UNIVERSIDAD POLITÉCNICA DE MADRID

PROYECTO ENSAMBLADOR

MEMORIA

ESTRUCTURA DE COMPUTADORES

AUTORES

Fernando Luna Marcuello, 190394

Borja Pérez-Villacastín Palacín, 190395

ÍNDICE

1. Introducción

Este documento presenta los pasos que hemos seguido a la hora de realizar el proyecto, empezando por un planteamiento general de las subrutinas, realizando el pseudocódigo de las mismas para posteriormente implementarlas en el ensamblador 88110. A medida que avanzamos con la implementación, codificamos casos de prueba para verificar el funcionamiento de las subrutinas. Sin embargo, el orden que seguiremos en este documento será el indicado en el índice de contenido.

Por lo tanto, presentaremos en primer lugar, los pseudocódigos que hemos llevado a cabo para cada subrutina, antes de empezar a programar en ensamblador. A continuación, las subrutinas comentadas. Por último, los casos de prueba que hemos utilizado para cada subrutina.

Los programas que hemos utilizado para la realización de este proyecto han sido Notepad++, Visual Studio Code y MINGW64. Los dos primeros para la implementación de los pseudocódgios, subrutinas y casos de prueba, y MINGW64 es una terminal de bash utilizada para la compilación del código y ejecución del emulador.

2. Diagrama de flujo o pseudocódigo

Inicialmente, para seguir una estructura a la hora de realizar los pseudocódigos, decidimos implementar las subrutinas similares a cómo sería una función Java, lenguaje de programación con el que los dos autores nos sentíamos cómodos.

En primer lugar, realizamos el pseudocódigo de las subrutinas que engloban el primer hito: LongCad, BuscaCar y CoincidenCad. Al ser estas subrutinas más cortas y sencillas, no nos complicamos a la hora de realizar los siguientes pseudocódigos:

```
LongCad(cadena) {
    // Inicializar puntero a la cadena y contador
    ptrCadena = dirección(cadena)  // r20
    contador = 0                   // r29
    char = 0                       // r3

    // Bucle para contar caracteres
    while (true) {
        char = *(ptrCadena + contador) // Cargar el carácter actual
        if (char == 0x00) {
            break // Salir del bucle si el carácter es nulo
        }
        contador++ // Incrementar el contador
    }

    // Devolver la longitud de la cadena
    return contador // r29
}
```

```
BuscaCar(caracter, cadena, from, to) {
    // Inicializar variables
    charABuscar = caracter     // r3
    ptrCadena = dirección(cadena) // r20
```

```
    inicio = from              // r29
    fin = to                   // r2
    charActual = 0             // r4

    // Bucle para buscar el carácter
    while (inicio < fin) {
        charActual = *(ptrCadena + inicio) // Obtener el carácter actual
        if (charActual == charABuscar) {
            return inicio // Retornar la posición si el carácter coincide
        }
        inicio++ // Incrementar la posición
    }

    // Si no se encuentra, retornar una indicación (por ejemplo, -1)
    return -1 // Indicando que no se encontró el carácter
}
```

```
CoincidenCad(cadena1, cadena2) {
    // Inicializar punteros a las cadenas y el índice
    ptrCadena1 = dirección(cadena1) // r21
    ptrCadena2 = dirección(cadena2) // r22
    indice = 0                      // r29
    char1 = 0                       // r3
    char2 = 0                       // r2

    // Bucle para comparar las cadenas
    while (true) {
        char1 = *(ptrCadena1 + indice) // Obtener el carácter de la primera cadena
        char2 = *(ptrCadena2 + indice) // Obtener el carácter de la segunda cadena

        // Verificar si alguno es terminador
        if (char1 == 0x00 || char2 == 0x00) {
            break // Salir del bucle si uno es terminador
        }

        // Comparar los caracteres
        if (char1 != char2) {
            return false // Las cadenas no coinciden
        }

        indice++ // Incrementar el índice
    }

    // Las cadenas coinciden si llegamos aquí
    return true
}
```

