

# **COMPRESIÓN MULTIMEDIA**

## **Práctica 4.- Compresión de imágenes**

**Basado en el Trabajo de ROQUE MARÍN**

**Dpto. de Ingeniería de la Información y las Comunicaciones  
Facultad de Informática, Universidad de Murcia, Campus de  
Espinardo, 30071 – Espinardo (Murcia)**

# 1.- COMPRESION JPEG EN MATLAB: `jcomdes`

- Función `jcomdes` (disponible en la librería):
  - **Comprime** un archivo de imagen BMP, aplicando una simplificación del método JPEG. Como resultado genera **tres secuencias** de caracteres que contienen las palabras código de las componentes Y, Cb y Cr.
  - A continuación (sin almacenar en archivo), **descomprime** las secuencias de caracteres **y reconstruye** una aproximación de la imagen original.
  - Está basada en la función `testquant` que desarrollaste en la práctica anterior donde cuantizabas y descuartizaba una imagen.
  - Ahora **se añade** un proceso de **codificación** Huffman canónico JPEG, **y** la correspondiente **decodificación**
  - **Utiliza las tablas de especificación Huffman** por defecto del documento de descripción del estándar JPEG (ITU T.81)
  - Admite un argumento `caliQ`: un entero  $\geq 1$ , que establece la calidad de la compresión y afecta a la relación de compresión.
    - Con `caliQ = 100`, se aplican las tablas estándar
    - Con `caliQ = 1`, escalones unidad (extremadamente lento) y se consigue una calidad de imagen descomprimida visualmente indistinguible de la original.
    - Con `caliQ > 100`, se aplican las tablas estándar multiplicadas por `caliQ %`, aumentando el tamaño de escalón y disminuyendo la calidad

# 1.- COMPRESION JPEG EN MATLAB: jcomdes

- Cabecera de la función jcomdes:

```
function jcomdes(fname,caliQ)

% jcomdes: Compresion y descompresion de imagenes basada en
% transformadas

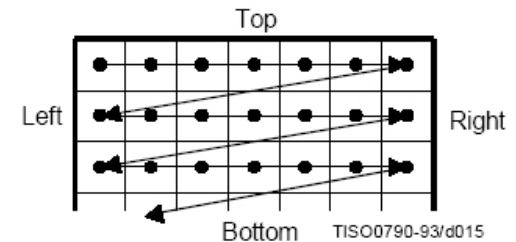
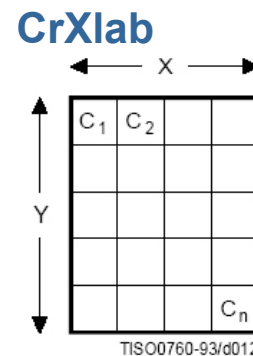
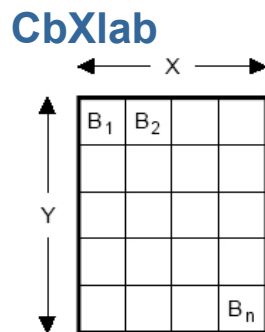
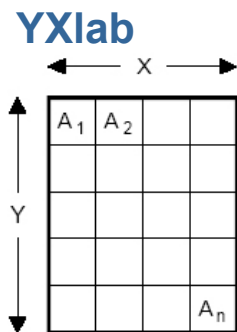
% Entradas:
%   fname: Un string con nombre de archivo, incluido sufijo
%           Admite BMP y JPEG, indexado y truecolor
%   caliQ: Factor de calidad (entero positivo >= 1)
%           100: calidad estandar
%           >100: menor calidad
%           <100: mayor calidad
% Salidas:
%   Ninguna. Solo visualización de imagenes original y procesada
```

# 1.- COMPRESION JPEG EN MATLAB: `jcomdes`

- Pasos aplicados por la función `jcomdes`:
  - Llama a la función auxiliar `imlee`:
    - Lee la imagen almacenada en el archivo `fname`, pasa la matriz RGB `x` al espacio de color YCbCr, y amplía dimensiones a múltiplos de 8: `Xamp`
  - Llama a la función auxiliar `imdct`:
    - Aplica DCT a la matriz ampliada `Xamp`, obteniendo la matriz transformada `Xtrans`
  - Llama a la función auxiliar `quantmat`:
    - Cuantiza los coeficientes `Xtrans`, obteniendo una matriz de etiquetas `Xlab`
  - Llama a la función auxiliar `scan`:
    - Reordena en zigzag cada bloque 8x8 de cada componente de color, obteniendo una matriz reordenada `Xscan` (formada por los scans `YScan`, `CbScan` y `CrScan`)
  - Llama a la función auxiliar `EncodeScans_dflt`:
    - Codifica en binario los tres scans, usando Huffman por defecto, y devuelve los strings binarios `CodedY`, `CodedCb` y `CodedCr`
  - Realiza el proceso inverso: `DecodeScans_dflt`, `invscan`, `desquantmat`, `imidct`. Obtiene una matriz reconstruida: `xrec`
  - Visualiza la imagen original `x` y la reconstruida `xrec`

## 2.- COMPRESION JPEG EN MATLAB: scan

- Los tres primeros pasos ya los conocemos (práctica previa)
- Scan según ITU T.81: Para modo baseline, secuencial, no intercalado
  - Un **scan** es un recorrido por los bloques 8x8 de la imagen de cada componente de color, en orden secuencial:
    - izquierda a derecha y arriba a abajo
  - El recorrido establecerá el orden en el que los datos serán codificados
  - Se hace un scan por cada componente de color YXlab, CbXlab, CrXlab
  - Entonces, para la codificación, los tres scans se ordenan secuencialmente



