Sistemas Digitales Avanzados y Microprocesadores: prácticas de simulación del microprocesador Motorola MC68000*

Miguel Moro Vallina**

Preliminares

El modo como están planteadas las prácticas se presta, en nuestra opinión, a emplear la metodología de la *programación estructurada*;[1] aun cuando el código ensamblador sea intrínsecamente no-estructurado, hemos tratado de mantenernos fieles a dicha metodología. Primero, considerando los algoritmos y subrutinas en primera instancia como "cajas negras" en las que, antes que los detalles operacionales, interesa definir rigurosamente los argumentos de entrada y salida y las pre y post-condiciones. Y segundo, codificando en "pseudolenguaje"[2] los algoritmos empleados, utilizando para ellos las estructuras iterativas propias de la programación estructurada y relegando a la fase final de codificación en ensamblador la traducción de dichas estructuras iterativas por los saltos condicionales correspondientes. No obstante, como *estadio intermedio* entre el pseudolenguaje y el código ensamblador —habida cuenta de las diferencias entre ambos "niveles" de programación, no sólo en cuanto a las estructuras iterativas sino también en los modos de direccionamiento o la forma de evaluar las expresiones aritméticas—, nos ha parecido de utilidad realizar una *descripción*

*Esta memoria corresponde a las prácticas obligatorias de simulación del ensamblador del MC68000 de la asignatura de Sistemas Digitales Avanzados y Microprocesadores, asignatura del Plan 2001 carrera de Ingeniería Industrial de la Universidad Nacional de Educación a Distancia. La composición de este documento se ha realizado mediante LaTeX.

**Correo: `narodnaia@gmail.com`. Web: `http://narodnaia.googlepages.com`

[1] Véase, por ejemplo, J. Castro, F. Cucker, X. Messeguer, A. Rubio, Ll. Solano y B. Valles (1993): *Curso de programación*. Madrid: McGraw–Hill.

[2] En realidad, emplearemos el lenguaje `WEB` como "pseudolenguaje" particular, puesto que proporciona, a nuestro juicio, un modo claro y riguroso de definir los algoritmos con un cierto nivel de abstracción. `WEB` es un sistema de *documentación estructurada* creado por Donald E. Knuth (véanse ejemplos prácticos en *TEX: The Program* y en *METAFONT: The Program*, volúmenes B y D de *Computers and Typesetting*, *op.cit.*). `WEB` se asemenja bastante al lenguaje Pascal —lo que lo hace ideal para ser fácimlmente comprensible por las personas—, con una serie de combios simbólicos que faciiltan su lectura y, especialmente, con una *organización* de las diversas partes que componen el programa minuciosamente orientada a facilitar su comprensión.

por pasos pormenorizada de cada algoritmo propuesto. Dichos "pasos" se reflejan posteriormente como comentarios al código ensamblador, lo cual facilita la lectura y eventual mantenimiento del programa.

Un programa de computadora, por complejo que sea, puede definirse, parafraseando a DONALD KNUTH, como una suerte de *tela de araña* compuesta de elementos simples; para comprender adecuadamente un programa es preciso comprender bien la naturaleza y características de dichos elementos y, más aún, las *relaciones* que los ligan unos a otros.[3] En ese contexto, una *documentación* adecuada, completa y precisa, que describa adecuadamente las estructuras de datos y los algoritmos empleados, deviene un elemento clave para construir programas eficientes e imprescindible de cada al mantenimiento, modificaciones y mejoras introducidas en los mismos. Más aún en un contexto en el que la *producción de software* se ha convertido en una tarea cada vez más colectiva y, por decir así, "taylorizada" en la que, a medida que la construcción de programas de creciente complejidad se disgrega en sus elementos simples, cobran una importancia estratégica las pautas y normas de comunicación e intercambio de información entre los diversos programadores, quienes, por añadidura, producen elementos cada vez más sometidos a estándares rigurosos.[4]

En ese contexto, y aun teniendo presente que los problemas propuestos en estas prácticas son pequeños y relativamente sencillos, nos hemos esforzado para tratar de que la documentación de los programas elaborados sea los maý clara y exhaustiva posible, tratando de presentar la descripción textual, el flujograma, la codificación en pseudolenguaje y la descripción por pasos de los algoritmos propuestos, como elementos complementarios y que, en cierto modo, forman una secuencia de precisión creciente que permite transitar "de los problemas a los programas".[5]

Pero, para preparar una documentación legible y bien estructurada, no es una cuestión baladí el disponer de una herramienta adecuada con la que elaborarla. En ese sentido, se ha optado por emplear el sistema de composición tipográfica LaTeXque provee, a nuestro modo de ver, la forma más potente y con mejores resultados para componer textos de índole científico-técnica.[6] Los flujogramas se han compuesto en PostScript con la ayuda del paquete PSTricks.[7]

---

[3] DONALD E. KNUTH: "Literate Programming", en *The Computer Journal*, vol. **27/2**, pp. 97–111.

[4] Es muy ilustrativo al respecto releer alguno de los textos clasivos de TAYLOR a la luz de la organización productiva de la industria del *software* actual. Véase, por ejemplo, F.W. TAYLOR (1903): *Shop Management*. Edición facsímil. Londres, 1993: Routlegde.

[5] Véase A. AHO, J. HOPCROFT y J. ULLMAN (1983): *Estructuras de datos y algoritmos*. México, 1988: Addison–Wesley Iberoamericana.

[6] Puede consultarse al respecto la monumental obra de D.E. KNUTH: *Computers and Typesetting*, vols. A–E. Reading, Massachussetts: Addison–Wesley, así como la obra de B. CASCALES SALINAS *et. al.*: *LaTeX: una imprenta en sus manos*. Madrid: Aula Documental de Investigación.

[7] Puede consultarse al respecto M. GOOSSENS, F. MITTELBACH, S. RAHTZ, D. ROEGEL y H. VOSS (1998): *The LaTeX Graphics Companion: Illustrating documents with TeX and PostScript*. Reagind, Massachussetts: Addison–Wesley.

Práctica I

Enunciado de la práctica

Se desea diseñar una subrutina 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 debería ser `[0E 0D 0C 0B 0A]`. La subrutina se denominará `INVERTIR` y recibirá la dirección de comienzo en el registro `A0` y el tamaño del mismo en el registro `D0`.

Argumentos de entrada y salida

La subrutina `INVERTIR` tendrá como argumentos de entrada el vector inicial y su tamaño (número de elementos de un byte cada uno). El argumento de salida será el vector invertido. Los vectores original e invertido se componen de un número indeterminado de elementos de un byte cada uno, tal como viene expuesto en el enunciado. Acotaremos el tamaño del vector reservando para su tamaño 16 bits de memoria (una palabra), longitud que nos permitiría operar con vectores de hasta 65536 elementos.

Descripción textual del algoritmo

El algoritmo propuesto realizará una suerte de *rotación lógica hacia la derecha* del vector inicial, desplazando hacia posiciones crecientes de memoria los elementos del vector. El dato que se desborda por la derecha se almacenará en la *primera* posición por la izquierda. A continuación se realizará la misma operación, pero desplazando sólo los datos situados *a partir* de dicha posición, y almacenando el dato desbordado en la *segunda* posición por la izquierda. El bucle continuará realizando estos desplazamientos hasta que el elemento (a_0) inicialmente situado en la posición inicial se encuentre en la última.

Descripción pormemorizada del algoritmo y datos intermedios empleados

Como puede comprobarse, el algoritmo se compondrá de n iteraciones, tantas como elementos tenga el vector. Deberá existir una variable —llamémosla, por ejemplo, `CONTADOR`— que indique el número de iteración en curso y que se incremente en cada una de ellas; la variable `CONTADOR` indicará además el punto en el cual deben introducirse los datos que se desborden por la derecha. Cada una de estas iteraciones, a su vez, estará compuesta por los pasos siguientes:

1. Almacenar el dato situado más a la derecha en una variable que denominaremos `AUX`.

2. Mover todos los elementos del vector situados en la izquierda de la posición indicada por la variable `CONTADOR` en una posición, situando el dato de la posición $n-2$ en la $n-1$, el de la $n-3$ en la $n-2$, y así sucesivamente, hasta la posición indicada por `CONTADOR`. Para ello será preciso realizar otro bucle que deberá hacer uso de una variable adicional denominada, verbigracia, `CONTADOR_BUCLE`.

3. Introducir el dato de la variable `AUX` en la posición de memoria indicada por la variable `CONTADOR`.

Codificación en `WEB`

A partir de la descripción pormenorizada del algoritmo, codificamos la subrutina `INVERTIR` como *procedimiento*. Supondremos que sus argumentos (que denominaremos *vector* y *tamaño*) se le pasan al procedimiento como *variables globales*.

```
procedure invertir;
  var aux: byte;   contador: integer;
  begin
    contador ← 0;
    repeat  begin
      aux ← vector[tamaño];
      for contador_bucle ← 0to contador do  begin
        vector[tamaño − contador_bucle] ← vector[tamaño − contador_bucle − 1];
        contador_bucle ← contador_bucle + 1;
        end;
      vector[contador] ← aux;
      contador ← contador + 1;
    until contador = tamaño;
  end;
```

Descripción por pasos del algoritmo

Una vez efectuada la codificación del algoritmo en "pseudolenguaje", realizaremos ahora una "traducción" del mismo a una código más cercano al lenguaje ensamblador. La *descripción por pasos*, en efecto, es ya mucho más próxima a la programación del MC6800, en varios sentidos: en primer lugar, sustituye las estructuras iterativas por saltos condicionales; en segundo lugar, realiza un uso intensivo de los registros (de datos y de memoria) en lugar de hacer referencia a *variables* y *tipos de datos abstractos*; por último, implementa, para cada estructura de datos utilizada, los modos de direccionamiento que se consideran más adecuados para su realización.

Los pasos del algoritmo propuesto serían, en definitiva, los siguientes. Se supone, tal como viene recogido en el enunciado, que la dirección inicial del

vector está almacenada en A0 y su tamaño en D0.[8]

Paso	Descripción	Pseudocód.
1	Inicializar CONTADOR.	D1 ← 0
2	Restar una unidad a la variable TAMAÑO, para poder operar adecuadamente con el modo de direccionamiento elegido.	D0 ← D0 − 1
3	Hacer que A1 apunte al *final* del vector de datos en memoria. Emplearemos direccionamiento directo en registro de direcciones.	A1 ← A1 + D0
4	Mover el contenido de la última posición del vector a la variable AUX. Para ello elmplearemos direccionamiento indirecto a registro.	AUX ← (A1)
5	Inicializar CONTADOR_2.	D4 ← 1
6	Hacer que A1 apunte a la posición de V1 TAMAÑO − CONTADOR_2.	A1 ← A0 + D0 − D4
7	Mover el elemento apuntado una posición a la derecha. Emplearemos direccionamiento indirecto a registro con post-autoincremento.	D5 ← (A1)+; (A1) ← D5
8	Incrementar CONTADOR_2.	D4 ← D4 − 1
9	Si CONTADOR_2 ≤ TAMAÑO − CONTADOR, saltar al paso *6*.	To *6* if D4 ≤ D2
10	Hacer que A1 apunte a V(CONTADOR).	A1 ← A0 + D1
11	Mover AUX a V(CONTADOR). Emplearemos nuevamente direccionamiento indirecto a registro.	(A1) ← D3
12	Incrementar CONTADOR.	D1 ← D1 + 1
13	Si CONTADOR ≤ TAMAÑO − 1, saltar al paso *3*.	To *3* if D1 ≤ D2
14	Ya se han realizado (TAMAÑO − 1) iteraciones y el vector se ha invertido: fin del programa.	END

El flujograma de la rutina INVERTIR se muestra en la figura 1.

