Proyecto de Programación en Ensamblador
Estructura de Computadores
Grado en Ingeniería Informática

Departamento de Arquitectura y Tecnología de Sistemas Informáticos

2021-2022. Primer semestre. Convocatorias de febrero y julio.

Introducción

El principal objetivo del proyecto de programación en ensamblador es que el estudiante pueda poner en práctica el conocimiento en profundidad de las posibilidades y el modo de funcionamiento de un procesador convencional de propósito general, utilizando para ello su lenguaje de programación nativo. Todo esto implica afrontar el diseño de programas, su desarrollo y, como parte fundamental, su depuración.

El trabajo se realizará en lenguaje ensamblador del Motorola 88110. Se trata de uno de los primeros procesadores RISC comerciales, lo que hace que sea lo suficientemente sencillo como para permitir que se desarrolle un proyecto interesante –al nivel que permite el conocimiento de un procesador al comenzar la asignatura "Estructura de computadores"– y, al mismo tiempo, suficientemente completo como para adquirir los conceptos básicos que se pretende que el estudiante alcance con su realización, que principalmente son los mencionados en el párrafo anterior.

El trabajo será sencillo para el estudiante si desde el primer momento ha seguido con normalidad las clases de teoría de la asignatura, ha realizado por su cuenta los problemas de programación en ensamblador propuestos en clase y ha asistido a la clase práctica de introducción a las herramientas utilizadas en el proyecto. Por el contrario, el arranque en el proyecto podría resultarle muy costoso si parte de una situación de desconocimiento del tema explicado en las clases de la asignatura.

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.

Se habla de *sin pérdidas* cuando el proceso de compresión seguido de la descompresión genera un resultado idéntico al original y es el tipo de compresión que se utiliza con ficheros de texto. Son compresores diferentes a los utilizados para la compresión de imágenes. Esta se basa normalmente en algoritmos que permiten comprimir de forma mucho más agresiva, que además es ajustable en función de los requisitos planteados. En estos compresores de imágenes se puede optar por un compromiso adecuado entre calidad obtenida y nivel de compresión alcanzado. En los compresores de texto o compresores sin pérdida de calidad, se pueden usar algoritmos más o menos complejos para tratar de alcanzar un nivel más alto de compresión, pero este nivel siempre está limitado por el requisito de obtener un resultado idéntico tras el proceso de compresión-descompresión.

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.

En el proyecto se programará en ensamblador una serie de subrutinas que permitan aplicar una compresión y descompresión a un texto almacenado en memoria. La compresión del texto almacenado en una zona de memoria Z1 dejará su resultado en otra zona Z1c y la descompresión del contenido de la zona Z1c depositará el texto obtenido en una tercera zona, Z2, de tal modo que el contenido de Z1 y Z2 deberán ser idénticos.

El hecho de haber partido la tarea de desarrollo en subrutinas elementales tiene el objetivo de facilitar el trabajo de depuración.

AVISO –2021/2022–

El enunciado de este proyecto **está 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 en el proyecto para que sea compensable con la teoría (un 3 sobre 10). De este modo, el proyecto planteado conlleva una carga de trabajo menor a la de cursos anteriores.

Aún tratándose de un proyecto nuevo, se verificará exhaustivamente que la implementación entregada por cada grupo sea la desarrollada únicamente por los miembros de dicho grupo, tratándose como caso de copia el uso de fragmentos de código copiados de otros grupos o realizados por otras personas o entidades que no sean las firmantes de los ficheros entregados.

Se recomienda que observe con especial atención los apartados de este documento que describen las normas de entrega y evaluación.

Aquellos estudiantes que tengan que repetir o corregir en la convocatoria de julio el proyecto que ellos mismos desarrollen en la convocatoria de febrero podrán partir de los programas que hubieran realizado anteriormente. Sin embargo, algunas o todas las pruebas establecidas podrán ser diferentes y sus resultados no tienen por qué coincidir, por lo que deberán adaptar la implementación de las subrutinas, de modo que *superen las pruebas que se establezcan.* Estas pruebas también podrán ser modificadas dentro de la convocatoria, si se considera necesario.

Independientemente del cambio realizado en el proyecto respecto al de cursos anteriores, se tendrá en cuenta lo siguiente:

- Los alumnos repetidores podrán formar grupo de proyecto con otro compañero.
- Los alumnos que formen parte de un grupo de proyecto en la convocatoria de febrero y no lo aprueben en dicha convocatoria **solo podrán establecer grupo con el mismo compañero** en sucesivas convocatorias o, alternativamente, realizar el proyecto **de forma individual**.
- Se realizará una revisión minuciosa de los proyectos realizados para descartar o localizar posibles **casos de copia** que desafortunadamente se siguen produciendo (y detectando) en la mayor parte de las convocatorias.

Compresión de texto

La compresión de texto aplicada en este programa se basa en un procedimiento elemental de búsqueda de cadenas de caracteres repetidas previamente en el texto y la sustitución de estas cadenas repetidas por una referencia a la posición previa en que aparece dicha cadena y el número de caracteres que coinciden entre ambas.

Un carácter se almacena en memoria como un byte (8 bits) que debe interpretarse sin signo. El carácter correspondiente estará codificado en ASCII.

Una cadena de caracteres está identificada por una dirección de comienzo a partir de la cual se interpreta que hay una secuencia de bytes (caracteres ASCII) que termina con un byte con código 0 (en hexadecimal `0x00` y frecuentemente representado como `\0`). No debe confundirse ese terminador de la cadena de caracteres con el carácter `0` (que en ASCII corresponde al código hexadecimal `0x30`).

Para que el programa y los datos utilizados sean manejables a nivel de programa ensamblador ejecutado en un emulador (limitado y lento en comparación con un procesador real), la identificación de la posición de las cadenas se codifica con un valor de 16 bits (entero sin signo `short`) que registra la distancia desde el comienzo del texto hasta la posición en que comienza dicha cadena). La longitud de la cadena coincidente se registra mediante un valor de 8 bits (entero sin signo `byte`) que hace referencia a cadenas de tamaño 1 a 255 caracteres.

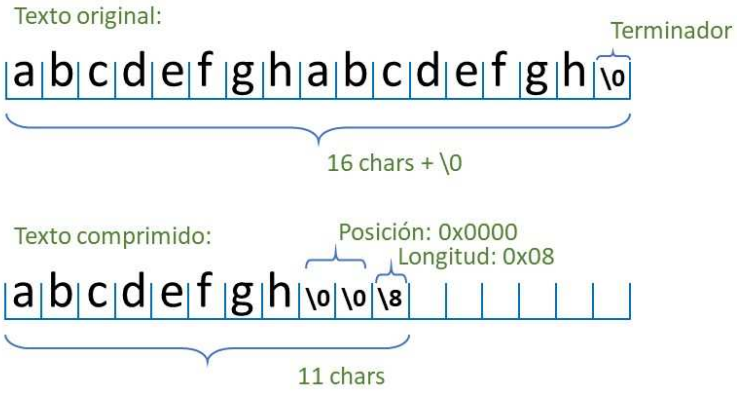

Figura 1. Ejemplo elemental de compresión de un texto reducido.

En la figura 1 se muestra un ejemplo sencillo y *simplificado* de compresión de un texto elemental:

La figura 2 ilustra el proceso de compresión que se lleva a cabo en este proyecto y que se describe a continuación. Dado que el programa emulador del procesador 88110 no incluye ningún tipo de periférico, los datos que se deben comprimir no corresponden a un fichero, como sería habitual, ya que para ello sería necesario disponer de un almacenamiento en disco. En su lugar, la información a comprimir se encontrará (en el ámbito de este proyecto) en una zona de memoria identificada por un nombre (una variable). Un ejemplo de esta variable, que denominamos de forma genérica `v.entrada` está representada en la primera línea de la figura 2, y estará formada por una cadena (o vector) de caracteres, terminada por un carácter `nul` (un byte con valor `0x00` que se representa gráficamente con la secuencia `\0`). El resultado del proceso de compresión será la información depositada en otra variable (sea ésta `v.salida`, representada en la tercera línea de la figura 2), cuyo contenido será una estructura formada por una cabecera (cinco bytes cuyo significado se describe más adelante) seguida por dos zonas de memoria consecutivas: un mapa de bits y un vector cuyos elementos son caracteres o bien referencias a otras secuencias de caracteres.

