Proyecto de programación en ensamblador
Estructura de Computadores

- El proyecto consiste en la programación, en ensamblador del Motorola 88110, de un conjunto de rutinas que realicen la ***compresión y descompresión de un texto almacenado en memoria.***

- El texto será una cadena de caracteres *ascii* almacenados en posiciones consecutivas de la memoria y que finaliza con el carácter *nul*. Cada carácter *ascii* es un ***byte que se interpreta sin signo***, teniendo por lo tanto un valor comprendido entre **0** y **255**. El valor **0** corresponde al terminador, *nul*, y se representa mediante la secuencia **\0**

- El proceso de compresión se basa en la búsqueda de subcadenas de caracteres repetidas, de las que se mantiene la primera ocurrencia y se sustituyen las demás por referencias a la primera.

Enunciado feb-jul 2021/2022

- El enunciado de este proyecto es nuevo y **esta adaptado a los cambios en la normativa de la asignatura producidos este curso**, entre los que se encuentra la reducción del peso asignado al proyecto (este curso es el **20%**) así como la nota mínima del proyecto para que sea compensable con la teoría (**3 sobre 10**). De este modo, el proyecto planteado conlleva una carga de trabajo menor a la de cursos anteriores.

- Todas las rutinas deben ser probadas exhaustivamente para determinar que su funcionamiento es correcto y acorde con su especificación.

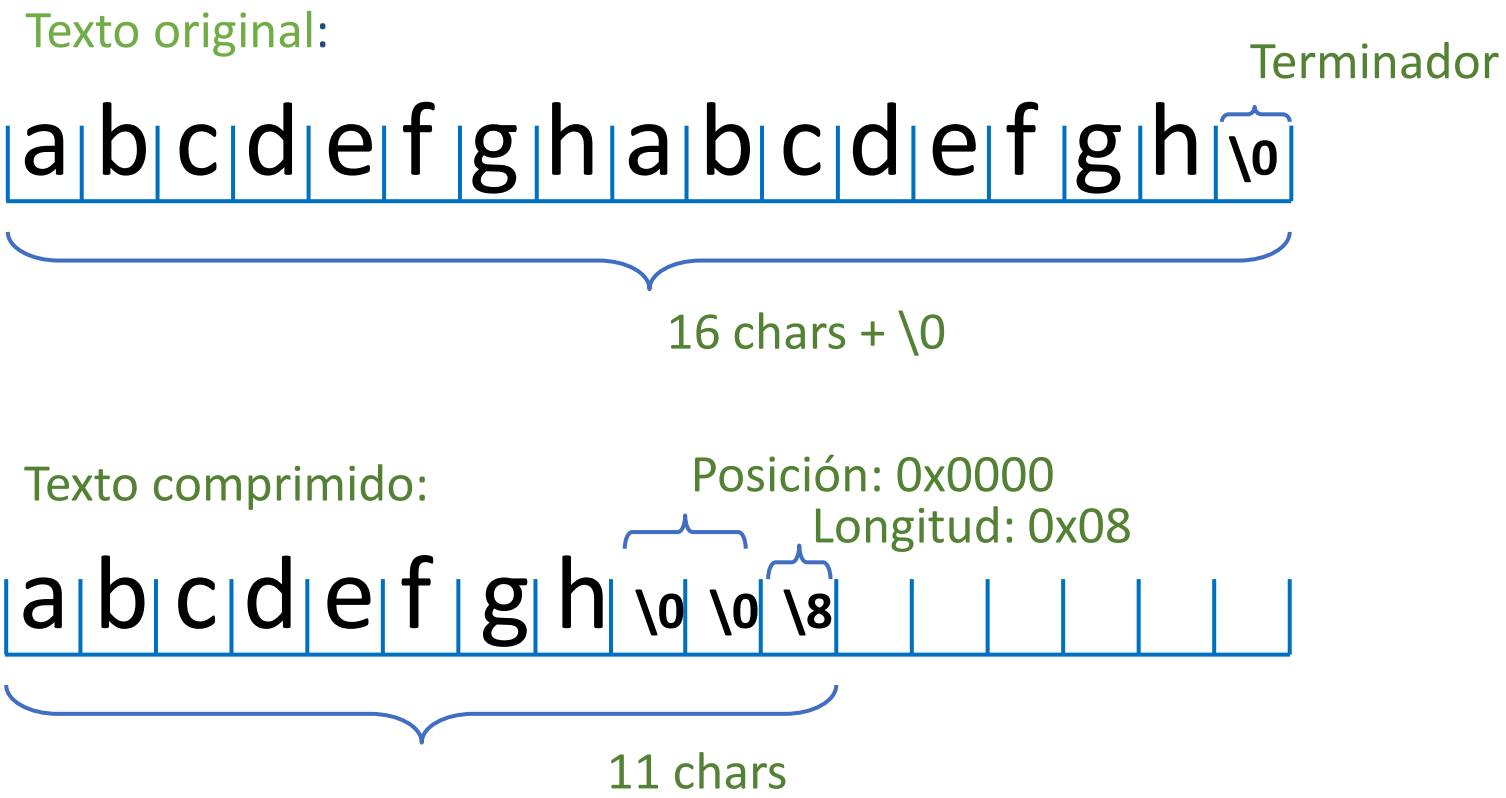
- Es importante observar con atención el apartado del enunciado que describe las normas de entrega y, en particular, la especificación del contenido que debe incluir la memoria del proyecto, así como las fechas de entrega de los ***hitos evaluables***.