Codificación en ensamblador

Se ha codificado un programa en ensamblador que, tal como viene expresado en el enunciado:

1. Dispone de una zona de datos en memoria para almacenar el vector, zona que se inicializa con los valores del ejemplo del enunciado ([0A 0B 0C 0D

[8] En la presentación de la descripción por pasos y en la forma del pseudocódigo correspondiente a cada uno de ellos, hemos tomado algunos elementos de la descripción del juego de instrucciones MIX, creado por Donald Knuth para un hipotético computador de 32 bits. Véase *The Art of Computer Programming*, Vol 1 (*Fundamental Algorithms*). Tercera edición. Reading, Massachussetts, 1997: Addison–Wesley.

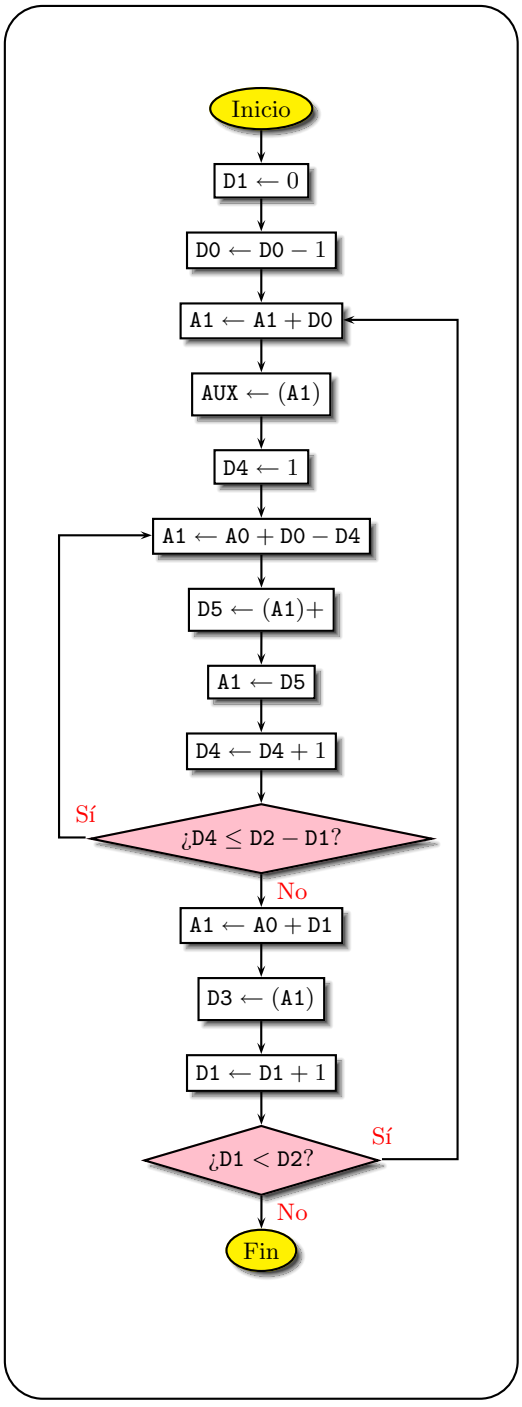

Figura 1: Diagrama de flujo de la subrutina `INVERTIR`.

0E]); dicha zona de memoria se etiqueta como VECTOR, y se ubica *detrás* del código del programa.

2. Carga la etiqueta VECTOR en el registro D0 y el valor "5" en D0.

3. Efectúa una llamada a la subrutina y seguidamente —una vez que el vector ubicado en la posición de memoria apuntada por A1 se halla totalmente invertido— termina.

El código en ensamblador propuesto se lista a continuación:

```
*-----------------------------------------------------------
* Program    :Programa invertir
* Written by :Miguel Moro Vallina
* Date       :23 de agosto de 2007
* Description:Dispone los datos adecuados y llama a la
*       subrutina INVERTIR
*-----------------------------------------------------------
START   ORG     $1000
        MOVEA.W #VECTOR,A0
        MOVEQ   #5,D0
        BSR     INVERT
        STOP    #$2000

*-----------------------------------------------------------
* Zona de datos
*-----------------------------------------------------------
DATOS   ORG     $2000
VECTOR  DS.B    5
        ORG     $2000
        DC.B    $0A,$0B,$0C,$0D,$0E

*-----------------------------------------------------------
* Zona de subrutinas
*-----------------------------------------------------------
SUBS    ORG     $3000
INVERT  MOVE.W  #0,D1           Inicializa CONTADOR
        MOVE.W  #0,D1           Paso 1. Inicializa CONTADOR
        SUBI    #1,D0           Paso 2.
BUCLE   MOVE.W  A0,A1           Paso 3. A1 apunta
        ADDA.W  D0,A1             al final del vector
*                               Paso 4.
        MOVE.B  (A1),D3           D3 contiene la variable AUX
*                               Paso 5.
        MOVEQ   #1,D4             Inicializa CONTADOR_BUCLE2
BUCLE2  MOVE.W  A0,A1           Paso 6. A1 apunta
        ADDA.W  D0,A1             a V(TAM-CONT_BUCLE2)
        SUBA.W  D4,A1
        MOVE.W  D0,D2             Copia la long. del vector en D2
        MOVE.B  (A1)+,D5        Paso 7.
        MOVE.B  D5,(A1)           Mueve un elemento a la derecha
*                               Paso 8.
        ADDI    #1,D4             Incrementa CONTADOR_BUCLE2
*                               Paso 9.
        SUB.W   D1,D2             Calcula TAM-CONTADOR
        CMP.W   D4,D2             Si D4 menor o igual que D2
        BGE     BUCLE2            repite BUCLE2 (Paso 6)
        MOVE.W  A0,A1           Paso 10.
        ADDA.W  D1,A1             A1 apunta a V(CONTADOR)
        MOVE.B  D3,(A1)         Paso 11.
```

```
*                               Mueve AUX a V(CONTADOR)
        ADDI    #1,D1           Paso 12. Incrementa CONTADOR
        MOVE.W  D0,D2           Paso 13.
        CMP.W   D1,D2             Si D1 menor que D2
        BGT     BUCLE             repite BUCLE
        RTS                     Paso 14. Retorno al programa
*                                 principal


        MOVE.B  #9,D0           Fin del programa
        TRAP    #15             Halt Simulator

        END     START
```

Traza de ejecución

En el fichero `programa_invertir_reg.txt` se muestra la traza de ejecución del anterior programa, generado con el simulador del *Easy68K*. La traza incluye el contenido de todos los registros de datos y memoria, así como el del puntero de pila, registros de estado y contador de programa, y de las posiciones de memoria involucradas (concretamente, las direcciones `$2000` a `$2005`, aquellas en las que está almacenado el vector que debe invertirse. La figura 2 muestra la captura de pantalla de la ventana de configuración de la traza de ejecución.

Simulación con `[DD CC BB AA]`

Se efectúa ahora una nueva simulación con la subrutina, esta vez con el vector `[DD CC BB AA]`. En este caso, introducimos directamente en memoria el vector pedido —ayudándonos de las órdenes disponibles en el simulador—; en el registro `A0` introducimos la posición de memoria en la que se ubica el vector (`$2000`) y en el `D0` su tamaño (`4`). La traza de la nueva simulación efectuada con estos datos se lista en el fichero `invertir_reg.txt`.

Práctica II

Enunciado de la práctica

Diséñese una subrutina para la conversión de formato binario a decimal de números sin signo de 16 bits. La subrutina recibirá el número binario en la palabra menossignificativa del registro `D0` y deberá generar una cadena de códigos ASCII (en decimal) en memoria de nombre `CADNUM`, correspondiente a la secuencia de cifras decimales que representa el número pedido.[9] Los sucesivos caracteres de la representación en ASCII se almacenarán en memoria de forma

[9] Por ejemplo, si se recibe el número `3F7B` en hexadecimal (= `16251` en decimal), la cadena `CADNUM` que se ha de generar es `[49,55,54,50,49,0]`, habida cuenta de que el código ASCII de cada cifra decimal se obtiene sin más que incrementarla en 48 unidades.

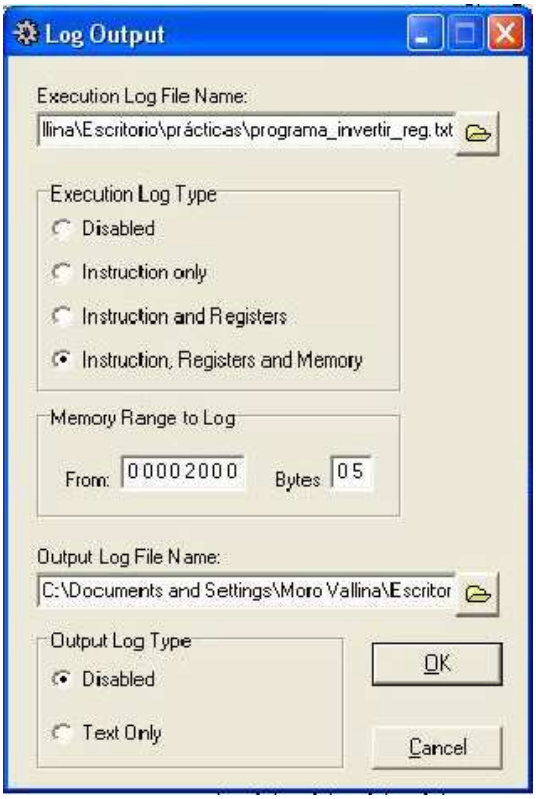

Figura 2: Captura de pantalla de la configuración de la traza de ejecución.

consecutiva y ordenada de mayor a menor peso. Detrás del último carácter de la cadena se añadirá un carácter nulo.

Descripción textual del algoritmo

El algoritmo efectuará la división entera entre 10 del número recibido. El resto (cifra de *menor peso* del número decimal correspondiente) se almacenará en memoria. El cociente obtenido se volverá a dividir entre 10, almacenándose el nuevo resto en la posición de memoria inmediatamente superior y así sucesivamente, hasta que el cociente obtenido sea nulo. El vector de datos situado en la memoria —en cuyo inicio se habrá situado un carácter nulo— se invertirá a continuación (invocando a la subrutina `INVERTIR` de la práctica anterior), obteniendo de este modo el resultado pedido.