En segundo lugar, comenzamos con el pseudocódigo de Comprime, en este punto tuvimos que repetir varias veces la implementación de la subrutina.Sin embargo, implementamos el código en ensamblador sobre este pseudocódigo:

```
 Comprime(texto, comprimido){
       crearMarcoPila();
       // 1. Determina Longitud Texto
       tamaño = LongCad(texto)
       // 2. Reserva en pila espacio
       //hay que reservar el tamaño en exceso a multiplo de 4 para que cuadre bien
       reservarPila(tamaño)

       // 3. inicializar variables
```

```
ptrTexto = /Texto
ptrPila = r31-1

// 4. copiar los 8*M primeros caracteres del texto al final de la pila
x=8*M
while(x!=0){
	(ptrTexto) --> (ptrPila)
	dirPila--
	ptrTexto++
	x--
}

// 5. Recorre  los caracteres del texto
bitMBits=7
mBits= /comprimido + 5
bytesMBits = 0
byte=00000000
while((ptrTexto)!=0x00){
	L = BuscaMax(Texto, ptrTexto, P)
	if(L < 4){
		(ptrTexto) --> (ptrPila)
		ptrPila--
		ptrTexto++
		clear(byte, bitMBits)
	} else{
		P --> (ptrPila)
		ptrPila= ptrPila-2
		L --> (ptrPila)
		ptrPila--
		ptrTexto= ptrTexto+L
		set(byte, bitMBits)
	}
	bitMBits--
	if(bitMBits==-1) {
		byte --> (mBits[bytesMBits])
		byte=00000000
		bitMBits=7
		bytesMBits++
	}
}

// 7. Ponemos en la cabecera el tamaño del texto comprimido
tamañoComprimido = 8*bytesMBits + bitMBits
cabecera = /comprimido
cabecera[0y1] = tamañoComprimido
// 8. Copiamos valor M
cabecera[2] = 1

// 6. copiamos el ultimo byte del mapa de bits
if(bitMBits != 0){
	byte --> (cabecera[bytesMBits])
	bytesMBits++
}

// 9. Pone en cabecera el comienzo del texto
comienzoComprimido= bytesMBits + 5
cabecera[3y4] = comienzoComprimido

// 10. Lo pasa de la pila a comprdo
ptrPila= r31-1
tamañoComprimido = tamañoComprimido 8*M
while(tamañoComprimido!=0){
	(ptrPila) --> (cabecera+comienzoComprimido)
	ptrPila--
	comienzoComprimido++
```

```
	}
	r29=comienzoComprimido
	destruirMarcoPila();
}
```

Antes de la implementación final de Comprime, comenzamos la implementación del pseudocódigo de Descomprime, siendo este el resultado:

```
Descomprime(com, desc){
	 // 1. Inicializar variables
	ptrCabecera = /com
	ptrDesc = /desc
	ptrMBits = /com + 5
	bitMBits = 7
	bitActivado= 128 //solo bit 7 activado
	posComienzoCom = ptrCabecera[3y4]
	ptrCom= /com + posComienzoCom
	tamDesc = 0

	// 2. Copiar a desc los primeros 8*M cars de com
	M = (ptrCabecera+2)
	x = M*8
	while(x!=0){
		(ptrCom) --> (ptrDesc+tamDesc)
		ptrCom++
		tamDesc++
		x--
	}

	// 3. Bucle recorriendo com y MBits
	byte = (ptrMBits)
	while( (ptrCom) != 0x00 ){
		if( byte AND bitActivado == 0){
			(ptrCom) --> (ptrDesc+tamDesc)
			ptrCom++
			tamDesc++
		}
		else{
			P = ptrCom[0y1]
			L = ptrCom[2]
			while(L!=0){
				(ptrDesc+P) --> (ptrDesc+tamDesc)
				P++
				tamDesc++
				L--
			}
			ptrCom = ptrCom + 3
		}
		bitMBits--
		if(bitMBits==-1){
			bitMBits=7
			bitActivado= 128
			ptrMBits++
			byte = (ptrMBits)
		} else {
			bitActivado= bitActivado/2
		}
	}

	// 4. Añade terminador 0x00
	(ptrDesc+tamDesc) <-- 0x00
```

```
	// 5. devuelve longitud texto descomprimido
	r29 = tamDesc
 }
```