**Scans:**

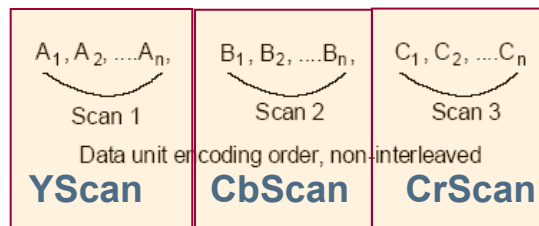
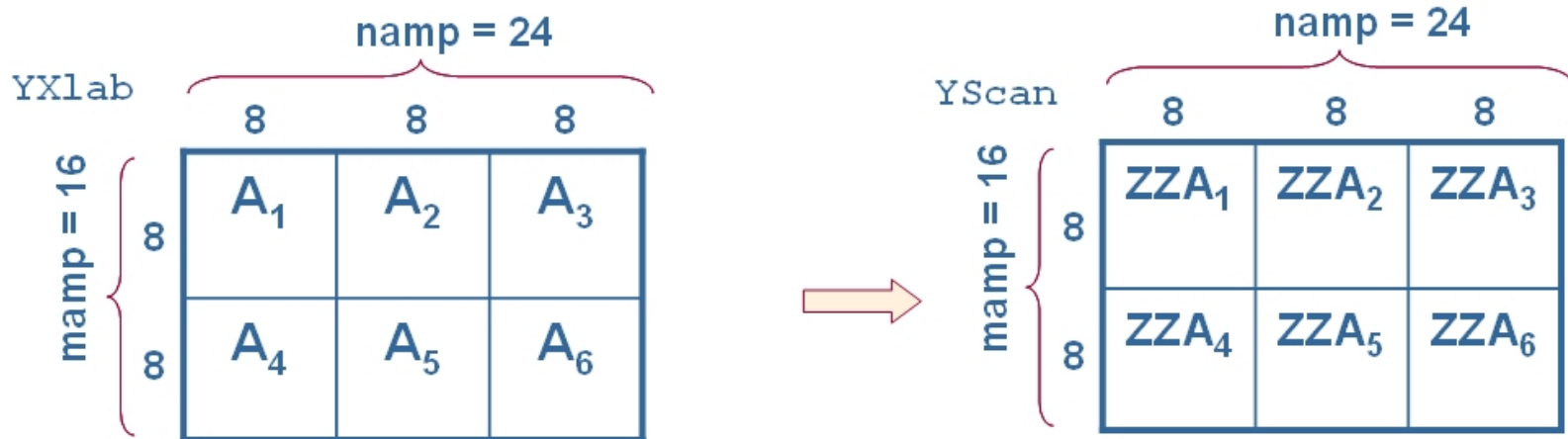


Figure A.2 – Non-interleaved data ordering



## 2.- COMPRESION JPEG EN MATLAB: scan

- Se define una **MCU** (Minimal Coding Unit) como un bloque de 8x8 que se utiliza en el scan. Una MCU es uno de los bloques  $A_i$
- Cada MCU deberá contener los datos listos para el Scan final
- Es decir deberá **contener los datos en orden zigzag** pero preparados para una lectura secuencial.
- La función **scan** prepara todas las MCU para cada componente de color.
- Al aplicar **scan**, no cambian las dimensiones de la matriz
- Por ejemplo,
  - supongamos que la matriz de etiquetas **YXlab** (componente Y -luminancia- de **Xlab** -valores cuantizados) tiene tamaño 16x24, y está dividida en 2x3 bloques de 8x8
  - Al aplicar **scan**, se obtiene una matriz **YScan** que tiene igual tamaño, pero los elementos de cada bloque han sido reordenados en zigzag



Nº bloques en vertical:  $nbv = mamp/8 = 2$ ;    Nº bloques en horizontal:  $nbh = namp/8 = 3$

## 2.- COMPRESION JPEG EN MATLAB: scan

- **Cabecera de la función scan:**

```
function XScan=scan(Xlab)
% Genera un scan por cada componente de color
% Cada scan es una matriz mamp x namp
% Cada bloque se reordena en zigzag

% Entradas:
% Xlab: Matriz de etiquetas a procesar: mamp x namp x 3
% Salidas:
% XScan: Scans de luminancia Y y crominancia Cb y Cr
% Es una matriz mamp x namp X 3 compuesta de:
% YScan: Scan de luminancia Y: Matriz mamp x namp
% CbScan: Scan de crominancia Cb: Matriz mamp x namp
% CrScan: Scan de crominancia Cr: Matriz mamp x namp
```

- La codificación se hará en el orden resultante del scan.
- El scan ordena secuencialmente los bloques (MCU) pero no ordena secuencialmente sus contenidos.
- Para reordenar los bloques 8x8 en vectores zig-zag, se utiliza:
  - La función Matlab `blkproc`
  - La función auxiliar `zigzag64`: Aplica una permutación a las etiquetas