- En la convocatoria de febrero, cada alumno podrá formar grupo con cualquier otro alumno o bien realizar el proyecto de forma individual.

- Se realizará una revisión minuciosa de los proyectos presentados en las dos convocatorias del curso para descartar o localizar posibles **casos de copia** que desafortunadamente se siguen produciendo (y detectando) en la mayoría de las convocatorias.

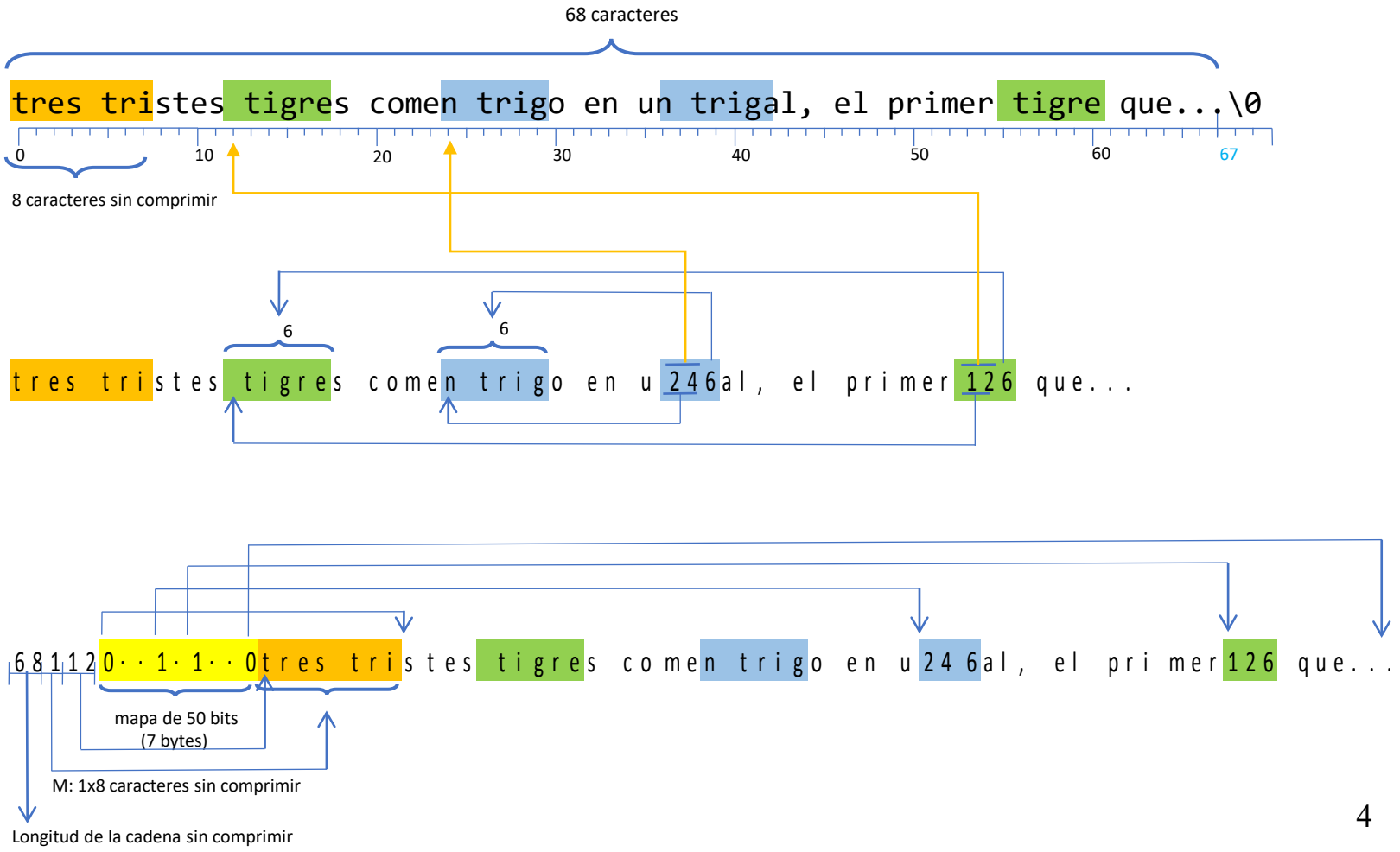
Compresión de texto

Ejemplo simplificado de compresión de un texto

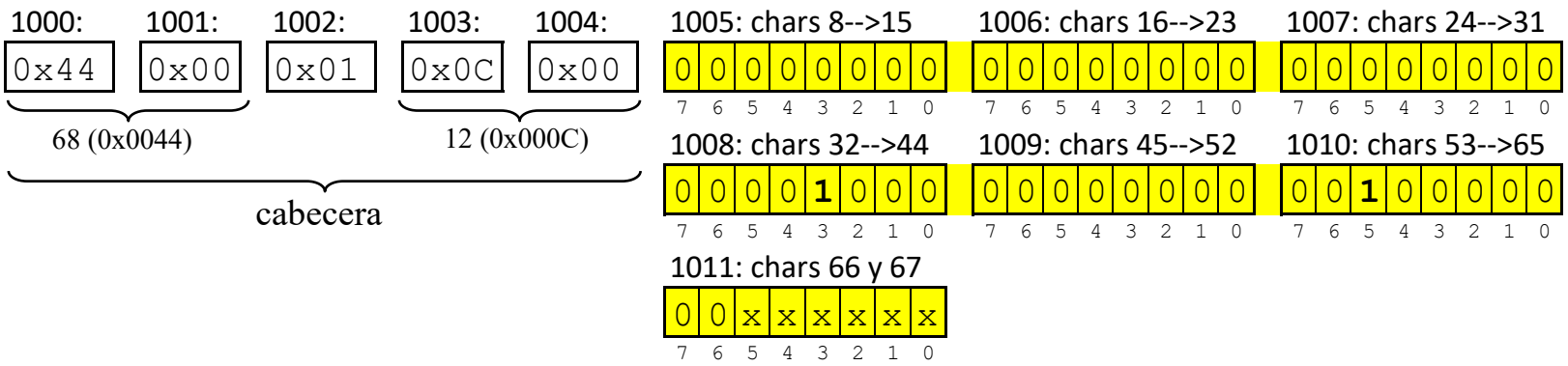


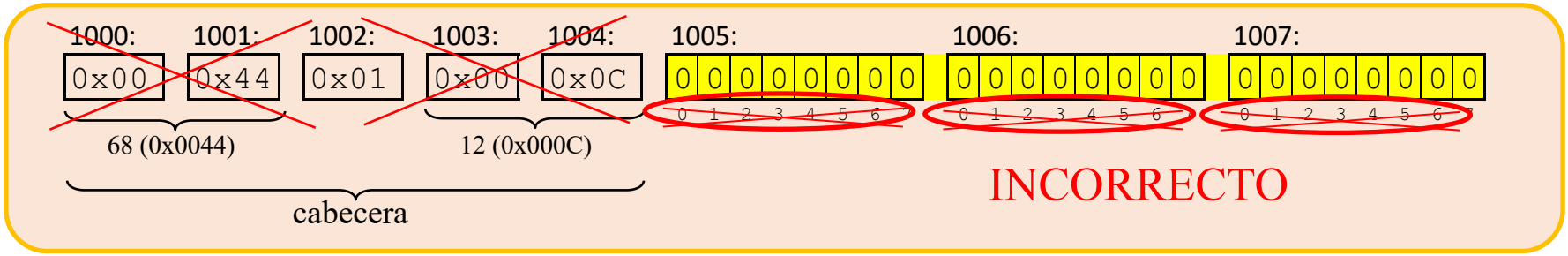
Compresión de texto

Ejemplo detallado de compresión en el formato del proyecto

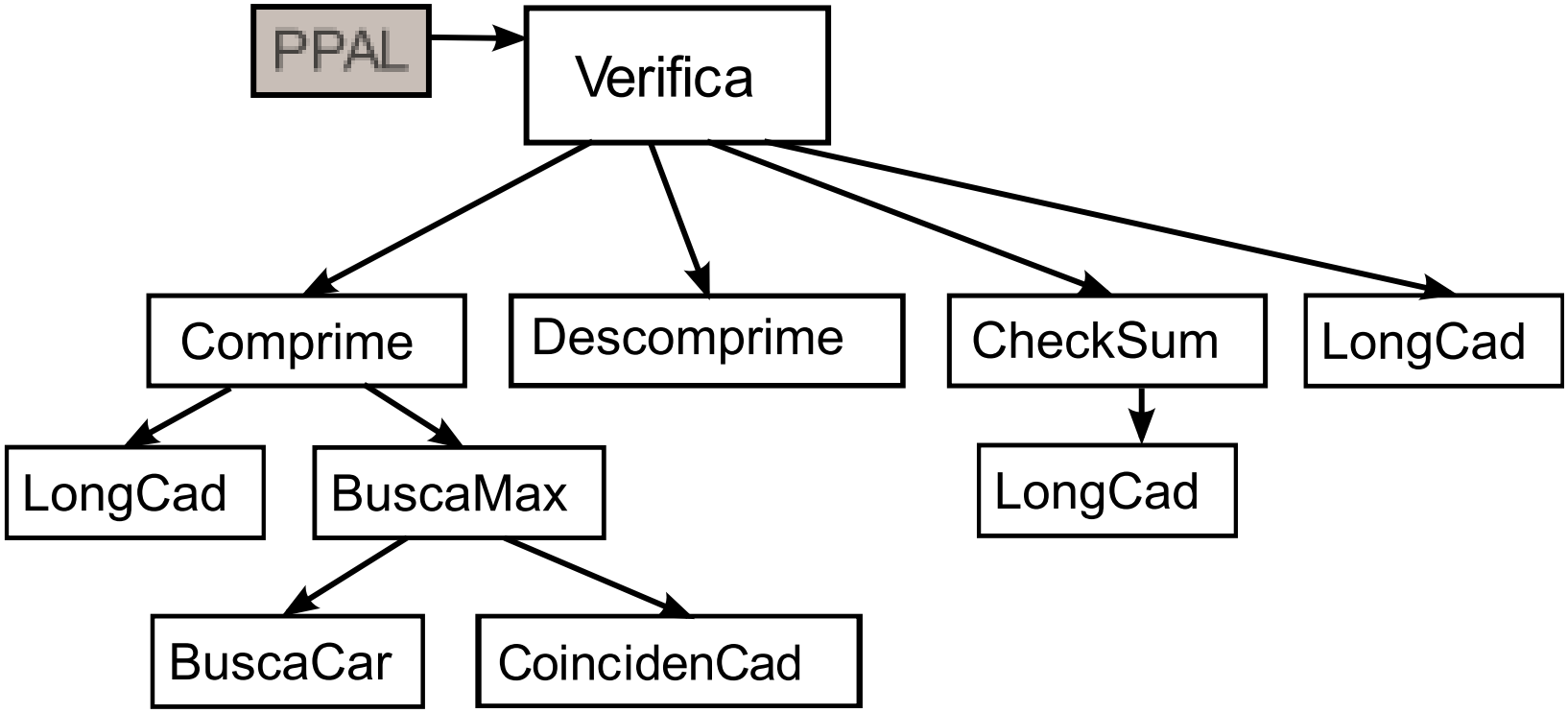


Mapa de bits


1000:
1001:
1002:
1003:
1004:
0x44
0x00
0x01
0x0C
0x00
68 (0x0044)
12 (0x000C)
cabecera
1005: chars 8-->15
1006: chars 16-->23
1007: chars 24-->31
1008: chars 32-->44