Por último, decidimos implementar la subrutina Verifica una vez terminadas Comprime y Descomprime. El pseudocódigo utilizado para Verifica fue el siguiente:

```
Verifica (texto, long1, long2){
	crearMarcoPila()
	tam= LongCad(texto)
	long1=tam
	tamAjustado= ajustar(tam)
	r21 = reservarPila(tamAjustado)
	r20= /texto
	Comprime(r20,r21)
	r22= reservarPila(tamAjustado)
	Descomprime(r21,r22)
	long2=r29
	if(long1!=long2){
		return -1
	} else{
		if(coincidencad!=long1) return -2
	}
	destruirMarcoPila();
	return 0;
 }
```

3. Casos de prueba

A medida que implementamos las subrutinas, tambíen probamos su funcionamiento diseñando los casos de prueba para cada una. En primer lugar, diseñamos casos de prueba que verificaban el correcto funcionamiento de las subrutinas del primer hito. Asimismo, creamos casos de prueba para comprobar el funcionamiento de Comprime, Descomprime y Verifica, las cuales nos devolvían los valores esperados, a pesar de que en el corrector automático las subrutinas no superasen la totalidad de las pruebas.

Estos son los casos de prueba para las subrutinas del primer hito:

LongCad:

Caso 1. Llamada a LongCad pasándole como parámetro la cadena “Prueba\0” con r29 inicializado a 100.

```
texto:      data "Prueba\0"

PPALLC:       LEA(r30, 86012)
              LEA(r20, texto)
              PUSH(r20)
              bsr LongCad
              POP(r20)
              stop
```

Inicial:

```
88110>
 PC=1056       bsr      57       Tot. Inst: 8   ¡¡ Ciclo : 114
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000100 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000064 h R30 = 00014FF4 h R31 = 00000000 h
```

Resultado: r29 = 6 (0x6)

```
88110> e
       Fin ejecución
 PC=1064       or       r30,r00,20472    Tot. Inst: 47  ¡¡ Ciclo : 693
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000424 h R02 = 00000000 h R03 = 00000000 h R04 = 00005AA4 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000100 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000006 h R30 = 00014FF4 h R31 = 00000000 h
```

Caso 2. Llamada a LongCad pasándole como parámetro la cadena “El Real Madrid ha ganado la final de la Champions\0”

```
texto:      data "El Real Madrid ha ganado la final de la Champions\0"

PPALLC:         LEA(r30, 86012)
                LEA(r20, texto)
                PUSH(r20)
                bsr LongCad
                POP(r20)
                stop
```

Inicial:

```
88110>
 PC=1048       bsr      56       Tot. Inst: 6   ¡¡ Ciclo : 88
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000100 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF4 h R31 = 00000000 h
```

Resultado: r29 = 49 (0x31)

```
88110> e
        Fin ejecución
 PC=1056       or       r30,r00,20472    Tot. Inst: 260 ¡¡ Ciclo : 3849
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 0000041C h R02 = 00000000 h R03 = 00000000 h R04 = 00005AA4 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000100 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000031 h R30 = 00014FF4 h R31 = 00000000 h
```

BuscaCar:

Caso 3. Llamada a BuscaCar pasándole una cadena de 30 caracteres y trata de localizar el situado en la posición 0 y en otras. El valor del parámetro from es 0.

```
C:  data "A"
REF: data "AAAABBBBCCCCDDDDEEEE_Fin.__.\0"
from: data 0
to: data 25
PPALBC:     LEA(r30, 0x14FF8)
            LOAD(r20, C)
            LEA(r21, REF)
            LOAD(r22, from)
            LOAD(r23, to)
            PUSH(r23)
            PUSH(r22)
            PUSH(r21)
            PUSH(r20)
            bsr BuscaCar
            POP(r20)
            POP(r21)
            POP(r22)
            POP(r23)
            stop
```

Inicial:

```
88110>
 PC=1148       bsr      73       Tot. Inst: 21  ¡¡ Ciclo : 337
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000041 h R11 = 0000030C h R12 = 00000000 h
 R13 = 00000019 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000000 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FE8 h R31 = 00000000 h
```

