UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

CURSO 2021 - 2022

“PROYECTO DE PROGRAMACIÓN EN ENSAMBLADOR”

AUTORES:

Julio Manso Sánchez-Tornero c200320

Nihel Kella Bouziane c200315

IDENTIFICADOR:

mansokella

Índice

1. Introducción

El MC88110 es un microprocesador RISC superescalar que forma parte de la familia 88000 de *Motorola*. Es capaz de iniciar dos instrucciones cada ciclo de reloj, respetando siempre la apariencia de ejecución secuencial del programa a través del mecanismo de *pipeline* del secuenciador. Las instrucciones se despachan hasta diez unidades funcionales que trabajan en paralelo. El simulador del MC88110 que se utiliza en este proyecto permite configurar distintos parámetros de la memoria principal, de las memorias caché de instrucciones y datos y de la CPU.

El proyecto consiste en la programación de un conjunto de rutinas que realizan la compresión y descompresión de un texto definido en memoria mediante una versión simplificada de uno de los compresores sin pérdidas habituales, del tipo de zip, gzip.

El texto a comprimir estará almacenado en memoria y no en un archivo, ya que el objetivo del proyecto es conocer las posibilidades del procesador y no el uso de periféricos como son los discos de almacenamiento ni las ayudas que proporciona un sistema operativo. De hecho, el software utilizado para el proyecto solamente emula un procesador y su memoria, no incluyendo la simulación de ningún dispositivo periférico.

Además, entre los objetivos del proyecto se encuentra aprender a depurar nuestras rutinas para comprobar el funcionamiento correcto del programa mediante la elaboración de un conjunto de casos de prueba. También se llevará a cabo un informe que incluya la bitácora de trabajo, así como lo aprendido y destacable durante el desarrollo del proyecto.

2. Histórico del desarrollo de las rutinas

Ambos miembros del grupo realizaron el trabajo de manera conjunta y de forma telemática. Fueron dedicadas 53 horas totales de trabajo.

En primer lugar revisamos el enunciado del proyecto así como las subrutinas que había que realizar junto al emulador de la *arquitectura 88110* y su juego de instrucciones.

Realizar esta tarea nos tomó un par de días ya que no entendíamos muy bien cómo funcionaba dicho juego de instrucciones.

31/10/2021 (Comienzo de Proyecto)

31/10/2021: (La duración fue de 3 horas aprox.)

Como se ha comentado anteriormente, este día fue dedicado a mirar el juego de instrucciones de la arquitectura 88110 para tener una toma de contacto previa a programar. De esta manera lograríamos entender satisfactoriamente lo que teníamos que hacer. Tuvimos dificultades sobre todo en el tratamiento de la pila y con el funcionamiento de las macros.

6/11/2021: (la duración fue de 2 horas aprox.)

Comenzamos con la primera subrutina *LongCad*. No encontramos dificultades en la implementación de la subrutina, pero sí en entender e implementar las *MACROS (LEA, POP, PUSH, DBNZ)*. Finalmente, conseguimos acabarla así como tener el *PPAL1* que se encargará de llamar a la subrutina.

7/11/2021 por la mañana (la duración fue de 2 horas aprox.)

En esta sesión hicimos la subrutina *BuscarCar*, además del *PPAL2* que se encargará de llamarla. No tuvimos grandes dificultades durante la implementación. El único obstáculo encontrado fue pasar los parámetros requeridos a la pila.

7/11/2021 por la tarde (la duración fue de 4 horas aprox.)

Por la tarde comenzamos la implementación de la subrutina *CoincidenCad* y redactamos el *PPAL3* que emplearíamos para hacer las correspondientes pruebas. No encontramos dificultades a excepción de que no tuvimos en cuenta el caso en el que ambas palabras fueran iguales. En ese caso el programa entraba en un bucle infinito. Añadimos al código lo que faltaba y lo conseguimos acabar.

