Proyecto de Programación en Ensamblador
Estructura de Computadores
Grado en Ingeniería Informática

Departamento de Arquitectura y Tecnología de Sistemas Informáticos

2019-2020. Primer semestre. Convocatorias de febrero y julio.

Introducción

El principal objetivo del proyecto de programación en ensamblador es que el estudiante pueda poner en práctica el conocimiento en profundidad de las posibilidades y el modo de funcionamiento de un procesador convencional de propósito general, utilizando para ello su lenguaje de programación nativo. Todo esto implica afrontar el diseño de programas, su desarrollo y, como parte fundamental, su depuración.

El trabajo se realizará en lenguaje ensamblador del Motorola 88110. Se trata de uno de los primeros procesadores RISC comerciales, lo que hace que sea lo suficientemente sencillo como para permitir que se desarrolle un proyecto interesante –al nivel que permite el conocimiento de un procesador al comenzar la asignatura "Estructura de computadores"– y, al mismo tiempo, suficientemente completo como para adquirir los conceptos básicos que se pretende que el estudiante alcance con su realización, que principalmente son los mencionados en el párrafo anterior.

El trabajo será sencillo para el estudiante si desde el primer momento ha seguido con normalidad las clases de teoría de la asignatura y ha realizado por su cuenta los problemas de programación en ensamblador propuestos en clase. Por el contrario, el arranque en el proyecto podría resultarle muy costoso si parte de una situación de desconocimiento del tema explicado en las clases de la asignatura.

El proyecto consiste en la programación de un conjunto de rutinas que realizan el filtrado de una imagen mediante un filtro programable.

La imagen es una matriz de píxeles, cada uno de los cuales se representa mediante un byte sin signo que especifica su nivel de gris (0 equivale a negro y 255 a blanco). El filtro está basado en una operación recursiva de convolución con un núcleo representado por una matriz cuadrada, de dimensión 3, formada por números fraccionarios.

AVISO –2019/2020–

El enunciado de este proyecto **está basado en el planteado en cursos anteriores**, pero difiere en algunas de las subrutinas que se deben desarrollar así como en la descripción de ciertas características o requisitos. Se recomienda que observe con especial atención los apartados de este documento que describen las normas de entrega y evaluación.

Aquellos estudiantes que tengan que repetir o corregir el proyecto que ellos mismos desarrollaron en una convocatoria previa, podrán partir de los programas que ya realizaron anteriormente. Sin embargo, algunas o todas las pruebas establecidas serán diferentes y sus resultados no coincidirán con los de otros cursos, por lo que deberán adaptar la implementación de las subrutinas, de modo que *superen las pruebas que se establezcan. Estas pruebas podrán ser diferentes en las convocatorias de febrero y julio; también podrán ser modificadas dentro de la convocatoria si se considera necesario.* El hecho de plantear un proyecto que coincide en buena parte con el del curso anterior tiene, además, las siguientes implicaciones:

- Los alumnos que hayan formado parte de un grupo de proyecto en alguna convocatoria anterior y que hubieran realizado al menos una entrega **solo podrán establecer**

grupo con el mismo compañero de dicha convocatoria o, alternativamente, realizar el proyecto **de forma individual**.

- Se realizará una revisión minuciosa de los proyectos realizados en este semestre para descartar o localizar posibles **casos de copia** que desafortunadamente se siguen produciendo (y detectando) en la mayor parte de las convocatorias.

En el proyecto se programará en ensamblador una serie de subrutinas que permitan aplicar un filtro recursivo y configurable a una imagen de entrada, obteniendo como resultado una imagen modificada de acuerdo a la especificación del filtro. El hecho de haber partido la tarea de desarrollo en subrutinas elementales tiene el objetivo de facilitar el trabajo de depuración.

Transformación de una imagen mediante la aplicación de un filtro genérico

El filtrado de una imagen consiste en realizar una serie de operaciones sobre cada uno de los píxeles que la componen. En el proyecto se trabaja con imágenes en escala de grises en las que cada píxel corresponde a un byte sin signo que indica su luminosidad. El filtro empleado pertenece a la categoría de los filtros lineales FIR ("finite impulse response") al que se han incorporado algunas variaciones para facilitar que el estudiante profundice en un concepto importante de la programación como es la *recursividad*. A nivel de programación en ensamblador, ello incidirá en la gestión de la pila y en el conocimiento del soporte que se ofrece a los lenguajes de alto nivel para implementar la recursividad. El filtro se basa en la convolución discreta de dos matrices: la que representa la imagen a filtrar, formada por tantos elementos como píxeles tiene la imagen, con la que define el filtro (también llamada "núcleo"del filtro), que normalmente es una matriz cuadrada de pequeño tamaño. De hecho, en el presente proyecto, el núcleo del filtro es una matriz de tamaño fijado en 3x3 elementos.

El modo de aplicación del filtro 3x3 consiste en efectuar, sobre cada píxel de la matriz que representa la imagen, ciertas operaciones en las que interviene el valor de dicho píxel, el de los ocho vecinos que rodean a este y el de los nueve elementos del núcleo del filtro.

Puesto que se trabaja con filtros cuyos valores numéricos pueden ser tanto positivos como negativos y no necesariamente enteros, es posible que la aplicación de un filtro produzca como resultado un valor no entero o que esté fuera del rango de valores que se pueden representar mediante un byte sin signo (de 0 a 255). En estos casos, el valor final se aproximará truncando ese resultado y, si es negativo, se ajustará a 0 y, si es positivo y mayor que 255, se limitará a 255. Además, puesto que los píxeles de las filas y de las columnas pertenecientes a los bordes de la imagen tienen un número de vecinos inferior a ocho, no se aplicará la operación de filtrado a dichos píxeles.

Denominando $f_{i,j}$ al elemento situado en la fila i y en la columna j de la matriz que representa el núcleo del filtro, $m_{i,j}$ al elemento de la fila i y la columna j de la matriz que representa la imagen y $r_{i,j}$ al correspondiente elemento en la imagen filtrada, la operación de filtrado es la siguiente:

$$\begin{aligned} r_{i,j} &= f_{0,0} \times m_{i-1,j-1} + f_{0,1} \times m_{i-1,j} + f_{0,2} \times m_{i-1,j+1} + \\ &+ f_{1,0} \times m_{i,j-1} + f_{1,1} \times m_{i,j} + f_{1,2} \times m_{i,j+1} + \\ &+ f_{2,0} \times m_{i+1,j-1} + f_{2,1} \times m_{i+1,j} + f_{2,2} \times m_{i+1,j+1} \end{aligned}$$

Por ejemplo, en el caso representado en la Figura 1, el nuevo valor del píxel $2,2$ de la imagen $im_{original}$ ($m_{2,2} = 1$) tras aplicar el filtro $F1$ será $r_{2,2} = 4$:

$$\begin{aligned} r_{2,2} &= f_{0,0} \times m_{1,1} + f_{0,1} \times m_{1,2} + f_{0,2} \times m_{1,3} + f_{1,0} \times m_{2,1} + f_{1,1} \times m_{2,2} + f_{1,2} \times m_{2,3} + \\ &\quad + f_{2,0} \times m_{3,1} + f_{2,1} \times m_{3,2} + f_{2,2} \times m_{3,3} = \\ &= 6 \times 0,125 + 7 \times 0,125 + 8 \times 0,125 + 0 \times 0,125 + 1 \times 0,0 + 2 \times 0,125 + \\ &\quad + 4 \times 0,125 + 5 \times 0,125 + 6 \times 0,125 = 4 \quad (\textit{tras el ajuste de } 4,75) \end{aligned}$$

Para obtener esta expresión se ha utilizado una matriz de filtro cuyo elemento central tiene valor 0 y los 8 restantes, es decir, todos los pertenecientes al borde tienen valor $1/8$.

En el caso de la implementación solicitada en este proyecto, el filtro a utilizar está formado por una matriz de 3x3 coeficientes fraccionarios, cada uno de ellos definido por dos números enteros, el numerador y el denominador de dicha fracción. Por consiguiente, también puede interpretarse que la matriz de filtro es una matriz de 3x6 elementos enteros, en la que los elementos que ocupan posiciones pares (comenzando por la posición 0) corresponden al numerador y el elemento que sigue a un numerador es el denominador de esa misma fracción.