El mapa de bits correspondiente a la primera de las zonas mencionadas (`_vb[i]`) está representado con fondo amarillo en la figura, y cada uno de sus elementos (cada bit) está asociado a un carácter del texto original e indica si dicho carácter se ha copiado directamente de `v.entrada` a `v.salida` (en cuyo caso `_vb[i]=0`), o bien se trata del primer carácter de una cadena que se repite, es decir, de una secuencia de caracteres que ya ha aparecido previamente en `v.entrada`. El almacenamiento de bits en este mapa se hace en orden de peso decreciente, comenzando por el `bit7` (el más significativo) y asociando los caracteres sucesivos a los siguientes bits hasta completar el byte con su `bit0`, pasando entonces a rellenar el siguiente byte). Este caso (`_vb[i]=1`) se ha representado con fondo verde o azul en la figura 2. Independientemente del número de bits que constituyen el mapa de bits, se almacena utilizando un número entero de bytes; en el ejemplo de la figura 2 se utilizan 50 bits que ocupan 7 bytes en `v.salida`.

La segunda zona, representada en la segunda línea de la figura 2, está formada por un *vector de bytes* (`_vBYT[j]`), que contiene un carácter cada vez que el correspondiente `_vb[i]` sea `0`, o una referencia a otra secuencia previa, cada vez que el `_vb[i]` asociado sea `1`. En este último caso, la secuencia de caracteres de longitud **L** que comienza en la posición **P** determinada por `_vb[i]` quedará representada en el fichero comprimido como el par **P** (posición desde el comienzo de `v.entrada`) con formato de 16 bits (con valor `24` para la cadena azul y `12` para la cadena verde), seguido de **L**, que se almacena con un formato entero de 8 bits (con valor `6` para las dos cadenas del ejemplo). Ambos valores, P y L, se representan sin signo. Por otra parte, en el proceso de compresión se usará una constante **M** que será por definición múltiplo de 8. Con ella se define el tamaño de una primera parte de `v.salida` en la que los correspondientes **M** caracteres del texto a comprimir (representados con fondo naranja en la figura 2) irán

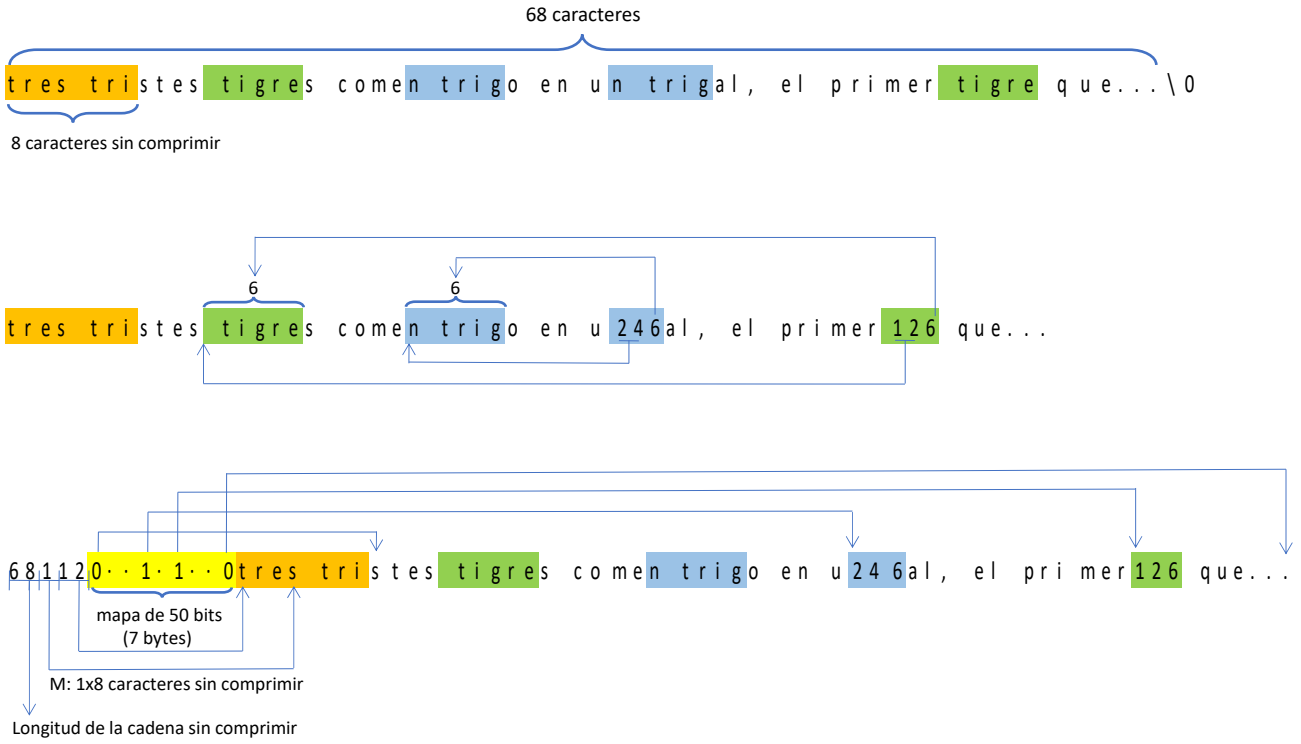

Figura 2. Ejemplo detallado de compresión de un texto reducido (N=6).

siempre copiados, evitando que sean sustituidos por cadenas previas. Por lo tanto tendrán `_vb[i]=0`. Dado que sus correspondientes **M** bits `_vb[i]` serán siempre cero, no se incluirán explícitamente en `v.salida`, evitando así que ocupen ese espacio.

Como se indicaba, el texto comprimido final estará formado por la cabecera de 5 bytes: 2 bytes iniciales que contienen la longitud del texto original sin comprimir (máx = 65535 chars), cuyo valor es `68` en el ejemplo de la figura; el siguiente byte, que codificará el valor de `M`, de modo que en cualquier caso `M` sea un múltiplo de 8 almacenado como `M/8` (con valor `1` en la figura); y, finalmente, otros 2 bytes, con valor `12` en la figura, que identifican la posición dentro de la zona de memoria en la que se encuentra el texto comprimido en la que comienza el vector de bytes `_vBYT[j]`; dicha posición se determina como el desplazamiento desde el comienzo del texto comprimido (en el ejemplo `5 + 7` bytes) a partir del que está almacenado el vector de bytes `_vBYT[j]`.

Tras los 5 bytes de la cabecera deberían ir los `M/8` bytes de `_vb[i]` correspondientes a los `M` caracteres iniciales, pero se obvian por ser todos ellos iguales a 0 (son *bits implícitos*). Después de esos `M` bits implícitos (que realmente no están) irán los restantes `_vb[i]` (en amarillo en la figura), y, justo a continuación, los caracteres que forman la salida.

En el momento de efectuar la compresión del texto se emplea también una constante `N` que define la longitud mínima de una cadena de caracteres que se almacene por referencia, es decir, con `_vb[i]=1`. Las secuencias de caracteres de tamaño igual o superior a `N` se copiarán por referencia, mediante los 3 bytes ya mencionados: dos de `P` (posición previa de esa cadena en `v.entrada`) y uno de `L` (longitud o número de caracteres de la cadena, que siempre será mayor o igual a `N`); el resto de los caracteres del texto se copiarán a `v.salida` como caracteres individuales, sin comprimir.

En el ámbito de este proyecto se utilizará `M=1` y `N=4`, aunque en general podrán ser parámetros configurables.

Habrá muchos textos que no se lleguen a comprimir, es decir, que la memoria ocupada por la cadena comprimida sea de mayor tamaño que la cadena original sin compresión. Eso se debe a las características del texto original y también a que el método de compresión es muy elemental, dado que no pretende ser más que un ejemplo sencillo y no está optimizado porque eso lo haría muy complejo para los propósitos de este proyecto. En muchos ejemplos utilizados en el proyecto se usarán cadenas sencillas que se presten a ser comprimidas.