Resultado: r29 = 0 (0x0)

```
88110> e
        Fin ejecución
 PC=1188       or       r30,r00,20472    Tot. Inst: 43  ¡¡ Ciclo : 704
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000480 h R02 = 00000019 h R03 = 00000041 h R04 = 00000041 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000041 h R11 = 0000030C h R12 = 00000000 h
 R13 = 00000019 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 0000030C h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF8 h R31 = 00000000 h
```

Caso 4. Llamada a BuscaCar pasándole una cadena de 30 caracteres y trata de localizar el situado en la posición 0 y en otras. El valor del parámetro from es mayor que cero.

```
C:  data "A"
REF: data "AAAABBBBCCCCDDDDEEEE_Fin.__.\0"
from: data 4
to: data 25
PPALBC:     LEA(r30, 0x14FF8)
            LOAD(r20, C)
            LEA(r21, REF)
            LOAD(r22, from)
            LOAD(r23, to)
            PUSH(r23)
            PUSH(r22)
            PUSH(r21)
            PUSH(r20)
            bsr BuscaCar
            POP(r20)
            POP(r21)
            POP(r22)
```

```
            POP(r23)
            stop
```

Inicial:

```
88110>
 PC=1148       bsr      73       Tot. Inst: 21  ¡¡ Ciclo : 337
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000041 h R11 = 0000030C h R12 = 00000004 h
 R13 = 00000019 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000000 h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FE8 h R31 = 00000000 h
```

Resultado: r29 = 25 (0x19)

```
88110> e
        Fin ejecución
 PC=1188       or       r30,r00,20472    Tot. Inst: 207 ¡¡ Ciclo : 3016
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000480 h R02 = 00000019 h R03 = 00000041 h R04 = 0000002E h
 R05 = 00000000 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000041 h R11 = 0000030C h R12 = 00000004 h
 R13 = 00000019 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 0000030C h
 R21 = 00000000 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000019 h R30 = 00014FF8 h R31 = 00000000 h
```

CoincidenCad:

Caso 5. Llama a CoincidenCad, pasándole dos cadenas de 10 caracteres que comienzan por un carácter diferente. Todos los registros parten de valores iniciales distintos de cero.

```
Cad1: data "1234567890\0"
Cad2: data "0123456789\0"

PPALCC:     LEA(r30, 0x14FF8)
            LEA(r20, Cad1)
            LEA(r21, Cad2)
            PUSH(r21)
            PUSH(r20)
            bsr CoincidenCad
            POP(r20)
```

```
            POP(r21)
            stop
```

Inicial:

```
88110>
 PC=1228       bsr      83       Tot. Inst: 10  ¡¡ Ciclo : 149
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000334 h
 R21 = 00000340 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF0 h R31 = 00000000 h
```

Resultado: r29 = 0 (0x0)

```
88110> e
        Fin ejecución
 PC=1252       or       r30,r00,20472    Tot. Inst: 30  ¡¡ Ciclo : 463
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 000004D0 h R02 = 00000030 h R03 = 00000031 h R04 = 00005668 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000334 h
 R21 = 00000340 h R22 = 00000340 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF8 h R31 = 00000000 h
```

Caso 6. Llama a CoincidenCad, pasándole dos cadenas de tamaño mediano en las que coinciden cerca de la primera mitad de sus caracteres.

```
   Cad1: data "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua\0"

   Cad2: data "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nemo enim
ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia
consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.\0"


   PPALCC:     LEA(r30, 0x14FF8)
               LEA(r20, Cad1)
               LEA(r21, Cad2)
               PUSH(r21)
               PUSH(r20)
               bsr CoincidenCad
               POP(r20)
```

```
            POP(r21)
            stop
```

Inicial:

```
88110>
 PC=2252       bsr      83       Tot. Inst: 10  ¡¡ Ciclo : 149
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000334 h
 R21 = 000003B0 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF0 h R31 = 00000000 h
```