En el ejemplo de la Figura 1, elfiltro estaría definido por la matriz:
$MFiltro = \{1,8,1,8,1,8, \quad 1,8,0,8,1,8, \quad 1,8,1,8,1,8\}$, equivalente a
$MFiltro = \{1/8,1/8,1/8, \quad 1/8,0/8,1/8, \quad 1/8,1/8,1/8\}$

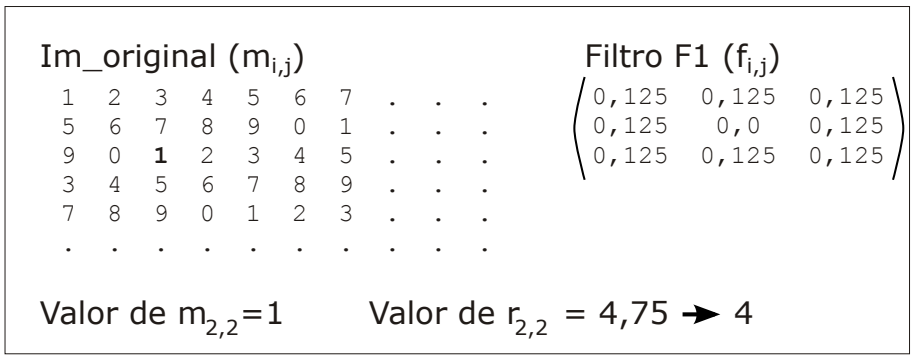

Figura 1. Aplicación de un filtro de imagen.

Según se señalaba anteriormente, el filtro de imagen presenta una modificación respecto al uso más frecuente de este tipo de filtros: consiste en que su aplicación será *recursiva*, de modo que hasta que no se cumpla alguna condición de finalización, se volverá a aplicar otro filtrado. En cuanto a las condiciones de continuación o terminación de la recursividad, se aplicará un nuevo filtrado siempre que se cumpla lo siguiente:

- En la última aplicación del filtro y de acuerdo a cierta medida, que es específica de este proyecto (definida más adelante), la imagen resultante difiere en una cierta magnitud de la imagen original.
- El número de veces que se ha aplicado el filtro es inferior a un máximo determinado (definido mediante otra subrutina).

En la descripción de las subrutinas que componen el proyecto quedarán aclarados los aspectos concretos de implementación.

Estructura del proyecto

El proyecto estará compuesto por ocho subrutinas que se relacionan tal como se indica en la Figura 2. Además, el alumno o grupo construirá un programa principal (PPAL en la figura) para cada una de las pruebas que sea necesario efectuar durante la implementación y depuración de las subrutinas del proyecto. Se ha desglosado el programa de filtrado en un número alto de subrutinas para facilitar al alumno su depuración, ya que así se enfrentará a fragmentos de código de tamaño manejable. Además, al tener el programa segmentado en varias partes, el sistema automático de corrección proporcionará información más precisa sobre las partes que se han implementado correctamente y las que necesitan revisión.

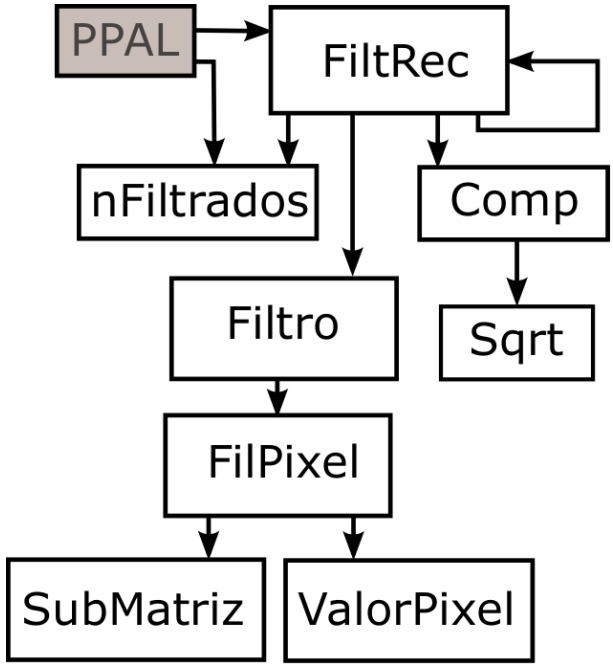

Figura 2. Jerarquía de las rutinas del proyecto.

Tipos de datos

En el proyecto se usan datos de diferentes tipos y tamaños:
Byte sin signo. Se usan para representar los píxeles de las imágenes.
Entero sin signo. Se usan para representar enteros positivos, por ejemplo M y N (el número de filas y de columnas de una imagen) así como las direcciones de memoria. Su longitud es 32 bits (4 bytes).
Entero con signo. Se usan para representar datos que pueden tomar valores enteros positivos o negativos, por ejemplo los coeficientes con los que se construye la matriz de filtro. Su longitud es 4 bytes.

En relación con otros tipos de datos, compuestos a partir de los anteriores, además de la matriz de filtro formada por 3×6 números enteros con signo que ya se ha descrito, se trabaja con imágenes, cuya representación está formada en este proyecto por M y N, dos enteros sin signo que definen su tamaño (número de filas, M, y número de columnas, N),

seguidos por $M \times N$ bytes sin signo que definen el nivel de gris de cada uno de los píxeles que forman la imagen.

Si bien en el caso de la matriz de filtro se trabaja con fracciones, todos los datos utilizados en este proyecto son de tipos enteros, en ningún caso se emplean representaciones fraccionarias o de coma flotante. Eso implica que cada vez que se realiza una operación entera (en particular una división), el resultado perderá su parte fraccionaria, lo que podría representar un error de redondeo muy significativo. Se indicará más adelante cómo evitar este potencial problema.

Programa Principal

El programa principal se encargará de inicializar la pila de usuario, almacenar en ella los parámetros que se deben pasar, e invocar a las distintas rutinas objetivo de este proyecto.

Los parámetros de las rutinas se pasarán siempre en la pila salvo que se especifique lo contrario. Los parámetros que se pueden representar mediante una palabra, 32 bits, se pasarán por valor. Los parámetros que ocupen más de 32 bits (vectores o matrices) se pasarán por dirección. El resultado se recogerá normalmente en el registro $r29$, salvo que se especifique de otro modo.

Raíz Cuadrada

```
rc = Sqrt ( Num )
```

Parámetros:

- **Num:** Es un parámetro de entrada que contiene el número del que se quiere encontrar la raíz cuadrada. Es un entero positivo que se pasa por valor y ocupa 4 bytes.

Valor de retorno:

- **rc:** La función devuelve una aproximación entera a la raíz cuadrada del número que se pasa como parámetro. Es un número entero positivo o nulo.

Descripción:

La rutina `Sqrt` realiza el cálculo de la raíz cuadrada entera de un número positivo o nulo. Utiliza el *algoritmo babilónico*, que se basa en encontrar el lado de un cuadrado de área conocida mediante aproximaciones sucesivas a la longitud de sus lados.

El funcionamiento de esta rutina será el siguiente:

1. Si el número contenido en el parámetro de entrada es menor que dos, se devolverá como valor de retorno ese mismo número en `r29`, devolviendo el control al programa llamante.

2. Definirá una pareja de números a y b que serán las aproximaciones a la raíz cuadrada. Inicialmente se asignará a a el parámetro de entrada `Num` y a b el valor 1.

3. Recorrerá un bucle en el que realizará, en cada iteración, las operaciones siguientes:

   a) Sustituirá *a* por *(a+b)/2* y *b* por `Num/a`.

   b) Si *b* es mayor que *a*, se intercambiarán ambos valores.

   c) Si la diferencia *a-b* es menor o igual a 1, se dará por finalizado el bucle. En caso contrario se procederá con la siguiente iteración.

4. Devolverá el control al programa llamante, dando como resultado en `r29` el último valor del número *b*.

Número de filtrados

```
NFiltrados = nFiltrados ( oper )
```

Parámetros:

- **oper:** Es un parámetro de entrada que se pasa por valor. Es un número entero (una palabra) que especifica la operación que debe realizar esta subrutina con la variable estática nF. Si el parámetro *oper* es mayor o igual a 0, la operación consistirá en inicializar nF asignándole el valor de *oper*. Si el parámetro *oper* es menor que cero, la operación consistirá en decrementar en una unidad el valor de nF limitando el resultado a cero: si al decrementar se alcanza un valor de nF menor que cero, este resultado se sustituirá por cero.

Valor de retorno:

- **NFiltrados:** La función devuelve en el registro $r29$ el valor de la variable nF.