## 2.- COMPRESION JPEG EN MATLAB: scan

- Función auxiliar zigzag64:

```
function matzz=zigzag64(mat)
% Reordena una matriz 8x8 en el orden natural
% y produce una matriz 8x8 reordenada en zigzag

% Entradas:
%   mat: Matriz 8x8 en orden natural
% Salidas
%   matzz: Matriz 8x8 en orden zig-zag

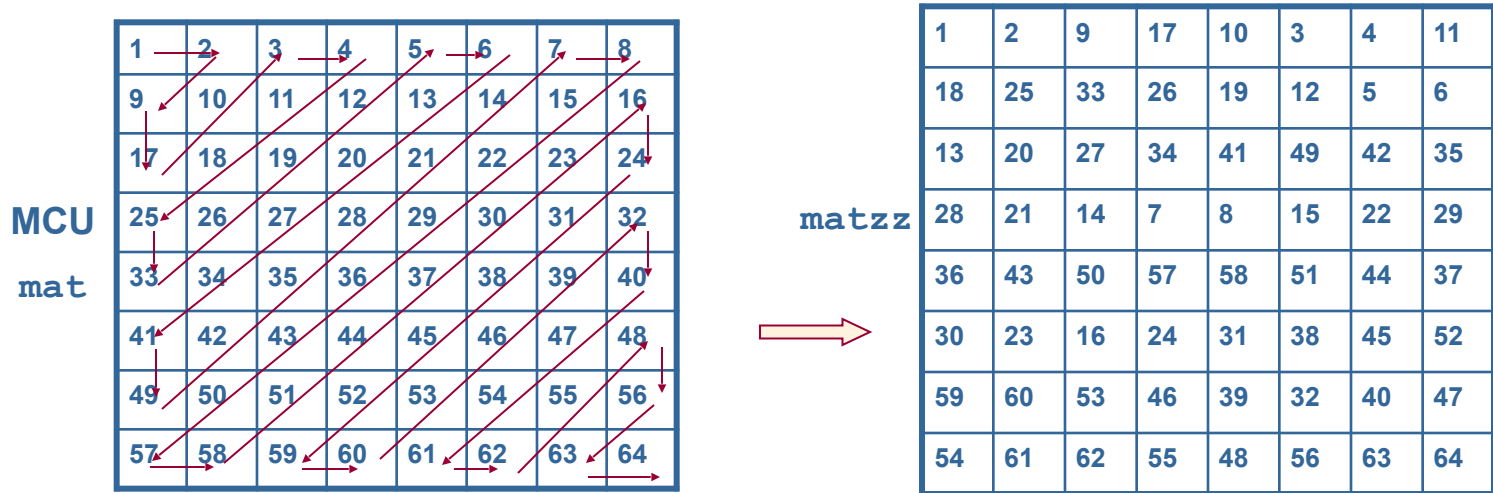
% Permutacion
perm = [1 2 9 17 10 3 4 11; ...
        18 25 33 26 19 12 5 6; ...
        13 20 27 34 41 49 42 35; ...
        28 21 14 7 8 15 22 29; ...
        36 43 50 57 58 51 44 37; ...
        30 23 16 24 31 38 45 52; ...
        59 60 53 46 39 32 40 47; ...
        54 61 62 55 48 56 63 64];

matt=mat';
matzz=matt(perm);
```



## 2.- COMPRESION JPEG EN MATLAB: scan

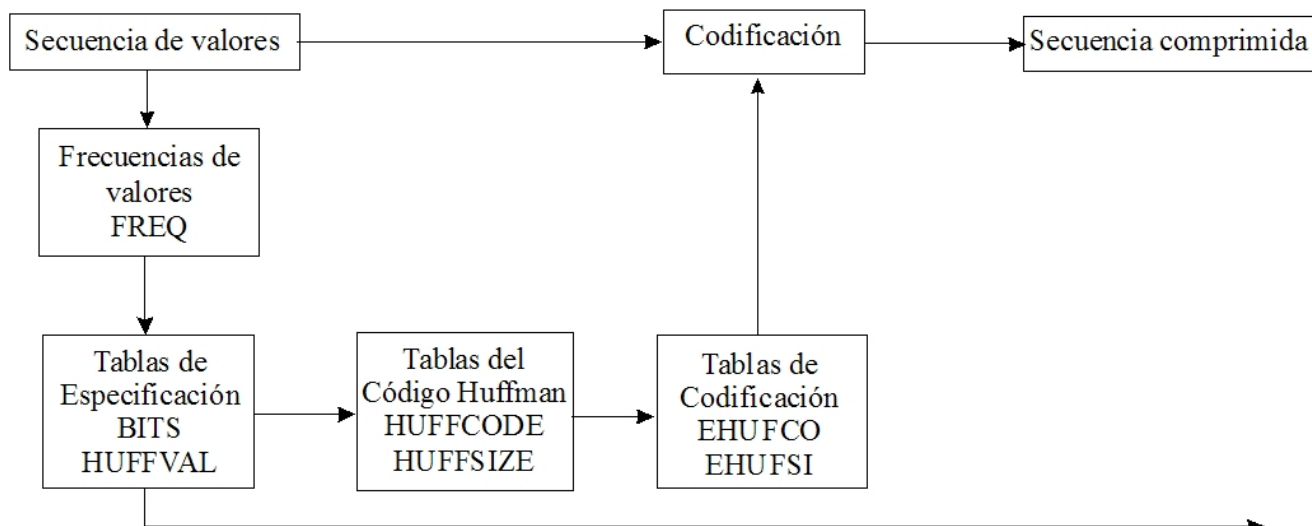
- La permutación zigzag64 es:



- Al recorrer `matzz` en orden secuencial (por filas) se está siguiendo el orden zigzag

### 3.- COMPRESION JPEG EN MATLAB: `EncodeScans_dflt`

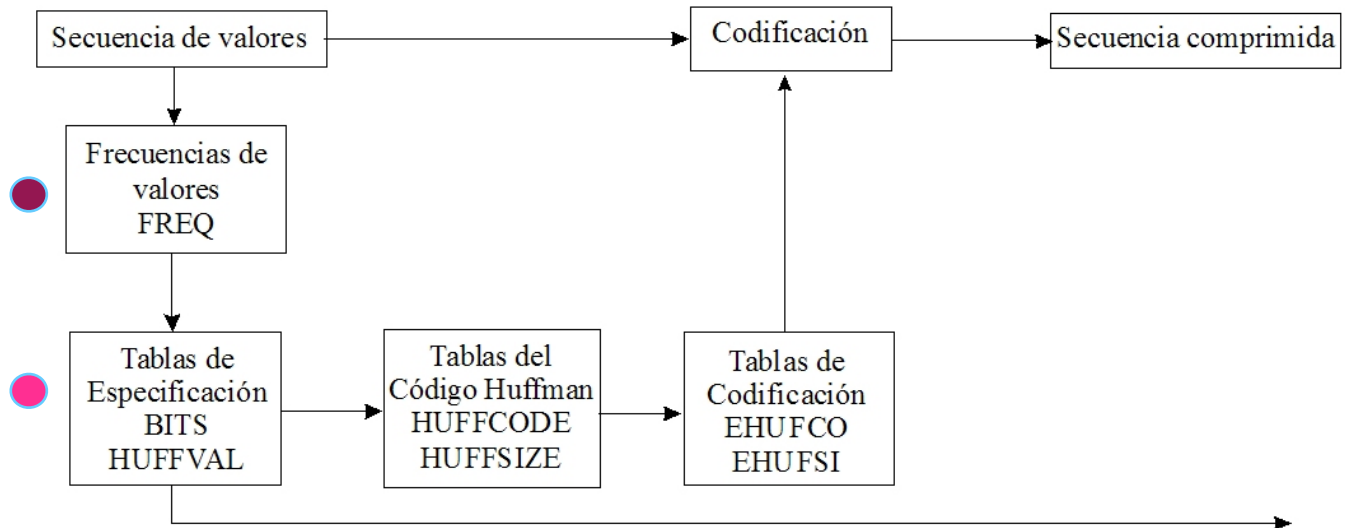
- Función auxiliar `EncodeScans_dflt`:
  - Codifica en binario los tres scans, usando Huffman por defecto, y devuelve los strings binarios `CodedY`, `CodedCb` y `CodedCr`
- Recordemos el esquema del proceso de compresión Huffman en JPEG (Práctica 2):



