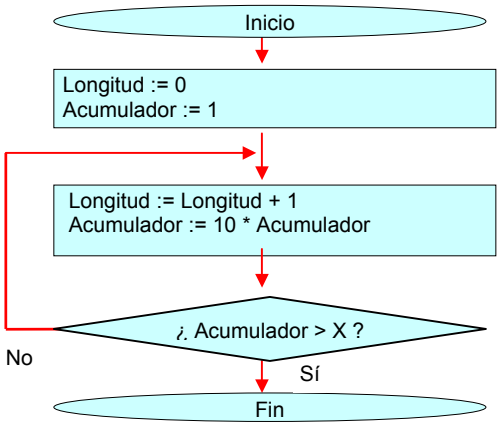
3º.- Descripción detallada del algoritmo.

a) Fragmento para medir el número de dígitos decimales:

Constante:
Número analizado. Almacenado en D0[15:00].

Variables intermedias:
Longitud. Almacenado en D1[31:00]. Se inicializa a 0.
Acumulador. Almacenado en D2[07:00]. Se inicializa a 1.

Diagrama de flujo:



b) Resto de la subrutina

Constantes:

En D3[31:00] almacenamos el número #16, pues `lsr.l #16,D2` tendría el primer operando fuera de rango.

En D4[31:00] almacenamos el número #10.

Variables intermedias:

D1[31:00] es la procedente del bloque anterior.

D0[31:00] inicialmente contiene el número dado. Funciona como divisor en la operación $\frac{(D4)}{(D0)}$, tras la cual:

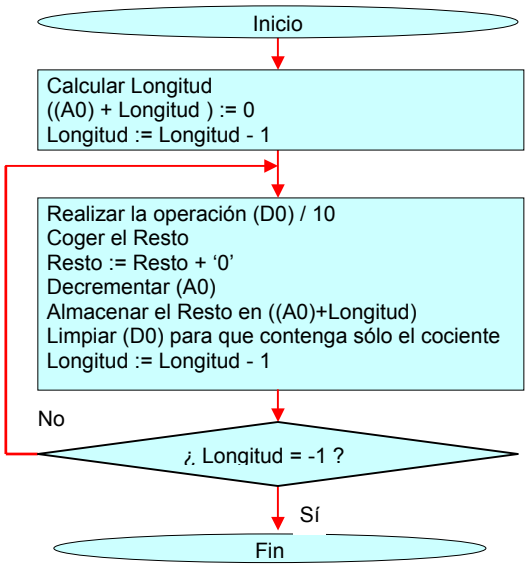
D0[31:16] contiene el resto.
D0[15:00] contiene el cociente.

D2. Sirve para recoger el resto, en D2[31:16], filtrándolo con la máscara #$FFFF0000. Este resto, tras desplazarlo a la palabra menos significativa de D2, se almacena en memoria.

Códigos ASCII de los caracteres numéricos:

Carácter:	0	1	2	3	4	5	6	7	8	9
Código en decimal:	48	49	50	51	52	53	54	55	56	57
Código en hexadecimal:	30	31	32	33	34	35	36	37	38	39

Diagrama de flujo:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 28-01-97
*
*
* Propósito: Crear una subrutina de conversión binario a decimal
* Autor:     José Garzía

*       ZONA DE CÓDIGO

Principal       org     $1000           ; Comienzo del programa principal
                jsr     Convierte       ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

Convierte       move.w  SR,-(SP)        ; Preserva el SR
                movem.l D0-D4,-(SP)     ; Preserva los registros que va a utilizar

* Bloque que mide el número de dígitos decimales

                clr.l   D1              ; Longitud := 0
                move.l  #1,D2           ; Acumulador := 1
MideLongitud    addq.l  #1,D1           ; Longitud := Longitud + 1
                mulu    #10,D2          ; Acumulador := 10 * Acumulador
                cmp.l   D0,D2
                bls     MideLongitud