Descripción:

La rutina `nFiltrados` está encargada de manejar la variable estática nF[1], siendo la única vía de acceso en todo el código a dicha variable. El resto de subrutinas (en realidad únicamente $FiltRec$) no debe acceder directamente a nF. Esta variable nF **estará ubicada obligatoriamente en la dirección 0 de la memoria principal** y será un entero sin signo de 32 bits.

Si el parámetro *oper* con el que se llama a esta función tiene valor 0 o mayor que 0, entonces la variable estática se inicializará a ese mismo valor ($nF = oper$). Por el contrario, si el parámetro es menor que cero, la función realizada será la de decrementar la variable estática ($nF \leftarrow (nF - 1)$) y después ajustar a cero en caso de que el resultado obtenido sea negativo.

La forma correcta de definir el máximo número de filtrados que se puede realizar de forma recursiva será mediante una llamada desde el programa principal a `nFiltrados` en la que se pase como parámetro el valor requerido para dicho máximo.

[1] Esta variable será utilizada a través de $nFiltrados$ por la subrutina $FiltRec$, y representará el número de veces que todavía se puede filtrar una imagen

Compara imágenes

```
Diferencia = Comp ( Imagen1, Imagen2 )
```

Parámetros:

- **Imagen1:** Es la matriz que contiene la primera de las imágenes que se han de comparar. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos:
  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (MxN bytes, MxN/4 palabras).
- **Imagen2:** Es la matriz que contiene la segunda de las imágenes que se han de comparar. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos con idéntica estructura que `Imagen1`.

Valor de retorno:

- **Diferencia:** La función devuelve una *medida* de la diferencia entre las dos imágenes que se han pasado como parámetro. Esta diferencia es la raíz cuadrada de la suma de los cuadrados de las diferencias de todos los píxeles de la imagen.

Descripción:

La rutina `Comp` recorre los píxeles de las dos imágenes a las que hacen referencia los parámetros y va acumulando el cuadrado de la diferencia entre cada par de píxeles situados en la misma fila y columna de una y otra imagen. Finalmente obtiene y devuelve al llamante la raíz cuadrada del valor acumulado.

El funcionamiento de esta rutina será el siguiente:

1. Inicializará a 0 un acumulador de diferencias `Dif` (podrá alojarse en un registro).
2. Recorrerá los `MxN` elementos de cada matriz, sumando para cada uno de ellos al acumulador `Dif` el cuadrado del valor de la diferencia entre el píxel perteneciente a la primera imagen y el de la segunda.
3. Calculará la raíz cuadrada del acumulador `Dif`. Para ello, llamará a la subrutina `Sqrt`, pasándole como parámetro dicho acumulador `Dif`.
4. Devolverá el control al programa llamante, dando como resultado en `r29` el valor de retorno recogido de la subrutina `Sqrt` en el paso anterior.

Extracción de Submatriz

`SubMatriz ( Imagen, SubImg, i, j )`

Parámetros:

- **Imagen:** Es la matriz que contiene la imagen de la que se ha de extraer una submatriz 3×3. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos:
  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (en total MxN bytes $\simeq$ MxN/4 palabras).
- **SubImg:** Es la matriz de tamaño 3×3 en que la subrutina debe depositar la submatriz de la imagen de entrada que se determina tal como se especifica más adelante, en la descripción de esta subrutina. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de salida que quedará relleno con los nueve valores (bytes sin signo) que forman la submatriz resultado.
- **i:** Es el número de fila (comprendido entre 1 y M-2) del píxel de la imagen proporcionada como primer parámetro *Imagen* que será el píxel central de la submatriz resultado. Es un parámetro de entrada que se pasa por valor.
- **j:** Es el número de columna (comprendido entre 1 y N-2) del píxel de la imagen proporcionada como primer parámetro *Imagen* que será el píxel central de la submatriz resultado. Es un parámetro de entrada que se pasa por valor.

Descripción:

La rutina `SubMatriz` obtiene una subimagen de tamaño 3×3 obtenida de la imagen de entrada *Imagen* sin más que copiar de la misma nueve valores de píxeles que se determinan tal como queda descrito en la explicación del funcionamiento de esta subrutina.

El funcionamiento de esta rutina consistirá en rellenar la submatriz resultado con los siguientes nueve valores (I representa la Imagen de entrada):

```
I(i-1,j-1), I(i-1,j), I(i-1,j+1),
I(i,j-1), I(i,j), I(i,j+1),
I(i+1,j-1), I(i+1,j), I(i+1,j+1)
```

Valor del píxel filtrado

`VPixel = ValorPixel( SubImg, MFiltro )`

Parámetros:

- **SubImg:** Es la matriz de tamaño 3×3 que contiene el conjunto de 9 valores (bytes sin signo) sobre los que se ha de aplicar el filtro especificado por el segundo parámetro. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

- **MFiltro**. Es una matriz cuadrada de 3x3 fracciones que definen el filtro que se aplicará a la imagen. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes. Define la posición de memoria en que se encuentra la especificación del filtro, formada por 3x3 parejas de coeficientes (numerador, denominador), cada una de las cuales representa la fracción que se utiliza para actualizar la imagen. Cada uno de los dieciocho elementos con los que se forma el filtro es un entero con signo.

Valor de retorno:

- **VPixel:** La función devuelve en $r29$ una aproximación al valor que se asignará al píxel seleccionado una vez aplicado el filtro. El valor de retorno es un entero que puede tener cualquier valor positivo o negativo, es decir, no tiene por qué cumplir con los requisitos para ser considerado un pixel válido.

Descripción:

La rutina `ValorPixel` aplica el filtro $MFiltro$ que se pasa como segundo parámetro a la submatriz que se pasa en el primer parámetro. Dada la estructura de ambos parámetros, la aplicación del filtro se limita a realizar el producto escalar de $SubImg$ por $MFiltro$ considerado el primero como un vector de nueve elementos enteros y el segundo como un vector de nueve elementos fraccionarios. De forma detallada, consiste en acumular sobre un registro los productos parciales de cada elemento de la submatriz por el correspondiente elemento del filtro, considerando en cada caso el formato de representación de dichos elementos (pares de enteros con signo los de $Mfiltro$, bytes sin signo los de $Subimg$). Para evitar que al dividir el valor de cada píxel entre el denominador de cada coeficiente fraccionario de la matriz de filtro se pueda generar un importante error de redondeo, que se acumularía en el resultado del píxel filtrado, en la implementación de esta subrutina se aplica también una función de "amplificación", consistente en multiplicar el valor de cada píxel por una constante K (en este caso $K = 256$). Después, una vez que se han acumulado los productos parciales sobre el registro resultado, se divide el valor obtenido entre la misma constante K.

El funcionamiento de esta rutina será el siguiente:

1. Inicializará a 0 un acumulador entero de una palabra Acc (normalmente se utilizará un registro).

2. Asignará a un puntero la dirección del comienzo de la submatriz de entrada $SubImg$, es decir, la que apunta a su primer elemento, y otro puntero a la dirección donde está almacenada la primera de las nueve fracciones $MF_{Numerador}/MF_{Denominador}$ de la matriz de filtro (que se denominarán MF_N/MF_D).

3. Recorrerá en un bucle los nueve pares MF_N/MF_D de la matriz de filtro y los correspondientes nueve elementos de $SubImg$ previamente multiplicados por $K = 256$, obteniendo su producto y acumulandolo en Acc. En cada iteración de este bucle se efectuarán las siguientes operaciones:

   a) Se leerá sobre un registro general el byte sin signo que determina el nivel de intensidad del píxel seleccionado en la submatriz. Con esta lectura, el registro contendrá el mismo valor pero representado como un entero positivo de 32 bits.

   b) Se multiplicará el valor obtenido en el paso anterior por la constante $K = 256$.
   c) Se multiplicará el valor obtenido en el paso anterior por el correspondiente elemento del filtro MF_N y se dividirá entre MF_D, acumulando el resultado sobre *Acc*.

4. Dividirá el resultado parcial almacenado en *Acc* entre la constante $K = 256$, dejando el resultado obtenido en $r29$.