- Pues recordemos ....

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

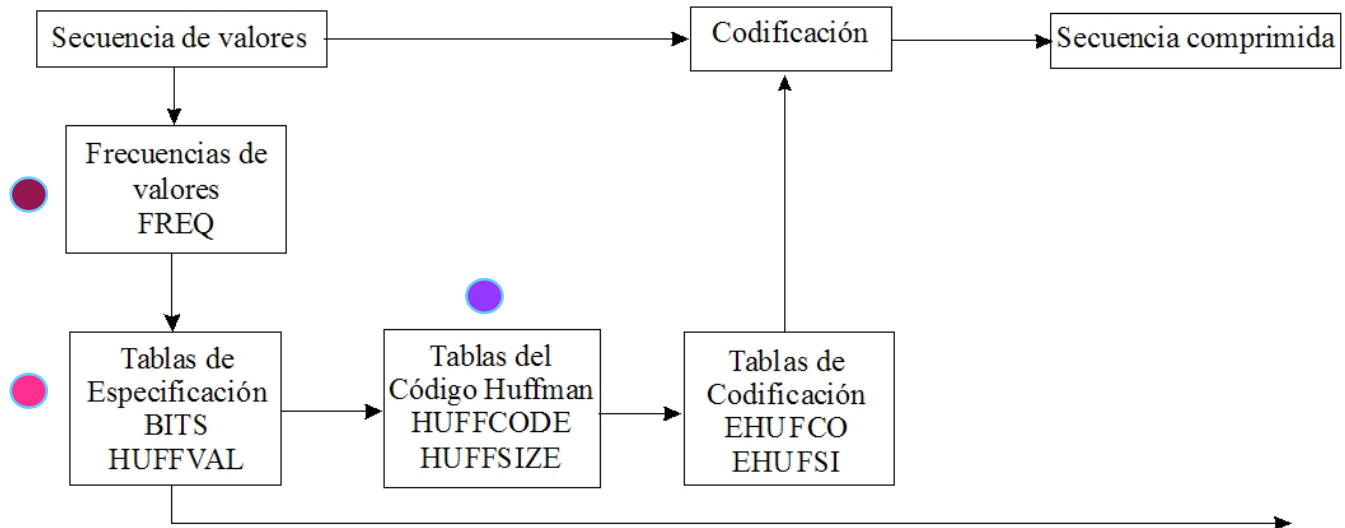
- Recordemos el esquema del proceso de compresión Huffman en JPEG (Práctica 2):



- Pasos generales del proceso:**
  - Calcular la frecuencia con la que aparece cada mensaje en la secuencia a codificar: Función auxiliar `Freq256`
  - Generar las tablas de especificación de un código Huffman: Función auxiliar `HSpecTables`
    - **BITS**: Número de palabras que hay de una longitud.
    - **HUFFVAL**: Los símbolos de esas longitudes

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

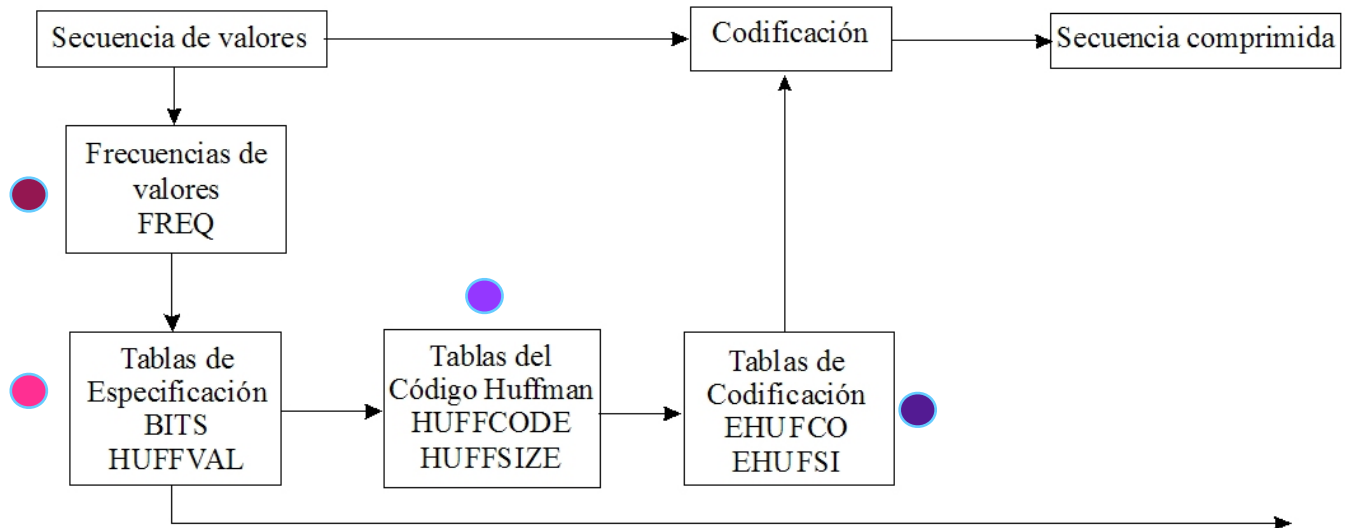
- Recordemos el esquema del proceso de compresión Huffman en JPEG (Práctica 2):