* Bloque que prepara el espacio de memoria reservado para el almacenamiento

                movea.l #Almacen,A0     ; A0 apunta al comienzo del espacio reservado
                adda.l  D1,A0           ; A0 apunta al byte inmediatamente posterior a la cadena
                move.b  #0,(A0)
                subq.l  #1,D1
                move.l  #16,D3
                move.l  #10,D4

* Bloque que realiza las operaciones de conversión y almacenamiento

InicioBucle     divu    D4,D0
                move.l  #$FFFF0000,D2   ; Introducimos una máscara en D2
                and.l   D0,D2           ; Filtramos el resto de la división en D2[31:16]
                lsr.l   D3,D2           ; Ahora el resto está en D2[15:00]
                addi.l  #Cero,D2        ; Se le suma el código ASCII de '0'
                andi.l  #$0000FFFF,D0   ; Borramos el resto que había en D0[31:00]
                move.b  D2,-(A0)         ; Almacenamos en memoria el código ASCII del número
                dbf     D1,InicioBucle

                movem.l (SP)+,D0-D4     ; Restaura los registros utilizados
                rtr                     ; Restaura el SR y sale de la subrutina

*       ZONA DE DATOS

                org     $2000
Almacen         ds.b    20              ; Reservamos espacio para 20 dígitos decimales
                cnop    0,2
Cero            equ     '0'             ; '0' en ASCII es $30=48
                end                     ; Fin del ensamblado
```

1997. Febrero. Segunda semana.

Diseñe una subrutina en el ensamblador del MC68000 que calcule la distancia entre dos combinaciones binarias cualesquiera. La subrutina recibirá dos datos de 32 bits en los registros D0 y D1; y devolverá la distancia entre ellos (que obviamente será un número entre 0 y 32) en el byte menos significativo del registro D2

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[31:00] recibe el primer operando.
En D1[31:00] recibe el segundo operando.

b) Resultado devuelto por la subrutina al programa principal:

En D2[07:00] devuelve la distancia entre los operandos: Nº de veces en las que para una misma posición en los operandos, no coincide el bit de un operando con el del otro.

2º.- Descripción textual del algoritmo.

Vamos comparando los dígitos de ambos operandos; desde la posición 31 hasta la 0. Previamente se extraen de cada operando; mediante sendas máscaras y una operación AND.

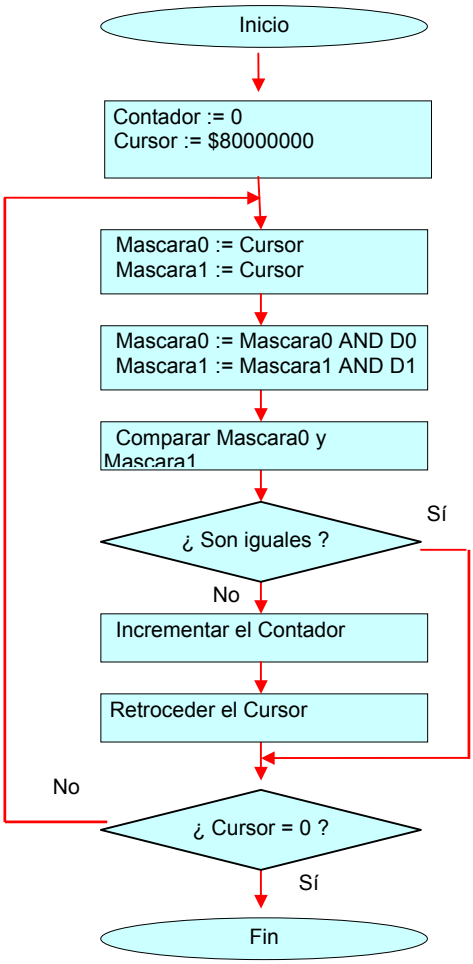
3º.- Descripción detallada del algoritmo.

Constantes:
D0 y D1. Son los operandos.

Variables intermedias:
D3. Hace las funciones de cursor. Apunta a la posición examinada en cada pasada por el bucle. Es el valor inicial de las máscaras.
D4. Máscara para el primer operando.
D5. Máscara para el segundo operando.

Variable resultado final:
D2. Variable tipo acumulador (aditivo, se inicializa a cero).

Diagrama de flujo de la subrutina:



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 11-02-97
*
*
* Propósito: Crear una subrutina que calcule la distancia entre dos combinaciones binarias
*

Principal  org     $1000           ; Comienzo del programa principal

           jsr     Subrutina       ; Llamada a la subrutina

           move.b  #228,D7
           trap    #14             ; Fin del programa principal

Subrutina  move.w  SR,-(SP)        ; Preserva el SR
           movem.l D3-D5,-(SP)     ; Preserva los registros que va a utilizar

           clr.b   D2
           move.l  #$80000000,D3

InicioBucle     move.l D3,D4       ; Actualiza la 1ª Máscara
                move.l D3,D5       ; Actualiza la 2ª Máscara

                and.l  D0,D4
                and.l  D1,D5

                cmp.l  D4,D5
                beq    SonIguales
SonDiferentes   addi.b #1,D2       ; Incrementa el contador de pares diferentes
SonIguales      lsr.l  #1,D3       ; Avanza el Cursor

                tst.l  D3
                bgt    InicioBucle ; Vuelve al comienzo del bucle siempre que (D3) > 0


           movem.l  (SP)+,D3-D5    ; Restaura los registros utilizados
           rtr                     ; Restaura el SR y sale de la subrutina

           end                     ; Fin del ensamblado
```

