DIARIO DE PRÁCTICAS

Luis Blanco de la Cruz
Roberto Gozalo Andrés

Primera Práctica 19/09/18

Ejercicio 1:

a)
En el simulador Mars, tras ejecutar nuestro programa vemos que es Little Endian porque en la dirección más baja del registro se guarda el bit más significativo, en este caso el 0, si fuera Big Endian se guardaría un 1. Ambas instrucciones son sintéticas, porque el compilador las cambia por otras básicas.

b)
Después de muchos errores, conseguimos compilar y ejecutar el programa correctamente.

c)
En nuestro programa utilizamos estas instrucciones:
-li: La cambia por addi.
-la: La cambia por lu+ori.
-b: La cambia por bgz.
-lw: Cuando se utiliza una etiqueta se transforma en lui.

d)
Todas las instrucciones sintéticas las marca como errores. Si, ahora el programa es mucho más difícil de leer que antes.

Ejercicio 2:

a)
No se traducen las instrucciones de la misma manera. la instrucción li se traduce como addiu en Mars y como ori en QtSPIM.

b)
bne no nos la marca como instrucción sintética pero probamos con beq y en Mars la traduce como addi+beq y en QtSPIM como ori+beq. Ambas son correctas, la traducción depende de cómo esté escrito el compilador de cada programa.

Ejercicio 3:

Como pone en el guión, al marcar la casilla de Bkpt se para el programa, si se marca en la casilla de un bucle el programa se para en cada iteración.

Segunda Práctica 26/09/18

Ejercicio 1:

Como se indica en el guión vamos a crear un programa para realizar el siguiente cálculo sobre matrices cuadradas:
B[i][j] = A[i][j] − A[j][i]
Dicho programa tiene como argumentos las direcciones de ambas matrices así como el número de filas de ellas (ya que son matrices cuadradas no necesitamos conocer el número de columnas porque es igual al de las filas).
Para comenzar con nuestro programa decidimos crear una función que hemos llamado calcula y que realizará el cálculo de la dirección
A[i][j] aplicando la fórmula obtenida en el guión.
A[i][j] = A +4 ∗ m ∗ i + 4 ∗ j
Durante la creación de dicha porción de código nos damos cuenta que nuestra función se puede optimizar sacando factor común quedando de la siguiente manera
A[i][j] = A +4 ∗ (m ∗ i + j)

Al principio nuestro programa no funciona correctamente, se nos había olvidado la gestión de la pila. Para solucionar esto primero debemos decrementar el valor de la pila, ya que esta apunta a la dirección más alta y antes de salir de la función devolver el apuntador a la dirección más alta.

Para recorrer las funciones necesitamos dos bucles for anidados, al primero lo llamaremos loopI y al otro loopJ. Tras pasar el profesor por nuestro puesto y solucionarnos algunas dudas que nos habían surgido conseguimos que se recorran las funciones correctamente. Lamentablemente la sesión de laboratorio llega a su fin y debemos de pausar nuestro trabajo.

Ya en nuestras casas y mediante videoconferencia retomamos el trabajo donde lo habíamos dejado en clase.

Nos damos cuenta de que nuestro programa no realiza el cálculo correctamente, ya que nos aparecen los valores correctos multiplicados por 4.

Tras revisar nuestro código nos damos cuenta de que no estábamos almacenando $t2 y $t3. Almacenamos mediante la instrucción lw dichos

registros en $t6 y $t7 respectivamente. Ahora si nuestro programa realiza la operación correctamente para matrices de 4*4.

28/09/18

Nos ponemos ahora con la función que imprimirá la matriz, a la que llamamos funcionB en previsión de que la que pida la matriz por pantalla sea la A.

Creamos dos cadenas ascii para imprimir un espacio y una tabulación para que quede mejor la matriz al imprimir.

Creamos dos bucles que iterarán hasta el tamaño de matriz que tengamos introducido en $a2, así podemos cambiar el tamaño más fácilmente cambiando solo un registro. Cogemos el código de la función calcula para calcular la posición a imprimir y lo introducimos en el bucle. Añadimos bifurcaciones para que imprima tabulaciones o saltos de línea cuando toque.