Descripción pormenorizada del algoritmo y constantes y variables empleadas

La primera operación que deberá efectuar la subrutina es situar en memoria (en la zona etiquetada como `CADNUM`) un carácter nulo. Realizado lo cual, el algoritmo consistirá básicamente en un bucle que realizará las siguientes operaciones:

1. Tomar el dato inicial (llamaremos `DIVIDENDO` a la variable en la que se encuentra) y realizar su división entre 10, almacenando el resto en la variable `MODULO` y el resultado en `COCIENTE`. A la variable módulo se le sumará el número 48 y se almacenará el resultado en la posición consecutiva del vector `CADNUM`.

2. Comprobar si `COCIENTE` es igual a cero y, en caso contrario, cargar su contenido en `DIVIDENDO` y repetir la operación.

3. En el caso de que `COCIENTE` sea nulo, el bucle finalizará y se llamará a la subrutina `INVERTIR` para que invierta el vector situado e la posición de memoria `CADNUM` y lo ordene, tal como pide el enunciado, con los dígitos de *menor* peso en posiciones *crecientes* de memoria.

Deberemos además disponer de una variable `TAM` que "cuente", a medida que se van cargando datos en el vector `CADNUM`, el *tamaño* de dicho vector. La variable `TAM` tendrá una utilidad doble: por una parte, indicará en qué posición de memoria se debe introducir el dato siguiente; por otra, una vez finalizado el bucle, se pasará como argumento de la subrutina `INVERTIR`.

Veamos, por último, los *tamaños* de los datos de entrada y salida. Para el vector de códigos ASCII es suficiente, como es obvio, reservar un byte por elemento; además, ello debe ser así, por coherencia con el tamaño de los datos de entrada de la subrutina `INVERTIR`. Por la misma razón, para la variable `TAM` reservaremos una palabra de memoria, si bien dicho tamaño —que permitiría

operar con un número de hasta 65536 cifras— excede con creces las longitudes que habrán de tener los datos y la propia capacidad de cálculo de la unidad aritmético-lógica del microprocesador.

En el juego de instrucciones de la familia Motorola 68000 existen dos instrucciones que efectúan la división entera de números codificados en binario: `DIVS` y `DIVU`, empleadas respectivamente para operar con números con signo (*signed*) o sin él (*unsigned*). Ambas efectúan la división de un dato de 32 bits (en destino) entre otro de 16 bits (en fuente), generando dos resultados de 16 bits: el cociente y el resto. El cociente se guarda en los 16 bits menos significativos y el resto en los 16 bits más significativos del registro destino.

Codificación en `WEB`

De la descripción del algoritmo surje fácilmente su codificación en `WEB`. Asumiremos, como hicimos en la anterior práctica, que el argumento de dicho procedimiento (*numero_binario*) se pasaría como variable global. Por coherencia con el criterio de la práctica anterior, en la llamada al procedimiento *invertir*, el paso de los argumentos se hará mediante las variables globales *tamaño* y *vector*.

```
procedure convertir;
  var dividendo, cociente, resto, tam: integer;
  begin
    tam ← 0;
    tam ← tam + 1;
    dividendo ← numero_binario;
    repeat  begin
      cociente ← dividendo div 10;
      resto ← dividendo mod 10;
      r ← r + 48;
      cadnum[tam] ← r;
      tam ← tam + 1
    until dividendo = 0;
    tamaño ← tam;
    vector ← cadnum;
    invertir;
  end;
```

Descripción por pasos del algoritmo propuesto

Los pasos del algoritmo que se propone para resolver el problema planteado serían los siguientes. Suponemos, tal como viene propuesto en el enunciado, que el vector de códigos ASCII generado (que vendrá expresado en base hexadecimal) se sitúa en la zona de memoria etiquetada como `CADNUM`[10] y que el argumento de

[10] Puesto que el número decimal de partida ocupa una palabra de memoria, habremos de operar con números de cinco cifras a lo sumo. Teniendo en cuenta que en el vector de códigos ASCII deberemos situar un cero en la última posición, reservaremos para `CADNUM` *seis* octetos de memoria.

entrada al algoritmo (el número con cuyas cifras decimales se debe componer el vector ASCII, NUM_BIN) se introducirá como una constante en la zona de datos del programa, representándolo, por comodidad, en base hexadecimal.

Paso	Descripción	Pseudocód.
1	Inicializar la variable TAM, que "contará" los elementos del vector ASCII, dato necesario para invocar posteriormente a la subrutina INVERTIR.	D0 $\leftarrow 0$
2	Hacer que A0 apunte a la zona de memoria etiquetada como CADNUM.	A0 $\leftarrow$ #CADNUM
3	Situar un "0" en la posición inicial del vector CADNUM e incrementar la posición a la que apunta A0 en una unidad. Emplearemos direccionamiento indirecto a registro con post-autoincremento.	(A0)+ $\leftarrow 0$
4	Cargar el dato de entrada NUM_BIN en D1.	D1 $\leftarrow$ #NUMBIN
5	Efectuar la división entera de D1 entre 10; el cociente se almacenará en la palabra menos significativa de D1 y el resto en la palabra más significativa.	D1 $\leftarrow$ D1 mod 10
6	Intercambiar las dos palabras de D1 para poder operar con el resto de la operación anterior.	SWAP D1
7	Situar en la zona de memoria apuntada por A1 el número $r + 48$. Para ello emplearemos direccionamiento indirecto a registro con post-autoincremento.	(A1)+ $\leftarrow$ D1 + 48
8	Incrementar la variable TAM.	D0 $\leftarrow$ D0 + 1
9	Poner a cero los 16 bits menos significativos de D1, para evitar resultados espurios en la siguiente división.	D1 $\leftarrow 0_W$
10	Intercambiar las palabras de D1 para situar el cociente en la palabra menos significativa.	SWAP D1
11	Si COCIENTE $\neq 0$, saltar al paso *5*	To *5* if D1 $\neq 0$
12	Situar en el registro A0 la dirección de CADNUM, argumento de la subrutina INVERTIR.	A0 $\leftarrow$ #CADNUM
13	Incrementar la variable TAM.	D0 $\leftarrow$ D0 + 1
14	Invocar a la subrutina INVERTIR.	To INVERT
15	Fin del programa.	END

El flujograma del programa CONVERTIR se muestra en la figura 3

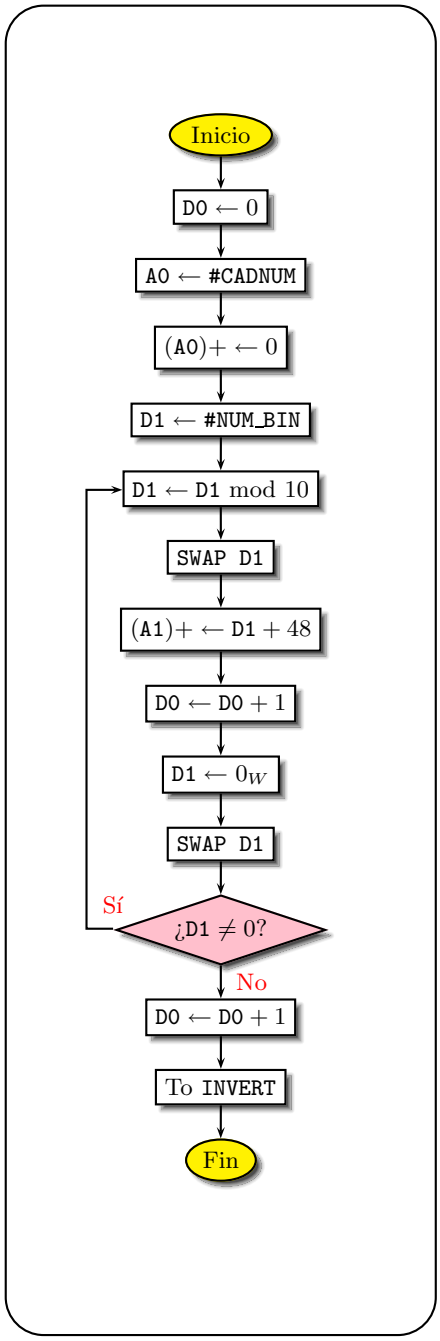

Figura 3: Diagrama de flujo del programa `CONVERTIR`.

Codificación en ensamblador

Se ha codificado en el ensamblador del MC68000 la subrutina pedida. El código propuesto se lista a continuación:

```
*-----------------------------------------------------------
* Program    :CONVERTIR
* Written by :Miguel Moro Vallina
* Date       :18 de agosto de 2007
* Description:Convierte binario a vector de códigos ASCII
*       que representan los dígitos de cada número en decimal
*       El vector se situará en la dirección indicada por
*       CADNUM
*-----------------------------------------------------------
CONVERT ORG     $1000
        MOVEQ   #0,D0           Paso 1. Inicializa TAM
        MOVEA.W #CADNUM,A0      Paso 2.
        MOVE.B  #0,(A0)+        Paso 3. Sitúa un 0 en CADNUM e
*                                 incrementa el puntero en una
*                                 posición
        MOVE.W  #NUM_BIN,D1     Paso 4.
BUCLE   DIVU    #10,D1          Paso 5.
        SWAP    D1              Paso 6. Intercambia las mitades
*                                 de D1
        ADDI    #48,D1          Paso 7. Suma r+48
        MOVE.B  D1,(A0)+          Mueve el código a la posición
*                                 correspondiente del vector
*                                 y postautoincrementa la pos.
        ADDI    #1,D0           Paso 8. Incrementa TAM
        MOVE.W  #0,D1           Paso 9. Pone a cero los 16 bits
*                                 menos significativos de D1
        SWAP    D1              Paso 10. Sitúa el cociente en la
*                                 palabra menos significativa
        CMP.B   #0,D1           Paso 11. Si el COCIENTE es distinto
        BNE     BUCLE             de cero, comienza de nuevo
*                                 el bucle
        MOVEA.W #CADNUM,A0      Paso 12.
        ADDI    #1,D0           Paso 13. Incrementa TAM
        BSR     INVERT          Paso 14.
*                               Paso 15. Fin del programa
        STOP    #$2000            Detiene la ejecución


*-----------------------------------------------------------
* Zona de datos
*-----------------------------------------------------------
DATOS   ORG     $2000
CADNUM  DS.B    6               Se reservan 6 bytes de memoria
*                               para CADNUM
        ORG     $2010
NUM_BIN EQU     $3F7B           El número binario que
*                               se quiera convertir

*-----------------------------------------------------------
* Zona de subrutinas
*-----------------------------------------------------------
SUBS    ORG     $3000
INVERT  MOVE.W  #0,D1           Inicializa CONTADOR
        MOVE.W  #0,D1           Paso 1. Inicializa CONTADOR
        SUBI    #1,D0           Paso 2.
BCL_IN  MOVE.W  A0,A1           Paso 3. A1 apunta
```

```
        ADDA.W  D0,A1             al final del vector
*                                 Paso 4.
        MOVE.B  (A1),D3           D3 contiene la variable AUX
*                                 Paso 5.
        MOVEQ   #1,D4             Inicializa CONTADOR_BUCLE2
BCL2_IN MOVE.W  A0,A1             Paso 6. A1 apunta
        ADDA.W  D0,A1             a V(TAM-CONT_BUCLE2)
        SUBA.W  D4,A1
        MOVE.W  D0,D2             Copia la long. del vector en D2
        MOVE.B  (A1)+,D5          Paso 7.
        MOVE.B  D5,(A1)           Mueve un elemento a la derecha
*                                 Paso 8.
        ADDI    #1,D4             Incrementa CONTADOR_BUCLE2
*                                 Paso 9.
        SUB.W   D1,D2             Calcula TAM-CONTADOR
        CMP.W   D4,D2             Si D4 menor o igual que D2
        BGE     BCL2_IN           repite BUCLE2 (Paso 6)
        MOVE.W  A0,A1             Paso 10.
        ADDA.W  D1,A1             A1 apunta a V(CONTADOR)
        MOVE.B  D3,(A1)           Paso 11.
*                                   Mueve AUX a V(CONTADOR)
        ADDI    #1,D1             Paso 12. Incrementa CONTADOR
        MOVE.W  D0,D2             Paso 13.
        CMP.W   D1,D2               Si D1 menor que D2
        BGT     BCL_IN              repite BUCLE
        RTS                       Paso 14. Retorno al programa
*                                   principal

        MOVE.B  #9,D0             Fin del programa
        TRAP    #15               Halt Simulator

        END     CONVERT
```