En la descripción de las subrutinas que componen el proyecto quedarán aclarados algunos aspectos concretos de implementación.

Estructura del proyecto

El proyecto estará compuesto por ocho subrutinas que se relacionan tal como se indica en la Figura 3.

La comprobación del funcionamiento de las subrutinas es una parte esencial del proyecto y cada grupo deberá preparar todas las pruebas que considere necesarias para verificar el funcionamiento correcto en todas las situaciones particulares que pudieran tener influencia en el resultado. Por ello, el alumno o grupo construirá al menos un programa principal (PPAL en la figura) para llamar a cada una de las subrutinas que forman parte del proyecto. Utilizará estos programas con distintos argumentos que correspondan a las diferentes situaciones (cadenas de caracteres cortas, largas, de tamaño máximo, con muchas o pocas repeticiones de fragmentos de texto, etc.).

Se ha desglosado el programa de compresión en un número alto de subrutinas, algunas de las cuales son de implementación directa y casi inmediata, para facilitar al alumno su depuración, ya que así se enfrentará a fragmentos de código de tamaño reducido. Además, al tener el programa segmentado en varias partes, el sistema automático de corrección proporcionará información más precisa sobre las partes que se han implementado correctamente y las que necesitan revisión.

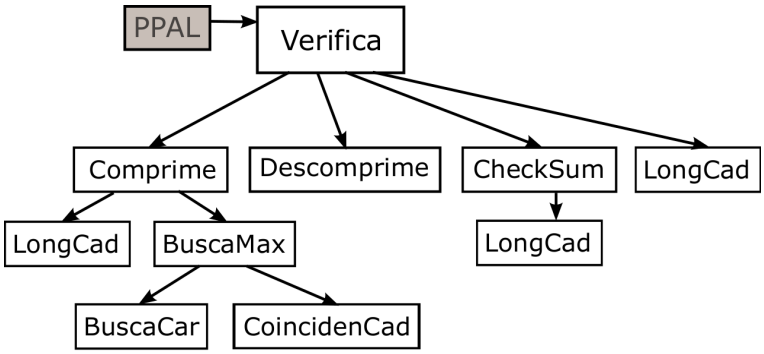

Figura 3. Jerarquía de las rutinas del proyecto.

Tipos de datos

En el proyecto se usan datos de diferentes tipos y tamaños:

Byte sin signo. Se usan para representar los caracteres de las cadenas de texto y para almacenar en el texto comprimido la longitud `L` de las subcadenas. Su longitud es de 8 bits.

Entero sin signo. Se usan para representar números enteros positivos en el código del programa, por ejemplo `L` o `P` (la Longitud o la Posición de una cadena de caracteres repetida), así como las direcciones de memoria. Su longitud es 32 bits (4 bytes).

Entero con signo. Se usan para representar cualquier dato que pueda tomar valores enteros positivos o negativos, por ejemplo los que se puedan utilizar para hacer algunos cálculos necesarios para la implementación del código. Su longitud es 32 bits (4 bytes).

Entero corto sin signo. Se usan para representar números enteros positivos en la zona de memoria que almacena el resultado de la compresión, ya que el valor de la variable `P` (posición) se debe almacenar en memoria con este formato. Su longitud es 16 bits (2 bytes que se almacenan en memoria siguiendo el convenio *little-endian*).

Todos los datos utilizados en este proyecto son de tipos enteros, en ningún caso se emplean representaciones fraccionarias o de coma flotante.

Además de los tipos básicos anteriores, se trabajará con cadenas de caracteres. Cada una de ellas será un conjunto de caracteres consecutivos identificado por una dirección de memoria `DirM` en la que se considera que empieza la cadena, que finalizará con el primer carácter con valor 0 (`0x00`) que se encuentre a partir de `DirM`.

Programa Principal

El programa principal se encargará de inicializar la pila de usuario, almacenar en ella los parámetros que se deben pasar, e invocar a las distintas rutinas objetivo de este proyecto.

Los parámetros de las rutinas se pasarán siempre en la pila salvo que se especifique lo contrario. Los parámetros que se puedan representar mediante una palabra, 32 bits, se pasarán por valor. Los parámetros que ocupen más de 32 bits (por ejemplo las cadenas de caracteres) se pasarán por dirección. El resultado se recogerá normalmente en el registro $r29$, salvo que se especifique de otro modo.

Longitud de una Cadena

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

Parámetros:

- **cadena:** Es la cadena de caracteres de la que se calculará su longitud. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **long:** La función devuelve la longitud de la cadena proporcionada en su único parámetro y sin incluir el carácter terminador `0x00`. Es un número entero (positivo o nulo) que se devuelve en `r29`.

Descripción:

La rutina `LongCad` recorre los caracteres de la cadena que se ha pasado como parámetro, incrementando un contador previamente inicializado a cero hasta que encuentre que el valor del carácter considerado es `0x00`, evitando incrementar el contador en este caso y devolviendo como resultado su valor.

Obsérvese que la cadena puede ser nula (si su primer carácter es `0x00`, que marca el final), en cuyo caso la longitud de la cadena será 0.

Búsqueda de un carácter

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

Parámetros:

- **C:** Es el carácter que se trata de localizar en la cadena `ref`. Es un parámetro de entrada de tipo byte, que se pasa por valor en una palabra de la pila.
- **ref:** Es la cadena de caracteres en la que se hace la búsqueda del carácter indicado en el primer parámetro (`C`). Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.
- **from:** Es el desplazamiento desde el inicio de la cadena `ref` en el que comenzará la búsqueda del carácter indicado en el primer parámetro. Es un parámetro de entrada, de tipo entero, que se pasa por valor en la pila.
- **to:** Es el desplazamiento desde el inicio de la cadena `ref` en el que finalizará la búsqueda del carácter indicado en el primer parámetro de la pila. La búsqueda se realizará entre `Dir(ref[from])` y `Dir(ref[to-1])`. Es un parámetro de entrada, de tipo entero, que se pasa por valor en la pila.

Valor de retorno:

- **rv:** La función devuelve la distancia en caracteres desde el comienzo de la cadena `ref` hasta la posición en que se localiza el carácter `C`. Si dicho carácter no figura en ese tramo de la cadena, `rv` tomará el valor del parámetro `to`. Es un número entero (positivo o nulo) cuyo valor se devuelve en `r29`.

Descripción:

La función busca la posición en que se encuentra por primera vez el carácter `C` en el tramo de la cadena `ref` que va desde la posición `from` hasta la posición `to-1`, es decir, desde `Dir(ref[from])` hasta `Dir(ref[to])`, sin incluir este último carácter. Si al llegar a la dirección `Dir(ref[to])` no se ha localizado el carácter que se busca, se devolverá el valor `to` en `r29`.

Coincidencia de cadenas

`long = CoincidenCad ( cadena1, cadena2 )`

Parámetros:

- **cadena1:** Es la cadena de caracteres que se debe comparar con la que se proporciona como segundo parámetro de esta misma subrutina. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.
- **cadena2:** Es la cadena de caracteres que se debe comparar con la que se proporciona como primer parámetro de esta misma subrutina. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **long:** La función devuelve la longitud de la zona de caracteres coincidentes entre ambas cadenas. Es un número entero que será nulo si el primer carácter de las cadenas es diferente y positivo en cualquier otro caso. Se devuelve en `r29`.

Descripción:

La rutina `CoincidenCad` recorrerá los caracteres de ambas cadenas mientras sean iguales y no se encuentre el final de ninguna de ellas, llevando la cuenta del número de caracteres idénticos.