- Pasos generales del proceso:
  - Calcular la frecuencia de los símbolos: `Freq256`
  - Generar las tablas de especificación de un código Huffman: `HSpecTables`
  - Construir las tablas del código Huffman: Función auxiliar `HCodeTables`
    - `HUFFSIZE`, otra forma de representar `BITS`. Contiene las longitudes de los códigos tantas veces como palabras haya de esa longitud.
    - `HUFFCODE`, los códigos (en decimal)

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Recordemos el esquema del proceso de compresión Huffman en JPEG (Práctica 2):

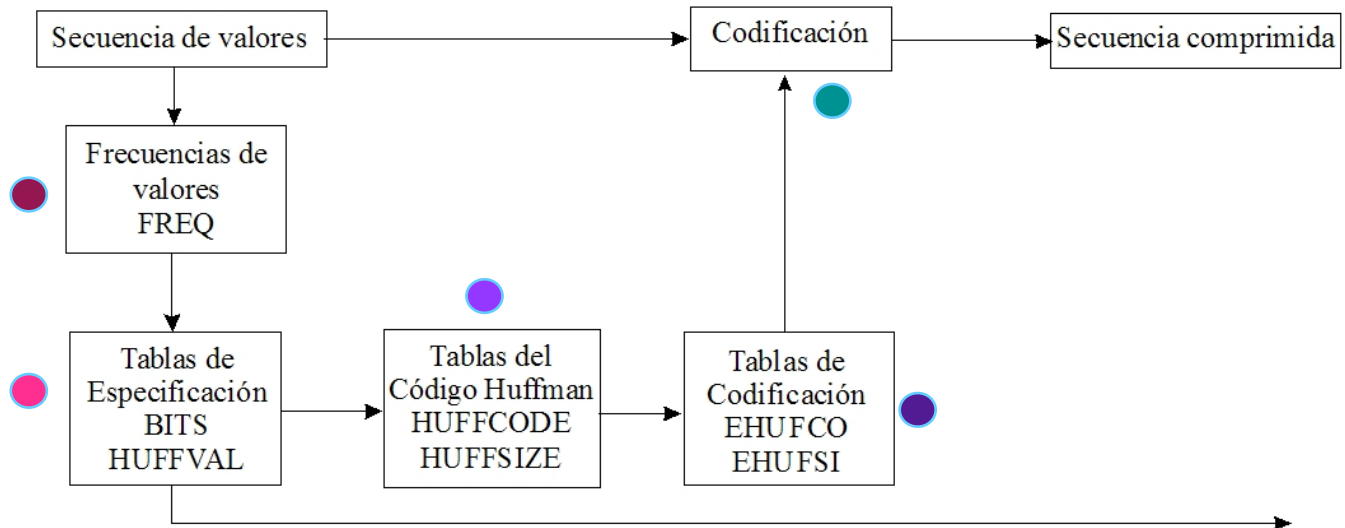


- Pasos generales del proceso:
  - Calcular la frecuencia de los símbolos: `Freq256`
  - Generar las tablas de especificación de un código Huffman: `SpecTables`
  - Construir las tablas del código Huffman: `HCodeTables`
  - Construir las tablas para codificación: Función auxiliar `HCodingTables`
    - `EHUFCO`: Los códigos según orden creciente de los símbolos.
    - `EHUFISI`: Las longitudes de las palabras código en ese orden.



### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Recordemos el esquema del proceso de compresión Huffman en JPEG (Práctica 2):



- Pasos generales del proceso:
  - Calcular la frecuencia de los símbolos: `Freq256`
  - Generar las tablas de especificación de un código Huffman: `SpecTables`
  - Construir las tablas del código Huffman: `HCodeTables`
  - Construir las tablas para codificación: `HCodingTables`
  - Codificar los valores de la secuencia a partir de las tablas de codificación: Función auxiliar EncodeSingleScan

# 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Cabecera de la función EncodeScans\_dflt:

```
function [CodedY,CodedCb,CodedCr]=EncodeScans_dflt(XScan)
% EncodeScans_dflt: Codifica en binario los scan con Huffman por defecto

% Entradas:
% XScan: Scans de luminancia Y y crominancia Cb y Cr
% Es una matriz mamp x namp X 3 compuesta de:
% YScan: Scan de luminancia Y: Matriz mamp x namp
% CbScan: Scan de crominancia Cb: Matriz mamp x namp
% CrScan: Scan de crominancia Cr: Matriz mamp x namp
% Salidas:
% CodedY: String binario con scan Y codificado
% CodedCb: String binario con scan Cb codificado
% CodedCr: String binario con scan Cr codificado
```

- La función EncodeScans\_dflt aplica tres pasos:
  - Recolectar la secuencia de valores a codificar en cada scan
  - Generar las tablas Huffman por defecto
  - Codificar en binario aplicando las tablas Huffman a los valores

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Recolectar los valores a codificar en cada scan:
  - Se aplica la función `CollectScan` a cada scan
  - Se generan dos tablas por scan: `nn_DC_CP` y `nn_AC-ZCP`

```
% Recolectar valores a codificar
[Y_DC_CP, Y_AC_ZCP]=CollectScan(YScan);
[Cb_DC_CP, Cb_AC_ZCP]=CollectScan(CbScan);
[Cr_DC_CP, Cr_AC_ZCP]=CollectScan(CrScan);
```

- Se generan dos tablas por scan: `nn_DC_CP` y `nn_AC-ZCP`
- **`nn_DC_CP`: Codificación Residual de los DC**
  - Contiene los pares categoría y posición (C, P) para las diferencias DIFF entre etiquetas DC de bloques adyacentes del scan nn
  - La columna 1 contiene la Categoría C en la que cae el valor DIFF
  - La columna 2 contiene la posición P en la categoría C, que permite identificar completamente al valor DIFF
  - Hay una fila por bloque (por valor de DIFF)
  - El tamaño de la tabla es  $(nbv * nbh) \times 2$ 
    - nbv es el número de bloques en vertical
    - nbh es el número de bloques en horizontal.