1009: chars 45-->52
1010: chars 53-->65
1011: chars 66 y 67
7 6 5 4 3 2 1 0


1000:
1001:
1002:
1003:
1004:
0x00
0x44
0x01
0x00
0x0C
68 (0x0044)
12 (0x000C)
cabecera
1005:
1006:
1007:
INCORRECTO

Jerarquía de las rutinas



Longitud de una cadena

```
long = LongCad (cadena)
```

- Parámetros (en la pila):

  `cadena`: De entrada. Se pasa por dirección (32 bits sin signo). Es la cadena de caracteres cuya longitud se debe calcular.

- Valor de retorno:

  `long`: Se devuelve en **`r29`**. Es la longitud de la cadena sin incluir el carácter terminador **`\0`**. Es un entero (32 bits) sin signo (positivo o nulo).

Búsqueda de carácter

```
rv = BuscaCar (C, ref, from, to)
```

- Parámetros (en la pila):

  C: De entrada. Se pasa por valor. Es el carácter que se trata de localizar en la cadena **ref**.

  ref: De entrada. Se pasa por dirección. Es la dirección de comienzo de una cadena de caracteres.

  from: De entrada. Se pasa por valor. Es la posición de la cadena a partir de la que se debe comenzar a buscar.

  to: De entrada: Se pasa por valor. Es la posición de la cadena en la que debe finalizar la búsqueda.

- Valor de retorno:

  rv: Se devuelve en **r29**. Es la posición de la primera ocurrencia del carácter **C** en la subcadena que va de **Dir(ref[from])** a **Dir(ref[to-1])**. Si **C** no está en esa subcadena, **rv** tomará el valor del parámetro **to**. Es un entero positivo o nulo.

Coincidencia de cadenas

```
long = CoincidenCad (cadena1, cadena2)
```

- Parámetros (en la pila):

  `cadena1`: De entrada. Se pasa por dirección: Es la primera de las dos cadenas de caracteres con que opera esta función.

  `cadena2`: De entrada. Se pasa por dirección: Es la segunda de las dos cadenas de caracteres con que opera esta función.