Búsqueda de la subcadena más larga

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

Parámetros:

- **ref:** Es la cadena de caracteres en la que se buscará la coincidencia más larga con la subcadena de esta misma que comienza en la posición que se pasa como segundo parámetro de esta misma función. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.
- **max:** Es el desplazamiento desde el inicio de la cadena `ref` en que se encuentra la subcadena de la que se buscará la copia más larga que esté localizada en una posición anterior de esta misma cadena, es decir, entre `Dir(ref[0])` y `Dir(ref[max-1])`. Es un parámetro de entrada de tipo entero sin signo que se pasa por valor y ocupa 4 bytes.
- **jj:** Es una variable entera en la que la función deberá devolver el desplazamiento desde el comienzo de la cadena `ref` en el que se encuentra la copia más larga de la subcadena que comienza en `Dir(ref[max])`. Es un parámetro de salida que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **rv:** es la longitud del tramo coincidente más largo entre cualquier subcadena de `ref` que comience antes de `Dir(ref[max])` y la subcadena que comienza precisamente en esa dirección `Dir(ref[max])`. Es un número entero (positivo o nulo) cuyo valor se devuelve en `r29`.

  En caso de que `rv` sea nulo, se devolverá un valor -1 en `jj`.

Descripción:

La función busca el tramo más largo de caracteres coincidentes entre la subcadena que comienza en la posición `max` de la cadena `ref`, y cualquier otra subcadena previa (desde el origen hasta la posición `max-1`).

En la dirección indicada en el último parámetro devolverá el valor de la distancia al comienzo de la cadena `ref` donde se encuentra la subcadena más larga que coincide con `max`. En el valor de retorno se devolverá su longitud o el valor -1 en caso de no haber localizado ninguna subcadena.

Si la subcadena coincidente más larga se localiza en varias posiciones a partir de `ref`, se devolverá en `jj` el desplazamiento correspondiente a la primera de ellas.

El funcionamiento de esta rutina será el siguiente:

1. Inicia dos variables a valor `0`: un marcador de posición `P`, que usará para recorrer `ref` y otra que mantendrá la longitud de la coincidencia más larga encontrada hasta el momento.
2. Recorre en un bucle la cadena `ref`, desde `P=0` hasta `max` o hasta encontrar una cadena de 255 caracteres, incrementando en cada iteración el marcador de posición indicado en el paso anterior. Para ello:

a) Localiza la siguiente posición del carácter `ref[max]` llamando a la subrutina `BuscaCar`, para lo que utiliza los siguientes parámetros: el carácter situado en `ref[max]`, la dirección de comienzo de la cadena `ref`, el desplazamiento actual del marcador `P` y el parámetro `max`.

b) Si `BuscaCar` devuelve `max`, continúa por el último paso.

c) Avanza el marcador `P` hasta la posición devuelta por `BuscaCar`. itemDetermina el número de caracteres que coinciden en las subcadenas que comienzan en la posición actual del puntero y en `ref[max]` llamando a la subrutina `CoincidenCad`, a la que pasará los parámetros `Dir(ref[P])` y `Dir(ref[max])`.

d) Si la longitud devuelta por `CoincidenCad` supera el valor encontrado hasta el momento, lo actualiza (limitado a un máximo de 255 caracteres), y almacena la posición de comienzo de la subcadena en la dirección indicada por el parámetro de salida `jj`.

e) Avanza el marcador `P` para que apunte al siguiente carácter.

3. Devuelve el control al llamante una vez que ha dejado en `r29` la longitud final de la subcadena más larga.

Cálculo del checksum (suma de comprobación)

```
rv = CheckSum( texto )
```

Parámetros:

- **texto:** Es la dirección de comienzo de la cadena de caracteres cuya suma de comprobación (checksum) se tiene que calcular. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **rv:** Es el resultado del cálculo de la suma de comprobación (checksum) obtenida tal como se indica en la descripción de esta función. Es un número entero sin signo cuyo valor se devuelve en `r29`.

Descripción:

La función calcula la suma de las palabras de 32 bits que componen el texto proporcionado como parámetro, interpretando dicho texto como un vector de enteros sin signo de 32 bits y conservando únicamente los 32 bits menos significativos de esa suma, es decir, obteniendo el resultado de la suma módulo 2^{32}. Se considerará que los elementos del vector de enteros están almacenados en memoria siguiendo el convenio *little-endian*, que es como está configurado el emulador 88110 en el ámbito de este proyecto. En caso necesario completará con ceros por la izquierda la última palabra de la cadena texto, tal como se describe a continuación:

El número total de caracteres de la cadena `texto` no tiene por qué ser múltiplo de cuatro, de modo que cuando no lo sea tendrá que tratarse ese caso particular, sabiendo que la última palabra de la cadena estará formada por un número de bytes entre 1 y 4 (en memoria), y que deberá llevarse a un registro de tal forma que quede en él la misma información (el mismo valor entero) que hubiera quedado si tras el terminador de la cadena se encontraran varios ceros repetidos (bytes con valor `0x00`).

Por ejemplo, si la cadena fuese `ABCDE` (en hexadecimal, en direcciones consecutivas de memoria: `41424344 45`), seguida a continuación por el terminador (`\0`) y varios caracteres arbitrarios, digamos `AA` (`4141`), se encontraría todo ello en memoria como `ABCDE\0AA`, es decir `41424344 45004141` y se leería sobre registros como las palabras `0x44434241` y `0x41410045`. Sin embargo, según se ha indicado, debería haberse interpretado como si tras el terminador (`\0`) hubiera otros bytes con valor cero hasta completar la palabra: `41424344 45000000`, lo que se habría leído como `0x44434241 0x00000045`, y su suma de comprobación (el checksum) habría sido `0x44434286`.

Comprime

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

Parámetros:

- **texto:** Es la cadena de caracteres que se debe comprimir. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.
- **comprdo:** Es la zona de memoria en la que deberá quedar almacenado el texto comprimido. Es un parámetro de salida que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **rv:** es la longitud de la estructura que contiene el texto comprimido (el texto de entrada una vez que ha sido comprimido), incluyendo sus tres componentes: la cabecera, el vector de bits y el vector de caracteres. Es un número entero sin signo cuyo valor se devuelve en `r29`.

Descripción:

La función comprime el texto original que está almacenado a partir de `texto` y deja el resultado a partir de `comprdo`.

El proceso de compresión se realiza según se ha descrito en la sección *Compresión de texto* de este documento, pág. 4 y corresponde a los siguientes pasos:

1. Determina la longitud de la cadena `texto` llamando a la subrutina `LongCad`.
2. Reserva espacio en la pila para una variable local donde almacenará la secuencia de caracteres y referencias a subcadenas previas, que constituye la última de las tres estructuras que forman el texto comprimido. Recuérdese que el texto comprimido está formado por tres zonas: la cabecera, el mapa de bits y la zona de caracteres y referencias a subcadenas. En el caso peor, es decir, si no se logra ninguna compresión del texto original, esta última parte ocupará el mismo tamaño que dicho texto y, por lo tanto, será necesario reservar en pila la longitud de la cadena `texto` ajustada por exceso a múltiplo de 4.
3. Inicializa las variables necesarias para realizar la compresión, por ejemplo los punteros o marcadores que sirvan para identificar la posición dentro de la cadena de caracteres o dentro de la estructura de bits que describe las subcadenas (número de byte y número de bit, de 7 a 0).
4. Copia los `8xM` caracteres iniciales sin comprimir de `texto` en la última zona del texto comprimido, la situada en la pila, avanzando sus punteros o marcadores.
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 marcador que se usa para recorrer `texto` y la dirección de una variable local para recoger la posición de la subcadena, `Dir(P)`.
   b) Si la longitud `L` de la subcadena devuelta por `BuscaMax` es < 4:
   . Copia el siguiente carácter de `texto` en la zona reservada en la pila para almacenar el texto comprimido 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 o bit7 a bit0).
   . Incrementa en una unidad el número de bits, así como el número de bytes de la zona de caracteres/referencias del texto comprimido.

c) Si no (la longitud `L` de la subcadena devuelta por `BuscaMax` es >= 4):
. Copia `P` y `L` en la zona reservada en la pila para almacenar el texto comprimido y avanza su puntero en 3 unidades.
. Avanza el puntero que recorre `texto` en `L` unidades.
. Escribe un 1 en el siguiente bit del byte mapa de bits (recuerde que los bits de un byte se van rellenando en orden decreciente, de `bit7` a `bit0`).
. Incrementa en una unidad el número de bits y en tres unidades el número de bytes de la zona de caracteres/referencias del texto comprimido.