Diseñe una subrutina en el ensamblador del MC68000 que compruebe si un año es o no bisiesto. Un año puede codificarse como un número entero sin signo en 16 bits. Como se sabe, un año es bisiesto si es divisible entre 4 pero no entre 100, excepto aquellos años que son divisibles entre 400; que sí son bisiestos.

La subrutina recibirá como parámetro el año en cuestión en la palabra menos significativa del registro D1; y devolverá en el registro D0 un valor igual a 1 si el año es bisiesto; e igual a 0 si no lo es.

Siga el proceso indicado a continuación:

1º.- Especificar los argumentos de la subrutina mencionados en el enunciado.

2º.- Realizar una descripción textual del algoritmo propuesto (10 líneas como máximo).

3º.- Describir los pasos del algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.

4º.- Codificar en el lenguaje ensamblador del MC68000 la subrutina pedida, comentando adecuadamente las instrucciones utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D1[15:00] recibe el año analizado.

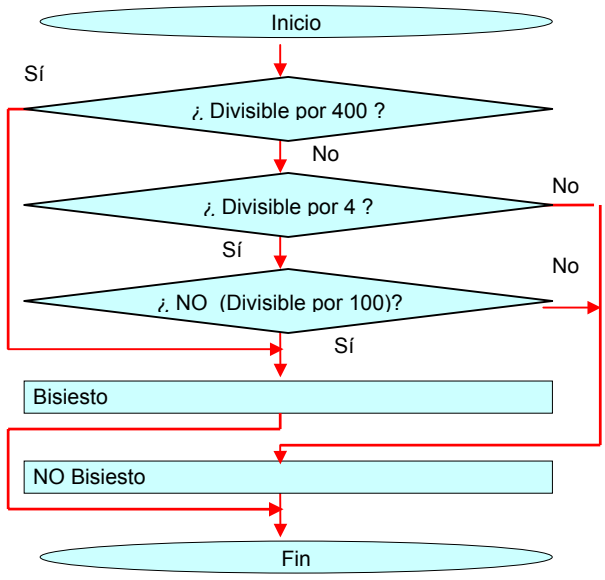
b) Resultado devuelto por la subrutina al programa principal:
En D0[31:00] devuelve el resultado: un 1 si es bisiesto; o un 0 si no lo es.

2º.- Descripción textual del algoritmo.

Se trata de implementar la función lógica: *(Divisible por 400) OR [(Divisible por 4) AND (NO (Divisible por 100))]*

3º.- Descripción detallada del algoritmo.

Diagrama de flujo:



Constante:
D1. Contiene el año.

Variable intermedia:
D2. Necesaria para realizar las divisiones sin perder el número original contenido en D1.