Traza de ejecución

En el fichero `convertir_reg.txt` se muestra una traza de la ejecución del programa del programa para el número $16251_{10} = 3F7B_{16}$; en la traza se muestra el contenido de los registros de datos y memoria, así como el puntero de pila, registros de estado y contador de programa; también se representan las posiciones de memoria (`$2000` a `$2006` en las que se situará el vector `CADNUM`.

Práctica III

Enunciado de la práctica

Primera parte Realícese un programa en el ensamblador del MC68000 que nos diga el día de la semana que fue (*lunes* a *domingo*) un día concreto facilitado por teclado. En concreto, el programa preguntará en primer lugar el año; en segundo lugar, el número de mes (1 a 12) y, en tercer lugfar, el día. Calcularemos y mostraremos el día de la semana de la fecha introducida sabiendo que el 1 de enero de 2007 fue *lunes*. Las fechas introducidas tienen que ser anteriores a ese valor.

Segunda parte Mejorar el programa anterior para que, en vez de preguntarnos el número de mes, nos pregunte el *nombre* del mes (de *enero* a *diciembre*). Para ello deberemos escribir una subrutina que compare dos cadenas de texto, cuyas direcciones serán proporcionadas en `A1` y `A2`. La subrutina devolverá `1` en `D1` si son iguales y `0` en `D1` si son diferentes.

Tercera parte Mejorar el apartado anterior para comprobar que los tres datos introducidos por teclado son válidos: para el año, estará comprendido en el rango $[1, \ldots, 2006]$; el nombre del mes tiene que ser uno de los doce posibles, todo en minúsculas; el día del mes tendrá que ser válido para el mes que hemos introducido (por ejemplo, si el mes introducido es "febrero", el día del mes deberá pertenecer al intervalo $[1, \ldots, 28]$).

N.B. Con el objetivo de simplificar la práctica, no se tendrán en cuenta en ningún apartado los años bisiestos, *i.e.*, se supondrá que todos los años tienen 365 días.

Primera parte

Descripción textual del algoritmo

La solución que se propone consta de dos componentes. El primero de ellos debe realizar la conversión de la fecha introducida a términos "absolutos", *i.e.*, contar el número de días que median entre la fecha introducida y el 1 de enero de 2007. Seguidamente, el algoritmo deberá efectuar una *correspondencia* entre dicho *número* de días y un determinado *nombre* de día de la semana, *i.e.*, "encasillar" este número en una de siete posibles celdas.[11]

El cáculo del número de días que median entre la fecha introducida y el 1 de enero de 2007 puede determinarse mediante un ejemplo. Supongamos, en efecto, que la fecha introducida es el *25 de julio de 2004*. El número de días sería entonces:

$$D = 365 \cdot (2006 - 2004) + \underbrace{31}_{12} + \underbrace{30}_{11} + \underbrace{31}_{10} + \underbrace{30}_{9} + \underbrace{31}_{8} + \underbrace{31}_{7} - 25,$$

donde los números $12, 11, 10 \ldots$ indican el *índice del mes* al que corresponden los números de días $31, 30, 31, \ldots$. El hecho de que a cada mes corresponde un múmero diferente de días invita a situar dichas cifras en un vector (llamémoslo `V1`) y acceder a ellas mediante direccionamiento relativo para facilitar los cálculos.

Una vez efectuado el cálculo del número de días, efectuaremos con el resultado la operación $N = D \mod 7$, haciendo corresponder a N, *resto* de la

[11] Así, si por ejemplo introdujésemos la fecha del 31 de diciembre de 2006, el algoritmo debería determinar que entre ambas fechas media *1* día, número al que debería corresponder la palabra *domingo*, exactamente igual que si mediasen *8*, *15*, *22*, etcétera. Es obvio que la implementación matemática de dicha correspondencia es la operación *módulo*. Por cierto que este método es idéntico (salvo por el número de celdas o casillas) al modo como en España se realiza la asignación de las letras del número de identificación fiscal a partir de la clave proporcionada por el número del Documento Nacional de Identidad.

división de N entre 7, un determinado significante, según la siguiente tabla de equivalencia:

D mod 7	*Día de la semana*
0	lunes
1	domingo
2	sábado
3	viernes
4	jueves
5	miércoles
6	martes

La implementación de esta tabla de equivalencia, trivial en un lenguaje de alto nivel, requeriría en ensamblador la utilización de un *direccionamiento indirecto*. En memoria estarán almacenados los nombres de los días de la semana o, dicho con más precisión, los códigos ASCII de los caracteres (en minúsculas) de los que están compuestos; las cadenas se hallarían separadas por un carácter apropiado (por ejemplo, el carácter nulo). Otra tabla situada en memoria contendría las *direcciones* de inicio de cada nombre y, por último, a las celdas de dicha tabla se accedería mediante un puntero compuesto de su dirección absoluta (supongámosla almacenada en el registro `A1`) más un número de posiciones dado, precisamente, por la operación $D \mod 7$.

Descripción pormenorizada del algoritmo y variables empleadas

Para la mayor parte de las variables utilizadas en el algoritmo (códigos ASCII, número del mes, día del mes, etcétera) será suficiente contar con un octeto de memoria, que permitirá representar números naturales pertenecientes al intervalo $[0, 255]$. Para representar el año de la fecha introducida reservaremos una palabra de memoria —asumiremos que se introducen solamente años positivos, correspondientes por tanto a fechas de nuestra era— y para la cuenta de días transcurridos entre aquella y el 1 de enero de 2007, una palabra larga, que permitirá albergar un entero positivo comprendido entre 0 y aproximadamente $4{,}295 \cdot 10^9$, rango más que suficiente habida cuenta de la acotación antedicha.

El algoritmo transcurrirá básicamente según la siguiente secuencia de pasos:[12]

1. Leer los números de año, mes y día, que se introducirán por teclado y se almacenarán, respectivamente, en las variables `AÑO`, `MES` y `DIA`.

2. Calcular el número de días transcurridos entre la fecha introducida y el 1 de enero de 2007, cifra que se almacenará en la variable `NUM_DIAS`. Dicho cálculo deberá, tal como se ha indicado, leer *hacia atrás* los datos (número de días de cada mes) del vector `V1` desde el correspondiente a diciembre

[12] Asumiremos que los vectores `V1`, `V2` y `V3` que se han definido están presentes en memoria como datos, para lo cual deberán, obviamente, introducirse como constantes en la zona correspondiente del programa.

hasta el correspondiente al introducido por teclado (incluyendo éste), para lo cual habrá de implementarse un bucle que deberá disponer de una variable que cuente los "pasos" efectuados, variable que denominaremos `CONTADOR`. Será necesaria también una variable auxiliar para almacenar datos parciales; denominaremos `AUX` a tal variable.

3. Calcular el *resto* de la división entera de `NUM_DIAS` entre 7; el resultado de dicha operación se almacenará en la variable N.

4. Mediante el direccionamiento indirecto descrito anteriormente, leer en memoria el nombre del día de la semana que corresponda al N calculado y representarlo en el periférico de salida. Para ello haremos uso nuevamente de la variable `AUX` antes citada, así como de una variable `CHAR` en la que iremos almacenando los códigos ASCII de los caracteres del nombre del día de la semana; la lectura finalizará cuando encontremos el carácter nulo.

Descripción por pasos del algoritmo

Los pasos del algoritmo que se propone para resolver el problema planteado se muestran a continuación.[13]

Paso	Descripción	Pseudocód.
1	Ejecutar la interrupción de lectura de dato numérico desde teclado para leer la variable `AÑO` y situarla en `D3`. Este registro se empleará también para computar el número de días transcurridos entre la fecha introducida y el 1 de enero de 2007.	input `AÑO` → `D3`
2	Ejecutar la interrupción de lectura de dato numérico desde teclado para leer `MES` y situar la variable en `D2`.	input `MES` → `D2`
3	Ejecutar la interrupción de lectura de dato numérico desde teclado para leer `DIA` y situar la variable en `D1`.	input `DIA` → `D1`
4	Realizar la operación $365 \cdot (2006 - \texttt{AÑO})$ y almacenar su resultado en `D3`.	`D3` ← $365 \cdot (2006 - \texttt{D3})$
5	Inicializar la variable `CONTADOR`, que emplearemos para acceder a los datos del vector `V1`, que contiene el número de días de cada mes.	`D5` ← 12

[13] Para implementar las operaciones de entrada y salida haremos uso de las excepciones en modo supervisor previstas en el *Easy68K*, invocadas a través de la instrucción `TRAP`. Concretamente, la instrucción `TRAP #15` se emplea para intercambiar datos con los periféricos de entrada y salida, debiéndose situar en el registro `D0` el código de la tarea que se desea ejecutar. Emplearemos las siguientes:

- `2` Lee una cadena de caracteres del teclado y la almacena en (`A1`), devolviendo su longitud (80 caracteres como máximo) en `D1.W`.
- `4` Lee un número del teclado y lo almacena en `D1.L`.
- `6` Muestra en pantalla el carácter cuyo código ASCII se halle almacenado en `D1.B`.