NOTA: Obsérvese que el recorrido por el mapa de bits rellena un único bit en cada paso, lo que conviene que se haga en una variable temporal, de modo que al completar los 8 bits de un byte, este se copie en el mapa de bits, se incremente el puntero o marcador correspondiente al recorrido del mapa de bits y se reinicie la variable temporal que contiene el byte en curso, para que comience con su `bit7`.

6. Copia el último byte del mapa de bits del texto comprimido e incrementa el puntero o marcador que indica el siguiente, excepto que se haya hecho esta copia en los puntos 5.b ó 5.c del último paso por el bucle, lo que habrá ocurrido si en ese último paso se han completado los 8 bits de un byte.

7. Copia la longitud de `texto` en la cabecera en los dos primeros bytes del texto comprimido (`comprdo[0]` y `comprdo[1]`).

8. Copia el valor 1 (`M`) en el tercer byte del texto comprimido `comprdo[2]`.

9. Determina el número de bytes del mapa de bits, le suma 5 (que son los bytes de la cabecera) y copia este resultado en los bytes `comprdo[3]` y `comprdo[4]`.

10. Lee los bytes del texto comprimido que está en la variable local de la pila y los copia en `comprdo` a continuación del mapa de bits.

11. Retorna dejando en `r29` la suma de los tamaños de las tres partes del texto comprimido: la cabecera, el mapa de bits y la parte final con los caracteres y las referencias a otras subcadenas.

Descomprime

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

Parámetros:

- **com:** Es la estructura que forma el texto comprimido. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

- **desc:** Es zona de memoria en la que debe quedar la cadena de caracteres correspondiente al texto una vez que se ha descomprimido. Es un parámetro de salida que se pasa por dirección, por lo que ocupa 4 bytes.

Valor de retorno:

- **rv:** Es la longitud de la cadena que contiene el texto una vez que se ha descomprimido. Es un número entero sin signo cuyo valor se devuelve en `r29`.

Descripción:

La función realiza la descompresión del texto que está almacenado a partir de `com` (en el formato comprimido de este proyecto) y deja el resultado a partir de `desc`.

El funcionamiento de esta rutina será el siguiente:

1. Inicializa las variables necesarias para realizar la descompresión, por ejemplo los punteros o marcadores de desplazamiento que identifican la posición dentro del texto comprimido y en la zona en que ha de quedar el texto descomprimido, así como los punteros o marcadores necesarios para recorrer el mapa de bits (número de byte y número de bit, de `bit7` a `bit0`).

2. Copia de `com` a `desc` los `8xM` caracteres iniciales que no se comprimen, avanzando sus punteros o marcadores.

3. Recorre en un bucle los bytes de `com` hasta alcanzar su final:

   a) Si el siguiente bit del mapa de bits es 0:
   . Copia el siguiente carácter de `com` a `desc` avanzando sus punteros. Recuerde que los bits de un byte se habrán rellenado en orden decreciente, desde el `bit7` al `bit0`.

   b) Si no (si el siguiente bit del mapa de bits es 1):
   . Copia los `L` caracteres situados a partir de `Dir(desc[P])` a la posición que marque 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.

   NOTA: Obsérvese que, de modo similar al funcionamiento descrito para la subrutina `Comprime`, el recorrido por el mapa de bits recoge un único bit en cada paso. En este caso conviene recoger en una variable temporal cada byte de los que forman el mapa de bits, tratar cada uno de sus 8 bits y, al terminar, avanzar al siguiente byte en la misma variable, continuando así hasta que se haya completado el proceso de descompresión.

4. Añade el terminador `\0` al final de `desc`.

5. Retorna dejando en `r29` la longitud del texto descomprimido.

Verifica

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

Parámetros:

- **texto:** Es la cadena de caracteres que se ha de comprimir, descomprimir y verificar para comprobar el funcionamiento global del sistema. Es un parámetro de entrada que se pasa por dirección, por lo que ocupa 4 bytes.

- **CheckSum1:** Es la variable en la que la subrutina escribirá la suma de comprobación (checksum) de las palabras que forman el texto original que se pasa en el primer parámetro. Es un parámetro de salida, de tipo entero sin signo y se pasa por dirección.

- **CheckSum2:** Es la variable en la que la subrutina escribirá la suma de comprobación (checksum) de las palabras que forman el texto obtenido tras comprimir y después descomprimir el texto original que se pasa en el primer parámetro. Es un parámetro de salida, de tipo entero sin signo y se pasa por dirección.

Valor de retorno:

- **rv:** Es un valor que se devuelve en `r29` y determina el resultado global de la verificación. Es un número entero con valor 0 (si la verificación ha sido correcta) ó -1 (si se ha detectado algún error).

Descripción:

La función se encarga de verificar que el funcionamiento de los procedimientos de compresión y descompresión es coherente. Para ello deberá reservar dos zonas de la pila de tamaño suficiente para almacenar respectivamente el texto comprimido (sea `PilaCom`) y el texto descomprimido (sea `PilaDes`).

La zona `PilaCom` deberá reservar el tamaño máximo que podrá ocupar el texto comprimido, que en el caso peor (cuando no se logre ninguna compresión del texto original) y suponiendo que `LO` sea la longitud del texto original, corresponderá a la suma de: la cabecera (5 bytes); el mapa de bits del texto comprimido (para el que se necesitarán `(7+LO)/8` bytes menos uno, que es el correspondiente a los 8 *bits implícitos* del comienzo); y, finalmente, los `LO` bytes necesarios para almacenar (en el caso peor) todos los caracteres del texto original. El tamaño así calculado deberá ajustarse al siguiente múltiplo de cuatro, de modo que pueda reservarse en la pila sin que produzca un desalineamiento de los accesos a memoria.

La zona `PilaDes` deberá reservar el tamaño máximo que podrá ocupar el texto descomprimido, que será igual al ocupado por el texto original pero ajustado por exceso al múltiplo de 4 más cercano, tal como se ha indicado para `PilaCom`.

Una vez reservada la memoria en pila, la función debe comprimir el texto original almacenado en `texto`, dejando el resultado en `PilaCom`. Después descomprimirá el texto que se ha almacenado en `PilaCom`, dejando el resultado de su descompresión en `PilaDes`. Finalmente, tras verificar que los tamaños del texto original y ese mismo texto una vez que ha sido comprimido y después descomprimido son iguales, procederá a calcular los checksum de ambas zonas de texto. devolviendo al programa llamante un valor `0` o `-1` en función de que los dos checksum calculados sean idénticos o sean diferentes.

Creación de una pila de usuario

Debido a que el 88110 no dispone de un registro de propósito específico para la gestión de la pila, se asignará como puntero de pila uno de los registros de propósito general. Se utilizará el registro **r30** como puntero de pila. Éste apuntará a la cima de la pila, es decir, a la palabra que ocupa la cabecera de la pila (donde estará la última información introducida) y ésta crecerá hacia direcciones de memoria decrecientes.

A modo de ejemplo se muestran las operaciones elementales a realizar sobre la pila: PUSH y POP (véase la Figura 4).

Supongamos que el registro **r30** contiene el valor `10000` (decimal) y el registro **r2** contiene el valor hexadecimal `0x04030201`. La operación PUSH, que introduce este registro en la pila, se implementa con la siguiente secuencia de instrucciones:

```
subu r30,r30,4
st   r2,r30,0
```

quedando r30(SP) = 9996 y la pila como se indica en la Figura 4.