Resultado: r29 = 55 (0x37)

```
88110> e
        Fin ejecución
 PC=2276       or       r30,r00,20472    Tot. Inst: 580 ¡¡ Ciclo : 8603
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 000008D0 h R02 = 0000002E h R03 = 0000002C h R04 = 00005998 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000334 h
 R21 = 000003B0 h R22 = 000003B0 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000037 h R30 = 00014FF8 h R31 = 00000000 h
```

Comprime y Descomprime:

A continuación, implementamos los casos de prueba para Comprime y Descomprime. Hemos realizado un programa principal conjunto que prueba las subrutinas Comprime y Descomprime en la misma ejecución, se muestra en el siguiente código:

```
PPALC:      LEA(r30,0x14FF8)
            or r31,r30,r30
            LEA(r20,comprdo)
            PUSH(r20)
            LEA(r21,texto)
            PUSH(r21)
            bsr Comprime
            or r18,r29,r0
            POP(r21)
            POP(r20)
            LEA(r20,descom)
```

```
            PUSH(r20)
            LEA(r21,comprdo)
            PUSH(r21)
            bsr Descomprime
            POP(r21)
            POP(r20)
            stop
```

Caso 7. Llama a Comprime pasándole una cadena de caracteres como la del enunciado del proyecto.

```
  texto:      data "tres tristes tigres comen trigo en un trigal, el primer tigre
que...\0"
  org 0x188
  comprdo:    data 0
```

Inicial:

```
88110>
 PC=2320       bsr      82       Tot. Inst: 11  ¡¡ Ciclo : 162
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000188 h
 R21 = 00000134 h R22 = 00000000 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FF0 h R31 = 00014FF8 h
```

```
88110> v 0x100 18

      256       74726573     20747269     73746573     20746967
      272       72657320     636F6D65     6E207472     69676F20
      288       656E2075     6E207472     6967616C     2C20656C
      304       20707269     6D657220     74696772     65207175
      320       652E2E2E     00000000
```

Resultado: r29 = 70 (0x64)

```
88110> e
       Alcanzado punto de ruptura, dirección 2376
 PC=2376       bsr      227      Tot. Inst: 28493       ¡¡ Ciclo : 439075
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000914 h R02 = 00000006 h R03 = 0000000B h R04 = 0000002E h
 R05 = 00000044 h R06 = 00000001 h R07 = 00005AA4 h R08 = 00000000 h
 R09 = 00000000 h R10 = 0B000000 h R11 = 00000043 h R12 = 00014F7C h
 R13 = 00000176 h R14 = 00000177 h R15 = 00000044 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000046 h R19 = 00000000 h R20 = 00000200 h
 R21 = 00000188 h R22 = 00014FAC h R23 = 0000018D h R24 = 00000041 h
 R25 = 00000000 h R26 = 00000000 h R27 = 000005A5 h R28 = 0000209D h
 R29 = 00000046 h R30 = 00014FF0 h R31 = 00014FF8 h
```

```
88110> v 0x188 18

       384                                4400010B    00241010
       400       00400074    72657320     74726973    74020004
       416       69670100    04636F6D     656E0400    04676F20
       432       656E2075    18000661     6C2C2065    6C207072
       448       696D6572    0C000620     7175652E    2E2E0000
```

Caso 8. Llama a Descomprime, pasándole la cadena comprimida ‘comprdo’ resultado del caso anterior. Por lo que el estado *Inicial* es el mismo que el *Resultado* del **Caso 7.**

```
org 0x200

descom:     data 0
```

Resultado: r29 = 68 (0x44)

```
88110> e
       Fin ejecución
 PC=2400       or       r30,r00,20472    Tot. Inst: 29445       ¡¡ Ciclo : 453073
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 0000094C h R02 = 00000006 h R03 = 00000040 h R04 = 00000044 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000012 h
 R09 = 00000000 h R10 = 0B000000 h R11 = 00000043 h R12 = 00014F7C h
 R13 = 00000142 h R14 = 00000143 h R15 = 00000044 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000046 h R19 = 00000000 h R20 = 00000200 h
 R21 = 00000188 h R22 = 000001CE h R23 = 00000200 h R24 = 00000041 h
 R25 = 00000000 h R26 = 00000000 h R27 = 000005A5 h R28 = 0000209D h
 R29 = 00000044 h R30 = 00014FF8 h R31 = 00014FF8 h
```

```
88110> v 0x200 18

       512       74726573    20747269     73746573    20746967
       528       72657320    636F6D65     6E207472    69676F20
       544       656E2075    6E207472     6967616C    2C20656C
       560       20707269    6D657220     74696772    65207175
       576       652E2E2E    00000000
```