5. Devolverá el control al programa llamante.

Filtro de un píxel

```
VPixel = FilPixel ( Imagen, i, j, MFiltro )
```

Parámetros:

- **Imagen:** Es la matriz que contiene la imagen de entrada a uno de cuyos píxeles se ha de aplicar el filtro. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos:
  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (en total MxN bytes $\simeq$ MxN/4 palabras).
- **i:** Es el número de fila (comprendido entre 0 y M-1) del píxel de la imagen proporcionada como primer parámetro *Imagen* al que se quiere aplicar la máscara definida por $MFiltro$. Es un parámetro de entrada que se pasa por valor.
- **j:** Es el número de columna (comprendido entre 0 y N-1) del píxel de la imagen proporcionada como primer parámetro *Imagen* al que se quiere aplicar la máscara definida por $MFiltro$. Es un parámetro de entrada que se pasa por valor.
- **MFiltro**. Es una matriz cuadrada de 3x3 fracciones que definen el filtro que se aplicará a la imagen. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes. Define la posición de memoria en que se encuentra la especificación del filtro, formada por 3x3 parejas de coeficientes (numerador, denominador), cada una de las cuales representa la fracción que se utiliza para actualizar la imagen. Cada uno de los dieciocho elementos con los que se forma el filtro es un entero con signo.

Valor de retorno:

- **VPixel:** La función devuelve el valor que se ha de asignar al píxel (i, j) de la imagen filtrada. El valor de retorno es un entero sin signo que se devuelve en el registro $r29$.

Descripción:

La rutina `FilPixel` aplica la máscara de filtrado definida por la matriz $MFiltro$ al píxel seleccionado por los parámetros i y j de la imagen proporcionada como primer

parámetro, $Imagen$. La rutina devuelve como resultado el valor equivalente al mismo píxel en la imagen filtrada. No será necesario comprobar que los valores de i y j facilitados como parámetros hacen referencia a un elemento válido de la imagen, sino que se supondrá que el programa o subrutina llamante solo proporcionará valores adecuados para dichos parámetros.

El funcionamiento de esta rutina será el siguiente:

1. Comprobará si el píxel (i, j) que se desea filtrar pertenece o no al borde de la imagen. En caso afirmativo, se asignará el valor de dicho píxel al registro $r29$ y se devolverá el control al programa llamante.

2. Reservará espacio en el marco de pila para almacenar una submatriz cuadrada de tamaño 3×3, $SubImg$, construida por la subrutina `SubMatriz` a partir del píxel que se está filtrando. Dado que la submatriz ocupará 9 bytes, el espacio que se reservará será de 3 palabras (12 bytes), ya que es el menor número de palabras en que se pueden almacenar los 9 bytes de la submatriz.

3. Llamará a la subrutina `SubMatriz`. Esta obtendrá y dejará almacenada en el espacio reservado en el paso anterior la submatriz 3×3 $SubImg$ de la que tras aplicar el filtro definido por $MFiltro$ se obtendrá el nuevo valor del píxel que se está filtrando. Para realizar esta llamada se habrán preparado sus parámetros: la imagen original $Imagen$, la zona de memoria en que quedará la submatriz resultado $SubImg$ (zona que se reservó en el marco de pila en el paso anterior) y los valores de i y j.

4. Llamará a la subrutina `ValorPixel`. Esta obtendrá un valor numérico provisional para el píxel filtrado (VP), obtenido simplemente como el producto escalar de dos vectores.

5. A partir del valor VP retornado por `ValorPixel` se obtiene el valor correspondiente al pixel filtrado sin más que ajustar el resultado a los límites establecidos para un pixel (entre 0 y 255).

6. Devolverá el control al programa llamante, dando como resultado en $r29$ el valor obtenido en el paso anterior.

Filtro de imagen

`Filtro ( Imagen, ImFiltrada, MFiltro )`

Parámetros:

- **Imagen:** Es la matriz que contiene la imagen de entrada a la que se ha de aplicar el filtro. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos:
  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (en total MxN bytes $\simeq$ MxN/4 palabras).

- **ImFiltrada:** Es la matriz que contendrá el resultado de aplicar el filtro sobre la imagen de entrada. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de salida que tiene tres campos:
  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (MxN bytes, MxN/4 palabras).
- **MFiltro**. Es una matriz cuadrada de 3x3 fracciones que definen el filtro que se aplicará a la imagen. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes. Define la posición de memoria en que se encuentra la especificación del filtro, formada por 3x3 parejas de coeficientes (numerador, denominador), cada una de las cuales representa la fracción que se utiliza para actualizar la imagen. Cada uno de los dieciocho elementos con los que se forma el filtro es un entero con signo.

Descripción:

La rutina `Filtro` aplica la máscara de filtrado definida por la matriz $MFiltro$ a la imagen proporcionada como primer parámetro, $Imagen$, dejando el resultado en la imagen especificada como segundo parámetro $ImFiltrada$.

El funcionamiento de esta rutina será el siguiente:

1. Copiará las constantes que definen el número de filas M y el número de columnas N de la imagen original sobre la imagen filtrada $ImFiltrada$.
2. Desde la primera ($i = 0$) hasta la última fila ($i = (M - 1)$), de la matriz original, realizará las siguientes operaciones:
   a) Desde el primero ($j = 0$) al último ($j = (N - 1)$) píxel de la fila que se está procesando, se realizarán las siguientes operaciones:
   - Preparará los parámetros y realizará una llamada a la subrutina `FilPixel` para el elemento actual. Los parámetros que se proporcionan son los siguientes: la imagen original $Imagen$, el número de fila i, y de columna j, y el filtro, $MFiltro$.
   - Almacenará el valor de retorno recogido de la subrutina `FilPixel` en la posición que le corresponde (i, j) de la imagen filtrada $ImFiltrada$.

Filtro recursivo

```
Diferencia = FiltRec ( ImagenIn, ImagenOut, MFiltro, NCambios )
```

Parámetros:

- **ImagenIn:** Es la matriz que contiene la imagen de entrada, a la que se ha de aplicar el filtro. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de entrada que tiene tres campos:

  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (en total MxN bytes $\simeq$ MxN/4 palabras).

- **ImagenOut:** Es la matriz que contiene la imagen de salida, a la que se habrá aplicado el filtro una o varias veces de forma recursiva. Se pasa por dirección, por lo que ocupa 4 bytes. Es un parámetro de salida que tiene tres campos:

  - El número de filas, de tipo entero (M).
  - El número de columnas, de tipo entero (N).
  - Los elementos almacenados por filas, cada uno de tipo byte sin signo (en total MxN bytes $\simeq$ MxN/4 palabras).

- **MFiltro**. Es una matriz cuadrada de 3x3 fracciones que definen el filtro que se aplicará a la imagen. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes. Define la posición de memoria en que se encuentra la especificación del filtro, formada por 3x3 parejas de coeficientes (numerador, denominador), cada una de las cuales representa la fracción que se utiliza para actualizar la imagen. Cada uno de los dieciocho elementos con los que se forma el filtro es un entero con signo.

- **NCambios:** Es el valor mínimo de la diferencia entre la imagen original y la filtrada que determina la necesidad de realizar un nuevo filtrado (una nueva llamada recursiva). Es un parámetro de entrada de tipo entero, se pasa por valor y ocupa 4 bytes. La diferencia entre una imagen y otra se especifica en la descripción de la subrutina `Comp`, ya que coincide con su valor de retorno.

Valor de retorno:

- **Diferencia:** La función devuelve una medida de la diferencia entre la imagen de entrada y la imagen filtrada obtenida en la última llamada recursiva a esta subrutina, salvo que la subrutina haya finalizado por haber realizado ya el número máximo de filtrados (es decir, que nF haya llegado a 0), en cuyo caso devolverá -1. En caso de devolver una medida de la diferencia, esta será la proporcionada por la subrutina `Comp`. El valor de retorno es un entero con signo que se devuelve en el registro $r29$.

Descripción:

La rutina `FiltRec` realiza el filtrado **recursivo** de una imagen. La imagen de entrada está definida por el primer parámetro de la subrutina, que especifica el número de filas y columnas y los bytes que representan cada uno de sus píxeles. La labor básica de esta subrutina consiste en aplicar una función de filtrado a cada uno de los píxeles de la imagen (mediante la llamada a la subrutina `Filtro`) y, a continuación, comprobar si se ha alcanzado alguna de las condiciones de terminación de la recursividad, en cuyo caso se devuelve el control al llamante. La condición de salida de la recursividad es que se detecte cualquiera de las siguientes situaciones:

- El número de veces que se ha aplicado el filtro ha alcanzado el máximo permitido, condición que vendrá determinada porque la llamada a la subrutina `nFiltrados` devuelva un valor nulo.