6	Hacer que `A1` apunte al inicio del vector `V1`, situado en la zona de memoria etiquetada como `DIAS_M`.	$\texttt{A1} \leftarrow \texttt{\#DIAS\_M}$
7	Inicializar la variable `AUX`, en la que se irán almacenando los elementos leídos de `V1`.	$\texttt{D6} \leftarrow 0$
8	Copiar el dato situado en la posición de `V1` apuntada por `A1` en la variable `AUX`. Emplearemos direccionamiento indirecto a registro con post-autoincremento.	$\texttt{D6} \leftarrow (\texttt{A1})+$
9	Decrementar la variable `CONTADOR`.	$\texttt{D5} \leftarrow \texttt{D5} - 1$
10	Sumar el contenido de la variable `AUX` al cómputo total de días, almacenado en `D3`.	$\texttt{D3} \leftarrow \texttt{D3} + \texttt{D6}$
11	Si `CONTADOR` $\geq$ `MES`, saltar al paso *8*.	To *8* if $\texttt{D5} \geq \texttt{D2}$
12	Restar la variable `DIA` del cómputo total de días transcurridos. (Tras este paso, el registro `D3` contiene el número de días transcurridos hasta el *31 de diciembre de 2006*).	$\texttt{D3} \leftarrow \texttt{D3} - \texttt{DIA}$
13	Incrementar el número de días en una unidad.	$\texttt{D3} \leftarrow \texttt{D3} + 1$
14	Efectuar la división entera del núymero de días (`D3`) entre 7 (el cociente se almacenará en la palabra menos significativa de `D3` y el resto en la palabra más significativa). Permutar las dos palabras de `D3`.	$\texttt{D3} \leftarrow \texttt{D3} \mod 7$
15	Hacer que `A1` apunte al inicio del vector `V2` (situado en la zona de memoria etiquetada como `DIR_D`), que contiene las *direcciones de inicio* de las cadenas ASCII correspondientes a los nombres de los días de la semana.	$\texttt{A1} \leftarrow \texttt{\#DIR\_D}$
16	Posicionar el apuntador de `A1` en la celda de `V2` que contenga la dirección del día de la semana correspondiente a la anterior operación `D3` mod 7. Puesto que las direcciones ocupan una *palabra* de memoria, debe multiplicarse por dos el resultado de dicho cálculo para que la operación se efectúe correctamente.	$\texttt{A1} \leftarrow \texttt{A1} + 2 \cdot \texttt{D3}$
17	Situar en `A2` el *contenido* de la dirección de memoria apuntada por `A1`. Se emplea direccionamiento indirecto a registro.	$\texttt{A2} \leftarrow (\texttt{A1})$
18	Inicio del bucle de escritura. Situar en `D1` el carácter apuntado por `A2`. Se emplea direccionamiento indirecto con postautoincremento.	$\texttt{D1} \leftarrow (\texttt{A2})+$
19	Si el código leído corresponde al carácter nulo (bit de estado `Z` a 1), saltar al paso *22* para finalizar el programa.	To *22* if $\texttt{Z} = 1$
20	En caso contrario, ejecutar la interrupción de escritura en pantalla del carácter almacenado en `D1`.	output `D1`

21	Si el código leído *no* corresponde al carácter nulo (bit de estado Z a 0), saltar al paso *18* para repetir el bucle de escritura.	To *18* if $Z = 0$
22	Fin del programa.	END

El flujograma del programa CONVERTIR_FECHA se muestra en la figura 4.

Codificación en ensamblador

Se ha codificado en ensamblador el programa pedido en el enunciado. El código propuesto se lista a continuación.

```
*-----------------------------------------------------------
* Program    :CONVERTIR_FECHA
* Written by :Miguel Moro Vallina
* Date       :18 de agosto de 2007
* Description:Muestra el día de la semana de una fecha
*       introducida por teclado
*-----------------------------------------------------------
CONV_F  ORG     $1000
        MOVE.B  #4,D0           Paso 1. Ejecuta la interrupción
        TRAP    #15               de lectura de dato de teclado
*                                 para leer el AÑO.
        MOVE.W  D1,D3             Sitúa AÑO en D3
        TRAP    #15             Paso 2. Lectura de MES
        MOVE.W  D1,D2             Sitúa MES en D2
        TRAP    #15             Paso 3. Lectura de DÍA
*                                 Queda almacenado en D1
        NEG     D3              Paso 4. Resta
        ADDI    #2006,D3          2006-AÑO
        MULU    #365,D3           Multiplica 365*(2006-AÑO)
        MOVEQ   #12,D5          Paso 5. Inicializa CONTADOR
        MOVEA.W #DIAS_M,A1      Paso 6. Sitúa en A1 la
*                                 dirección de DIAS_M
        MOVEQ   #0,D6           Paso 7. Inicializa AUX
BUCLE   MOVE.B  (A1)+,D6        Paso 8.
        SUBI    #1,D5           Paso 9. Decrementa CONTADOR
        ADD.W   D6,D3           Paso 10. D3=D3+AUX
        CMP     D5,D2           Paso 11. Si CONTADOR mayor
        BLE     BUCLE             que MES, salta a BUCLE
        SUB.W   D1,D3           Paso 12. D3=D3-DIA
        ADDI    #1,D3           Paso 13.
        DIVU    #7,D3           Paso 14.
        SWAP    D3                D3=D3 mod 7
        MOVEA.W #DIR_D,A1       Paso 15. A1 apunta a V2[0]
        MULU    #2,D3           Paso 16.
        ADDA.W  D3,A1             A1 apunta a V3[D3]
        MOVEA.W (A1),A2         Paso 16. A2 apunta a la
*                                 dirección de inicio de
*                                 la cadena corresp.
BUC_S   MOVE.B  (A2)+,D1        Paso 18. Bucle de escritura
        BEQ     FIN             Paso 19. Si el código leído
*                                 es $00, salta a FIN
        MOVE.B  #6,D0           Paso 20. Muestra en pantalla el
        TRAP    #15               caracter contenido en D1.B
        BNE     BUC_S           Paso 21. Si el código leído no
*                                 es $00, salta a BUC_S
```

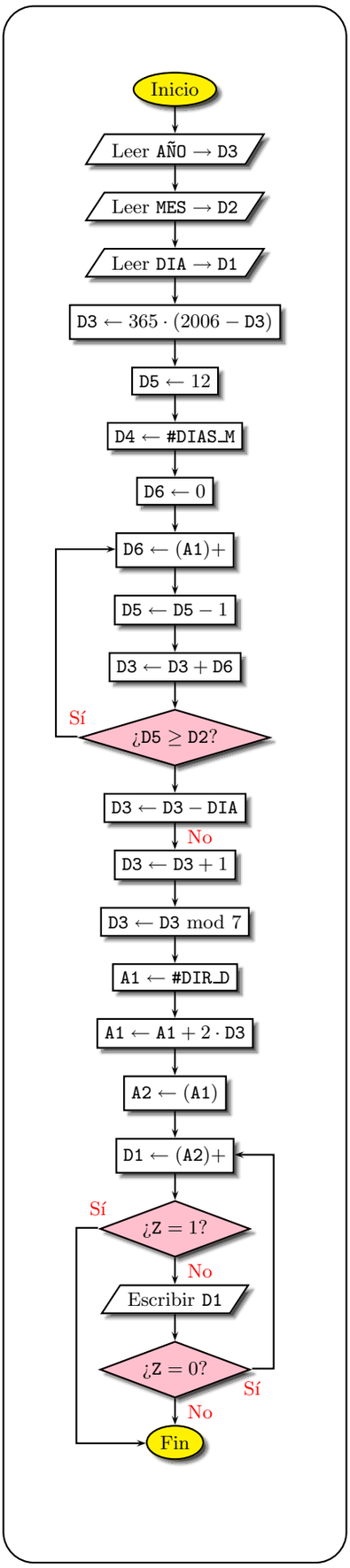

Figura 4. Diagrama de flujo del programa `CONVERTIR_FECHA`.

```
FIN     STOP    #$2000          Paso 22. Fin de programa

DATOS   ORG     $3000
DIAS_M  DS.B    12
        ORG     $3000
DIAS    DC.B    31,30,31,30,31,31,30,31,30,31,28,31
        ORG     $300C
DIR_D   DS.W    7
        ORG     $300C
DIR_LU  DC.W    $3020           Dirección de la cadena "lunes"
DIR_DO  DC.W    $3026           Dirección de la cadena "domingo"
DIR_SA  DC.W    $302E           Dirección de la cadena "sabado"
DIR_VI  DC.W    $3036           Dirección de la cadena "viernes"
DIR_JU  DC.W    $303E           Dirección de la cadena "jueves"
DIR_MI  DC.W    $3046           Dirección de la cadena "miercoles"
DIR_MA  DC.W    $3050           Dirección de la cadena "martes"
        ORG     $3020
NOM_D   DS.B    6
        ORG     $3020
LUN     DC.B    $6C,$75,$6E,$65,$73
        DC.B    0               Final de "lunes"
        ORG     $3026
        DS.B    8
        ORG     $3026
DOM     DC.B    $64,$6F,$6D,$69,$6E,$67,$6F
        DC.B    0               Final de "domingo"
        ORG     $302E
        DS.B    7
        ORG     $302E
SAB     DC.B    $73,$61,$62,$61,$64,$6F
        DC.B    0               Final de "sábado"
        ORG     $3036
        DS.B    8
        ORG     $3036
VIE     DC.B    $76,$69,$65,$72,$6E,$65,$73
        DC.B    0               Final de "viernes"
        ORG     $303E
        DS.B    7
        ORG     $303E
JUE     DC.B    $6A,$75,$65,$76,$65,$73
        DC.B    0               Final de "jueves"
        ORG     $3046
        DS.B    10
        ORG     $3046
MIE     DC.B    $6D,$69,$65,$72,$63,$6F,$6C,$65,$73
        DC.B    0               Final de "miercoles"
        ORG     $3050
        DS.B    7
        ORG     $3050
MAR     DC.B    $6D,$61,$72,$74,$65,$73
        DC.B    0               Final de "martes"

        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     CONV_F
```

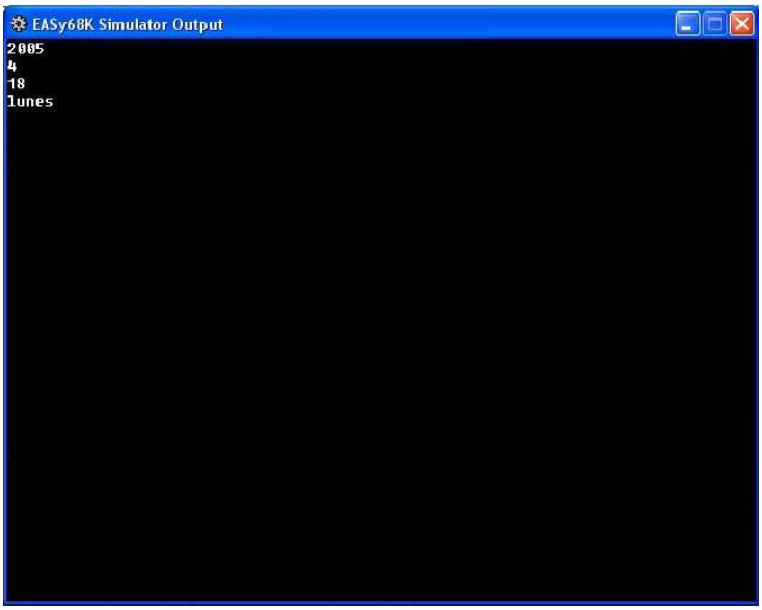

Figura 4: Captura de pantalla de la ventana de entrada y salida del simulador.