8/11/2021 (la duración fue de 2 horas aprox.)

En esta sesión realizamos pruebas para comprobar que las subrutinas estaban implementadas correctamente. Estas pruebas se encuentran en el *punto 3* de la memoria.

ENTREGA HITO Nº1 : 08/11/2021 -> Fallan 11 pruebas de la subrutina BuscaCar

9/11/2021 (la duración fue de 15 min aprox.)

En esta sesión revisamos la subrutina *BuscaCar* para averiguar qué ha podido pasar para que no pase ninguna de las pruebas. Nos parecía extraño, ya que el código parecía correcto y las pruebas que hicimos previas a la entrega pasaban satisfactoriamente. Tras buscar en la documentación proporcionada ejemplos de paso de parámetros en pila, observamos que pasábamos los parámetros a la pila en orden inverso (es decir, de izquierda a derecha). Por tanto, procedimos a hacer las modificaciones correspondientes en la subrutina y, seguidamente, realizamos pruebas para verificar que el comportamiento era correcto.

17/11/2021 (la duración fue de 1 hora aprox.)

Comenzamos a realizar *BuscaMax*, tuvimos grandes problemas a la hora de entender el enunciado. Hicimos una parte de esta subrutina además de tener el *PPAL4* que se encargará de llamar a la subrutina.

18/11/2021 por la mañana (la duración fue de 2 horas aprox.)

En esta sesión continuamos realizando *BuscaMax*. Tuvimos serios problemas debido a que teníamos errores de alineamiento de palabra al no desplazar el puntero de forma correcta. También tuvimos problemas con el tratamiento de la pila a la hora de guardar los registros al llamar a otras subrutinas, es decir, al crear variables locales.

18/11/2021 por la tarde (la duración fue de 3 horas aprox.)

Terminamos de hacer *BuscaMax* e hicimos varias pruebas para ver que su funcionamiento era correcto y en efecto lo era, ya que conseguimos arreglar los errores de alineamiento a palabra y de tratamiento de la pila.
Además empezamos a plantear la subrutina *Checksum*, así como tener el *PPAL5* que se encargará de llamar a la subrutina.

19/11/2021 (la duración fue de 3 horas aprox.)

En esta sesión, conseguimos acabar *Checksum*, aunque tuvimos problemas con el caso de una cadena cuya longitud no es múltiplo de 4 al tener que desplazar una palabra para interpretarla como ceros. Posteriormente, comenzamos a realizar pruebas y encontramos una falla, ya que con esa implementación no se tenía en cuenta el caso de tener una longitud inferior a 4. La solventamos y entregamos el fichero para realizar una corrección de las que se nos permite.

CORRECCIÓN ADICIONAL Nº1 : 19/11/2021 -> Fallan 5 pruebas de la subr *Checksum*

20/11/2021 (la duración fue de 2 horas aprox.)

En esta sesión, revisamos la subrutina y arreglamos el problema encontrado: error de alineamiento a palabra en cadenas de más de 10 caracteres. Identificamos la falla en el código y hallamos una implementación mejor que ya tenía en cuenta los casos en los que la longitud sea menor que 4 o mayor que 10.

ENTREGA HITO Nº2 : 22/11/2021 -> Pasan todas las pruebas

01/12/2021 (la duración fue de 5 horas aprox.)

Comenzamos a realizar *Comprime* con problemas a la hora de entender el enunciado, ya que no tenemos muy claro cómo empezar ni qué variables locales reservar. Decidimos elaborar una parte de esta subrutina además de tener el *PPAL6* con el que haremos las pruebas correspondientes.

Comienzan a aparecer problemas debido a que no sabemos cómo concatenar los carácteres que se leen en cada iteración sin tocar la memoria. Por ejemplo, al llamar a *BuscaMax* y obtener una longitud de cadena inferior a 4, hay que copiar el carácter leído en la variable local dedicada a almacenar el texto comprimido, pero no entendemos de qué manera esto puede ocurrir. Mandamos un correo y se nos indica que debemos emplear una variable local compleja en la pila.