- En la última aplicación del filtro y de acuerdo a la medida proporcionada por la subrutina `Comp`, la diferencia entre la matriz de entrada y la obtenida tras retornar de la subrutina `Filtro` sea menor que el valor especificado en el parámetro $NCambios$.

La implementación de esta subrutina debe realizarse **obligatoriamente** mediante el algoritmo recursivo aquí descrito. En caso de entregar una implementación no recursiva, el proyecto será evaluado como suspenso aún en el caso de que supere todas las pruebas del corrector automático.

El funcionamiento de la rutina debe ser el descrito a continuación:

1. Reservará espacio en el marco de pila para almacenar una imagen completa, $ImagenTMP$, del mismo tamaño (mismas dimensiones) que el ocupado por la imagen de entrada, $ImagenIn$. Para ello, reservará el mismo espacio que ocupa la imagen sin filtrar, pero ajustado por exceso para que sea un múltiplo de 4 bytes.

2. Llamará a la subrutina `Filtro`, que aplicará (una sola vez) el filtro definido por la matriz de coeficientes $MFiltro$. Para ello, `FiltRec` habrá preparado los parámetros de `Filtro`, que serán: la imagen de entrada $ImagenIn$, la dirección de la imagen resultante $ImagenOut$, y la matriz de coeficientes de filtrado, $MFiltro$.

3. Almacenará una copia de la imagen filtrada $ImagenOut$ sobre la zona reservada en el marco de pila en el paso 1, $ImagenTMP$. Esta copia de la imagen pasará a ser la imagen de entrada en la siguiente llamada recursiva.

4. Decrementará en una unidad la variable estática nF que mantiene el número de veces que aún se puede pasar el filtro, lo que hará mediante una llamada a `nFiltrados` en la que pasará como parámetro un valor negativo cualquiera (por ejemplo, -1). Si el valor retornado por `NFiltrados` en $r29$ es cero, se abandonará el proceso recursivo (por requerir un número demasiado alto de aplicaciones del filtro), asignará un valor -1 a $r29$ y continuará la ejecución en el paso 7 de este procedimiento.

5. Calculará la diferencia entre la imagen de entrada y la filtrada, para decidir si el proceso recursivo debe continuar o finalizar. Para ello, llamará a la subrutina `Comp`, pasándole como parámetros la imagen de entrada $ImagenIn$ y la filtrada $ImagenOut$. Si el valor retornado por `Comp` en $r29$ es menor que el especificado por el parámetro $NCambios$, se dará por terminado el proceso recursivo y la ejecución continuará en el paso 7.

6. Al no haberse alcanzado el final de la recursividad, llamará a la propia subrutina `FiltRec`, con los siguientes parámetros: la copia de la imagen obtenida tras el último filtrado, $ImagenTMP$, la imagen de salida $ImagenOut$, la matriz de filtro $MFiltro$, y el valor máximo de la diferencia entre imágenes para que finalice el proceso recursivo, $NCambios$.

7. Devolverá el control al programa llamante.

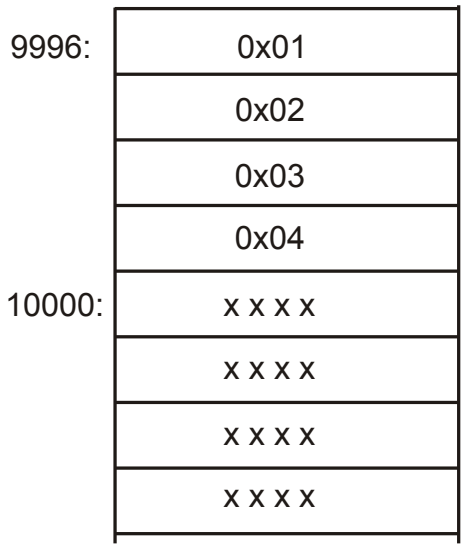

Figura 3. Operaciones PUSH y POP.

Creación de una pila de usuario

Debido a que el 88110 no dispone de un registro de propósito específico para la gestión de la pila, se asignará como puntero de pila uno de los registros de propósito general. Se utilizará el registro **r30** como puntero de pila. Éste apuntará a la cima de la pila, es decir, a la palabra que ocupa la cabecera de la pila (donde estará la última información introducida) y ésta crecerá hacia direcciones de memoria decrecientes.

A modo de ejemplo se muestran las operaciones elementales a realizar sobre la pila: PUSH y POP (véase la Figura 3).

Supongamos que el registro **r30** contiene el valor 10000 (decimal) y el registro **r2** contiene el valor hexadecimal 0x04030201. La operación PUSH, que introduce este registro en la pila, se implementa con la siguiente secuencia de instrucciones:

```
subu r30,r30,4
st r2,r30,0
```

quedando r30(SP) = 9996 y la pila como se indica en la Figura 3.

Supongamos que después de realizar la operación anterior se realiza una operación POP sobre la pila con el registro **r3** como operando. La secuencia de instrucciones resultante sería la siguiente:

```
ld r3,r30,0
addu r30,r30,4
```

quedando r3 = 0x04030201 y r30(SP) = 10000

Subrutinas anidadas, variables locales, paso de parámetros y uso de registros

Puesto que las instrucciones de salto con retorno que proporciona el 88110 (**jsr** y **bsr**) salvaguardan la dirección de retorno en el registro **r1**, hay que incluir un mecanismo que permita realizar llamadas anidadas a subrutinas. Por este motivo es necesario guardar el contenido de dicho registro en la pila. Este mecanismo sólo es estrictamente necesario cuando una subrutina realiza una llamada a otra, pero, para sistematizar la realización de este proyecto, se propone realizar siempre a la entrada de una subrutina la salvaguarda del registro **r1** en la pila mediante una operación PUSH.

El espacio asignado para variables locales se reserva en el marco de pila de la correspondiente rutina. Para construir dicho marco de pila basta con asignar a uno de los registros de la máquina el valor del puntero de pila **r30**, después de haber salvaguardado el valor que tuviera el registro que actúa como puntero al marco de pila del llamante. Para la realización del proyecto se utilizará el registro **r31**. Por tanto, las primeras instrucciones de una subrutina que desea activar un nuevo marco de pila serán las siguientes:

```
RUTINA:   subu r30,r30,4     ; Se realiza una operacion PUSH r1
          st r1,r30,0
          subu r30,r30,4     ; Se realiza una operacion PUSH r31
          st r31,r30,0
          addu r31,r30,r0    ; r31 <- r30
```

En este proyecto se utilizarán los dos métodos clásicos de paso de parámetros:

- **Paso por dirección:** Se pasa la dirección de memoria donde está contenido el valor del parámetro sobre el que la subrutina tiene que operar.
- **Paso por valor:** Se pasa el valor del parámetro sobre el que la subrutina tiene que operar.

El paso de parámetros a todas las subrutinas del proyecto se realizará utilizando la pila de la máquina salvo cuando se especifique de otro modo. El parámetro que queda en la cima de la pila es el primero de la lista de argumentos. Por ejemplo, para una invocación de la rutina `FiltRec` los parámetros en pila quedarían tal y como se especifica en la Figura 4, en la que se ilustra también la reserva de espacio en la pila que realiza `FiltRec` para almacenar una copia de la imagen filtrada. El comienzo de esta rutina quedaría como sigue:

```
FiltRec:
    PUSH  (r1)          ; Se guarda la direccion de retorno
    PUSH  (r31)         ; Se salva el FP del llamante
    addu  r31,r30,r0    ; r31<-r30    Activación puntero de marco de pila
    ld    r10, r31, 8   ; lee los parámetros de la subrutina
    . . . . .           ; . . . . . .
```

En este proyecto todos los parámetros de salida se pasan por dirección. Esto implica que las subrutinas no pueden modificar ninguno de los datos que reciben en la pila y deben

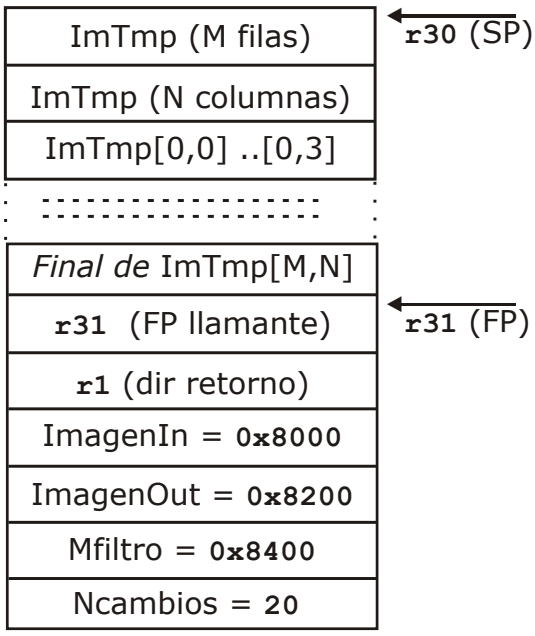

Figura 4. Estado de la pila durante la ejecución de la rutina `FiltRec`.

devolver el control al programa llamante dejando la pila en el mismo estado que tenía al comienzo de la subrutina.