Simulación del programa

Se realiza una simulación del programa elaborado con el *Easy68K*. Los datos introducidos por teclado y el resultado mostrado en pantalla se muestra en la captura de pantalla de la figura 4. La traza —registros de datos y memoria, registro de estado, puntero de pila y contador de programa— correspondiente a la ejecución del algoritmo para dichos datos se muestra en el fichero `convertir_fecha_reg.txt`.

Segunda parte

Descripción textual del algoritmo

La subrutina `COMPARA_CADENAS`[14] realizará la comparación, carácter a carácter, de dos cadenas de texto situadas en sendas posiciones de memoria, almacenadas en los registros de direcciones `A1` y `A2`. Asumiremos que dichas cadenas contienen un "carácter de terminación" (emplearemos nuevamente el carácter nulo) tal que, al llegar a él, la comparación finalizará. Tan pronto como la comparación por caracteres encuentra una pareja distinta, el proceso finaliza situando un `0` en `D1`. En caso contrario, se continúa avanzando hasta llegar a

[14] En esta segunda parte de la práctica, nos limitaremos a implementar la subrutina mencionada, pero no replantearemos aún el programa `CONVERTIR_FECHA` para incluir la mejora propuesta en el enunciado de la práctica. La incluiremos en la tercera parte de la misma, junto con las relativas a asegurara la coherencia de los datos introducidos por teclado

los caracteres de terminación de ambas cadenas, en cuyo caso se colocará un `1` en el registro `D1` indicando con ello que ambas cadenas son idénticas.

Descripción pormenorizada del algoritmo y variables empleadas

Supondremos que las cadenas son sendos vectores —que llamaremos `V1` y `V2`— de caracteres o, más bien, de códigos ASCII, con lo que reservaremos un octeto de memoria para cada uno de ellos. Sus direcciones de inicio, como se ha indicado, se almacenan en los registros de direcciones `A1` y `A2`. Emplearemos dos variables (`CHAR1` y `CHAR2`, ambas de un byte de longitud) en las que cargaremos cada pareja de caracteres a comparar. El bucle avanzará hacia posiciones crecientes de memoria hasta alcanzar sendos caracteres de terminación. La comparación finalizará con resultado positivo, lógicamente, si dichos caracteres se alcanzan *a la vez* en ambas cadenas.

Descripción por pasos del algoritmo propuesto

La subrutina `COMPARA_CADENAS` propuesta se compondría de los pasos siguientes:

Paso	Descripción	Pseudocód.
1	Inicio del bucle de comparación. Cargar en `D2` el contenido de la posición a la que apunta `A1`. Se emplea direccionamiento indirecto a registro con post-autoincremento.	`D2` ← (`A1`)+
2	Cargar en `D3` el contenido de la posición a la que apunta `A2`. Se emplea también direccionamiento indirecto a registro con post-autoincremento.	`D3` ← (`A2`)+
3	Si `D2` ≠ `D3`, saltar al paso *7*.	To *7* if `D2` ≠ `D3`
4	En caso contrario, comprobar si `D2` = 0 (condición que indicaría el fin de la cadena). Si `D2` ≠ 0, saltar al paso *1* para repetir la comparación con la siguiente pareja de caracteres.	To *1* if `D2` ≠ `$00`
5	En caso contrario (`D2` = 0), situar un "`1`" en `D1` para indicar que ambas cadenas son iguales.	`D1` ← 1
6	Saltar al paso *8* (fin de programa).	To *8*.
7	(Las cadenas son desiguales). Situar un "`0`" en `D1`.	`D1` ← 0
8	Fin del programa.	`END`

El flujograma de la subrutina propuesta se muestra en la figura 5.

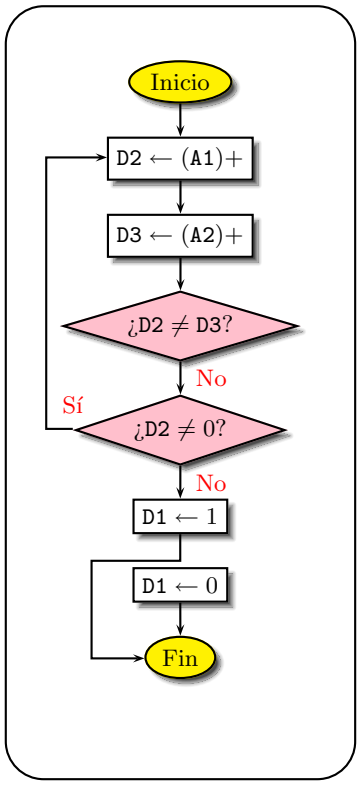

Figura 5. Diagrama de flujo de la subrutina `COMPARA_CADENAS`.

Codificación en ensamblador

El código elaborado para la subrutina propuesta en el enunciado de la práctica es el que se lista a continuación:

```
*-----------------------------------------------------------
* Program    :COMPARA_CADENAS
* Written by :Miguel Moro Vallina
* Date       :19 de agosto de 2007
* Description:Compara dos cadenas de caracteres
*-----------------------------------------------------------
CMP_CAD ORG     $1000
BUC_CMP MOVE.B  (A1)+,D2        Paso 1.
        MOVE.B  (A2)+,D3        Paso 2.
        CMP.B   D2,D3           Paso 3. Si D2 no es igual que
        BNE     DESIG             D3, salta al paso 7
        CMP.B   #$00,D2         Paso 4. Si D2 no es igual a
        BNE     BUC_CMP           $00 (car. nulo), salta a
*                                 paso 1 para repetir bucle.
IGUAL   MOVE.B  #1,D1           Paso 5. Iguales: sitúa 1 en D1
        BRA     FIN             Paso 6. Salta al paso 8 (fin
*                                 de programa
DESIG   MOVE.B  #0,D1           Paso 7. Desiguales: sitúa 0
*                                 en D1
FIN     STOP    #$2000          Paso 8. Fin del programa

        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     CMP_CAD
```

Simulaciones de la subrutina

Se realizarán un par de simulaciones de la subrutina elaborada. En la primera de ellas se situarán dos cadenas de caracteres iguales (concretamente, la cadena "`diciembre`") en sendas posiciones de mamoria (las posiciones `$0000` y `$0010`). La captura de pantalla mostrada en la figura 5 muestra la presencia de dichas cadenas en memoria. La traza generada por la subrutina es la mostrada en el fichero `compara_cadenas_iguales.txt`. Si, por el contrario, hacemos que las cadenas de texto sean desiguales (concretamente, se introducen las cadenas "`diciembr`" y "`diciembre`" en las posiciones de memoria antes indicadas, tal y como recoge la figura 6), la traza generada por el simulador al ejecutar la subrutina es la listada en el fichero `compara_cadenas_desiguales.txt`.

Tercera parte

Descripción textual del algoritmo

El programa, "mejorado" para asegurar la coherencia de los datos introducidos, es bastante similar al descrito en la primera parte de esta práctica. Su

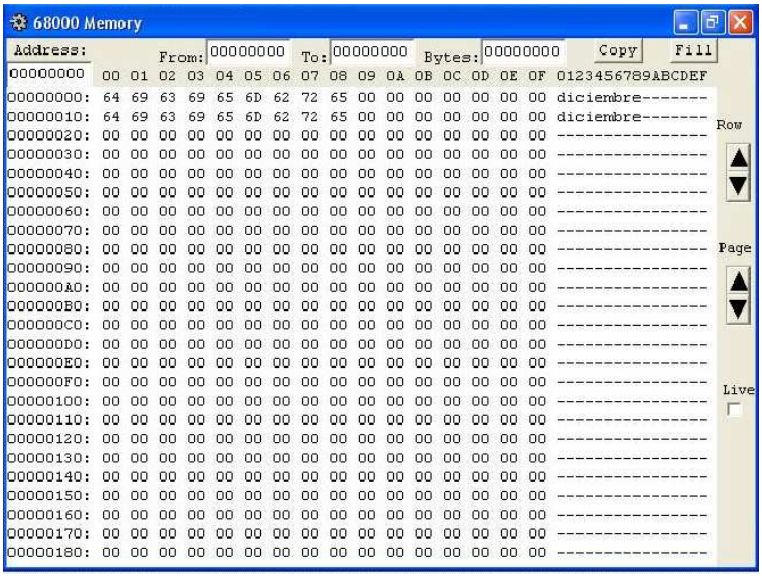

Figura 5: Memoria del simulador para cadenas iguales.

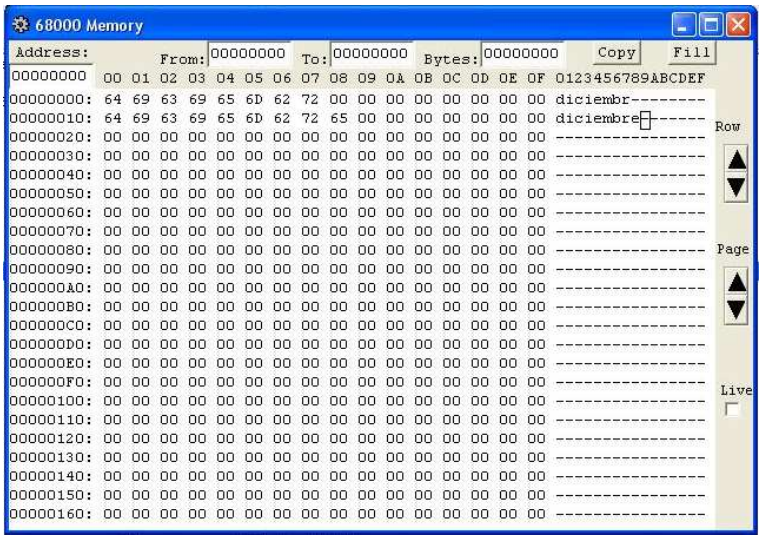

Figura 6: Memoria del simulador para cadenas desiguales.