07/12/2021 (la duración fue de 2 horas aprox.)

Nos damos cuenta de que la implementación que estamos realizando no es correcta ya que seguimos sin entender del todo cómo concatenar los caracteres tras la compresión. Además, empleamos una dirección de memoria (p. ej. 12000) para guardar el texto comprimido, lo que está prohibido en las indicaciones del proyecto.

08/12/2021 (la duración fue de 4 horas aprox.)

A pesar de los pocos avances, tratamos de avanzar con los conceptos a tratar aun sabiendo que la implementación no es del todo correcta. De esta manera pretendemos que con el tiempo y la toma de contacto con problemas que aparecerán más adelante pudiésemos afinar la forma de implementar la subrutina.

09/12/2021 (la duración fue de 2 horas aprox.)

Comenzamos a entender la manera de tratar la pila, reservando una zona que almacene el texto comprimido en la pila. Por lo que calculamos la memoria necesaria en el peor caso (no hay compresión) y la reservamos para almacenar cada carácter del texto comprimido. Además, actualizamos las variables locales que teníamos.

11/12/2021 (la duración fue de 2 horas aprox.)

Se consiguen avances con la compresión de los caracteres pero no conocemos la manera de tratar el mapa de bits al no saber cómo escribir 0 y 1 a nivel de bit. Escribimos un correo especificando este problema.

13/12/2021 (la duración fue de 2 horas aprox.)

Se plantea una manera de rellenar un byte y se observa que funciona. Surgen complicaciones de por medio que se terminan resolviendo. Por lo tanto, la subrutina *Comprime* queda acabada en esta sesión.

15/12/2021 (la duración fue de 2 horas aprox.)

En esta sesión comenzamos a plantear la subrutina *Descomprime*. Redactamos el *PPAL7* y empezamos el planteamiento. La primera duda que surge tiene que ver con la lectura de los bits del mapa de bits, es decir, cómo leer de un byte los 8 bits que tiene para saber cuándo se debe copiar el carácter y cuándo hay que emplear la referencia al carácter para copiarlo. Tras mandar un correo, logramos entender el procedimiento a seguir.

CORRECCIÓN ADICIONAL Nº2 : 15/12/2021 -> Falla 1 prueba de *BuscaMax* y 1 de *Comprime*. (Se cree que el fallo en *Comprime* está relacionado con el fallo de *BuscaMax*)

16/12/2021 (la duración fue de 2 horas aprox.)

Tras ver los resultados, quedamos extrañados ya que la última vez que hicimos una corrección adicional, todas las pruebas de *BuscaMax* pasaban. Esto es porque el día 25/11/2021 se añadió un test que nos ayudaría a detectar un problema en *BuscaMax* que

aparecería en *Comprime* de no solventarlo. Se decide dejar eso para otro momento y continuar con *Descomprime*. Se logra acabar la subrutina tras pasar la siguiente corrección especificada.

CORRECCIÓN ADICIONAL Nº3 : 16/12/2021 -> Falla 1 prueba de *BuscaMax*, 1 de *Comprime* y 9 de *Descomprime*. (No se han realizado aún cambios en *BuscaMax* y *Descomprime* aún sin terminar)

17/12/2021 (la duración fue de 4 horas aprox.)

Se comienza a implementar la subrutina *Verifica*. Para ello, y siguiendo la metodología empleada durante todo el proyecto, escribimos el *PPAL8*, el último del proyecto. Hicimos la llamada a *LongCad* para calcular así el número de bytes del mapa de bits a partir de la fórmula especificada en el enunciado del proyecto. También se reservan las variables locales oportunas. Se reserva espacio suficiente en la pila para guardar el resultado de la llamada a *Comprime* y el de la llamada a *Descomprime.* Después, se realiza la llamada a *Checksum,* primero para el texto original, y luego para el texto obtenido tras la compresión y descompresión del mismo. Por tanto, no se han encontrado complicaciones en la realización de esta subrutina. Retomando el error en *BuscaMax*, decidimos hacer una corrección en el código añadiendo que, en el caso de encontrar una coincidencia mayor a 255, se actualizara la longitud de la coincidencia a 255 (limitándola).