Al probarlo nos imprimía la matriz correctamente tabulada pero solo con 0s. Después de revisar el código, nos dimos cuenta que estábamos imprimiendo B, que estaba vacía ya que solo ejecutábamos la función de imprimir y no la de calcular. Al añadirla al bucle, ya funcionaba correctamente e imprimía los valores adecuados.

29/09/18

Nos disponemos a hacer la función de pedir los datos por pantalla. Después de echar un ojo a la referencia de mips conseguimos que almacene el dato introducido en la matriz.

Ahora creamos otro bucle y volvemos a copiar el codigo para calcular la posición en la que introducir el dato. Después de un par de errores por repetir registros en varias funciones, vemos que funciona correctamente.

Decidimos poner un poco más bonito la parte de introducción de datos y añadimos otras cadenas ascii. Ahora al pedirte los datos te mostrará un mensaje y la posición en la que introduces los datos; así será más fácil saber donde pones cada número.

Tercera Práctica 03/10/18

Ya que en el segundo ejercicio vamos a emplear la función del primero decidimos comentar nuestro trabajo a partir del segundo ejercicio, que incluye todo lo realizado en el primero.

Primero vamos a solicitar al usuario que introduzca un número en decimal, que como se indica en el guión de la práctica almacenaremos en $a1 y después guardamos el valor de la máscara que utilizaremos en la función con el valor 0xF0000000. También guardamos un 0 con la instrucción lw y una X con la instrucción lb ya que los número en hexadecimal vienen siempre precedidos por 0x.

Después realizamos un bucle que se repite tantas veces como dígitos tiene el número en hexadecimal (8 en concreto) dentro del bucle se realiza la operación and con la máscara para poder obtener los 4 bits más significativos y después son desplazados 28 posiciones a la derecha.Comprobamos después, si ese número es mayor
o igual a 10, de ser así se le suma 55 de esta manera, mediante el código Ascii somos capaces de representar las letras de la A a la F. Sino, se le suma 48, para poder representar los dígitos del 0 al 9.

Guardamos ahora mediante la instrucción lb el número o la letra obtenido e incrementamos la dirección de guardado. Después desplazamos hacia la izquierda 4 posiciones para poder coger los siguientes bits en la siguiente iteración. Cuando se termina el bucle se añade el fin de carrera al número en formato hexadecimal.

Tras salir de la función se imprime el número mediante la instrucción
li $v0, 4

Como se indica en el guión vamos ahora a probar con los diferentes datos que nos indican, tras realizar las pruebas y comprobar los resultados con la ayuda de un conversor de decimal a hexadecimal, podemos comprobar que nuestro programa funciona correctamente.

06/10/18

Vamos a modificar nuestro programa para que ahora el número mostrado en pantalla sea el resultado de multiplicar el número introducido por el usuario por 4 y convertido a formato hexadecimal. Decidimos utilizar la función del apartado 1 y simplemente meter una multiplicación por 4 dentro de la propia función.

Para ello y tras corregir varios fallos, finalmente añadimos en nuestra función la instrucción sll $t1,$t1,2 de esta manera desplazando a la izquierda dos posiciones multiplicamos por 4 el número introducido por el usuario.

Siguiendo con el guión vamos ahora a probar los números del apartado 2b observamos que con los 3 últimos números, tras realizar la multiplicación por 4, se obtiene un número que no puede ser representado en hexadecimal con 8 dígitos. En principio, el resto de números en principio correctos .

Ejercicio 3

Continuando con el guión de la práctica vamos ahora a activar las opciones correspondientes en Mars.

Tras activar las diferentes opciones que se especifican en el guión vemos como Mars nos muestra la ruta de datos. En ella aparecen varios colores. Rosa op code, verde rs, azul rt, azul clarito desplazamiento (aunque también el desplazamiento va por otro cable al ALU Control), amarillo PC y el 4 que se suma al PC, en naranja rd, en gris y rojo las señales de control dependiendo de si están en desactivadas o activadas respectivamente, en un marrón anaranjado el cable que va por el sign extended al shift left y al add, también va a la ALU si es una instrucción donde la ALU necesite el desplazamiento. Por arriba tenemos varios cables que se utilizan para determinar si es una instrucción jump o de bifurcación y el cable marrón oscuro es el que lleva el PC+4. Por último el cable azul oscuro de abajo que sale de la ALU y si no es una instrucción lw o sw pasa por el multiplexor y llega al write data del banco de registros, sino es un cable negro que sale de la memoria de datos y llega al mismo sitio pero no escribe nada en el banco de registros

Los caminos críticos son:

R: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux
lw: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux + Memoria de datos
sw: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux
Bifurcación: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux

Después de ver todos los caminos críticos, no se nos ha ocurrido ninguna representación más óptima.