Variable resultado final:
D0. Contiene un 1 si el año es bisiesto; o un 0 en caso contrario.

4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen del 09-09-97
*
* Propósito: Crear una subrutina para comprobar si un año es bisiesto
* Autor:     José Garzía

*       ZONA DE CÓDIGO

Principal       org     $1000           ; Comienzo del programa principal
                jsr     Comprueba       ; Llamada a la subrutina
                move.b  #228,D7
                trap    #14             ; Fin del programa principal

Comprueba       move.w  SR,-(SP)        ; Preserva el SR
                move.l  D2,-(SP)        ; Preserva los registros que va a utilizar

*       ¿ Es divisible por 400 ?
        clr.l   D2
        move.w  D1,D2
        divu    #400,D2
        swap    D2
        cmpi    #0,D2
        beq     Bisiesto

*       ¿ Es divisible por 4 ?
        clr.l   D2
        move.w  D1,D2
        divu    #4,D2
        swap    D2
        cmpi    #0,D2
        bne     NoBisiesto

*       ¿ Es divisible por 100 ?
        clr.l   D2
        move.w  D1,D2
        divu    #100,D2
        swap    D2
        cmpi    #0,D2
        beq     NoBisiesto

Bisiesto        move.l  #SI,D0
                bra     Fin
NoBisiesto      move.l  #NO,D0

Fin             move.l (SP)+,D2         ; Restaura los registros utilizados
                rtr                     ; Restaura el SR y sale de la subrutina