- Valor de retorno:

  `long`: Se devuelve en **`r29`**. Es el número de caracteres consecutivos que coinciden a partir del comienzo de sendas cadenas de caracteres. Es un entero que será nulo si el primer carácter de las cadenas es diferente y positivo en cualquier otro caso.

Búsqueda de la subcadena más larga (1)

```
rv = BuscaMax (ref, max, jj)
```

- Parámetros (en la pila):
  - **`ref`**: De entrada. Se pasa por dirección. Es la cadena de caracteres en la que se busca la coincidencia más larga con cierta subcadena.
  - **`max`**: De entrada. Se pasa por valor. Es el desplazamiento desde el comienzo de **`ref`** donde se encuentra la subcadena que se trata de localizar en otra posición (más cercana al comienzo de **`ref`**).
  - **`jj`**: De salida. Se pasa por dirección. Es una variable entera en la que se devolverá el desplazamiento desde el comienzo de **`ref`** donde se encuentra la subcadena más larga que coincide con la que comienza en la dirección **`Dir(ref[max])`**.
- Valor de retorno:
  - **`rv`**: Se devuelve en **`r29`**. Es la longitud de la subcadena más larga encontrada en el tramo **`Dir(ref[0]) a Dir(ref[max-1])`** que coincide con la que comienza en **`Dir(ref[max])`**. Es un valor entero, positivo o nulo.

Búsqueda de la subcadena más larga (2)

```
rv = BuscaMax (ref, max, jj)
```

- Descripción:

  La función busca en la cadena **`ref`**, desde su origen hasta la posición **`max-1`**, el tramo más largo de caracteres coincidentes con los que se encuentran a partir de **`Dir(ref[max])`**.

  Proporciona, en la dirección que se pasa en el parámetro **`jj`**, la distancia al comienzo de **`ref`** de la subcadena más larga que coincide y en el valor de retorno, el número de caracteres coincidentes.

Búsqueda de la subcadena más larga (3)

```
rv = BuscaMax (ref, max, jj)
```

- Algoritmo:
  1. Inicializa un marcador de posición **P** para recorrer **ref** y una variable con la longitud máxima encontrada hasta el momento **L**, ambos con valor 0.
  2. Recorre **ref** desde la posición **P=0** hasta **max** o hasta encontrar una cadena de **255** caracteres:
     - Busca la siguiente posición del carácter **C=ref[max]** llamando a la subrutina **BuscaCar** con los parámetros **C**, la cadena **ref**, el valor actual del marcador **P** y el parámetro **max.**
     - Si **BuscaCar** devuelve **max**, continúa en paso **3**.
     - Avanza el puntero hasta la posición **P** devuelta por **BuscaCar**.
     - Llama a la subrutina
       CoincidenCad(Dir(ref[P]), Dir(ref[max]))
     - Si el valor devuelto por **CoincidenCad** es mayor que **L,** actualiza el valor de **L** limitado al máximo, **255**, y almacena la posición **P** en la dirección que indica el parámetro de salida **jj.**
     - Avanza el puntero **P** hasta el siguiente carácter.
  3. Asigna a **r29** la longitud máxima final y retorna al llamante.

Checksum

`rv = Checksum (texto)`

- Parámetros (en la pila):

  `texto`: De entrada. Se pasa por dirección: Es la dirección de comienzo de la cadena de caracteres cuyo *checksum* se ha de calcular.

- Valor de retorno:

  `rv`: Se devuelve en r29. Es la suma de comprobación o *checksum* de la cadena **`texto`**. Es un entero (32 bits) sin signo.

- Descripción:

  Interpreta la cadena **`texto`** como un vector de enteros sin signo de 32 bits y calcula los 32 bits menos significativos de la suma de sus elementos (suma módulo 2^{32}). En caso necesario, completa con ceros por la izquierda la última palabra de la cadena **`texto`**: **`0x000000`**t_{n-1} **`ó 0x0000`**$t_{n-1}t_{n-2}$ **`ó 0x00`**$t_{n-1}t_{n-2}t_{n-3}$.

Comprime (1)

```
rv = Comprime (texto, comprdo)
```

- Parámetros (en la pila):

  `texto`: De entrada. Se pasa por dirección: Es la cadena de caracteres que se ha de comprimir.

  `comprdo`: De salida. Se pasa por dirección: Es la zona de memoria en la que ha de quedar el texto comprimido.

- Valor de retorno:

  `rv`: Se devuelve en **`r29`**. Es el tamaño en bytes de la estructura que contiene el texto comprimido. Incluye la cabecera, el mapa de bits y el vector de caracteres o referencias. Es un entero positivo (32 bits).

- Descripción:

  La función comprime el texto original almacenado a partir de **`texto`** y deja el resultado a partir de **`comprdo`**, de acuerdo al formato detallado en el enunciado de este proyecto.