CORRECCIÓN ADICIONAL Nº4 : 17/12/2021 -> Falla 1 prueba de *BuscaMax*, 1 de *Comprime* y 1 de *Verifica*. (Sin mejoras tras modificar *BuscaMax*, pasan pruebas de *Descomprime*)

18/12/2021 por la mañana (la duración fue de 1 hora aprox.)

Al no saber lo que puede estar ocurriendo con el test de *BuscaMax* añadido el día 25/11/2021, decidimos leer de nuevo el enunciado del proyecto. Nos dimos cuenta de que tal vez habíamos podido malinterpretar el enunciado, pensando que habría que terminar el bucle al contar 255 caracteres leídos. Al continuar leyendo, entendimos que la longitud de la coincidencia debía estar limitada a 255 caracteres, por lo que en el caso en el que se encontrara una coincidencia mayor o igual a 255, se actualizaría la longitud a 255 y se finalizaría el bucle. Lo corregimos en el código y esperamos confirmación de los profesores con una aclaración de lo que se espera de la subrutina.

18/12/2021 por la tarde (la duración fue de 2 horas aprox.)

En esta sesión realizamos pruebas para comprobar que las subrutinas estaban implementadas correctamente. Estas pruebas se encuentran en el *punto 3* de la memoria.

20/12/2021 por la tarde (la duración fue de 30 min aprox.)

Dedicamos está sesión para realizar las últimas revisiones y pruebas antes de la última corrección.

ÚLTIMA CORRECCIÓN: 20/12/2021 -> Falla 1 prueba de *Verifica* y el resto de pruebas pasan.

20/12/2021 por la noche (la duración fue de 15 min aprox.)

Tras ver los resultados de la corrección, decidimos no modificar más el fichero, por si se cometiera algún error y no pudiésemos modificarlo al no tener más correcciones disponibles.

ENTREGA DEFINITIVA: 22/12/2021

Última actualización: 21/12/2021 a las 11:36

3. Conjuntos de casos prueba

Para todos los casos de prueba, iniciamos el puntero de pila (SP) en la dirección de memoria 0xEA60 (60000 en decimal) mediante el mandato LEA (r30, 60000). Las pruebas reflejadas en este conjunto han sido extraídas de los ejemplos de casos de prueba publicados por el departamento. Agrupamos las pruebas según la subrutina:

3.1. LongCad

1. Caso 1

Llama a *LongCad* pasándole como parámetro la cadena **"Pienso, luego existo\0AA"** con r29 inicializado a 0.

Resultado. r29= 20(0x14)

2. Caso 2

Llama a *LongCad* pasándole como parámetro la cadena **"Pablito clavo un clavito &&&&\0"** con r29 inicializado a 0.

Resultado. r29= 29 (0x1D)

3.2. BuscaCar

1. Caso 1

Llama a *BuscarCar* pasándole como parámetro la cadena "**Pienso, luego existo\0AA**" por referencia (20000), la 'o' como char por valor (111, 0x6F), el FROM '2' por valor (2, 0x02) y el TO '13' por valor (13, 0x0D) con r29 inicializado a 0.

Resultado. r29= 5 (0x05)

2. Caso 2

Llama a *BuscarCar* pasándole como parámetro la cadena "**Pienso, luego existo\0AA**" por referencia (20000), la '?' como char por valor (63, 0x3F), el FROM '2' por valor (2, 0x02) y el TO '13' por valor (13, 0x0D) con r29 inicializado a 0.