*       ZONA DE DATOS

                org     $2000
NO              equ     0
SI              equ     1

                end                      ; Fin del ensamblado
```

1997. Septiembre (reserva).

Se dispone de un computador dotado de un microprocesador M68000, en cuya memoria está grabada una secuencia de datos. Los datos son números enteros positivos y negativos de 16 bits. Sin embargo, la secuencia se encuentra almacenada en un formato de codificación diferencial, de forma que el primer dato ocupa 16 bits y cada diferencia ocupa 8 bits. El primer dato ($i=1$) está en un apalabra de memoria etiquetada como PRIMERO, mientras que las sucesivas diferencias se encuentran almacenadas consecutivamente a partir de la posición de memoria etiquetada como DIFEREN. Tanto el primer dato como las diferencias están expresadas en complemento a dos.

Diseñe una subrutina en ensamblador del M68000 que, dado un número entero $i \geq 1$, devuelva el valor del dato i-ésimo de la secuencia almacenada. El valor i se pasará en la palabra menos significativa del registro D0. el dato i-ésimo se devolverá en D1.

Siga el procedimiento indicado a continuación:
1.- Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2.- Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3.- Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4.- Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Datos de entrada y resultados de salida de la subrutina.

a) Parámetros pasados a la subrutina por el programa principal:
En D0[15:00] recibe el índice del elemento.

b) Resultado devuelto por la subrutina al programa principal:
En D1[31:00] devuelve el valor del elemento.

2º.- Descripción textual del algoritmo.

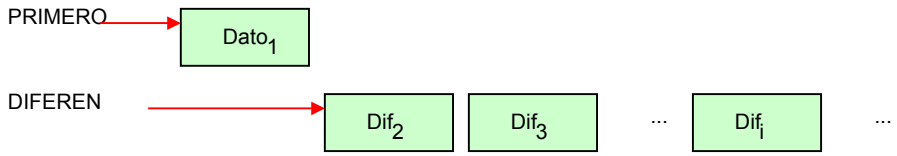
La codificación diferencial consiste en que en el lugar i-ésimo se coloca la diferencia del dato i-ésimo respecto del dato (i-1)-ésimo:

$$\text{dato } i\text{-ésimo} = \begin{cases} \text{dato inicial} & \text{si} \quad i = 1 \\ \text{dato } (i\text{-}1)\text{-ésimo} + \text{diferencia } i\text{-ésima} & \text{si} \quad i > 1 \end{cases} \quad \text{o bien} \quad \text{Dato}_i = \begin{cases} \text{Dato}_1 & \text{si} \quad i = 1 \\ \text{Dato}_{i-1} + \text{Dif}_i & \text{si} \quad i > 1 \end{cases}$$

Resolviendo la recurrencia:

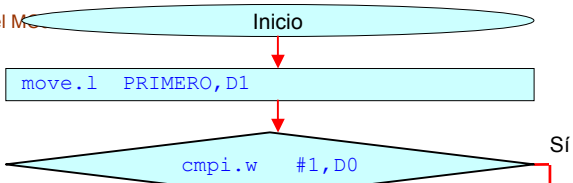
$$\text{Dato}_i = \text{Dato}_{i-1} + \text{Dif}_i = \text{Dato}_{i-2} + \text{Dif}_{i-1} + \text{Dif}_i = \text{Dato}_1 + \text{Dif}_2 + \ldots + \text{Dif}_{i-1} + \text{Dif}_i$$

Como vemos, la estrategia será acumular en un bucle desde el segundo hasta el i-ésimo; y sumar el primer elemento a lo acumulado.



Para procesar el elemento Dif_i, fijémonos que para Dif_2, se le suma cero al puntero DIFEREN. Por tanto, para utilizar el puntero DIFEREN, al índice del elemento (el pasado a la subrutina), debemos restar dos unidades.

3º.- Descripción detallada del algoritmo.



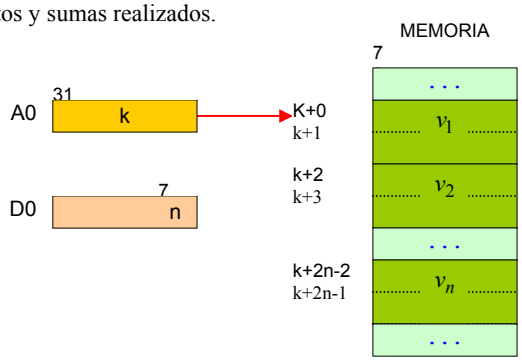
4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*       Escuela de Informática de la UNED
*       Asignatura: Estructura y Tecnología de Computadores I
*       Tutoría del Centro Asociado de Plasencia
*       Examen del 13-09-97
*
*
* Propósito:    Encontrar el valor de un dato almacenado en una lista en
*               la que se utiliza codificación diferencial


*               ZONA DE CÓDIGO
Principal       org     $1000           ; Comienzo del programa principal

                jsr     LeerNumero      ; Llamada a la subrutina

                move.b  #228,D7
                trap    #14             ; Fin del programa principal

LeerNumero      move.w  SR,-(SP)        ; Preserva el SR
                movem.l A0/D0,-(SP)     ; Preserva los registros que va a utilizar

                clr.l   D1
                move.w  PRIMERO,D1
                cmpi.l  #1,D0
                beq     Fin             ; Era el primero
                subq.l  #2,D0
                movea.l #DIFEREN,A0

InicioBucle     add.b   (A0)+,D1
                dbf     D0,InicioBucle

Fin             movem.l  (SP)+,A0/D0    ; Restaura los registros utilizados
                rtr                     ; Restaura el SR y sale de la subrutina

*               ZONA DE DATOS
                org     $2000
PRIMERO         ds.w    1
                org     $2002
DIFEREN         ds.b    20              ; Por poner alguna cantidad

                end                     ; Fin del ensamblado
```

Dado un vector v de n componentes $v = (v_1, v_2, ...v_n)$, la norma de v se define como $\|v\| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$. Se dispone de un computador basado en un microprocesador M68000. En él, el vector se representa almacenando ordenadamente en memoria sus componentes en palabras consecutivas, tal como muestra la figura (cada componente ocupa una palabra de 16 bits).

Diseñe una subrutina en ensamblador del M68000 que, dado un vector v almacenado en memoria según el procedimiento antedicho, calcule el cuadrado de su norma. El número de componentes del vector es a priori desconocido, y lo recibe la subrutina como parámetro de entrada en el byte menos significativo del registro D0. Además, la subrutina recibirá la dirección de comienzo del vector en el registro A0, y devolverá la norma en el registro D1. Las componentes del vector son números enteros de 16 bits almacenados en complemento a 2. La norma será un número entero de 32 bits. Suponer que en ningún caso se producirá desbordamiento en los productos y sumas realizados.