Se puede comparar la cadena de las posiciones de memoria desde 0x100, la cual es el texto inicial, a la cadena de las posiciones de memoria desde 0x200, la cual es la cadena comprimida en primer lugar, esta cadena comprimida está guardada a partir de las posiciones de memoria

0x188 y posteriormente descomprimida. Todos estos casos cumplen con los requisitos pedidos en las subrutinas Comprime y Descomprime.

Caso 9. Para este caso, utilizaremos el código de la prueba anterior, con el que comprimiremos la cadena de caracteres “012345670123456701234567012345670/0” para luego descomprimirla y comprobar sus resultados guardados en las posiciones de memoria 0x188 Y 0x200.

```
org 0x100
texto:      data "012345670123456701234567012345670\0"
org 0x188
comprdo:    data 0
org 0x200
descom:     data 0
```

```
 R01 = 00000468 h R02 = 00000006 h R03 = 00000040 h R04 = 00000020 h
 R05 = 00000080 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000018 h
 R09 = 00000000 h R10 = 06000000 h R11 = 00000008 h R12 = 00014FA0 h
 R13 = 00000100 h R14 = 00000108 h R15 = 00000020 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000011 h R19 = 00000000 h R20 = 00000200 h
 R21 = 00000188 h R22 = 00000199 h R23 = 00000200 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 000005A5 h R28 = 0000209D h
 R29 = 00000020 h R30 = 00014FF8 h R31 = 00014FF8 h
88110> v 0x100

       256       30313233     34353637     30313233     34353637
       272       30313233     34353637     30313233     34353637
       288       00000000     00000000     00000000     00000000
       304       00000000     00000000     00000000     00000000
       320       00000000     00000000     00000000     00000000
88110> v 0x188

       384                                  20000106     00803031
       400       32333435     36370000     18000000     00000000
       416       00000000     00000000     00000000     00000000
       432       00000000     00000000     00000000     00000000
       448       00000000     00000000     00000000     00000000
       464       00000000     00000000
88110> v 0x200

       512       30313233     34353637     30313233     34353637
       528       30313233     34353637     30313233     34353637
       544       00000000     00000000     00000000     00000000
       560       00000000     00000000     00000000     00000000
       576       00000000     00000000     00000000     00000000
```

Como podemos observar la cadena de caracteres se almacena en la posición de memoria 0x100, luego se muestra comprimida en la posición 0x188 y finalmente se encuentra descomprimida en la posición 0x200.

Este caso ha sido determinante para entender el funcionamiento de Comprime y Descomprime, y evaluar su correcto funcionamiento, ya que aun siendo correcto el resultado de Comprime, este no se correspondía con nuestro resultado esperado.

A la hora de comprimir la cadena que hemos mencionado antes, creíamos que el resultados que se debía obtener debía ser “01234567 P(0),L(8) P(0),L(8) P(0),L(8)” (sin tener en cuenta

cabecera y Mapa de Bits), sin embargo, el resultado correcto es “01234567 P(0),L(24)”. Esta equivocación al entender Comprime ocasionaba algunos fallos en la lógica del código de Descomprime y por ende en su funcionamiento.

Tras depurar y seguir la traza generada por esta prueba conseguimos entender y arreglar finalmente el código para que funcionase correctamente.