Cada subrutina puede usar a su conveniencia todos los registros de propósito general, salvo $r1$, $r30$ y $r31$; no pueden presumir que tengan un valor determinado; deben asignarles un valor inicial antes de usarlos como operandos de entrada; y no pueden usarlos para devolver al programa llamante ningún tipo de información, salvo $r29$ y cuando así esté especificado en la descripción de la subrutina. Las subrutinas que llaman a una segunda subrutina deben salvar en la pila todos los datos que estén almacenados en registros y sean necesarios tras el retorno de dicha segunda subrutina, recuperando su valor de la pila antes de reanudar la ejecución. Por ejemplo, si una rutina necesita usar los valores almacenados en $r10$ y $r11$ tras llamar a la subrutina `SUBRUTINA` a la que se pasa como parámetros $r20$ y $r21$, una codificación correcta es la siguiente:

```
RUTINA:   PUSH  (r1)          ; Se guarda la direccion de retorno
          . . . . .           ; . . . . . .
          PUSH  (r10)         ; Se salva el valor de r10
          PUSH  (r11)         ; Se salva el valor de r11
          PUSH  (r21)         ; Se pasa como parámetro r21
          PUSH  (r20)         ; Se pasa como parámetro r20
          bsr   SUBRUTINA     ; Se llama a la subrutina
          POP   (r20)         ; Se recupera el parámetro r20
          POP   (r21)         ; Se recupera el parámetro r21
          POP   (r11)         ; Se recupera el valor de r11
          POP   (r10)         ; Se recupera el valor de r10
          . . . . .           ; . . . . . .
```

Asignación de etiquetas y de memoria

El punto de entrada de cada una de las subrutinas deberá ir asociado a las etiquetas `nFiltrados`, `FiltRec`, `Filtro`, `FilPixel`, `SubMatriz`, `ValorPixel`, `Comp` y `Sqrt` (respetando el tipo de letra mayúsculas/minúsculas). Por ejemplo, la primera instrucción perteneciente a la subrutina `Comp` deberá ir precedida de esta etiqueta:

```
Comp: subu r30,r30,4
              .
              .
              .
```

Las direcciones de memoria **0x00010000 y siguientes se deben reservar para el programa corrector**, por lo que el alumno **no debe utilizarlas** para almacenar código, datos ni el espacio de pila. Además, deberá situar la pila en posiciones altas de memoria pero **siempre inferiores a la 0x00010000** que se ha mencionado. Por ejemplo, podrá inicializarse el puntero de pila en la dirección `0x0000F000`.

Avisos importantes

Los proyectos que no se atengan a las siguientes normas se evaluarán como no aptos en la parte de implementación aunque superen la batería de pruebas establecida por el Departamento:

1. Todas las subrutinas deberán finalizar con el mismo valor del puntero de pila que tenía al principio de la subrutina (antes de que ejecute su primera instrucción).

2. Cuando una subrutina requiera memoria de trabajo en la que almacenar temporalmente información, empleará **siempre** memoria de la pila y **nunca** variables definidas en direcciones absolutas.

3. La implementación de la subrutina `FiltRec` debe realizarse obligatoriamente mediante el algoritmo recursivo descrito en este enunciado.

4. La variable estática nF descrita en la subrutina `nFiltrados` se tratará exclusivamente desde dicha subrutina, no estando autorizada su consulta o modificación desde otra parte del código del proyecto.

Los proyectos que no se atengan a las siguientes normas se verán penalizados pudiendo llegar a ser calificados como no aptos:

5. Las tareas de este proyecto incluyen la definición e implementación por parte de los alumnos de la batería de pruebas a emplear para verificar el funcionamiento correcto de las subrutinas del proyecto, ya que constituye una parte fundamental de cualquier desarrollo de software. Debido a ello no se considera válido copiar dichas baterías de pruebas de unos grupos a otros. En caso de no respetar esta norma, el proyecto podrá ser calificado como suspenso.

6. Los programas de prueba usarán obligatoriamente macros `PUSH` para incluir en la pila los parámetros de la subrutina que estén probando. Los datos de prueba se definirán individualmente, de forma clara, mediante pseudo-instrucciones `data` y `org` y se se les asignará etiquetas independientes. Se muestra un ejemplo en la sección "Ejemplos", pag. 23. Las consultas de los alumnos que no sigan estas instrucciones no serán atendidas. El código del proyecto incluirá al menos un programa de prueba para cada subrutina implementada.

Sugerencias y recomendaciones

- Para calcular el número entero múltiplo de 4 más pequeño pero igual o superior a un número dado, se pueden usar varios métodos sencillos y se considera válido utilizar cualquiera de ellos. Sin embargo, recomendamos evitar el uso de bucles que realicen más de cuatro iteraciones. Como sugerencia, tenga en cuenta que puede comprobar si un número cualquiera es múltiplo de 4 simplemente observando si sus dos bits menos significativos tienen valor cero.

- En ningún apartado de este proyecto se requiere utilizar instrucciones que operen con números en coma flotante. En particular, las multiplicaciones y divisiones que se utilicen deben trabajar con operandos enteros.

Ejemplos

A continuación se incluye un ejemplo de caso de prueba con los argumentos que se pasan a la subrutina `FiltRec` y las direcciones de memoria que se modifican.

En la página web se puede encontrar un documento con ejemplos de casos de prueba de todas las subrutinas del proyecto, que se pueden seguir como guía para la elaboración de juegos de ensayo más completos.

Todos los ejemplos se han descrito utilizando el formato de salida que ofrece el simulador del 88110. En este procesador el direccionamiento se hace a nivel de byte y se utiliza el formato *little-endian*. En consecuencia, cada una de las palabras representadas a continuación de la especificación de la dirección debe interpretarse como formada por 4 bytes con el orden que se muestra en el ejemplo siguiente:

```
Direcciones de memoria, tal como las representa el simulador:
60000       04050607     05010000

Direcciones de memoria, tal como se deben interpretar:
60000       04
60001       05
60002       06
60003       07

60004       05
60005       01
60006       00
60007       00

Valor de las palabras almacenadas en las posiciones 60000 y
60004, tal como las interpreta el procesador:

60000       0x07060504 = 117.835.012
60004       0x00000105 = 261
```

Caso de prueba de la subrutina `FiltRec`:

Llama a `FiltRec` pasándole una imagen no nula de 4x8 elementos y un filtro que sustituye cada elemento por la media de los que le rodean. El parámetro NCambios tiene valor 0, por lo que la ejecución se detiene al alcanzar el máximo número de llamadas recursivas, que se fija en 3.