Cuarta Práctica 10/10/18

Ejercicio 1

Primero pedimos al usuario que introduzca un número en hexadecimal, este número estará compuesto como máximo de 8 dígitos y lo guardamos en $a0. En $a1 reservamos espacio para 12 dígitos, por si acaso el usuario se equivoca al introducir los valores. Después llamamos a nuestra función (hexadecimalABinario) que primero almacena la cadena en $t0 para poder trabajar con ella. Decidimos trabajar directamente pidiendo el número por teclado y con toda las opciones (que las letras puedan ser mayúsculas o minúsculas) porque nos parece más cómodo realizar todas las comprobaciones desde el principio.

Como en ascii las letras mayúsculas van antes que las minúsculas creamos un bucle que será el que haga el trabajo de identificar el carácter introducido. Inicialmente, hacemos dos comprobaciones para ir reduciendo el rango: si es menor que 48 (0 en ascii) o si es mayor que 102 (f en ascii). Ahora para saber si es un número, miramos a ver si es menor que 58 (9 en ascii). A continuación descartamos que esté entre el 0 y la A (65 en ascii). Seguimos comprobando si es una de las letras mayúsculas hasta la F (70 en ascii). Ahora comprobamos si es un número erróneo entre la F y la a (97 en ascii). Finalmente, si no es ninguno de estos casos es que es una letra mayúscula y la restamos 87 para conocer su valor. Si en la comprobación hemos visto que es un número le restamos 48 para conocer su valor y si era una letra minúscula le restamos 55.

Ejercicio 2

a)
Tras un par de errores, el programa funciona perfectamente

b)
000000ff-> 255
0000ffff-> 65535
ffffffff-> -1
7fffffff-> 2147483647
80000000-> -2147483648
Comprobamos los números con la calculadora y vemos que funciona correctamente.

c)
Lo hacíamos desde el principio y funciona correctamente

d)
Añadimos una instrucción srl después de la ejecución de la función y nuestro programa funciona correctamente.

Ejercicios 3 y 4
Ya hacíamos las dos cosas desde el principio (nos fijamos en el valor de $v1) y funcionan correctamente.

13/10/18

Ejercicio 5:
Al probar las mayúsculas en este apartado, nos damos cuenta que antes habíamos puesto mal un valor límite y fallaba con la F mayúscula. Arreglamos este despiste y probamos con los valores que pide el ejercicio:

000000FF->255
0000FFff->65535
fffffffg->Error, g no es hexadecimal
Affffffff->-1342177281
80000000->-2147483648
FFfffffff0->Error, se sale de rango
Affffffg->Error, g no es hexadecimal.

Ejercicio 6:
Añadimos dos mensajes que se imprimirán cuando la cadena sea incorrecta y cuando la cadena sea demasiado larga dependiendo del código de error que devolvemos de anteriores apartados.

Ejercicio 7:
Empleamos parte del código de nuestra práctica anterior y lo introducimos en una función que la llamaremos binarioAHexadecimal. Primero convertimos el número introducido a binario y lo dividimos entre 4 y si no hay mensaje de error en $v1 continúa ejecutando binarioAHexadecimal. Si ha habido un error, muestra uno de los mensajes de error anteriores, volviendo a solicitar la cadena. Si todo ha ido bien, imprime el número introducido dividido por 4 por pantalla.