única diferencia con aquel es que, para cada uno de los datos DIA, MES y AÑO introducidas, el algoritmo comprobará que los datos son coherentes, volviendo a realizar la petición del dato en caso contrario. En el caso de la variable AÑO, dicha comparación es trivial: deberá cumplirse que AÑO sea *mayor que* cero *y menor que* 2007. Para comprobar que el nombre del mes esté entre los doce posibles habremos de emplear nuevamente un modo de *direccionamiento indirecto*. En efecto, en memoria estarán situados —en posiciones consecutivas y separadas por caracteres nulos, tal como se hizo anteriormente con los nombres de los días de la semana— los nombres de los meses del año, en un vector que denominaremos V5. Una tabla de apuntadores, V4, contendrá en sus celdas la dirección de inicio de cada nombre.

El algoritmo que habrá de comprobar la coherencia del nombre de mes consistirá en un bucle que obrará del modo siguiente. Un bucle permitirá recorrer la tabla de apuntadores antes mencionada. La dirección a la que apunta cada celda (*i.e.*, la dirección de comienzo de la cadena que contiene el nombre del mes) se cargará en A2 y, seguidamente, se invocará a la subrutina COMPARA_CADENAS para que coteje el texto introducido por teclado (situado en una dirección de memoria apuntada por el registro A1) y la cadena a la que apunta A2. Si el registro D1 contiene un 0, el bucle se ejecutará de nuevo para comprobar si el texto introducido por teclado coincide con el nombre del mes siguiente. Si, por el contrario, D1 contiene un 1, el bucle finalizará y en MES se almacenará el contenido del contador del bucle (número del mes cuyo nombre ha coincidido con la cadena introducida). Si se llega al último mes y ninguno ha coincidido, el programa volverá a pedir que se introduzca el dato por teclado.

Restará, por último, comprobar la coherencia del dato DIA. Para ello emplearemos el vector V1 definido en la primera parte de la práctica, que contiene en sus celdas el número de días de cada mes. Conocido el dato MES, bastará comprobar que DIA es *menor que* cero y *menor o igual* que el dato almacenado en la posición MES de V1. En el caso de que la intersección de ambas condiciones sea cierta, el prgrama continuará ya de modo idéntico a como se ha descrito en la primera poarte. En caso contrario, volverá a pedirse el dato DIA hasta que se introduzca correctamente.

Descripción por pasos del algoritmo

El programa para la conversión de fecha, mejorado para incluir la comprobación de que los datos introducidos son coherentes, se compondría de los siguientes pasos:

Paso	Descripción	Pseudocód.
1	Ejecutar la interrupción de lectura de dato numérico desde teclado para leer AÑO y situarlo en D1.	input AÑO → D1
2	Si AÑO ≤ 0, saltar al paso *1* para volver a pedir el dato.	To *1* if AÑO ≤ 0
3	Si AÑO ≥ 2007, saltar al paso *1* para volver a pedir el dato.	D3 ← D1

4	En caso contrario, almacenar AÑO en D3.	To *1* if AÑO $\geq$ 2007
5	Ejecutar la interrupción de lectura de la variable MES (cadena de caracteres). El registro A1 apuntará a la zona de memoria ($00) en la que el simulador sitúa la cadena leída.	input MES $\rightarrow$ (A1)
6	Hacer que el registro A3 apunte a V4, vector que contiene las direcciones de inicio de las cadenas ASCII correspondientes a los nombres de los meses; dicho vector está almacenado en la zona de memoria etiquetada como DIR_M.	A3 $\leftarrow$ #DIR_M
7	Inicializar CONTADOR, variable que emplearemos para comparar los sucesivos nombres con la cadena introducida.	D4 $\leftarrow$ 0
8	Inicio del bucle de comparación de la cadena con el nombre del mes. Hacer que A2 apunte a la dirección contenida en la zona de memoria apuntada por A3. Emplearemos direccionamiento indirecto con post-autoincremento.	A2 $\leftarrow$ (A3)+
9	Hacer que A1 apunte a la dirección de memoria $00, en la que el simulador ha situado la cadena introducida por teclado.	A1 $\leftarrow$ $00
10	Invocar a la subrutina COMPARA_CADENAS.	To CMP_CAD
11	Si D1 = 1, el resultado de la comparación devuelto por la subrutina es positivo (las cadenas son iguales). Saltar al paso *15*.	To *15* if D1 = 1
12	Incrementar CONTADOR.	D4 $\leftarrow$ D4 + 1
13	Si CONTADOR $\leq$ 11, saltar al paso *8* para efectuar la comparación con el siguiente nombre de mes.	To *8* if D1 $\leq$ 11
14	En caso contrario, las cadenas introducidas no coinciden con ninguno de los nombres de mes. Saltar al paso *5* para volver a leer MES.	To *5*
15	Hacer que A1 apunte al inicio de V1, vector que contiene el número de días de cada mes, situados en orden inverso.	A1 $\leftarrow$ #DIAS_M
16	Lectura del día del mes. Ejecutar la interrupción de lectura de dato numérico desde teclado para leer DIA y situarlo en D1.	input DIA $\rightarrow$ D1
17	Hacer que A1 apunte a la posición de V1 indicada por 11 $-$ CONTADOR.	D4 $\leftarrow$ 11 $-$ D4; A1 $\leftarrow$ A1 + D4
18	Copiar el contenido de la celda de V1 apuntada por A1 a AUX. Emplearemos direccionamiento indirecto a registro.	D2 $\leftarrow$ (A1)
19	Si DIA $\leq$ 0, saltar al paso *16* para volver a pedir el dato.	To *16* if D1 $\leq$ 0.

20	Si DIA $>$ AUX, saltar al paso *16* para volver a pedir el dato. En caso contrario, el programa continúa normalmente.	To *16* if D1 $>$ D2
21	Sitúa en D2 el contenido de la operación $12 -$ CONTADOR.	D2 $\leftarrow 12 -$ D4
22	Realizar la operación $365 \cdot (2006 -$ AÑO$)$ a almacenar su resultado en D3.	D3 $\leftarrow 365 \cdot (2006 -$ D3$)$
23	Inicializar la variable CONTADOR, que emplearemos para acceder a los datos del vector V1, que contiene el número de días de cada mes.	D5 $\leftarrow 12$
24	Hacer que A1 apunte al inicio del vector V1, situado en la zona de memoria etiquetada como DIAS_M.	A1 $\leftarrow$ #DIAS_M
25	Inicializar la variable AUX, en la que se irán almacenando los elementos leídos de V1.	D6 $\leftarrow 0$
26	Copiar el dato situado en la posición de V1 apuntada por A1 en la variable AUX. Emplearemos direccionamiento indirecto a registro con post-autoincremento.	D6 $\leftarrow$ (A1)+
27	Decrementar la variable CONTADOR.	D5 $\leftarrow$ D5 $- 1$
28	Sumar el contenido de la variable AUX al cómputo total de días, almacenado en D3.	D3 $\leftarrow$ D3 $+$ D6
29	Si CONTADOR $\geq$ MES, saltar al paso *26*.	To *26* if D5 $\geq$ D2
30	Restar la variable DIA del cómputo total de días transcurridos. (Tras este paso, el registro D3 contiene el número de días transcurridos hasta el *31 de diciembre de 2006*).	D3 $\leftarrow$ D3 $-$ DIA
31	Incrementar el número de días en una unidad.	D3 $\leftarrow$ D3 $+ 1$
32	Efectuar la división entera del número de días (D3) entre 7 (el cociente se almacenará en la palabra menos significativa de D3 y el resto en la palabra más significativa). Permutar las dos palabras de D3.	D3 $\leftarrow$ D3 mod 7
33	Hacer que A1 apunte al inicio del vector V2 (situado en la zona de memoria etiquetada como DIR_D), que contiene las *direcciones de inicio* de las cadenas ASCII correspondientes a los nombres de los días de la semana.	A1 $\leftarrow$ #DIR_D
34	Posicionar el apuntador de A1 en la celda de V2 que contenga la dirección del día de la semana correspondiente a la anterior operación D3 mod 7. Puesto que las direcciones ocupan una *palabra* de memoria, debe multiplicarse por dos el resultado de dicho cálculo para que la operación se efectúe correctamente.	A1 $\leftarrow$ A1 $+ 2 \cdot$ D3

35	Situar en A2 el *contenido* de la dirección de memoria apuntada por A1. Se emplea direccionamiento indirecto a registro.	A2 ← (A1)
36	Inicio del bucle de escritura. Situar en D1 el carácter apuntado por A2. Se emplea direccionamiento indirecto con postautoincremento.	D1 ← (A2)+
37	Si el código leído corresponde al carácter nulo (bit de estado Z a 1), saltar al paso *40* para finalizar el programa.	To *40* if Z = 1
38	En caso contrario, ejecutar la interrupción de escritura en pantalla del carácter almacenado en D1.	output D1
39	Si el código leído *no* corresponde al carácter nulo (bit de estado Z a 0), saltar al paso *36* para repetir el bucle de escritura.	To *36* if Z = 0
40	Fin del programa.	END

El diagrama de flujo del programa CONVERTIR_FECHA_MEJORADO se muestra en la figura 6, situada al final de esta memoria de prácticas.