Siga el procedimiento indicado a continuación:

1.- Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2.- Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3.- Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4.- Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Estructuras de datos iniciales y argumentos de la subrutina.

a) Existe una formación unidimensional; el vector de elementos dado.

b) Parámetros pasados a la subrutina por el programa principal:
En D0[07:00] recibe el número de componentes del vector.
En A0[31:00] recibe la dirección de comienzo del vector.

c) Resultado devuelto por la subrutina al programa principal:
En D1[31:00] devuelve el valor de la norma del vector.

2º.- Descripción textual del algoritmo.

Recorreremos el vector en un bucle. En cada pasada leemos un elemento, lo llevamos a un registro auxiliar (D2), lo multiplicamos por sí mismo en este registro; y lo sumamos a un acumulador aditivo (D1).

3º.- Descripción detallada del algoritmo.

a) Constantes: Ninguna.

b) Variables:

D0 se utiliza como contador. El bucle lo implementamos con el salto condicional DBF, es decir, en el última pasada, el índice debe valer cero. Por ello decrementamos el índice en una unidad antes de entrar en el bucle.

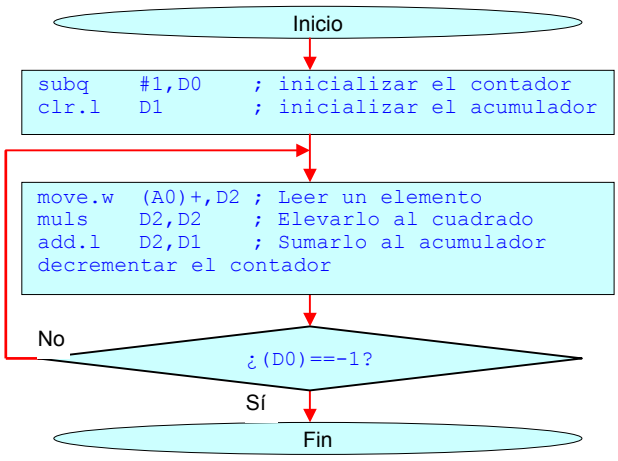
D1 se utiliza como acumulador aditivo. Por ello se inicializa con el elemento neutro de la suma.

D2 es un registro auxiliar donde se lleva cada elemento y se eleva al cuadrado.

c) Punteros:

A0 apunta a la dirección de comienzo del vector. La forma más natural de recorrer el vector de comienzo a fin es utilizar el modo de direccionamiento indexado con posincremento.

d) Pasos del algoritmo



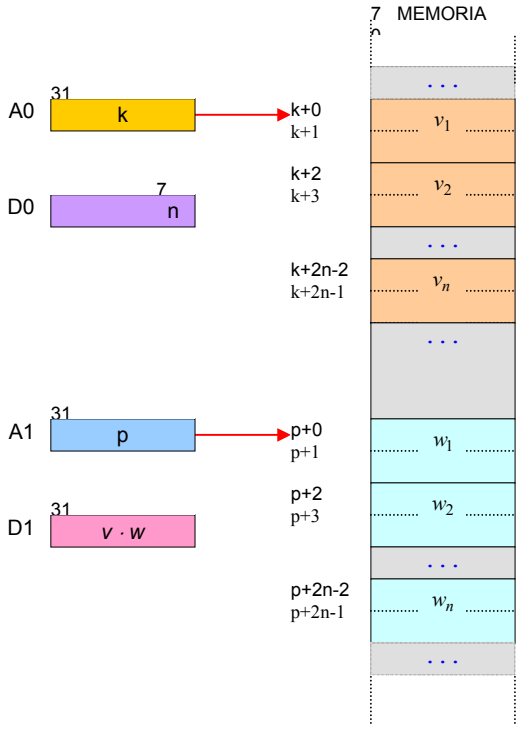
4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*       Escuela de Informática de la UNED
*       Asignatura: Estructura y Tecnología de Computadores I
*       Tutoría del Centro Asociado de Plasencia
*       Examen del 19-09-98
*
* Propósito:    Calcular el cuadrado de la norma de un vector