Supongamos que después de realizar la operación anterior se realiza una operación POP sobre la pila con el registro **r3** como operando. La secuencia de instrucciones resultante sería la siguiente:

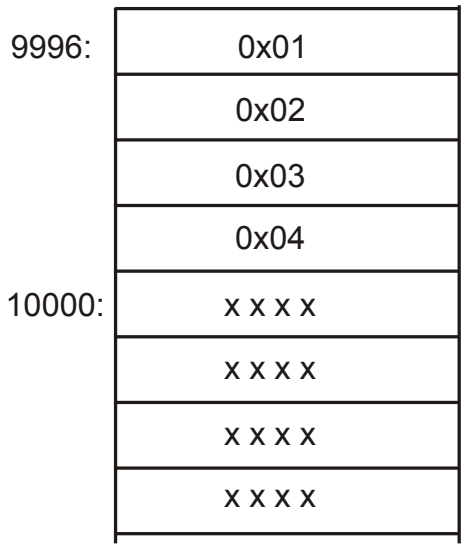

Figura 4. Operaciones PUSH y POP.

```
ld   r3,r30,0
addu r30,r30,4
```

quedando r3 = `0x04030201` y r30(SP) = `10000`

Subrutinas anidadas, variables locales, paso de parámetros y uso de registros

Puesto que las instrucciones de salto con retorno que proporciona el 88110 (**jsr** y **bsr**) salvaguardan la dirección de retorno en el registro **r1**, hay que incluir un mecanismo que permita realizar llamadas anidadas a subrutinas. Por este motivo es necesario guardar el contenido de dicho registro en la pila. Este mecanismo sólo es estrictamente necesario cuando una subrutina realiza una llamada a otra, pero, para sistematizar la realización de este proyecto, se propone realizar siempre a la entrada de una subrutina la salvaguarda del registro **r1** en la pila mediante una operación PUSH.

El espacio asignado para variables locales se reserva en el marco de pila de la correspondiente rutina. Para construir dicho marco de pila basta con asignar a uno de los registros de la máquina el valor del puntero de pila **r30**, después de haber salvaguardado el valor que tuviera el registro que actúa como puntero al marco de pila del llamante. Para la realización del proyecto se utilizará el registro **r31**. Por tanto, las primeras instrucciones de una subrutina que desea activar un nuevo marco de pila serán las siguientes:

```
RUTINA:   subu r30,r30,4     ; Se realiza una operacion PUSH r1
          st r1,r30,0
          subu r30,r30,4     ; Se realiza una operacion PUSH r31
          st r31,r30,0
          addu r31,r30,r0    ; r31 <- r30
```

En este proyecto se utilizarán los dos métodos clásicos de paso de parámetros:

- **Paso por dirección:** Se pasa la dirección de memoria donde está contenido el valor del parámetro sobre el que la subrutina tiene que operar.

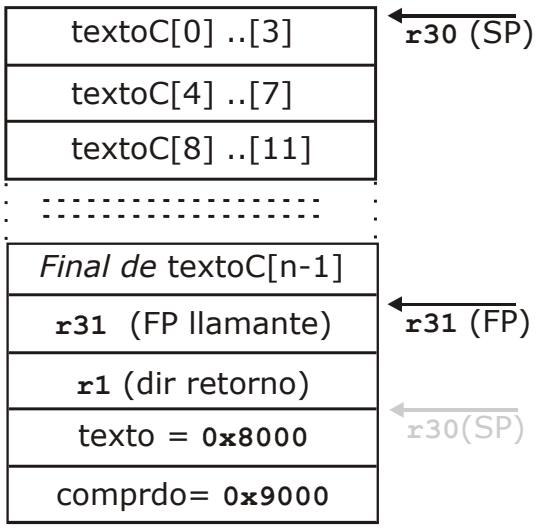

Figura 5. Estado de la pila durante la ejecución de la rutina `Comprime`.

- **Paso por valor:** Se pasa el valor del parámetro sobre el que la subrutina tiene que operar.

El paso de parámetros a todas las subrutinas del proyecto se realizará utilizando la pila de la máquina salvo cuando se especifique de otro modo. El parámetro que queda en la cima de la pila es el primero de la lista de argumentos. Por ejemplo, para una invocación de la rutina `Comprime`, los parámetros en la pila quedarían tal y como se especifica en la Figura 5, en la que se ilustra también la reserva de espacio en la pila que realiza `Comprime` para almacenar la variable local `textoC`. El comienzo de esta rutina quedaría como sigue:

```
Comprime:
    PUSH  (r1)          ; Se guarda la direccion de retorno
    PUSH  (r31)         ; Se salva el FP del llamante
    addu  r31,r30,r0    ; r31<-r30    Activación puntero de marco de pila
    ld    r10, r31, 8   ; lee los parámetros de la subrutina
    . . . . .           ; . . . . . .
```

En este proyecto los parámetros de salida se pasan por dirección. Las subrutinas únicamente pueden modificar los datos que reciben en la pila para actualizar los valores de salida de parámetros de salida o de entrada/salida que se pasan por valor; dado que no se usa este tipo de parámetros, los datos recibidos en la pila no pueden ser modificados y se debe devolver el control al programa llamante dejando en la pila el mismo estado que tenían al comienzo de la subrutina.

Cada subrutina puede usar a su conveniencia todos los registros de propósito general, salvo $r1$, $r30$ y $r31$; no pueden presumir que tengan un valor determinado; deben asignarles un valor inicial antes de usarlos como operandos de entrada; y no pueden usarlos para devolver al programa llamante ningún tipo de información, salvo $r29$ y cuando así esté especificado en la descripción de la subrutina. Las subrutinas que llaman a una segunda subrutina deben salvar en la pila todos los datos que estén almacenados en registros y sean necesarios tras el retorno de dicha segunda subrutina, recuperando su valor de la pila antes de reanudar la ejecución. Por ejemplo, si una rutina necesita usar los valores almacenados en $r10$ y $r11$ tras llamar a la subrutina `SUBRUTINA` a la que se pasa como parámetros $r20$ y $r21$, una codificación correcta es la siguiente:

```
RUTINA:   PUSH  (r1)         ; Se guarda la direccion de retorno
          . . . . .          ; . . . . . .
          PUSH  (r10)        ; Se salva el valor de r10
          PUSH  (r11)        ; Se salva el valor de r11
          PUSH  (r21)        ; Se pasa como parámetro r21
          PUSH  (r20)        ; Se pasa como parámetro r20
          bsr   SUBRUTINA    ; Se llama a la subrutina
          POP   (r20)        ; Se recupera el parámetro r20
          POP   (r21)        ; Se recupera el parámetro r21
          POP   (r11)        ; Se recupera el valor de r11
          POP   (r10)        ; Se recupera el valor de r10
          . . . . .          ; . . . . . .
```

Asignación de etiquetas y de memoria

El punto de entrada de cada una de las subrutinas deberá ir asociado a las etiquetas `LongCad`, `BuscaCar`, `CoincidenCad`, `BuscaMax`, `Checksum`, `Comprime`, `Descomprime` y `Verifica` (respetando el tipo de letra mayúsculas/minúsculas). Por ejemplo, la primera instrucción perteneciente a la subrutina `LongCad` deberá ir precedida de esta etiqueta:

```
LongCad: subu r30,r30,4
              .
              .
              .
```

Las direcciones de memoria `0x00010000` **y siguientes se deben reservar para el programa corrector**, por lo que el alumno **no debe utilizarlas** para almacenar código, datos ni el espacio de pila. Además, deberá situar la pila en posiciones altas de memoria pero **siempre inferiores a la** `0x00010000` que se ha mencionado. Por ejemplo, podrá inicializarse el puntero de pila en la dirección `0x0000F000`.

Avisos importantes

Los proyectos que no se atengan a las siguientes normas se evaluarán como no aptos en la parte de implementación aunque superen la batería de pruebas establecida por el Departamento:

1. Todas las subrutinas deberán finalizar con el mismo valor del puntero de pila que tenía al principio de la subrutina (antes de que ejecute su primera instrucción).

2. Cuando una subrutina requiera memoria de trabajo en la que almacenar temporalmente información, empleará **siempre** memoria de la pila y **nunca** variables definidas en direcciones absolutas.

Los proyectos que no se atengan a las siguientes normas se verán penalizados pudiendo llegar a ser calificados como no aptos:

5. Las tareas de este proyecto incluyen la definición e implementación por parte de los alumnos de la batería de pruebas a emplear para verificar el funcionamiento correcto de las subrutinas del proyecto, ya que constituye una parte fundamental de cualquier desarrollo de software. Debido a ello no se considera válido copiar dichas baterías de pruebas de unos grupos a otros. En caso de no respetar esta norma, el proyecto podrá ser calificado como suspenso.