Codificación en ensamblador

El código en el ensamblador del MC68000 propuesto para la resolución del problema se lista a continuación:

```
*-----------------------------------------------------------
* Program    :CONVERTIR_FECHA_MEJORADO
* Written by :Miguel Moro Vallina
* Date       :19 de agosto de 2007
* Description:Muestra el día de la semana de una fecha
*       introducida por teclado, comprobando
*       además la coherencia de los datos introducidos
*-----------------------------------------------------------
CNV_F_M ORG     $1000
BCL_A   MOVE.B  #4,D0           Paso 1. Lee la variable AÑO
        TRAP    #15               desde el teclado
        CMP.W   #0,D1           Paso 2. Si AÑO menor o igual
        BLE     BCL_A             que 0 salta a BCL_A para
*                                 volver a pedir dato
        CMP.W   #2007,D1        Paso 3. Si AÑO mayor o igual que
        BGE     BCL_A             2007, salta a BCL_A para volver
*                                 a pedir dato.
        MOVE.W  D1,D3           Paso 4. En caso contrario,
*                                 almacena AÑO en D3
BCL_M   MOVE.B  #2,D0           Paso 5. Lee la variable MES
        TRAP    #15               (string) desde el teclado.
        MOVEA.W #DIR_M,A3       Psdo 6. A3 apunta a V4
        MOVEQ   #0,D4           Paso 7. Inicializa CONTADOR
BCL_CMP MOVEA.W (A3)+,A2        Paso 8. A2 apunta a inicio de
*                                 cadenas
        MOVEA.W #$00,A1         Paso 9.
        BSR     CMP_CAD         Paso 10.
```

```
        CMP.B   #1,D1           Paso 11. Si D1=1, salta a la
        BEQ     CONT              lectura del dia
        ADDI    #1,D4           Paso 12. Incrementa CONTADOR
        CMP.B   #11,D4          Paso 13. Si CONTADOR menor que
        BLE     BCL_CMP           11, salta a BCL_CMP.
        BRA     BCL_M           Paso 14. En caso contrario,
*                                 vuelve a pedir MES
CONT    MOVEA.W #DIAS_M,A1      Paso 15. Sitúa en A1 la dirección
*                                 de DIAS_M
BCL_D   MOVE.B  #4,D0           Paso 16. Lee la variable DIA desde
        TRAP    #15               el teclado
        NEG     D4              Paso 17.
        ADDI    #11,D4            D4=11-D4
        ADDA.W  D4,A1             A1 apunta a V1[D4]
        MOVE.B  (A1),D2         Paso 18. Copia el número de días
*                                 del mes a D2
        CMPI.B  #0,D1           Paso 19. Si D1 menor o igual que
        BLE     BCL_D             0, vuelve a pedir DIA
        CMP.B   D1,D2           Paso 20. Si D1 mayor que D2,
        BLT     BCL_D             vuelve a pedir DIA. En caso
*                                 contrario, continúa con el
*                                 programa normalmente.

        NEG     D4              Paso 21.
        ADDI    #12,D4            D2=12-D4
        MOVE.B  D4,D2
        NEG     D3              Paso 22. Resta
        ADDI    #2006,D3          2006-AÑO
        MULU    #365,D3           Multiplica 365*(2006-AÑO)
        MOVEQ   #12,D5          Paso 23. Inicializa CONTADOR
        MOVEA.W #DIAS_M,A1      Paso 24. Sitúa en A1 la
*                                 dirección de DIAS_M
        MOVEQ   #0,D6           Paso 25. Inicializa AUX
BUCLE   MOVE.B  (A1)+,D6        Paso 26.
        SUBI    #1,D5           Paso 27. Decrementa CONTADOR
        ADD.W   D6,D3           Paso 28. D3=D3+AUX
        CMP     D5,D2           Paso 29. Si CONTADOR mayor
        BLE     BUCLE             que MES, salta a BUCLE
        SUB.W   D1,D3           Paso 30. D3=D3-DIA
        ADDI    #1,D3           Paso 31.
        DIVU    #7,D3           Paso 32.
        SWAP    D3                D3=D3 mod 7
        MOVEA.W #DIR_D,A1       Paso 33. A1 apunta a V2[0]
        MULU    #2,D3           Paso 34.
        ADDA.W  D3,A1             A1 apunta a V3[D3]
        MOVEA.W (A1),A2         Paso 35. A2 apunta a la
*                                 dirección de inicio de
*                                 la cadena corresp.
BUC_S   MOVE.B  (A2)+,D1        Paso 36. Bucle de escritura
        BEQ     FIN             Paso 37. Si el código leído
*                                 es $00, salta a FIN
        MOVE.B  #6,D0           Paso 38. Muestra en pantalla el
        TRAP    #15               caracter contenido en D1.B
        BNE     BUC_S           Paso 39. Si el código leído no
*                                 es $00, salta a BUC_S

FIN     STOP    #$2000          Paso 40. Fin de programa

*-----------------------------------------------------------
* Zona de datos
*-----------------------------------------------------------
```

```
DATOS   ORG     $3000
DIAS_M  DS.B    12
        ORG     $3000
DIAS    DC.B    31,30,31,30,31,31,30,31,30,31,28,31
        ORG     $300C
DIR_D   DS.W    7
        ORG     $300C
DIR_LU  DC.W    $3020           Dirección de la cadena "lunes"
DIR_DO  DC.W    $3026           Dirección de la cadena "domingo"
DIR_SA  DC.W    $302E           Dirección de la cadena "sabado"
DIR_VI  DC.W    $3036           Dirección de la cadena "viernes"
DIR_JU  DC.W    $303E           Dirección de la cadena "jueves"
DIR_MI  DC.W    $3046           Dirección de la cadena "miercoles"
DIR_MA  DC.W    $3050           Dirección de la cadena "martes"
        ORG     $3020
NOM_D   DS.B    6
        ORG     $3020
LUN     DC.B    $6C,$75,$6E,$65,$73
        DC.B    0               Final de "lunes"
        ORG     $3026
        DS.B    8
        ORG     $3026
DOM     DC.B    $64,$6F,$6D,$69,$6E,$67,$6F
        DC.B    0               Final de "domingo"
        ORG     $302E
        DS.B    7
        ORG     $302E
SAB     DC.B    $73,$61,$62,$61,$64,$6F
        DC.B    0               Final de "sábado"
        ORG     $3036
        DS.B    8
        ORG     $3036
VIE     DC.B    $76,$69,$65,$72,$6E,$65,$73
        DC.B    0               Final de "viernes"
        ORG     $303E
        DS.B    7
        ORG     $303E
JUE     DC.B    $6A,$75,$65,$76,$65,$73
        DC.B    0               Final de "jueves"
        ORG     $3046
        DS.B    10
        ORG     $3046
MIE     DC.B    $6D,$69,$65,$72,$63,$6F,$6C,$65,$73
        DC.B    0               Final de "miercoles"
        ORG     $3050
        DS.B    7
        ORG     $3050
MAR     DC.B    $6D,$61,$72,$74,$65,$73
        DC.B    0               Final de "martes"

        ORG     $3060
DIR_M   DS.W    13
        ORG     $3060
DIR_EN  DC.W    $3080           Dirección de "enero"
DIR_FE  DC.W    $3086           Dirección de "febrero"
DIR_MZ  DC.W    $308E           Dirección de "marzo"
DIR_AB  DC.W    $3094           Dirección de "abril"
DIR_MY  DC.W    $309A           Dirección de "mayo"
DIR_JN  DC.W    $30A0           Dirección de "junio"
DIR_JL  DC.W    $30A6           Dirección de "julio"
DIR_AG  DC.W    $30AC           Dirección de "agosto"
DIR_SE  DC.W    $30B4           Dirección de "septiembre"
```

```
DIR_OC  DC.W    $30C0           Dirección de "octubre"
DIR_NO  DC.W    $30C8           Dirección de "noviembre"
DIR_DI  DC.W    $30D2           Dirección de "diciembre"
        DC.W    $0000           Fin de direcciones meses

        ORG     $3080
NOM_M   DS.B    6
        ORG     $3080
ENE     DC.B    $65,$6E,$65,$72,$6F
        DC.B    0               Final de "enero"
        ORG     $3086
        DS.B    8
        ORG     $3086
FEB     DC.B    $66,$65,$62,$72,$65,$72,$6F
        DC.B    0               Final de "febrero"
        ORG     $308E
        DS.B    6
        ORG     $308E
MRZ     DC.B    $6D,$61,$72,$7A,$6F
        DC.B    0               Final de "marzo"
        ORG     $3094
        DS.B    6
        ORG     $3094
ABR     DC.B    $61,$62,$72,$69,$6C
        DC.B    0               Final de "abril"
        ORG     $309A
        DS.B    5
        ORG     $309A
MAY     DC.B    $6D,$61,$79,$6F
        DC.B    0               Final de "mayo"
        ORG     $30A0
        DS.B    6
        ORG     $30A0
JUN     DC.B    $6A,$75,$6E,$69,$6F
        DC.B    0               Final de "junio"
        ORG     $30A6
        DS.B    6
        ORG     $30A6
JUL     DC.B    $6A,$75,$6C,$69,$6F
        DC.B    0               Final de "julio"
        ORG     $30AC
        DS.B    7
        ORG     $30AC
AGO     DC.B    $61,$67,$6F,$73,$74,$6F
        DC.B    0               Final de "agosto"
        ORG     $30B4
        DS.B    11
        ORG     $30B4
SEP     DC.B    $73,$65,$70,$74,$69,$65,$6D,$62,$72,$65
        DC.B    0               Final de "septiembre"
        ORG     $30C0
        DS.B    8
        ORG     $30C0
OCT     DC.B    $6F,$63,$74,$75,$62,$72,$65
        DC.B    0               Final de "octubre"
        ORG     $30C8
        DS.B    10
        ORG     $30C8
NOV     DC.B    $6E,$6F,$76,$69,$65,$6D,$62,$72,$65
        DC.B    0               Final de "noviembre"
        ORG     $30D2
        DS.B    10
```

```
        ORG     $30D2
DIC     DC.B    $64,$69,$63,$69,$65,$6D,$62,$72,$65
        DC.B    0               Final de "diciembre"


*-----------------------------------------------------------
* Zona de subrutinas
*-----------------------------------------------------------

SUBS    ORG     $4000
CMP_CAD ORG     $4000
BUC_CMP MOVE.B  (A1)+,D5        Paso 1.
        MOVE.B  (A2)+,D6        Paso 2.

        CMP.B   D5,D6           Paso 3. Si D5 no es igual que
        BNE     DESIG             D6, salta al paso 7
        CMP.B   #$00,D5         Paso 4. Si D5 no es igual a
        BNE     BUC_CMP           $00 (car. nulo), salta a
*                                 paso 1 para repetir bucle.
IGUAL   MOVE.B  #1,D1           Paso 5. Iguales: sitúa 1 en D1
        BRA     FIN_S           Paso 6. Salta al paso 8 (fin
*                                 de programa
DESIG   MOVE.B  #0,D1           Paso 7. Desiguales: sitúa 0
*                                 en D1
FIN_S   RTS                     Fin de subrutina

        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     CNV_F_M
```

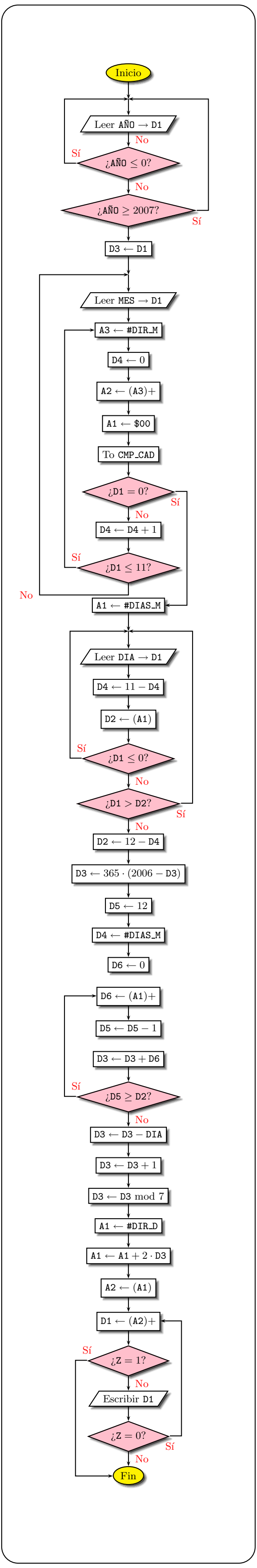

Figura 6. Diagrama de flujo del programa `CONVERTIR_FECHA_MEJORADO`.