Verifica: Por último, hemos realizado la implementación de la subrutina Verifica, siendo este el programa que hemos utilizado para los casos de prueba:

```
org 0x100
texto:  data "0123456789\0"
org 0x300
long1:  data 00
long2:  data 00

PPALV:      LEA(r30,0x14FF8)
            or r31,r30,r30
            LEA(r20,long2)
            PUSH(r20)
            LEA(r21,long1)
            PUSH(r21)
            LEA(r22,texto)
            PUSH(r22)
            bsr Verifica
            POP(r21)
            POP(r21)
            POP(r20)
            stop
```

Caso 10. Llama a Verifica pasándole una cadena de 10 caracteres que no admite compresión.

```
org 0x100
texto:  data "0123456789\0"
```

Inicial:

```
88110>
 PC=2460       bsr      273      Tot. Inst: 15  ¡¡ Ciclo : 223
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000304 h
 R21 = 00000300 h R22 = 00000100 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FEC h R31 = 00014FF8 h
```

Resultado: r29 = 0 (0x0)

```
88110> e
       Fin ejecución
 PC=2492       ld       r20,r30,r00      Tot. Inst: 30553       ¡¡ Ciclo : 469594
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 000009A0 h R02 = 00000000 h R03 = 00000000 h R04 = 00005AA4 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000044 h
 R09 = 00000000 h R10 = 0B000000 h R11 = 00000043 h R12 = 00014F10 h
 R13 = 00000142 h R14 = 00000143 h R15 = 00000044 h R16 = 00000000 h
 R17 = 00000044 h R18 = 00000048 h R19 = 00000000 h R20 = 00000304 h
 R21 = 00000300 h R22 = 00014F44 h R23 = 00014F44 h R24 = 00000041 h
 R25 = 00000000 h R26 = 00000000 h R27 = 000005A5 h R28 = 00014F44 h
 R29 = 00000000 h R30 = 00014FF8 h R31 = 00014FF8 h
```

```
88110> v 0x300 2

       768       0A000000     0A000000
```

Caso 11. Llama a Verifica pasándole una cadena de caracteres como la del enunciado del proyecto.

```
   texto:      data "tres tristes tigres comen trigo en un trigal, el primer tigre
que...\0"
```

Inicial:

```
88110>
 PC=2460       bsr      273      Tot. Inst: 15  ¡¡ Ciclo : 223
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 00000000 h R02 = 00000000 h R03 = 00000000 h R04 = 00000000 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00000000 h R08 = 00000000 h
 R09 = 00000000 h R10 = 00000000 h R11 = 00000000 h R12 = 00000000 h
 R13 = 00000000 h R14 = 00000000 h R15 = 00000000 h R16 = 00000000 h
 R17 = 00000000 h R18 = 00000000 h R19 = 00000000 h R20 = 00000304 h
 R21 = 00000300 h R22 = 00000100 h R23 = 00000000 h R24 = 00000000 h
 R25 = 00000000 h R26 = 00000000 h R27 = 00000000 h R28 = 00000000 h
 R29 = 00000000 h R30 = 00014FEC h R31 = 00014FF8 h
```

Resultado: r29 = 0 (0x0)

```
88110> e
       Fin ejecución
 PC=2492       ld       r20,r30,r00      Tot. Inst: 30553       ¡¡ Ciclo : 469594
 FL=1 FE=1 FC=0 FV=0 FR=0
 R01 = 000009A0 h R02 = 00000000 h R03 = 00000000 h R04 = 00005AA4 h
 R05 = 00000000 h R06 = 00000000 h R07 = 00005AA4 h R08 = 00000044 h
 R09 = 00000000 h R10 = 0B000000 h R11 = 00000043 h R12 = 00014F10 h
 R13 = 00000142 h R14 = 00000143 h R15 = 00000044 h R16 = 00000000 h
 R17 = 00000044 h R18 = 00000048 h R19 = 00000000 h R20 = 00000304 h
 R21 = 00000300 h R22 = 00014F44 h R23 = 00014F44 h R24 = 00000041 h
 R25 = 00000000 h R26 = 00000000 h R27 = 000005A5 h R28 = 00014F44 h
 R29 = 00000000 h R30 = 00014FF8 h R31 = 00014FF8 h
```

```
88110> v 0x300 2

       768       44000000     44000000
```

4. Resumen semanal de fechas y horas empleadas

Semana	**Horas**	**Trabajo realizado**
29/04/24 - 05/05/24	Borja: 5h Fer: 5h	Lectura y comprensión del enunciado. Implementación del primer hito.
27/05/24 - 02/06/24	Borja: 16h Fer: 13h	Implementación Comprime, Descomprime y Verifica
03/06/24 - 09/06/24	Borja: 12h Fer: 10h	Depuración y solventar errores
10/06/24 - 16/06/24	Borja: 3h Fer: 9h	Realización de la Memoria

No hemos hecho uso de tutorías, ni tampoco hemos resuelto dudas con ninguno de los profesores del proyecto.