6. Los programas de prueba usarán obligatoriamente macros `PUSH` para incluir en la pila los parámetros de la subrutina que estén probando. Los datos de prueba se definirán individualmente, de forma clara, mediante pseudo-instrucciones `data` y `org` y se se les asignará etiquetas independientes. Se muestra un ejemplo en la sección *Ejemplos*, pág. 21. Las consultas de los alumnos que no sigan estas instrucciones no serán atendidas. El código del proyecto incluirá al menos un programa de prueba para cada subrutina implementada.

Sugerencias y recomendaciones

- Para calcular el número entero múltiplo de 4 más pequeño pero igual o superior a un número dado, se pueden usar varios métodos sencillos y se considera válido utilizar cualquiera de ellos. Sin embargo, recomendamos evitar el uso de bucles que realicen más de cuatro iteraciones. Como sugerencia, tenga en cuenta que puede comprobar si un número cualquiera es múltiplo de 4 simplemente observando si sus dos bits menos significativos tienen valor cero.

- En ningún apartado de este proyecto se requiere utilizar instrucciones que operen con números en coma flotante. En particular, las multiplicaciones y divisiones que se utilicen deben trabajar con operandos enteros.

Ejemplos

A continuación se incluye un ejemplo de caso de prueba con los argumentos que se pasan a la subrutina `BuscaCar` y las direcciones de memoria que se modifican.

En la página web se puede encontrar un documento con ejemplos de casos de prueba de todas las subrutinas del proyecto, que se pueden seguir como guía para la elaboración de juegos de ensayo más completos.

Todos los ejemplos se han descrito utilizando el formato de salida que ofrece el simulador del 88110. En este procesador el direccionamiento se hace a nivel de byte y se utiliza el formato *little-endian*. En consecuencia, cada una de las palabras representadas a continuación de la especificación de la dirección debe interpretarse como formada por 4 bytes con el orden que se muestra en el ejemplo siguiente:

```
Direcciones de memoria, tal como las representa el simulador:
60000       04050607     05010000

Direcciones de memoria, tal como se deben interpretar:
60000       04
60001       05
60002       06
60003       07

60004       05
60005       01
60006       00
60007       00

Valor de las palabras almacenadas en las posiciones 60000 y
60004, tal como las interpreta el procesador:

60000       0x07060504 = 117.835.012
60004       0x00000105 = 261
```

Caso de prueba de la subrutina `BuscaCar`:

Llama a `BuscaCar` pasándole el carácter que ha de buscar (`C`), el comienzo de la cadena de caracteres que contiene la zona en la que debe realizar la búsqueda (`ref`), el desplazamiento hasta el comienzo de la zona de búsqueda en `ref` (`from`) y el desplazamiento hasta el final de esa misma zona de búsqueda (`to`).

La cadena de caracteres y el resto de parámetros se definen en ensamblador mediante las siguientes pseudo-instrucciones, en las que se ha utilizado como zona de datos la situada a partir de la dirección hexadecimal $0x8000$:

```
       org     0x8000
C:     data "_"
REF:   data   "AAAABBBBCCCCDDDDEEEE_Fin.__.\0"
from:  data 4
to:    data 25
```

El emulador 88110 mostrará estos datos de la siguiente forma usando el comando V:

```
88110> v 0x8000 12
     32768       5F000000     41414141     42424242     43434343
     32784       44444444     45454545     5F46696E     2E5F5F2E
     32800       00000000     04000000     19000000     00000000
```

El siguiente programa principal inicializa la pila asignando el valor `0x9000` al puntero de pila, registro $r30$. A continuación introduce en la pila los parámetros usando operaciones PUSH y llama a la subrutina `BuscaCar`. Por último, vacía la pila dejando $r30$ con el valor inicial, `0x9000`.

```
        org   0x8400
ppal:   or    r30, r0, 0x9000
        or    r31, r30, r30
                              ; Alternativamente se pueden poner:
        LOAD  (r11, C)        ; or    r11, r0, 0x5F ; igual a "_"
        LEA   (r10, REF)
        LOAD  (r12, from)     ; or    r12, r0, 4
        LOAD  (r13, to)       ; or    r13, r0, 25

        PUSH  (r13) ; to
        PUSH  (r12) ; from
        PUSH  (r10) ; REF
        PUSH  (r11) ; C

        bsr   BuscaCar
ret:    addu  r30, r30, 16

        stop
```

Al comienzo de la subrutina `BuscaCar`, el puntero de pila $r30$ tiene el valor `0x8FF0` (`36848`) y el contenido de la pila es el siguiente:

```
r30=36848 (0x8FF0)

36848       5F000000      04800000      04000000      19000000
```

El primer valor corresponde al primer parámetro, `C` (`0x5F`, que corresponde al carácter '_'), el segundo a la dirección `0x8004` que es donde comienza la cadena `ref`, el tercero al valor entero 4 (parámetro `from`) y el último al valor entero `0x19=25` (parámetro `to`).

Por último, el resultado de la ejecución de `BuscaCar`, mostrado antes de ejecutar la instrucción de la etiqueta `ret` es el siguiente:

```
r30=36848 (0x8FF0)  r29=20 (0x14)
```

NORMAS DE PRESENTACIÓN Y EVALUACIÓN

Toda la información relativa a este proyecto y, en particular, las aclaraciones o modificaciones de última hora, se encuentra disponible en:

`http://www.datsi.fi.upm.es/docencia/Estructura_09/Proyecto_Ensamblador`

Esta página contiene una sección de anuncios/noticias relacionadas con el proyecto.

EVALUACIÓN 2021/2022

El proyecto consta de tres partes: código y memoria del proyecto (*código-memoria*), pruebas de funcionamiento (*pruebas*) y examen del proyecto (*examen*). Para obtener una nota de proyecto compensable con la teoría (3 puntos) es necesario que la nota de código-memoria sea *Apto*, que la nota de pruebas sea de al menos 5 puntos y la nota del examen sea de al menos 3 puntos.

- Calificación de la parte de **pruebas**:

Independientemente de las pruebas superadas al concluir el periodo de correcciones del proyecto, hay dos fechas importantes (definidas para cada convocatoria) ya que cada una de ellas lleva asociado un *hito evaluable*. Estos hitos consisten en lo siguiente:

- Hito1: Superar en la fecha programada todas las pruebas establecidas para las subrutinas `LongCar`, `BuscaCar` y `CoincidenCad`.
- Hito2: Superar en la fecha programada las pruebas correspondientes al *Hito1* y además todas las pruebas establecidas para las subrutinas `BuscaMax` y `CheckSum`.

Teniendo en cuenta estos hitos, las distintas subrutinas que se deben programar, y que el objetivo del proyecto es completar la construcción de todas las subrutinas, incluyendo la que se utiliza para verificar el funcionamiento general (`Verifica`), la nota de la parte de pruebas se obtendrá del siguiente modo:

- Hito1: 1 punto (calificación del hito: 0 ó 1)
- Hito2: 1 punto (calificación del hito: 0 ó 1)
- Subrutina `Descomprime`: hasta 2 puntos.
- Subrutina `Comprime`: hasta 4 puntos. La calificación concreta se obtendrá proporcionalmente al número de pruebas superadas.
- Subrutina `Verifica`: hasta 2 puntos. La calificación concreta se obtendrá proporcionalmente al número de pruebas superadas.

La calificación concreta obtenida en cada una de las tres últimas pruebas (`Descomprime`, `Comprime` y `Verifica`) se obtendrá proporcionalmente al número de pruebas superadas.

Los alumnos que no obtengan al menos 5 puntos en la parte de pruebas no podrán presentarse al examen del proyecto y su calificación global del proyecto quedará establecida como la mitad de su nota de pruebas.