Resultado. r29= 13 (0x0D) devuelve *to* ya que no encuentra en carácter.

3.3. CoincidenCad

1. Caso 1

Llama a *BuscarCar* con r29 inicializado a 0 y pasándole como parámetros:

Cadena1: "**Pienso, luego existo\0AA**"

Cadena2: "**Pienso, luego no existo\0AA**"

Resultado. r29= 14 (0x0E)

2. Caso 2

Llama a *BuscarCar* con r29 inicializado a 0 y pasándole como parámetros:

Cadena1: "**Pienso, luego existo\0AA**"

Cadena2: "**Pienso, luego existo\0AA**"

Resultado. r29= 20 (0x14)

3.4. BuscaMax

1. Caso 1

Llamamos a *BuscaMax* pasándole como parámetros de entrada la cadena especificada abajo por referencia (26500) y el max 59 por valor (0x3B) y como parámetro de salida jj 0 por referencia (27000) con r29 inicializado a 0.

Cadena: **"Nuestra envidia siempre dura más que la felicidad de quien envidiamos\0"**

Resultado. r29 : 7 jj : 08

2. Caso 2

Llamamos a *BuscaMax* pasándole como parámetros de entrada la cadena especificada abajo por referencia (26000) y el max 2 por valor (0x02) y como parámetro de salida jj 0 por referencia (27000) con r29 inicializado a 0.

Cadena: **"hola\0"**

Resultado. r29 : 0 jj : -1

3.5. Checksum

1. Caso 1

Llamamos a *Checksum* pasándole como parámetros de entrada la cadena especificada abajo de 10 caracteres por referencia (28000) y con r29 inicializado a 0.

Cadena: **"SOLO SE QUE NO SE NADA\0AA"**

Resultado. r29 : 2445A89B

2. Caso 2

Llamamos a *Checksum* pasándole como parámetros de entrada la cadena especificada abajo por referencia (29000) y con r29 inicializado a 0.

Cadena: **"SOL\0"**

Resultado. r29 : 004C4F53

3.6. Comprime

1. Caso 1

Llamamos a *Comprime* pasándole como parámetros de entrada la cadena especificada abajo de 32 caracteres por referencia (31000) y un parámetro de salida por referencia (30000). *Nota: la cadena no admite compresión.*

Cadena: **"Supercalifragilisticoespialidoso\0"**

Resultado. r29 : 41 (0x29) → 32 (*LongCad*) + 5 (Cabecera) + 4 (Mapa de bits)

```
88110> v 31000

    30992                                53757065    7263616C
    31008       69667261    67696C69     73746963    6F657370
    31024       69616C69    646F736F     00000000    00000000
    31040       00000000    00000000     00000000    00000000
    31056       00000000    00000000     00000000    00000000
    31072       00000000    00000000
88110> v 30000

    30000       20000109    00000000     00537570    65726361
    30016       6C696672    6167696C     69737469    636F6573
    30032       7069616C    69646F73     6F000000    00000000
    30048       00000000    00000000     00000000    00000000
    30064       00000000    00000000     00000000    00000000
88110>
```

2. Caso 2

Llamamos a *Comprime* pasándole como parámetros de entrada la cadena especificada abajo de 66 caracteres por referencia (31000) y un parámetro de salida por referencia (30000). *Nota: la cadena admite compresión.*

Cadena: **"cuando yo digo Diego, digo digo, y cuando yo digo digo, digo Diego\0"**

Resultado. r29 : 48 (0x30) → 40 (Compr)+ 5 (Cabecera) + 3 (Mapa de bits)

```
88110> v 31000

    30992                                6375616E    646F2079
    31008       6F206469    676F2044     6965676F    2C206469
    31024       676F2064    69676F2C     20792063    75616E64
    31040       6F20796F    20646967     6F206469    676F2C20
    31056       6469676F    20446965     676F0000    00000000
    31072       00000000    00000000
88110> v  30000

    30000       42000108    0000061C     6375616E    646F2079
    30016       6F206469    676F2044     6965676F    2C090006
    30032       0A00042C    20792000     000F1B00    060A000A
    30048       00000000    00000000     00000000    00000000
    30064       00000000    00000000     00000000    00000000
88110>
```