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Ejemplo: Imagen 'Img06.bmp', de 300 x 240 (1200 bloques)



- Se muestran los dos primeros bloques del scan de luminancia YScan
  - En cada bloque, las etiquetas están ya reordenadas en zigzag
  - El coeficiente DC es el primero de cada bloque

Array Editor: YScan

File Edit View Web Window Help

Numeric format: shortG Size: 8 by 16

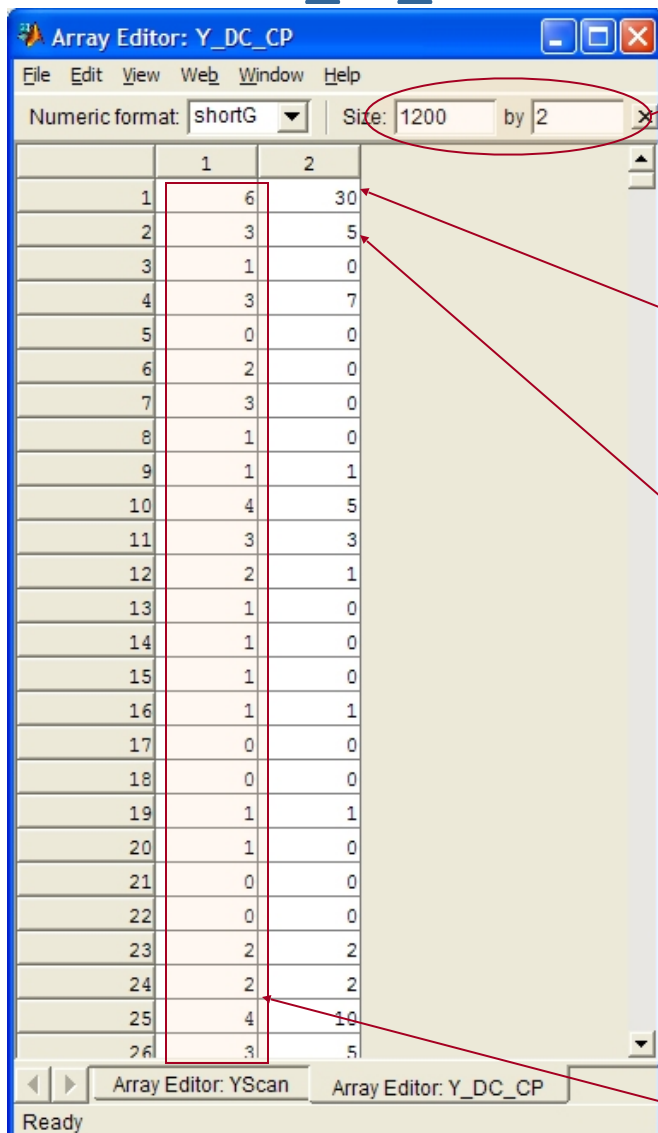
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-33	0	-1	0	1	0	0	0	-28	-2	0	0	0	-1	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ready



# 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- La tabla  $Y\_DC\_CP$  para el ejemplo es:



	1	2
1	6	30
2	3	5
3	1	0
4	3	7
5	0	0
6	2	0
7	3	0
8	1	0
9	1	1
10	4	5
11	3	3
12	2	1
13	1	0
14	1	0
15	1	0
16	1	1
17	0	0
18	0	0
19	1	1
20	1	0
21	0	0
22	0	0
23	2	2
24	2	2
25	4	10
26	3	5

Filas: 1200 (nº bloques)

-30 es negativo.  
30 es el complemento a 1 de 33

$$\text{DIFF}(1): DC_1 - 0 = -33 - 0 = -33$$

Para  $\text{DIFF} = -33$ ,  $(C, P) = (6, 30)$

```
>> catpos(-33)
ans =
     6     30
>>
```

$$\text{DIFF}(2): DC_2 - DC_1 = -28 - (-33) = 5$$

Para  $\text{DIFF} = 5$ ,  $(C, P) = (3, 5)$

```
>> catpos(5)
ans =
     3     5
>>
```

La secuencia a codificar es columna 1  
(categorías)



### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- **nn\_AC\_ZCP: Codificación Run Length**
  - Contiene los valores (Z, C, P) para las etiquetas AC del scan
  - La columna 1 contiene el par Z/C en decimal:  $ZC_{dec} = Z \cdot 16 + C$ 
    - Z: N° de ceros previos a la etiqueta (entre 0 y 15)
    - C: Categoría de la etiqueta
  - La columna 2 contiene la posición P de la etiqueta en la categoría C, que permite identificar completamente a la etiqueta
  - El final de un bloque (0/0 EOB) se representa mediante el par  $(ZC_{dec}, P) = [0 \ 0]$
  - Una secuencia de **16 ceros** consecutivos (F/0 ZRL) se representa mediante el par  $(ZC_{dec}, P) = (15 \cdot 16 + 0, 0) = [240 \ 0]$
  - El número de filas es indeterminado, pues las etiquetas nulas consecutivas se agrupan en un valor de Z

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Volvemos al ejemplo 'Img06.bmp':

Array Editor: YScan

File Edit View Web Window Help

Numeric format: shortG Size: 8 by 16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		0	-1	0	1	0	0	0		-2	0	0	0	-1	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ready

- Para el bloque 1, la secuencia de 63 etiquetas AC a codificar es:
  - 0, -1, 0, 1, 0, ....0
- Para el bloque 2, la secuencia de 63 etiquetas AC a codificar es:
  - -2, 0, 0, 0, -1, 0, ....0

# 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- La tabla Y\_AC\_ZCP para el ejemplo es:

Array Editor: Y\_AC...