Comprime (2)

`rv = Comprime (texto, comprdo)`

- Algoritmo:

1. Determina la longitud de la cadena **`texto`** llamando a la subrutina **`LongCad`**.
2. Reserva espacio en la pila para almacenar la secuencia de caracteres y referencias a cadenas del texto comprimido (la última parte del texto comprimido). Se reserva el número de bytes de **`texto`** ajustado por exceso a múltiplo de 4.
3. Inicializa las variables necesarias, por ejemplo los punteros o desplazamientos para recorrer **`texto`**, la variable local anterior o el mapa de bits.
4. Copia directamente los **`8xM`** caracteres iniciales de **`texto`** en la posición inicial de la variable local que almacena los caracteres y referencias (**`VL`**).
5. Recorre en un bucle los caracteres de **`texto`** hasta alcanzar su final:
   a) Localiza la siguiente subcadena repetida llamando a la subrutina **`BuscaMax`** con los parámetros **`texto`**, la posición actual del puntero que recorre **`texto`** y la dirección de una variable local en la que se recogerá la posición de la subcadena, **`P`**.

Comprime (3)

`rv = Comprime (texto, comprdo)`

- Algoritmo (cont.-1):

5. **`....`**

b) Si la longitud **`L`** de la subcadena devuelta por **`BuscaMax`** es < 4:

Copia el siguiente carácter de texto en la posición actual de **`VL`** y avanza los punteros en una unidad.

Escribe un 0 en el siguiente bit del mapa de bits (el orden de escritura en cada byte es de más a menos significativo, MSB a LSB).

Incrementa en una unidad el número de bits y el número de bytes del texto comprimido.

c) Si no, la longitud **`L`** de la subcadena es >= 4:

Copia **`P`** y **`L`** en la posición actual de **`VL`** y avanza puntero en 3 unidades.

Avanza el puntero que recorre **`texto`** en **`L`** unidades.

Escribe un **`1`** en el siguiente bit del mapa de bits (va de MSB a LSB).

Incrementa en una unidad el número de bits y en 3 unidades el número de bytes del texto comprimido.

Comprime (4)

`rv = Comprime (texto, comprdo)`

- Algoritmo (cont.-2):

6. Si es necesario, copia el último byte del mapa de bits en la zona indicada por el puntero a **`comprdo`**, y lo incrementa.

7. Rellena la cabecera de **`comprdo: 2`** primeros bytes con longitud de **`texto`**.

8. Rellena la cabecera de **`comprdo:`** valor **`1`** (**`M`**) en el tercer byte.

9. Rellena la cabecera de **`comprdo: 2`** últimos bytes con suma de: nº de bytes de mapa de bits + 5 (bytes de esta cabecera).

10. Lee los bytes del texto comprimido que se han escrito en la variable local de la pila **`VL`** y los copia en **`comprdo`** a continuación del mapa de bits.

11. Retorna dejando en **`r29`** la suma del número de bytes del mapa de bits, más los que ocupa la zona del texto comprimido, más los 5 de la cabecera.

Descomprime (1)

```
rv = Descomprime (com, desc)
```

- Parámetros (en la pila):

  com: De entrada. Se pasa por dirección: Es la zona de memoria en la que se encuentra el texto comprimido.

  desc: De salida. Se pasa por dirección: Es la cadena de caracteres en la que ha de quedar el texto una vez descomprimido.

- Valor de retorno:

  rv: Se devuelve en **r29**. Es la longitud de la cadena que contiene el texto una vez que se ha descomprimido. Es un entero (32 bits) positivo o nulo.

- Descripción:

  La función descomprime el texto que tiene el formato detallado en el enunciado de este proyecto y está almacenado a partir de **com** dejando el resultado a partir de **desc**.

Descomprime (2)

`rv = Descomprime (com, desc)`

- Algoritmo:

1. Inicializa las variables necesarias, por ejemplo los punteros o desplazamientos para recorrer **`com`** y **`desc`**, o el mapa de bits.
2. Copia de **`com`** a **`desc`** los **`8xM`** caracteres iniciales que no se comprimen, avanzando sus punteros.
3. Recorre en un bucle los bytes de **`desc`** hasta alcanzar su final:
   a) Si el siguiente bit del mapa de bits es 0:
      Copia el siguiente carácter de **`com`** a **`desc`** **`y`** avanza sus punteros.
   b) Si no (el siguiente bit del mapa de bits es **`1`**):
      Copia los **`L`** caracteres situados a partir de **`Dir(desc[P])`** a la posición que marca el puntero actual de **`desc`**, incrementando este en **`L`** unidades y el puntero a **`com`** en 3 unidades.
   c) Avanza el puntero a la posición del siguiente bit.
4. Añade el terminador "**`\0`**" al final de **`desc`**.
5. Retorna dejando en **`r29`** la longitud del texto.

Verifica (1)

```
rv = Verifica(texto, CheckSum1, CheckSum2)
```

- Parámetros (en la pila):

  `texto`: De entrada. Se pasa por dirección: Es la cadena de caracteres que se ha de comprimir, descomprimir y verificar.