Quinta Práctica 17/10/18

Ejercicios 1 y 2:

a)
Empezamos directamente haciendo que el programa recoja el número por pantalla. Creamos una función llamada decimalABinario dicha función empieza con un bucle que cuenta cuántas cifras tiene el número contando los caracteres hasta que encuentra el \0. También comprobamos si el número tiene el signo menos delante. Para ello, comprobamos si el 45 en ascii está presente, si lo está sumamos uno a un registro para utilizar de flag al final de la función. Ahora para empezar a recoger el número decrementamos puntero y contador para no tener en cuenta el \0. En otro bucle, vamos sumando el número multiplicándole por la potencia de 10 que le corresponde y aumentando dicha potencia de 10 dependiendo de la posición. Y para acabar, si había un signo menos hacemos 0-numero para que sea negativo y si no lo era lo devolvemos tal cual.

b)
25-> 25
-048-> -48
F024->22024 este valor no se corresponde con el número decimal correspondiente, ya que nuestra función está tomando el valor del caracter f en ascii para realizar los cálculos pertinentes

- 5-> 155 este valor no se corresponde con el número decimal correspondiente, ya que nuestra función está tomando el valor del espacio en ascii para realizar los cálculos pertinentes
2147483647-> 2147483647
-5000000000-> El programa no responde (el número es muy grande)
-2147483648-> El programa no responde (el número es muy grande)

20/10/2018

c)
Para realizar este apartado utilizamos la función del guión anterior. Al tratar de emplear esta función nos aparecía un error de desbordamiento se nos había olvidado reservar espacio. Para solucionar este despiste añadimos la instrucción la $a1,A. También corregimos un fallo en una instrucción habíamos confundido un 1 con un 4 y nuestro programa funciona correctamente

Ejercicio 3:

Vamos ahora a modificar nuestro programa para ello creamos varias funciones que detectarán si no se ha introducido nada, si se ha introducido un caracter invalido o si el número introducido es demasiado grande. Para esto en nuestra función decimaABinario debemos realizar las diferentes comprobaciones. Al probar nuestro programa nos damos cuenta de que no funciona correctamente ya que cuando introducimos un número demasiado grande se produce un desbordamiento provocando la detención del programa. Para solucionar este problema limitamos nuestro programa para que solo permita la introducción de números de 9 cifras ya que uno de 10 cifras produciría desbordamiento.

Ejercicio 4:

Siguiendo el guión de la práctica incorporamos ahora nuestra función que detecta los diferentes errores a nuestro código del ejercicio 2c y ejecutamos nuestro programa. El resultado no es el esperado, pero tras solucionar unos despistes y volver a ejecutar nuestro código funciona correctamente.
b)
25-> 00000019
-048-> FFFFFFD0
F024-> Mensaje de caracter incorrecto (hemos introducido una f)

- 5-> FFFFFFFB
2147483647-> Número demasiado grande
-5000000000-> Número demasiado grande
-2147483648->

Sexta Práctica 24/10/18

Ejercicio 1 y 2

Para comenzar tendremos nuestro número en $a1 y en $a0 nuestra dirección de memoria, como se indica en la práctica. Realizamos ahora la llamada a nuestra función binarioADecimal. Le sumamos a el registro donde almacenamos nuestro número, 98 para guardar el número en formato Ascii pero en el final de la zona de memoria. Después creamos un bucle para ir dividiendo el número entre 10 y guardamos el cociente en $t0 y el resto en $t3 al que le sumamos 48 para pasarlo a ascii, lo guardamos en memoria y disminuimos en una unidad el puntero para apuntar a la dirección de memoria anterior. Al final de nuestra función ya vamos guardando el número en el orden correcto hasta que encontramos el /0.

27/10/2018

Nos ayudamos ahora de la función que habíamos creado en la práctica anterior y de esta manera, ya tenemos implementado la validación de la entrada.

Realizamos las diferentes pruebas con los datos del guión de prácticas. Tenemos problemas con los valores extremos, ya que nuestro programa no realiza bien la comprobación, tras solucionar un despiste en nuestro código, la comprobación se realiza correctamente. Hemos tenido que incluir en nuestro programa el valor límite.

Ejercicio 3

Para este ejercicio hemos empleado la función hexadecimalABinario que previamente habíamos creado en una de las prácticas anteriores, también realizamos las modificaciones oportunas para poder introducir dos números con su correspondiente validación de entrada.
Nuestro programa cogerá los dos números introducidos en hexadecimal y realizará la transformación de ambos números a su correspondiente

número en binario, una vez transformados los dos números, realizamos la suma en binario y después transformamos dicho número (el resultado) a su correspondiente número en decimal, mediante la función binarioADecimal para después imprimirlo por pantalla.