3.7. Descomprime

1. Caso 1

Llamamos a *Descomprime* pasándole como parámetros de entrada la cadena especificada abajo de 22 caracteres tras la compresión por referencia (12000) y un parámetro de salida por referencia (10000). *Nota: la cadena no admite compresión.*

Cadena: **"Esternocleidomastoideo\0"**

CMPR: data 0x07010016, 0x45000000, 0x72657473, 0x6C636F6E
data 0x6F646965, 0x7473616D, 0x6564696F, 0x0000006F

Resultado. r29 : 22 (0x16)

```
88110> v 12000

    12000       16000107    00000045    73746572    6E6F636C
    12016       6569646F    6D617374    6F696465    6F000000
    12032       00000000    00000000    00000000    00000000
    12048       00000000    00000000    00000000    00000000
    12064       00000000    00000000    00000000    00000000
88110> v 10000

    10000       45737465    726E6F63    6C656964    6F6D6173
    10016       746F6964    656F0000    00000000    00000000
    10032       00000000    00000000    00000000    00000000
    10048       00000000    00000000    00000000    00000000
    10064       00000000    00000000    00000000    00000000
88110>
```

2. Caso 2

Llamamos a *Descomprime* pasándole como parámetros de entrada la cadena especificada abajo de 10 caracteres por referencia (12000) y un parámetro de salida por referencia (10000). *Nota: la cadena no admite compresión.*

Cadena: **"cuando yo digo Diego, digo digo, y cuando yo digo digo, digo Diego\0"**

CMPR: data 0x08010042, 0x1C060000, 0x6E617563, 0x79206F64
data 0x6964206F, 0x44206F67, 0x6F676569, 0x0600092C
data 0x2C04000A, 0x00207920, 0x001B0F00, 0x0A000A06

Resultado. r29 : 66 (0x42)

```
88110> v 12000

    12000       42000108    0000061C    6375616E    646F2079
    12016       6F206469    676F2044    6965676F    2C090006
    12032       0A00042C    20792000    000F1B00    060A000A
    12048       00000000    00000000    00000000    00000000
    12064       00000000    00000000    00000000    00000000
88110> v 10000

    10000       6375616E    646F2079    6F206469    676F2044
    10016       6965676F    2C206469    676F2064    69676F2C
    10032       20792063    75616E64    6F20796F    20646967
    10048       6F206469    676F2C20    6469676F    20446965
    10064       676F0000    00000000    00000000    00000000
88110>
```

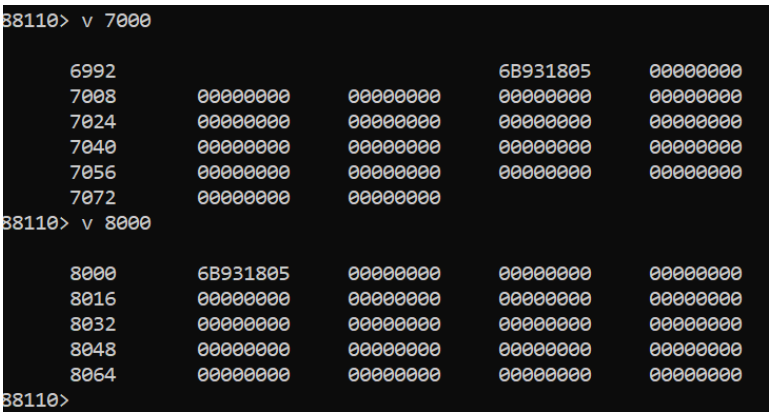
3.8. Verifica

1. Caso 1

Llamamos a *Verifica* pasándole como parámetros de entrada la cadena especificada abajo de 10 caracteres por referencia (9000) y como parámetros de salida para el *Checksum* 1 la referencia 7000 y para el *Checksum* 2 la referencia 7004. *Nota: la cadena no admite compresión.*