  `Checksum1` **:** De salida. Se pasa por dirección: Es la variable en la que la subrutina escribirá el *checksum* de las palabras que forman el texto original que se pasa en el primer parámetro.

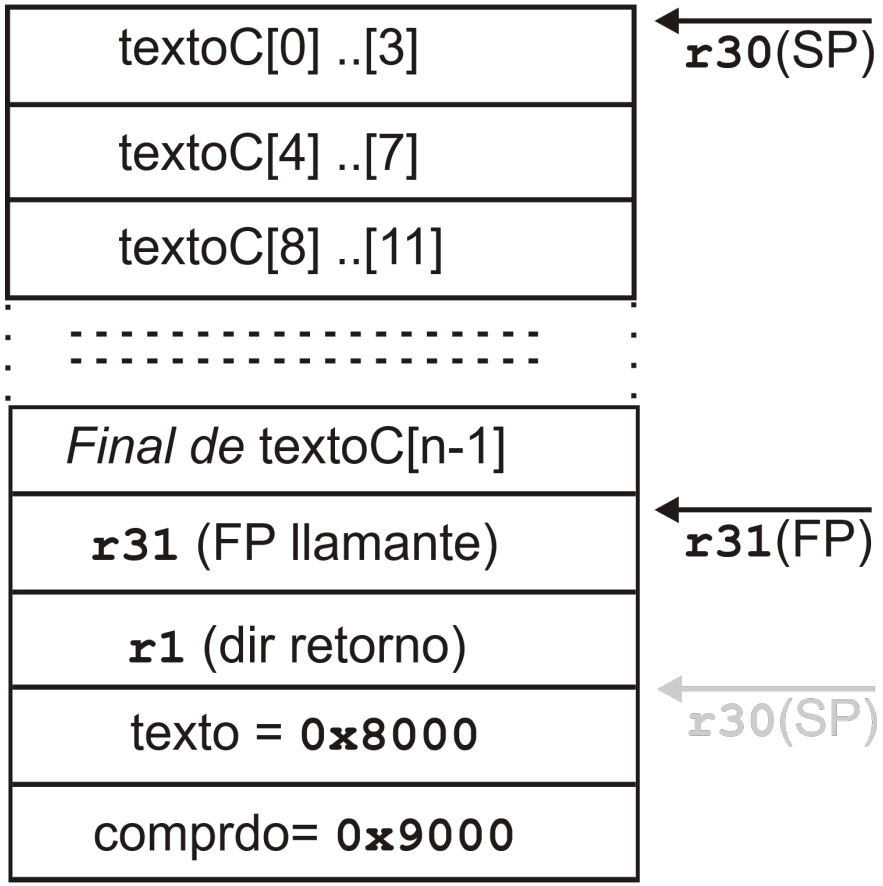
  `Checksum2` **:** De salida. Se pasa por dirección: Es la variable en la que la subrutina escribirá el *checksum* de las palabras que forman el texto obtenido tras comprimir y después descomprimir el texto original que se pasa en el primer parámetro.

- Valor de retorno:

  `rv`: Se devuelve en r29. Es el resultado global de la verificación: **`0`** si la verificación ha sido correcta o **`-1`** si se ha detectado algún error. Es un entero que solo puede tomar los valores **`0`** ó **`-1`**.

Tratamiento de la pila

```
Comprime:
      PUSH    (r1)
      PUSH    (r31)
      addu    r31,r30,r0
      ld      r10,r31,8
       ........
```



Ejemplo: Búsqueda de un carácter (1)

```
        org     0x8000
C:      data "_"
REF:    data "AAAABBBBCCCCDDDDEEEE_Fin\0"
from:   data 4
to:     data 24


88110> v 0x8000 12
     32768  5F000000  41414141  42424242  43434343
     32784  44444444  45454545  5F46696E  00000000
     32800  04000000  18000000  00000000  00000000
```

Ejemplo: Búsqueda de un carácter (2)

```
        org   0x8400
ppal:   or    r30, r0, 0x9000
        or    r31, r30, r30
                              ; Alternativamente se puede
                              ; poner:
        LOAD  (r11, C)        ; or    r11, r0, 0x5F ; "_"
        LEA   (r10, REF)
        LOAD  (r12, from)     ; or    r12, r0, 4
        LOAD  (r13, to)       ; or    r13, r0, 24

        PUSH  (r13) ; to
        PUSH  (r12) ; from
        PUSH  (r10) ; REF
        PUSH  (r11) ; C
        bsr   BuscaCar
ret:    addu   r30, r30, 16
        stop
```

Ejemplo: Búsqueda de un carácter (3)

```
r30=36848 (0x8FF0)

Direcciones de memoria:
36848   5F000000  04800000  04000000  18000000




Resultado:
r30=36848 (0x8FF0)  r29=20 (0x14000000)

Direcciones de memoria:
36848   5F000000  04800000  04000000  18000000
```

Programas de prueba

```
LEA:  MACRO...

        org   xxx
D1:     data...
D2:     data...
D3:     res...

        org   yyy
PPAL: "inicialización de la pila"
        ...
        "paso de parámetros (PUSH)"
        ...
        bsr subrutina
        "vaciado de la pila (POP)".
        stop
```

Entrega (fechas)

CONVOCATORIA DE FEBRERO 2022

El plazo de entrega del proyecto estará abierto para entregar código y memoria:
Desde el lunes día 25 de octubre hasta el lunes día 20 de diciembre de 2021.