La imagen de entrada y la matriz de filtro son las siguientes:

$$\begin{pmatrix} FF & 01 & 02 & FF & FF & 03 & 04 & FF \\ FF & 05 & 06 & FF & FF & 07 & 08 & FF \\ FF & 05 & 06 & FF & FF & 07 & 08 & FF \\ FF & 01 & 02 & FF & FF & 03 & 04 & FF \end{pmatrix} \qquad \begin{pmatrix} 1 & 8 & 1 & 8 & 1 & 8 \\ 1 & 8 & 0 & 8 & 1 & 8 \\ 1 & 8 & 1 & 8 & 1 & 8 \end{pmatrix}$$

Estos datos de entrada se definen en ensamblador mediante las siguientes pseudo-instrucciones (que también incluyen la definición de la variable estática nF con un valor inicial de 100 y la reserva de espacio, 40 bytes, para la imagen de salida FILTRADA):

```
        org     0
nF:     data    100

        org     0x8000
IMAGEN: data    4, 8
        data    0xFF0201FF, 0xFF0403FF
        data    0xFF0605FF, 0xFF0807FF
        data    0xFF0605FF, 0xFF0807FF
        data    0xFF0201FF, 0xFF0403FF

FILTRO: data    1, 8, 1, 8, 1, 8
        data    1, 8, 0, 8, 1, 8
        data    1, 8, 1, 8, 1, 8

        org     0x8100
FILTRADA: res   40
```

El emulador 88110 mostrará estos datos de la siguiente forma usando el comando V:

```
        0       64000000

    32768       04000000    08000000    FF0102FF    FF0304FF
    32784       FF0506FF    FF0708FF    FF0506FF    FF0708FF
    32800       FF0102FF    FF0304FF

    32800                               01000000    08000000
    32816       01000000    08000000    01000000    08000000
    32832       01000000    08000000    00000000    08000000
    32848       01000000    08000000    01000000    08000000
    32864       01000000    08000000    01000000    08000000
```

El siguiente programa principal inicializa la pila asignando el valor `0x9000` al puntero de pila, registro $r30$. Inicializa la variable nF llamando a la subrutina `nFiltrados`, a la que

pasa como parámetro el valor 3. A continuación introduce en la pila los parámetros usando operaciones PUSH y llama a la subrutina `FiltRec`. Por último, vacía la pila dejando $r30$ con el valor inicial, `0x9000`

```
        org   0x8400
ppal:   or    r30, r0, 0x9000
        or    r31, r30, r30

        addu  r10, r0, 3     ; máx: 3 filtrados
        PUSH  (r10)
        bsr   nFiltrados     ; inicializa nF
        addu  r30, r30, 4

        LEA   (r10, IMAGEN)
        LEA   (r11, FILTRADA)
        LEA   (r12, FILTRO)
        or    r13, r0, r0      ; NCambios=0

        PUSH  (r13) ; NCambios
        PUSH  (r12) ; FILTRO
        PUSH  (r11) ; FILTRADA
        PUSH  (r10) ; IMAGEN

        bsr   FiltRec
ret:    addu  r30, r30, 16

        stop
```

Al comienzo de la subrutina `FiltRec`, el puntero de pila $r30$ tiene el valor `0x8FEC` (36848) y el contenido de la pila es el siguiente:

```
r30=36848 (0x8FF0)

36848      00800000      00810000      28800000      00000000
```

Por último, el resultado de la ejecución de `FiltRec`, mostrado antes de ejecutar la instrucción de la etiqueta `ret`, y la imagen filtrada es el siguiente:

```
r30=36848 (0x8FF0)  r29=-1 (0xFFFFFFFF)

Direcciones de memoria modificadas:

00000       00000000

33024       04000000      08000000      FF0102FF      FF0304FF
33040       FF8B7393      93758CFF      FF8B7393      93758CFF
33056       FF0102FF      FF0304FF

36848       00800000      00810000      28800000      00000000
```

$$\begin{pmatrix} FF & 01 & 02 & FF & FF & 03 & 04 & FF \\ FF & 8B & 73 & 93 & 93 & 75 & 8C & FF \\ FF & 8B & 73 & 93 & 93 & 75 & 8C & FF \\ FF & 01 & 02 & FF & FF & 03 & 04 & FF \end{pmatrix}$$

NORMAS DE PRESENTACIÓN Y EVALUACIÓN

Toda la información relativa a este proyecto y, en particular, las aclaraciones o modificaciones de última hora, se encuentra disponible en:

`http://www.datsi.fi.upm.es/docencia/Estructura_09/Proyecto_Ensamblador`

Esta página contiene una sección de anuncios/noticias relacionadas con el proyecto.

EVALUACIÓN 2019/2020

El proyecto consta de tres partes: código y memoria del proyecto (*código*), pruebas de funcionamiento (*pruebas*) y examen del proyecto (*examen*). Para obtener una nota de proyecto compensable con la teoría (2 puntos) es necesario que la nota de código-memoria sea "Apto", que la nota de pruebas sea de al menos 4 puntos y la nota del examen sea de al menos 2 puntos.

- Calificación de la parte de **pruebas**:

Independientemente de las pruebas superadas al concluir el periodo de correcciones del proyecto, hay dos fechas importantes (definidas para cada convocatoria) ya que cada una de ellas lleva asociado un *hito evaluable*. Estos hitos consisten en lo siguiente:

- Hito1: Superar en la fecha programada todas las pruebas establecidas para las subrutinas `nFiltrados` y `Sqrt`.
- Hito2: Superar en la fecha programada las pruebas correspondientes al *Hito1* y además todas las pruebas establecidas para las subrutinas `Comp`, `ValorPixel` y `SubMatriz`.

Teniendo en cuenta estos hitos, las distintas subrutinas que se deben programar, y que el objetivo del proyecto es la construcción de la subrutina recursiva, la nota de la parte de pruebas se obtendrá del siguiente modo:

- Hito1: 1 punto (calificación del hito: 0 ó 1)
- Hito2: 1 punto (calificación del hito: 0 ó 1)
- Subrutina `FilPixel`: hasta 1 punto. La calificación concreta se obtendrá proporcionalmente al número de pruebas superadas.
- Subrutina `Filtro`: hasta 1 punto. La calificación concreta se obtendrá proporcionalmente al número de pruebas superadas.
- Subrutina `FiltRec`: hasta 6 puntos. La calificación concreta se obtendrá proporcionalmente al número de pruebas superadas.

Los alumnos que no obtengan al menos 4 puntos en la parte de pruebas no podrán presentarse al examen del proyecto y su calificación global del proyecto quedará establecida como la mitad de su nota de pruebas.

El número de pruebas que se pasa a cada subrutina, las que se superan y las que fallan estarán indicados en el resultado de la corrección. Estas pruebas podrán verse modificadas durante el desarrollo del proyecto, en cuyo caso se avisará mediante la sección de noticias de la página web del proyecto. Además de las pruebas de cada subrutina, podrán facilitarse otras que no se tendrán en cuenta para la calificación, pero que podrán aportar información de interés a cada grupo de cara a la depuración del código que ha generado. Por ejemplo, al probar la subrutina `FiltRec` se incluirán algunas pruebas en las que se utilizará una implementación realizada por el Departamento del resto de las subrutinas necesarias para la comprobación. Con ello se facilitará que puedan acotar el origen de los posibles errores,

ya que se podrá saber si estos se encuentran en la implementación de `FiltRec` o en las subrutinas de apoyo. Ese tipo de ayuda a la depuración podrá también incluirse para cualquier subrutina que no sea terminal, sino que necesite del funcionamiento correcto de otras subrutinas.

- Calificación del **examen del proyecto**:

Para que un alumno pueda presentarse al examen del proyecto es necesario que su grupo haya alcanzado una calificación de al menos 4 puntos en la parte de **pruebas**. El examen podrá ser a) de tipo test, b) de preguntas con respuesta corta o pequeñas cuestiones prácticas o c) una mezcla de (a) y (b). En caso de obtener una calificación de al menos 3 puntos en el examen, se hará una media ponderada entre la calificación de pruebas (70 %) y la del examen (30 %). Si se obtiene una calificación inferior a 3 puntos en el examen, la nota global del conjunto pruebas-examen será la obtenida en el examen.

- Calificación de **código-memoria**:

Para todos los grupos que obtengan calificación compensable en el conjunto pruebas-examen se revisará tanto el código como la memoria entregados, cuyo resultado podrá implicar que dicha nota quede modificada de acuerdo a lo siguiente:

- En general, siempre que se hayan seguido las normas de implementación indicadas, la calificación obtenida en el conjunto pruebas-examen, siendo ésta mayor o igual a 3 puntos, podrá ser incrementada o decrementada hasta en 1 punto en función de la calidad de la implementación y la memoria.
- El proyecto en conjunto pasará a ser "No Apto" si en esta revisión se detecta que se ha incumplido alguno de los requisitos básicos establecidos, especialmente los recogidos en la sección "Avisos importantes" (pág. 20), por ejemplo, que la subrutina `FiltRec` no se ha implementado por completo (incluyendo todos los pasos establecidos en este enunciado), que se haya implementado de forma no recursiva, que se haya utilizado memoria en direcciones absolutas para almacenar variables temporales, etc.

CONVOCATORIA DE FEBRERO 2020

El plazo de entrega del proyecto estará abierto desde el viernes día **18 de octubre** hasta el miércoles **día 11 de diciembre de 2019** en que se realizará la corrección definitiva y se recogerán las memorias del proyecto (en fichero pdf).