5. Observaciones finales y comentarios personales

En primer lugar, nos gustaría recalcar que la depuración del código utilizando el programa del 88110 ha supuesto un reto complicado, pero satisfactorio una vez conseguimos visualizar los resultados esperados.

Por otro lado, la realización del proyecto en ensamblador nos ha permitido comprender cómo funciona el código en ensamblador en profundidad, siendo un lenguaje de programación con el que no estamos familiarizados. Tuvimos que ejecutar instrucción a instrucción y analizar cómo variaban las posiciones de memoria y los registros en cada ejecución hasta comprender qué cambios realizar y cómo podíamos completar las subrutinas propuestas.

En consecuencia, dedicamos mucho tiempo a resolver problemas que nos surgían al no conocer plenamente el funcionamiento del emulador, lo que incrementó las horas de trabajo dedicadas a completar las subrutinas del primer hito. Sin embargo, esto no fue un problema mayor debido al conocimiento que adquirimos con el uso de este mismo emulador en el pasado y al amplio margen de tiempo con el que nos organizamos, empezando a trabajar en el proyecto desde la última semana de abril. Asimismo, para evitar los problemas que nos perjudicaron en las anteriores entregas de este proyecto, tuvimos que empezar desde el principio, dedicando entre los dos autores aproximadamente 10 horas durante la semana del 29 de abril al 3 de mayo en realizar las subrutinas del primer hito. A pesar de no contar con el gestor de prácticas para saber si las subrutinas realizadas superaban las pruebas de las correcciones automáticas, los casos de prueba diseñados para estas subrutinas nos ofrecían los resultados que esperábamos, por lo que continuamos con la implementación de las demás subrutinas. Sin embargo, al tener un margen tan amplio y no contar con las correcciones hasta el 31 de mayo, no dedicamos muchas más horas durante las semanas del 6 al 26 de mayo, retomando el proyecto la semana del 27 de mayo. Durante esa semana nos dedicamos a implementar las subrutinas Comprime, Descomprime y Verifica, estas tres subrutinas nos llevaron aproximadamente 51 horas grupales, incluyendo la lectura, comprensión e implementación inicial y la depuración de errores una vez recibida la primera corrección. El 31 de mayo, en la primera corrección superamos todas las pruebas del primer hito, pero no conseguimos superar casi ninguna prueba del Comprime, Descomprime y Verifica, viéndonos obligados a reestructurar nuestro código casi por completo, revisando cada una de las subrutinas y dedicando muchas más horas de las esperadas.

Por último, una vez implementadas todas las subrutinas, enviamos nuestra última versión del código el 3 de junio, antes de empezar a escribir en esta memoria el trabajo realizado. La última semana de trabajo, del 10 de junio al 16 de junio, la dedicamos a terminar la memoria y probar el código.

Fernando Luna Marcuello, 190394, y Borja Pérez-Villacastín Palacín, 190395, declaramos el conocimiento y conformidad con las siguientes normas sobre copia y fraude académico:

- El uso de cualquier material de otros grupos/alumnos, por cualquier motivo, para el desarrollo del proyecto está explícitamente prohibido.

- El departamento comprueba la copia entre todas las entregas de todos los grupos que intervienen en la convocatoria y se considera copia cuando aparezca esa condición en cualquiera de las entregas.

- El departamento realizará finalmente un análisis global de posibles casos de copia, incluyendo la inspección visual de implementaciones de los grupos cuyo código presenta un mayor índice de coincidencias. Igualmente, se comprobarán algunas implementaciones elegidas al azar.