File Edit View Web Window Help

Numeric format: shortG Siz X

	1	2
1	17	0
2	17	1
3	0	0
4	2	1
5	49	0
6	0	0
7	1	1
8	1	0
9	1	0
10	49	1
11	0	0
12	3	2
13	2	1
14	17	0
15	2	0
16	1	0
17	0	0
18	2	2
19	33	0
20	1	1
21	0	0
22	3	5
23	1	1
24	17	1
25	2	1
26	17	0
27	0	0
28	2	1

Ready

Bloque 1: 0, -1, 0, 1, 0, ....0

$(Z,C,P) = (1, 1, 0) \rightarrow [17 \ 0]$

```
>> catpos(-1)
ans =
      1      0
>>
```

$(Z,C,P) = (1, 1, 1) \rightarrow [17 \ 1]$

```
>> catpos(1)
ans =
      1      1
>>
```

$(Z,C,P) = \text{EOB} = [0 \ 0]$

Todos los demás son 0,  
se puede cerrar el bloque

# 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- La tabla Y\_AC\_ZCP para el ejemplo es:

Array Editor: Y\_AC...

File Edit View Web Window Help

Numeric format: shortG Siz X

	1	2
1	17	0
2	17	1
3	0	0
4	2	1
5	49	0
6	0	0
7	1	1
8	1	0
9	1	0
10	49	1
11	0	0
12	3	2
13	2	1
14	17	0
15	2	0
16	1	0
17	0	0
18	2	2
19	33	0
20	1	1
21	0	0
22	3	5
23	1	1
24	17	1
25	2	1
26	17	0
27	0	0
28	2	1

Ready

Bloque 2:  $-2, 0, 0, 0, -1, 0, \dots, 0$

$(Z,C,P) = (0, 2, 1) \rightarrow [2 \ 1]$

```
>> catpos(-2)
ans =
     2     1
>>
```

$(Z,C,P) = (3, 1, 0) \rightarrow [49 \ 0]$

```
>> catpos(-1)
ans =
     1     0
>>
```

$(Z,C,P) = \text{EOB} = [0 \ 0]$

La secuencia a codificar es columna 1  
(valores ZCdec)

### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Generar las tablas Huffman por defecto:
  - En JPEG, se utilizan 4 códigos Huffman:
    1. Luminancia DC: Se aplica a la columna de categorías de Y\_DC\_CP
    2. Crominancia DC: Se aplica a la columna de categorías de Cb\_DC\_CP y Cr\_DC\_CP
    3. Luminancia AC: Se aplica a la columna ZCdec de Y\_AC\_ZCP
    4. Crominancia AC: Se aplica a la columna ZCdec de Cb\_AC\_ZCP y Cr\_AC\_ZCP
- Para ello usamos la función auxiliar HufCodTables\_dflt:

```
function ehuf = HufCodTables_dflt(tipo)

% Carga tablas de especificacion Huffman por defecto
[BITS,HUFFVAL] = huffdflt(tipo); % Default tables DC
% Construye Tablas delCodigo Huffman
[HUFFSIZE, HUFFCODE] = HCodeTables(BITS, HUFFVAL);
% Construye Tablas de Codificacion Huffman
[EHUFCO, EHUFSI] = HCodingTables(HUFFSIZE, HUFFCODE, HUFFVAL);
ehuf=[EHUFCO EHUFSI];
```



### 3.- COMPRESION JPEG EN MATLAB: EncodeScans\_dflt

- Cargadas las 4 tablas Huffman, se utilizan para codificar en binario los valores:
  - Usamos la función auxiliar EncodeSingleScan
    - Cabecera de EncodeSingleScan:

```
function CodedScan=EncodeSingleScan(FScan, DC_CP, AC_ZCP, ehufDC, ehufAC)
% EncodeScans_dflt: Codifica en binario un scan usando Huffman por defecto

% Entradas:
%   FScan: Scan codificado de una componente Y, Cb o Cr: Matriz mamp x namp
%   DC_CP: Valores DC a codificar
%   AC_ZCP: Valores AC a codificar
%   ehufDC: Es la concatenacion [EHUFCO EHUFISI] para valores DC_CP del scan
%   ehufAC: Es la concatenacion [EHUFCO EHUFISI] para valores AC_ZCP del scan
% Salidas:
%   CodedF: String binario con scan F codificado
```

- El proceso de codificado es similar al aplicado en la Práctica 2, pero después de codificar un valor en binario, añadimos los bits de posición P

## 4.- DESCOMPRESION JPEG EN MATLAB: `jcomdes`

- Los tres strings binarios codificados, `CodedY`, `CodedCb` y `CodedCr`, contiene la imagen codificada en JPEG.
- Para abreviar, vamos a omitir el almacenamiento de la imagen comprimida en un archivo.
- Justo a continuación `jcomdes` procede a realizar el proceso inverso: descomprimir los strings binarios codificados para obtener una reconstrucción de la imagen original.
- Está basada en la 2ª parte de la función `testquant` de la práctica anterior, a la que se añade un proceso de decodificación Huffman
- Utiliza las tablas de especificación Huffman por defecto del documento de descripción del estándar JPEG (ITU T.81)
- Esencialmente, implementa los pasos inversos ejecutados hasta ahora por `jcomdes` y en orden inverso, por lo que no los describimos en detalle.
- Después, simplemente visualizamos la imagen original y la reconstruida, para obtener una apreciación visual de la pérdida de calidad asociada a la compresión (depende del factor de calidad).