Cada grupo podrá disponer de las siguientes correcciones:

- Primera corrección: **Viernes 29/10/2021 18:00**
- 1 corrección adicional: 2, 3, 4 ó 5 de noviembre
- 1er hito evaluable : Lunes 8/11/2021:
  subrutinas LongCad, BuscaCar y CoincidenCad: **1 punto**
- 1 corrección adicional: 10, 11, 12, 15, 16, 17, 18 o 19 de noviembre
- 2º hito evaluable : Lunes 22/11/2021:
  subrutinas del 1er hito más BuscaMax y Checksum: **1 punto**
- 4 correcciones adicionales: 23, 24, 25, 26, 29 y 30 de noviembre,
  1, 2, 3, 9, 10, 13, 14, 15, 16, y 17 de diciembre
- Última corrección: Lunes 20/12/2021

Todas las correcciones se realizarán a partir de las 21:00, salvo la primera del viernes 29 de octubre, que se realizará a las 18:00. Para solicitar una corrección bastará con entregar correctamente los ficheros del proyecto antes de dicha hora límite.

El examen del proyecto está planificado para el martes día 21 de diciembre de 2021.

Entrega (contenido)

Ficheros:

1. **autores (solo en alta de 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:

   ```
   N° Matrícula; DNI; apellido apellido, nombre; correo electrónico
   ```

   NOTA: este fichero solo se entrega en el momento de hacer el registro en el Gestor de Prácticas. Después de ese momento se accederá mediante el par [usuario:clave] utilizado durante el registro

2. **CDV.ens**: Contendrá las subrutinas que componen el proyecto debidamente comentadas junto con un programa principal y los datos de prueba para cada una de ellas que se haya utilizado para su depuración.

3. **memoria.pdf:** *Memoria*, en formato PDF y tamaño DINA4, 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.

Entrega (memoria)

La memoria debe contener los siguientes puntos:

- 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:

    - Número total de horas invertidas en el proyecto por cada miembro del grupo.
    - Relación de consultas 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. Una única prueba por cada subrutina.

- Observaciones finales y comentarios personales sobre este proyecto, entre los que se debe incluir una descripción de las principales dificultades surgidas para su realización.

Evaluación (2021/2022)

- El proyecto consta de tres partes: código y memoria (***código-memoria***), pruebas de funcionamiento (***pruebas***) y examen del proyecto (***examen***). Para superar el proyecto se deberá obtener la calificación de **apto en cada una de las tres partes**.

- Para que un grupo supere la parte de pruebas será necesario obtener al menos 5 puntos en la última corrección (o con anterioridad). La puntuación de esta parte es la siguiente:

  Hitos evaluables, **2 puntos** condicionados al logro de los hitos en las fechas anteriormente indicadas :

  – ***A****: Superar todas las pruebas de las subrutinas* **`LongCad, BuscaCar y CoincidenCad`** *(1 punto)*.

  – ***B****: Superar todas las pruebas de las subrutinas* **`LongCad, BuscaCar, CoincidenCad, BuscaMax`** *y* **`CheckSum`** *(1 punto)*.

  Subrutina **Descomprime, 2 puntos.** Proporcional al número de pruebas superadas.

  Subrutina **Comprime, 4 puntos.** Proporcional al número de pruebas superadas.

  Subrutina **Verifica, 2 puntos.** Proporcional al número de pruebas superadas.

- Examen: A partir de **3 puntos** hará media con la nota de pruebas.

- Código-memoria: **±1 punto** en función de la calidad. "**No apto**" requisitos básicos "Avisos importantes" pág. 18.

Evaluación (2021/2022)

- Calificación final:
  - Si (pruebas >= 5 puntos; examen >= 3 puntos; código-memoria Apto):
    Nota de proyecto = 0,7 * pruebas + 0,3 * examen ± 1 (código-memoria)
  - Si (examen < 3 puntos, código-memoria Apto):
    Nota de proyecto = Nota del examen del proyecto
  - Si (pruebas < 5 puntos o código-memoria "No apto"):
    Nota máxima de proyecto = 2,5 puntos *(la calificación real dependerá del número de pruebas superadas)*
- **Nota del proyecto** es compensable con la nota de teoría **sii es >= 3 puntos**