Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Organización del Procesador (1949)

Proyecto “*Zip*”

Integrantes:

- Abuzaid Karim – 35.472.429
- González Alan – 37.177.303

Herramientas:

- **GitHub** (Sistema de control colaborativo de revisión y desarrollo de software)
  https://github.com/vercryger/compressor
- **Nasm** (versión 2.09.10)
- **GCC** (versión 2.09.10)

Aclaraciones:

El código fuente **Assembly** de las subrutinas tanto para la compresión como para la descompresión, se encontrará en los archivos ***encode.asm*** y ***decode.asm*** respectivamente. Dentro de los mismos archivos, se utilizarán subrutinas *externas* declaradas en ***library.asm***.

Instrucciones:

Una vez descomprimido el archivo, en la carpeta ***src*** se encontrará todo el código fuente necesario para ejecutar el programa.
Para correr el programa, simplemente ejecutar el *bash script*:

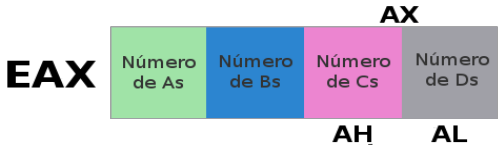
```
src$ ./run.sh
```

Dicho script, primero compilará el código **Assembly**, luego el de **C**, y por último se ejecutará el programa propiamente dicho.

Compresión

```
push dword [ebp + 8]  ; pointer to cadeChar[]
call counter
add esp, 4
```

Como punto de partida, se procesó la cadena pasada como parámetro desde **C** (*cadeChar*), utilizando la subrutina *counter* que cuenta la cantidad de cada una de las letras (*A, B, C* y *D*) de forma tal de que en un registro de propósito general (en este caso EAX), quede almacenada dicha información:



Finalmente la información esta lista para ser utilizado por el *llamante*.

```
mov [aux], eax

push dword eax
call sortbyte
add esp, 4
```

Luego para armar la *matriz de codificación* se copió el registro en una variable de 4 bytes *aux*, y al registro EAX se lo ordenó un registro de *bytes*. Para el ejemplo:

cadeChar = “AAACCBBAACBBBBAADDBB”

counter(*cadeChar*) --> EAX = 06080302 *aux* = 06080302
sortbyte(*eax*) --> EAX = 08060302

La subrutina *sortbyte* ordena de a bytes en orden no ascendente las frecuencias de cada una de las letras, pero ahora la cardinalidad no corresponde a la de cada una de las letras, es por esto que antes de ordenarlo se guardó en *aux* el orden original. Ahora sólo bastaría mapear cada byte de EAX con los de *aux* para saber a *qué* letra corresponde y por ende saber qué letra tiene mayor frecuencia.
Dicho cálculo se realiza desde la línea 36 hasta la línea 71. Una vez hecho esto en ECX se tendrá las posiciones de las letras a las cuales pertenecen dichas frecuencias. Por ejemplo:

aux			
0	*1*	*2*	*3*
06	08	03	02

EAX			
Extended		**AX**	
0	*1*	*2*	*3*
08	06	03	02

ECX			
Extended		**CX**	
0	*1*	*2*	*3*
1	**0**	**2**	**3**

Luego en base a estas posiciones calculadas en ECX, puedo saber en qué orden están en el *cadeChar[]*:

cadeChar + **0** = *A*
cadeChar + **1** = *B*
cadeChar + **2** = *C*
cadeChar + **3** = *D*

Y ahora sólo falta asignarle la correspondiente codificación a cada una de las letras, y por esto es que se ordenó anteriormente las frecuencias, por que a la letra que le pertenece la primer frecuencia, se la codificará como *01* ya que es la que más veces aparece, luego a la letra que le pertenece la segunda frecuencia se la codificará como *011* porque es la que le sigue en cuanto a apariciones, y así sucesivamente con las demás. Es decir, cada posición de las frecuencias en EAX está asociada a una codificación única:

EAX			
Extended		**AX**	
0	*1*	*2*	*3*
08	06	03	02
01	*011*	*0111*	*01111*

Finalmente guardo en *matrizCod* (de forma *aleatoria*) las codificaciones correspondientes a las letras. Primero guardo la codificación de *01111*, luego la de *0111* y así sucesivamente, con lo que obtenemos:

Dirección	Contenido	*cadeChar*
matrizCod + 0	000000011	**A**
matrizCod + 1	00000001	**B**
matrizCod + 2	00000111	**C**
matrizCod + 3	00001111	**D**

Una vez generada la matriz de codificación, lo que sigue es codificar la cadena de entrada. El pseudocódigo sería el siguiente:

```
i = 0
j = 0
AX = 1                                  // ¡Importante!*
BX = cadeChar[j]                        // obtengo la primer letra
while (BL != 0) {                       // no es el fin de la cadena
        BL = getCodification(BL)        // obtengo la codificación en matrizCod y la almaceno en BL
        shl AX, 1                       // necesario para marcar el inicio de una nueva letra

        while (BL != 0) {               // hay 1's en BL para pasar a AL
                if ( isNotFull(AL) ) {
                        shr BL, 1
                        rcl AX, 1       // rotación con el carry
                } else {                // AL está completo, hay que guardar el contenido en cadeZip
                        cadeZip[i] = AL
                        i++
                        AX = 1          // ¡Importante!*
                }
        }
        j++
        BL = cadeChar[j]                // obtengo la siguiente letra
}
```

(*) Es necesario para saber en qué momento AL estará completo y por ende se necesitará pedir nueva letra, pues AH será igual a **1**, ya que se shiftearán las codificaciones de BL en AX de derecha a izquierda. Por ejemplo:

Al comenzar el primer ciclo, la primer letra sería una *A*, entonces BL tendría el valor 65, en binario *1000001*:

BX	
BH	**BL**
00000000	01000001

AX	
AH	**AL**
00000000	00000001

Luego se debe buscar la codificación del valor 65 y colocarlo en BL, seguido de shiftear con 0 de derecha a izquierda el registro AX:

BX	
BH	**BL**
00000000	00000011

AX	
AH	**AL**
00000000	00000010

A continuación todos los bits de BL se pasarán a AX, y esto se repetirá varias veces hasta que AX tenga el siguiente estado:

AX	
AH	**AL**
0000000**1**	01101101

Es aquí cuando ya no hay mas lugar en AL y por lo tanto se debe mover su contenido al arreglo *cadeZip* y volver AL a su estado original.
En el caso en que aun queden bits en BL, luego de restaurar AX, se continuarán colocando los 1's restantes hasta agotar BL.