## 4.- DESCOMPRESION JPEG EN MATLAB: `jcomdes`

- **Pasos restantes:**
  - Llama a la función auxiliar `DecodeScans_dflt`:
    - Decodifica los tres strings binarios, usando Huffman por defecto, y devuelve la matriz 3-D reconstruida `xScanrec` (contiene etiquetas de cuantización de las tres componentes de color y en orden zigzag). La decodificación es similar a la de la práctica 2, pero usando los bits de posición
  - Llama a la función auxiliar `invscan`:
    - Deshace el orden zigzag en cada bloque 8x8 de cada componente de color, obteniendo una matriz 3-D `xlabrec`
  - Llama a la función auxiliar `desquantmat`:
    - Descuantiza los coeficientes `xlabrec`, obteniendo una matriz de coeficientes transformados reconstruida `Xtransrec`
  - Llama a la función auxiliar `imidct`:
    - Aplica IDCT a la matriz ampliada `Xtransrec`, obteniendo la matriz transformada `Xamprec`
  - Convierte del espacio de color YCbCr al RGB
  - Genera la matriz RGB `xrec` superponiendo las tres capas de color y recortando a las dimensiones originales
  - Visualiza la imagen original `x` y la reconstruida `xrec`

## 5.- EJERCICIOS DEL GUIÓN DE PRÁCTICAS

- **Ejercicio 1:** Escribir una pareja de funciones para comprimir y descomprimir archivos de imagen, usando las **tablas Huffman por defecto**
  - **Función de compresión:** `RC=jcom_dflt(fname,caliQ)`
    - Se basa en la primera mitad de la función `jcomdes` (librería)
    - Aplica tablas Huffman por defecto
    - Genera y almacena un archivo comprimido `*.hud`
    - Calcula y devuelve la relación de compresión `RC`
  - **Función de descompresión:** `[MSE,RC]=jdes_dflt(fname)`
    - Se basa en la segunda mitad de la función `jcomdes` (librería)
    - Aplica tablas Huffman por defecto
    - Lee un archivo comprimido `*.hud`
    - Genera y almacena un archivo descomprimido `*_des_def.bmp`
    - Calcula y devuelve:
      - La relación de compresión `RC`
      - El error cuadrático medio `MSE`
    - Visualiza la imagen bitmap original y la descomprimida



## 5.- EJERCICIOS DEL GUIÓN DE PRÁCTICAS

- **Ejercicio 2:** Escribir una pareja de funciones para comprimir y descomprimir archivos de imagen, usando las tablas Huffman a medida (custom)
- **Función de compresión:** `RC=jcom_custom(fname,caliQ)`
  - Se basa en la primera mitad de la función `jcomdes` (librería)
  - Aplica tablas Huffman a medida
  - Genera y almacena un archivo comprimido \* **.huc**
  - Calcula y devuelve la relación de compresión RC
- **Función de descompresión:** `[MSE,RC]=jdes_custom(fname)`
  - Se basa en la segunda mitad de la función `jcomdes` (librería)
  - Aplica tablas Huffman a medida, almacenadas en el archivo
  - Lee un archivo comprimido \* **.huc**
  - Genera y almacena un archivo descomprimido \* **\_des\_cus.bmp**
  - Calcula y devuelve:
    - La relación de compresión RC
    - El error cuadrático medio MSE
  - Visualiza la imagen bitmap original y la descomprimida



## 5.- EJERCICIOS DEL GUIÓN DE PRÁCTICAS

- **Ejercicio 3:** Aplicar los compresores y descompresiones a imágenes artificiales.
- En el ejercicio 2 del guión 3 estudiaste el efecto de la cuantización sobre un bloque 8x8.
  - De ahí debiste sacar varios criterios de cuándo se obtendrán coeficientes nulos
- Teniendo en cuenta que la codificación Huffman por defecto asigna códigos cortos a categorías bajas, responde a preguntas de este tipo
  - ¿cómo deberían ser las imágenes para conseguir mayor compresión?
  - ¿cómo deberían ser las imágenes más difíciles de comprimir?
  - ¿Cuáles perderán más calidad?
- Construye imágenes de tamaño no mayor a 80x80 en BMP que justifiquen las respuestas considerando  $\text{caliQ}=100$

## 5.- EJERCICIOS DEL GUIÓN DE PRÁCTICAS

- En los dos primeros ejercicios hay que almacenar y leer archivos. Seguir un procedimiento similar al de la Práctica 2 (revisar)
- El segundo ejercicio se basa en la función `jcomdes`, pero en lugar de usar las tablas Huffman JPEG por defecto, debe construir unas tablas a medida, en función de las frecuencias con las que aparecen los valores de categoría C en las tablas `nn_DC_CP` y `nn_AC-ZCP`
- Requiere reescribir la función de codificación y la de decodificación:
  - `[CodedY, CodedCb, CodedCr, % Se añade lo que viene a continuación  
BITS_Y_DC, HUFFVAL_Y_DC, BITS_Y_AC, HUFFVAL_Y_AC, BITS_C_DC,  
HUFFVAL_C_DC, BITS_C_AC, HUFFVAL_C_AC] = EncodeScans_custom(XScan);`
  - `XScanrec = DecodeScans_custom(CodedY, CodedCb, CodedCr, [mamp namp],  
% Se añade lo que viene a continuación  
BITS_Y_DC, HUFFVAL_Y_DC, BITS_Y_AC, HUFFVAL_Y_AC, BITS_C_DC,  
HUFFVAL_C_DC, BITS_C_AC, HUFFVAL_C_AC);`
- Para generar las tablas Huffman a medida, seguir un procedimiento similar al seguido en práctica 2 (revisar)
- Para el tercer ejercicio básate en las conclusiones que obtuviste en el ejercicio 2 del guión 3 (`testQuantDCT8x8.m`)

## 6.- CONCLUSIONES

---

- Al final de esta práctica, dispondremos de dos pares de compresores/descompresores:
  - Por defecto: `jcom_dflt` / `jdes_dflt`
  - A medida: `jcom_custom` / `jdes_custom`
- Se debe guardar copia de seguridad de estos compresores, ya que será necesario usarlos para el Proyecto Final de la asignatura (práctica siguiente)