28/10/2018

Nos damos cuenta de que si introducimos los números en hexadecimal con ocho cifras nuestro programa no realiza la suma correctamente.
Habíamos olvidado cargar el número antes de realizar la llamada a nuestra función. Tras solucionar este despiste el programa realiza la suma correctamente.

Práctica Final

31/10/2018

Tras leer el guión del trabajo final y después de los consejos del profesor decidimos que vamos a tratar de dividir el trabajo en la validación de los datos, por una parte y por la otra, nos encargaremos de la realización de los cálculos para obtener el tiempo que ha pasado.

Vamos ahora a buscar información en internet sobre los diversos tipos de gestionar las fechas para así poder elegir uno de ellos. Tras ver varios métodos como el epoc de los sistemas Linux, finalmente decidimos que almacenaremos las fechas en días y el tiempo en segundos.

07/11/2018

Como hemos dividido el trabajo, a partir de ahora nuestro bitácora estará dividido en dos partes para cada fecha.

Por un lado nos hemos centrado en planificar todo lo relativo a la resta de las fechas, contaremos con dos funciones, ya que almacenaremos las fechas en días y el tiempo en segundos, a las que llamaremos restaDias y restaSegundos, en las que comprobamos si se pasan las fechas correctamente (primero la mayor de ellas) y realizamos la resta, de no ser así, llamaremos a sendas funciones de invertir que nos dejarán como primera fecha la mayor de las dos para así poder realizar la resta correctamente. Una vez realizada la resta tendremos otras dos funciones.

17/11/2018

Como ya habíamos comentado tenemos dos funciones que realizarán la transformación del resultado de la resta. Por una parte, tenemos la función convertirDias que transformará el resultado en días al correspondiente valor en años, meses y días. Por otro lado, tendremos la función convertirSegundos que transformará los segundos obtenidos en la resta a su equivalente en horas, minutos y segundos.

Para la próxima semana deberemos de crear una función que sume los días correspondientes si las horas superan las 24 así como otra que imprima los resultados.

18/11/2018

Tras comentarlo con algunos compañeros, decidimos añadir una ventana al introducir las fechas y al mostrar los errores, para que quede mejor que introducirlo por consola. Tras varias pruebas y búsqueda por internet encontramos los valores necesarios para el syscall de la ventana.

Para tratar la fecha, inicialmente vamos a considerar el formato dado: XX/XX/XXXX XX:XX:XX. Por ahora solo con barras en la fecha y con los dos puntos en la hora. Tras introducir la fecha llamaremos la función comprobarFecha para leer la cadena introducida y guardarla si es correcta o devolver un error si algo ha sido introducido mal. En ella lo que haremos ser recorrer la cadena buscando cifras y distinguiendo los espacios y los separadores.

21/11/2018

Siguiendo con las fechas creamos las diferentes funciones que se encargarán de, una vez realizada la resta, comprobar que los segundos y los minutos son menores de 60 y las horas menores de 24 sumando la cantidad que sea necesaria a la siguiente unidad para transformar los segundos que obtenemos en la resta a su equivalente en horas minutos y segundos.

Realizamos el mismo procedimiento para los días, que serán transformados en años meses y días. Como esta función ha de diferenciar entre los meses de 30 y 31 días debemos invertir todo el tiempo del que disponemos en su creación.

25/11/2018

Hemos introducido una función llamada espacios, para que el usuario pueda introducir los dígitos de la fecha con espacios entre ellos.
Si después de comprobar ambas fechas, no ha habido ningún error, continuamos con la ejecución normal del programa llamando a la

función que convierte las fechas en días y las horas en segundos. En esta parte, usaremos la función decimalABinario de las prácticas anteriores.

26/11/2018

Dentro de comprobarFecha llamaremos, aparte de a la función espacios, a dos funciones que se encargarán de introducir en un segundo vector la fecha correctamente. Estas funciones se llamarán esUnNumero y separador. La primera se encargará de introducir las cifras y la segunda de introducir la barra o los dos puntos dependiendo de si es fecha u hora.

28/11/2018

Tras algunos problemas, decidimos que la comprobación de que el número de días concuerde con el mes lo haremos en otra función. En compruebaFecha sólo habrá errores si hay caracteres no válidos, o más de 4 cifras en los años o más de 2 en el resto. Por ende, nuestro programa sólo funcionará desde el año 1 hasta el año 9999.