*               ZONA DE CàDIGO
Principal       org     $1000               ; Comienzo del programa principal
                movea.l #Vector,A0          ; Dirección del vector
                move.b  #5,D0               ; Tamaño por omisión
                jsr     CalcularNorma       ; Llamada a la subrutina

                move.b  #228,D7
                trap    #14                 ; Fin del programa principal

CalcularNorma   move.w  SR,-(SP)            ; Preserva el SR
                movem.l A0/D0/D2,-(SP)      ; D1 no se preserva, pues devuelve el resultado

                andi.l  #$00FF,D0           ; En D0 sólo es relevante el byte menos significativo
                subq    #1,D0               ; Inicializa el contador
                clr.l   D1                  ; Inicializa el acumulador

InicioBucle     move.w (A0)+,D2             ; Lee un elemento
                muls    D2,D2               ; Lo eleva al cuadrado
                add.l   D2,D1               ; Lo suma al acumulador
                dbf     D0,InicioBucle

Fin             movem.l  (SP)+,A0/D0/D2     ; Restaura los registros utilizados
                rtr                         ; Restaura el SR y sale de la subrutina

*               ZONA DE DATOS
                org     $2000
Vector          dc.w    $FFFE,$FFFF,0,1,2   ; Vector por omisión

                end                         ; Fin del ensamblado
```

Dados dos vectores *v* y *w* de *n* componentes $v=(v_1,v_2,...v_n)$ y $w=(w_1,w_2,...w_n)$, el producto escalar de *v* por *w* se define como $v \cdot w = v_1 \cdot w_1 + v_2 \cdot w_2 + ... + v_n \cdot w_n$. Se dispone de un computador basado en un microprocesador M68000. En él, cada vector se representa almacenando ordenadamente en memoria sus componentes en palabras consecutivas, tal como muestra la figura (cada componente ocupa una palabra de 16 bits).

Diseñe una subrutina en ensamblador del M68000 que, dados dos vectores *v* y w almacenados en memoria según el procedimiento antedicho, calcule su producto escalar. El número de componentes de los vectores es a priori desconocido, y lo recibe la subrutina como parámetro de entrada en el byte menos significativo del registro D0. Además, la subrutina recibirá la dirección de comienzo del vector *v* en el registro A0, la dirección de comienzo del vector *w* en el registro A1, y devolverá el producto escalar en el registro D1. Las componentes de los vectores son números enteros de 16 bits almacenados en complemento a 2. El producto escalar será un número entero de 32 bits. Suponer que en ningún caso se producirá desbordamiento en los productos y sumas realizados.



Siga el procedimiento indicado a continuación:

1.- Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2.- Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3.- Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4.- Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Solución:

1º.- Estructuras de datos iniciales y argumentos de la subrutina.

a) Existen dos formaciones unidimensionales; los vectores de elementos dados.

b) Parámetros pasados a la subrutina por el programa principal:
En D0[07:00] recibe el número de componentes del vector.
En A0[31:00] recibe la dirección de comienzo del vector *v*.
En A1[31:00] recibe la dirección de comienzo del vector *w*.

c) Resultado devuelto por la subrutina al programa principal:
En D1[31:00] devuelve el valor del producto escalar de ambos vectores.

2º.- Descripción textual del algoritmo.

Recorreremos los vectores en un bucle. En cada pasada leemos un elemento de cada vector, los llevamos a sendos registros auxiliares (D2 y D3), lo multiplicamos en el registro D3; y lo sumamos a un acumulador aditivo (D1).

3º.- Descripción detallada del algoritmo.

a) Constantes: Ninguna.

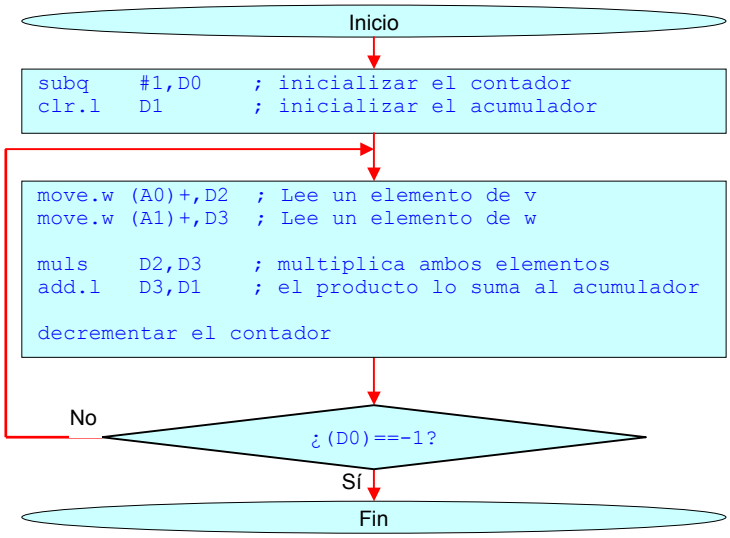
b) Variables:
D0 se utiliza como contador. El bucle lo implementamos con el salto condicional DBF, es decir, en el última pasada, el índice debe valer cero. Por ello decrementamos el índice en una unidad antes de entrar en el bucle.

D1 se utiliza como acumulador aditivo. Por ello se inicializa con el elemento neutro de la suma.
D2 y D3 son registros auxiliares donde se lleva cada par de elementos y se realiza el producto en D3.

c) Punteros:
A0 apunta a la dirección de comienzo del vector *v*.
A1 apunta a la dirección de comienzo del vector *w*.

La forma más natural de recorrer los vectores de comienzo a fin es utilizar el modo de direccionamiento indexado con posincremento.

d) Pasos del algoritmo



4º.- Codificación de la subrutina en el lenguaje ensamblador del MC68000:

Las instrucciones están explicadas en los comentarios; y los pasos del algoritmo los explican por sí mismos los nombres de las etiquetas.