En todas las correcciones se pasarán las pruebas de todas las subrutinas. Esto permitirá que los grupos avancen en la implementación y puedan probar en las primeras correcciones, no sólo las subrutinas asociadas a cada hito, sino también el resto. Por otra parte, obliga a que estén definidas desde el primer momento las etiquetas asociadas a todas las subrutinas del proyecto. Entre el 12 y el 19 de diciembre se dejará el sistema configurado de modo que permita entregar únicamente la memoria del proyecto, para facilitar que se incluyan las últimas modificaciones a la misma.

Cada grupo podrá disponer de las siguientes correcciones para comprobar el funcionamiento de su código:

- Primera corrección, día 23 de octubre
- Una única corrección adicional antes del 30 de octubre
- Corrección del primer hito, día 30 de octubre
- Una única corrección adicional antes del 15 de noviembre
- Corrección del segundo hito, día 15 de noviembre

- Un máximo de tres correcciones adicionales antes del 11 de diciembre
- Corrección definitiva, día 11 de diciembre

Las correcciones se realizarán los días laborables no festivos a partir de las 21:00. Para solicitar una corrección bastará con entregar correctamente los ficheros del proyecto antes de dicha hora límite.

El **examen del proyecto** se realizará el viernes día 13 de diciembre a las 15:00.

CONVOCATORIA EXTRAORDINARIA DE JULIO 2020

Todos los alumnos que se presenten a esta convocatoria del proyecto deberán realizar la **entrega completa de los ficheros del proyecto**, independientemente de que se hayan presentado o no y de las pruebas que hayan superado en la convocatoria de febrero.

El plazo de entrega del proyecto estará abierto desde el jueves día **28 de mayo** hasta el viernes **día 19 de junio de 2020** en que se realizará la corrección definitiva. El lunes día 22 de junio el sistema quedará configurado de modo que permita entregar únicamente la memoria del proyecto. El plazo de entrega para la memoria finalizará el martes 23 de junio a las 20:00.

Cada grupo podrá disponer de la primera y última de las correcciones, que se realizarán los días **1 y 19 de junio**, así como la corrección en que se comprobará si se superan los "hitos evaluables", que tendrá lugar el **día 8 de junio** (deberá pasar correctamente todas las pruebas de las subrutinas `nFiltrados`, `Sqrt`, `Comp`, `ValorPixel` y `SubMatriz`).

Además, podrá pasar correcciones un máximo de **cuatro** veces en las planificadas para los días 2 a 5, 9 a 12 y 15 a 18 de junio.

Todas las correcciones se realizarán a partir de las 21:00. Para solicitar una corrección bastará con entregar correctamente los ficheros del proyecto antes de dicha hora límite.

El **examen del proyecto** se realizará el mismo día del examen extraordinario de teoría (jueves 25 de junio). La hora de comienzo se anunciará en la Web de la asignatura.

TODAS LAS CONVOCATORIAS

La última entrega del proyecto no tiene carácter *obligatorio*. Es decir, una vez que los ficheros entregados por un grupo superan todas las pruebas del proyecto, no es necesario que el grupo realice una nueva entrega para la corrección del último día. En caso de no haber entregado la versión definitiva del fichero que contiene la memoria del proyecto, esta memoria se podrá entregar hasta el día de la última corrección.

Antes de cada examen del proyecto se indicará en la Web de la asignatura si se permitirá utilizar alguna documentación para realizar dicho examen y en caso afirmativo se especificará cuál. Asimismo, se indicará si se permite el uso de calculadora. En caso de permitirlo se referiría a un modelo básico, no programable. No se permitirá el uso de teléfonos móviles ni otros dispositivos con capacidad de almacenamiento y/o conexión remota.

TUTORÍAS DEL PROYECTO

Las preguntas relacionadas con este proyecto se atenderán por correo electrónico en la dirección (**pr_ensamblador@datsi.fi.upm.es**) y personalmente en los despachos 4105 y/o 4106. El horario de atención personal a los alumnos para cuestiones relacionadas con este proyecto es el indicado en la web del Departamento para los profesores encargados del proyecto durante el presente curso académico (José L. Pedraza, Manuel M. Nieto):

`http://www.datsi.fi.upm.es/docencia/tutorias.html`

ENTREGA DEL PROYECTO

La entrega se compone de:

1. Una **memoria**, en formato DIN-A4, en cuya portada deberá figurar claramente el nombre y apellidos de los **autores** del proyecto, identificador del **grupo de alumnos** (el mismo que emplean para realizar las entregas y consultas) y el nombre de la asignatura.

   Dicha memoria se entregará en formato electrónico, como un fichero PDF, mediante el sistema de entregas. Contendrá al menos los siguientes apartados:

   - Histórico del desarrollo de las rutinas, con fechas, avances y dificultades encontradas, especificando el trabajo que realiza cada miembro del grupo o si dicho trabajo es común. Se detallará en este apartado el número total de horas invertidas en el proyecto por cada miembro del grupo, así como la relación de visitas realizadas a los profesores del proyecto.
   - Descripción resumida del juego de ensayo (conjunto de casos de prueba) que el grupo haya diseñado y utilizado para probar el correcto funcionamiento del proyecto.
   - Observaciones finales y comentarios personales de este proyecto, entre los que se debe incluir una descripción de las principales dificultades surgidas para su realización.

   NOTA: Esta memoria no debe incluir el listado en ensamblador del código generado por el grupo, por lo que sí será necesario que todas las subrutinas se encuentren adecuadamente comentadas en el propio fichero que se entrega para su corrección automática (el descrito en el siguiente punto, *filtror.ens*), ya que dicho fichero será consultado en el proceso de evaluación del proyecto.

2. La entrega de los ficheros que contienen el proyecto. Será obligatorio entregar los siguientes ficheros:

   - **autores (solo al dar de alta el grupo)**: Es un fichero ASCII que deberá contener los apellidos, nombre, número de matrícula, DNI y dirección de correo electrónico de los autores del proyecto. El proyecto se realizará individualmente o en grupos de **dos alumnos**. Cada línea de este fichero contendrá los datos de uno de los autores de acuerdo al siguiente formato:

     Nº Matrícula; DNI ; apellido apellido, nombre; correo_electrónico
     El número de matrícula que se debe indicar en el fichero es el que **asigna la secretaría de la Facultad** (por ejemplo 990999) y no el que se utiliza como identificador para abrir cuentas en el Centro de Cálculo (por ejemplo a990999).
     La dirección de correo electrónico deberá ser una dirección válida que el alumno consulte frecuentemente. Se recomienda utilizar la dirección oficial asignada al alumno por la UPM, aunque se admiten otras direcciones personales.

- **filtror.ens (en cada entrega)**: Contendrá las subrutinas que componen el proyecto debidamente comentadas, junto con un programa principal para cada una de ellas que se haya utilizado para su depuración y que funcione correctamente.
- **memoria.pdf (en cada entrega)**: Será un fichero en formato PDF que contenga la memoria del proyecto. En la entrega inicial, el contenido del fichero `memoria.pdf` deberá contener al menos la identificación de los miembros del grupo que realiza el proyecto.

IMPORTANTE: Antes de efectuar cada entrega del proyecto se recomienda realizar el ensamblado del fichero `filtror.ens` asegurando que no genera ningún error, así como ejecutar el código del proyecto con varios casos de prueba. De este modo minimizará la probabilidad de malgastar alguna de las correcciones disponibles. Se recuerda también que no debe borrar los programas principales incluidos en `filtror.ens` ni los datos de este fichero antes de la entrega, aunque sí debe comprobar que no los ha situado en las direcciones reservadas al DATSI.

FORMA DE ENTREGA DE LOS FICHEROS

Los ficheros del proyecto se entregan mediante un navegador web convencional conectándose a la dirección:

`http://www.datsi.fi.upm.es/Practicas`

Recuerde que la información de su proyecto no debe ser conocida por otros alumnos o grupos, ya que si por cualquier razón lo fuera, podría verse involucrado en un caso de COPIA cuyas consecuencias le perjudicarán (están descritas en las normas de la asignatura). Por ello le recomendamos que siempre que trabaje en equipos de la Escuela o, en general, en cualquier equipo al que puedan acceder directa o indirectamente otros alumnos, lo haga siempre manteniendo los ficheros del proyecto en dispositivos privados (extraibles), evitando así dejar copias temporales en lugares accesibles por otros grupos.