En lo que a la resta de fechas se refiere, como ya habíamos comentado, ya tenemos la resta realizada y el resultado transformado a los correspondientes años, meses, días, horas, minutos y segundos basándonos en la función binarioADecimal que habíamos creado en las prácticas guiadas. Ahora vamos a crear una función a la que llamaremos CompletarCadena que será la encargada de mostrar por ventana el resultado obtenido.
Primero planteamos realizar esta función mediante un bucle, pero luego descartamos la opción porque las cifras y la longitud del resultado varía en función de las fechas de entrada introducidas (a veces pueden ser 10 días, que tiene dos cifras o a veces pueden ser 3 días, que tiene una cifra). Tras comprobar dicha función y subsanar algún despiste, conseguimos que muestre el resultado correctamente.

30/11/2018

Llamaremos a la segunda función, que será la encargada de pasar la fecha a días y las horas a segundos convertirASegundos y tras la ejecución de la misma tendremos un vector en el que la posición 0 será la suma de días de la fecha introducida, la posición 4 la suma de

segundos de la hora introducido y luego en las tres posiciones siguientes tendremos el mes, el año y el dia. Esto es así, porque al principio pensábamos que con almacenar el mes era suficiente para realizar el cálculo pero luego nos dimos cuenta de que era necesario también el año y el día.

En esta función realizaremos la comprobación de que el número de días concuerde con el mes. Como empezamos a comprobar por días, no podremos saber que el 29 de Febrero es una fecha válida, asique esa fecha la consideraremos no válida. Para comprobar esto llamaremos a una función llamada comprobarDiaMes que comprobará si el día es correcto con un vector que tiene almacenados los días de cada mes, comprobando que el número de días es igual o menor al del mes. Para comprobar las horas y los minutos/segundos simplemente miramos si es mayor que 23 o 59 respectivamente.

En la parte de la resta de fechas, no habíamos tenido en cuenta los años bisiestos, como tenemos tiempo y ya hemos completado la parte básica de la práctica decidimos tratar de implementarlo.

Buscamos en internet el algoritmo para detectar años bisiestos que es el siguiente:
si el año es divisible entre 4, pero no lo es entre 100 o bien si el año es divisible entre 400.

En nuestra función, aplicamos el algoritmo anteriormente citado y añadimos un día más por cada vez que se cumpla.
Tras implementar todo lo dicho, nos damos cuenta de que no estábamos teniendo en cuenta la fecha exacta, ya que solo nos fijábamos en el año.

01/12/2018

Una vez comprobado todo, ahora vamos a calcular los díias y segundos.
Para calcular los días, seguimos esta fórmula:
(año-1)x365+mes_en_dias-dias_del_mes+dia = fecha en dias.
Para calcular los segundos, simplemente multiplicamos por 60 los minutos y luego por 60 otra vez las horas. Todo esto lo hacemos en una función llamada multiplicar y al salir de ella si estamos calculando días guardamos el resultado en la posición 0 del vector y si estamos

calculado segundos en la 2º. Todo esto estará dentro de una función llamada multiplicar a la que llamaremos una vez comprobado que cada cifra es correcta.

Hoy vamos a tratar de arreglar nuestra función que detecta los años bisiestos y agrega un día más al resultado por cada año que pase.

Tras varios intentos fallidos de diferenciar las fechas, nos damos cuenta de que lo mejor es conocer cuál es la fecha mayor y cual es la menor. Después de realizar esta comprobación, también debemos tener en cuenta que aunque el año sea bisiesto, la fecha puede ser anterior al 29 de febrero de este año, por lo tanto no habría que agregar un día más al resultado (lo denominamos un año “bisiestoMalo”) y que dicho problema solo lo tenemos con los años que son introducidos por el usuario, ya que los bisiestos que tenemos entre medias de las dos fechas introducidas si que son años completos y por lo tanto ha de sumarse un día más por cada uno de ellos.

Finalmente, decidimos permitir la introducción del día 29 de febrero. El cálculo de los días que han pasado se realiza correctamente, pero debemos comentar que al realizar la transformación del resultado obtenido de la resta, a formato años, meses, días, horas, minutos y segundos, detectamos un error, ya que seguimos considerando que ese mes tiene 28 días.