Cadena: **"Esternocleidomastoideo\0"**

Resultado. r29 : 0 (0x0)

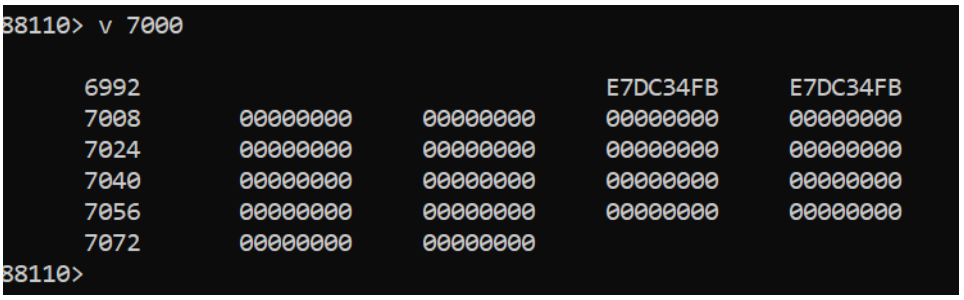


2. Caso 2

Llamamos a *Verifica* pasándole como parámetros de entrada la cadena especificada abajo por referencia (9000) y como parámetros de salida para el Checksum 1 la referencia 7000 y para el Checksum 2 la referencia 7004. *Nota: la cadena admite compresión.*

Cadena: **"cuando yo digo Diego, digo digo, y cuando yo digo digo, digo Diego\0"**

Resultado. r29 : 0 (0x0)



4. Observaciones finales y comentarios personales

Durante la realización del proyecto hemos podido ir observando la dificultad que conlleva desarrollar un programa en ensamblador. Si bien es cierto que a medida que avanzaba el proyecto la dificultad de las subrutinas se incrementaba, hemos de decir que ello fomentaba la adquisición progresiva de soltura, lo que lo hace más ameno.

También debemos comentar que, a pesar de que al principio del proyecto nos asustó el hecho de tener correcciones limitadas tanto en número como en tiempo, luego no resultó ser un gran problema al tener la opción de depurar el código mediante el simulador y los comandos pertinentes, ya que contábamos además con el conjunto de casos prueba proporcionado por el departamento. Sí que consideramos que para *Comprime* hubiese venido bien para poder encontrar errores, ya que depurar con más de 5.000.000 ciclos en cadenas de más de 255 caracteres (e incluso menos) es imposible.

El proyecto en sí estaba muy bien estructurado y las explicaciones en el propio enunciado eran bastante claras, excepto en el caso de *BuscaMax* que consideramos que era complejo de entender (un esquema de lo que se espera de la subrutina hubiera aclarado las dudas). El manual del emulador 88110 es muy útil, a excepción del punto referido a las instrucciones de campo de bit. No se entiende prácticamente nada de lo que se espera de las instrucciones y realmente lo averiguamos probándolo. Creemos que añadir esquemas y ejemplos concretos facilitaría la progresión en este sentido. Respecto al escalado de dificultad, es progresivo y coherente.

Agradecemos también la atención del profesorado a lo largo de todo el proyecto, ya que en particular empleamos el correo electrónico para preguntar las dudas y plantear los problemas que nos surgían y la atención era, además de rápida, eficaz.

Cabe destacar que el tema del proyecto (compresión de texto) es muy interesante y creemos que eso también ha fomentado que trabajemos en el proyecto de manera regular. En un principio lo vimos como un verdadero reto, implementar algo semejante al *WinRAR*, y haber conseguido algo, por pequeño que sea, nos hace sentir satisfechos.