```
*
*    Escuela de Informática de la UNED
*    Asignatura: Estructura y Tecnología de Computadores I
*    Tutoría del Centro Asociado de Plasencia
*    Examen de 1999-09-08 (reserva)
*
*
* Propósito:    Calcular el producto escalar de dos vectores

*                ZONA DE CÓDIGO
Principal        org     $1000              ; Comienzo del programa principal
                 movea.l #Vector_v,A0       ; Dirección del vector v
                 movea.l #Vector_w,A1       ; Dirección del vector w
                 move.b  #5,D0              ; Tamaño por omisión
                 jsr     ProductoEscalar    ; Llamada a la subrutina

                 move.b  #228,D7
                 trap    #14                ; Fin del programa principal

ProductoEscalar  move.w  SR,-(SP)           ; Preserva el SR
                 movem.l A0/D0/D2-D3,-(SP)  ; D1 no se preserva, pues devuelve el resultado

                 andi.l  #$00FF,D0          ; En D0 sólo es relevante el byte menos significativo
                 subq    #1,D0              ; Inicializa el contador
                 clr.l   D1                 ; Inicializa el acumulador

InicioBucle      move.w (A0)+,D2            ; Lee un elemento de v
                 move.w (A1)+,D3            ; Lee un elemento de w
                 muls    D2,D3              ; Multiplica ambos elementos
                 add.l   D3,D1              ; El producto lo suma al acumulador
                 dbf     D0,InicioBucle

Fin              movem.l (SP)+,A0/D0/D2-D3  ; Restaura los registros utilizados
                 rtr                        ; Restaura el SR y sale de la subrutina

*                ZONA DE DATOS
                 org     $2000
Vector_v         dc.w    $FFFE,$FFFF,0,1,2  ; Vector v por omisión
Vector_w         dc.w    1,1,0,1,3          ; Vector w por omisión

                 end                        ; Fin del ensamblado
```