El número de pruebas que el Departamento utilizará para probar cada cada subrutina, cuántas de ellas se superan y cuántas fallan, se indicará en el resultado de la corrección. Estas pruebas podrán verse modificadas durante el desarrollo del proyecto, en cuyo caso se avisará en la sección de noticias de la página web del proyecto. Además de las pruebas de cada subrutina, podrán facilitarse otras que no se tendrán en cuenta para la calificación, pero que podrán aportar información de interés a cada grupo de cara a la depuración del código que ha generado. Por ejemplo, al probar la subrutina `Verifica` se incluirán algunas pruebas en las que se utilizará una implementación realizada por el Departamento del resto de las subrutinas necesarias para la comprobación. Con ello se facilitará que se pueda acotar el origen de los posibles errores, ya que se podrá saber si estos se encuentran en la implementación de `Verifica` o en las subrutinas de apoyo. Ese tipo de ayuda a la depuración podrá también incluirse para cualquier subrutina que no sea terminal, sino que necesite del funcionamiento correcto de otras subrutinas.

- Calificación del **examen del proyecto**:

Para que un alumno pueda presentarse al examen del proyecto es necesario que su grupo haya alcanzado una calificación de al menos 5 puntos en la parte de **pruebas**. El examen podrá ser a) de tipo test, b) de preguntas con respuesta corta o pequeñas cuestiones prácticas o c) una mezcla de (a) y (b). En caso de obtener una calificación de al menos 3 puntos en el examen, se hará una media ponderada entre la calificación de pruebas (70 %) y la del examen (30 %). Si se obtiene una calificación inferior a 3 puntos en el examen, la nota global del conjunto pruebas-examen será la obtenida en el examen.

- Calificación de **código-memoria**:

Para todos los grupos que obtengan calificación compensable en el conjunto pruebas-examen se revisará tanto el código como la memoria entregados, cuyo resultado podrá implicar que dicha nota quede modificada de acuerdo a lo siguiente:

- En general, siempre que se hayan seguido las normas de implementación indicadas, la calificación obtenida en el conjunto pruebas-examen, siendo ésta mayor o igual a 3 puntos, podrá ser incrementada o decrementada hasta en 1 punto en función de la calidad (organización, comentarios, uso correcto de las diferentes instrucciones, etc) de la implementación y la memoria.
- El proyecto en conjunto pasará a ser *No Apto* si en esta revisión se detecta que se ha incumplido alguno de los requisitos básicos establecidos, especialmente los recogidos en la sección *Avisos importantes* (pág. 18), por ejemplo, que se haya utilizado memoria en direcciones absolutas para almacenar variables temporales, etc.

CONVOCATORIA DE FEBRERO 2022

El plazo de entrega del proyecto estará abierto desde el lunes día **25 de octubre** hasta el lunes **día 20 de diciembre de 2021** en que se realizará la corrección definitiva y se recogerán las memorias del proyecto (en fichero pdf).

En todas las correcciones se pasarán las pruebas de todas las subrutinas. Esto permitirá que los grupos avancen en la implementación y puedan probar todas las subrutinas en las primeras correcciones, no sólo las asociadas a cada hito. Por otra parte, esto obliga a que estén definidas desde el primer momento las etiquetas asociadas a todas las subrutinas del proyecto. Los días 21 y 22 de diciembre el gestor de prácticas estará configurado de modo que permita entregar únicamente la memoria del proyecto para facilitar que se incluya en la misma la información sobre las modificaciones de última hora.

Cada grupo podrá disponer de las siguientes correcciones para comprobar el funcionamiento de su código:

- Primera corrección, día 29 de octubre (a las 18:00)
- Una única corrección adicional antes del 8 de noviembre
- Corrección del primer hito, día 8 de noviembre
- Una única corrección adicional antes del 22 de noviembre
- Corrección del segundo hito, día 22 de noviembre
- Un máximo de cuatro correcciones adicionales (días lectivos) antes del 20 de diciembre
- Corrección definitiva, día 20 de diciembre

Las correcciones se realizarán los días lectivos 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** se realizará el día 21 de diciembre de 2021 en horario y aulas pendientes de determinar y que se anunciarán en la web del Proyecto.

CONVOCATORIA EXTRAORDINARIA DE JULIO 2022

Todos los alumnos que se presenten a esta convocatoria del proyecto deberán realizar la **entrega completa de los ficheros del proyecto, código y memoria**, independientemente de que se hayan presentado o no y de las pruebas que hayan superado en la convocatoria de febrero.

El plazo de entrega del proyecto estará abierto desde el viernes día **27 de mayo** hasta el lunes **día 20 de junio de 2022** en que se realizará la corrección definitiva. Los días 21 y 22 de junio el gestor de prácticas estará configurado de modo que permita entregar únicamente la memoria del proyecto para facilitar que se incluya en la misma la información sobre las modificaciones de última hora.

Cada grupo podrá disponer de la primera y última de las correcciones, que se realizarán los días **30 de mayo y 20 de junio**, así como la corrección en que se comprobará si se superan los *hitos evaluables*, que tendrá lugar el **día 13 de junio** (deberá pasar correctamente todas las pruebas de las subrutinas `LongCar`, `BuscaCar`, `CoincidenCad`, `BuscaMax` y `CheckSum`).

Además, podrá pasar correcciones un máximo de **cuatro** veces en las planificadas para los días laborables entre el 1 y el 17 de junio.

Todas las correcciones se realizarán a partir de las 21:00. Para solicitar una corrección bastará con entregar correctamente los ficheros del proyecto antes de dicha hora límite.

El **examen del proyecto** se realizará el mismo día del examen extraordinario de teoría (viernes 24 de junio) en horario que se anunciará con la suficiente antelación.

TODAS LAS CONVOCATORIAS

La última entrega del proyecto no tiene carácter *obligatorio*. Es decir, una vez que los ficheros entregados por un grupo superan todas las pruebas del proyecto y la memoria tiene su versión definitiva, *no es necesario* que el grupo realice una nueva entrega para la última corrección.

Antes de cada examen del proyecto se indicará en la Web de la asignatura si se permitirá utilizar alguna documentación para realizar dicho examen y en caso afirmativo se especificará cuál. Asimismo, se indicará si se permite el uso de calculadora. En caso de permitirlo se referiría a un modelo básico, no programable. No se permitirá el uso de teléfonos móviles ni otros dispositivos con capacidad de almacenamiento y/o conexión remota.

TUTORÍAS DEL PROYECTO

Las posibles preguntas relacionadas con el proyecto se atenderán por correo electrónico en la dirección **pr_ensamblador@datsi.fi.upm.es** o presencialmente. Se tratará de dar respuesta por correo electrónico en un plazo breve, o alternativamente se concederá una cita en los casos necesarios. En cualquier caso, solicitamos que para cualquier consulta se contacte previamente con los profesores a través de la dirección indicada **describiendo en lo posible de forma concreta cuál es el problema que se trata de resolver**.

ENTREGA DEL PROYECTO

La entrega se compone de:

1. Una **memoria**, en formato DIN-A4, 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.

   Dicha memoria se entregará en formato electrónico, como un fichero PDF, mediante el sistema de entregas. Contendrá al menos los siguientes apartados:

- 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 el número total de horas invertidas en el proyecto por cada miembro del grupo, así como la relación de visitas 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.
- Observaciones finales y comentarios personales de este proyecto, entre los que se debe incluir una descripción de las principales dificultades surgidas para su realización.

NOTA: Esta memoria no debe incluir el listado en ensamblador del código generado por el grupo, por lo que sí será necesario que todas las subrutinas se encuentren adecuadamente comentadas en el propio fichero que se entrega para su corrección automática (el descrito en el siguiente punto, *CDV.ens*), ya que dicho fichero será consultado en el proceso de evaluación del proyecto.

2. La entrega de los ficheros que contienen el proyecto. Será obligatorio entregar los siguientes ficheros:

   - **autores (solo al dar de alta el 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:

     Nº Matrícula; DNI; apellido apellido, nombre; correo_electrónico

     El número de matrícula que se debe indicar en el fichero es el que **asigna la secretaría de la Escuela** (por ejemplo 990999) y no el que se utiliza como identificador para abrir cuentas en el Centro de Cálculo (por ejemplo a990999).

     La dirección de correo electrónico deberá ser una dirección válida que el alumno consulte frecuentemente. Se recomienda utilizar la dirección oficial asignada al alumno por la UPM, aunque se admiten otras direcciones personales.
   - **CDV.ens (en cada entrega)**: 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 y que funcione correctamente.
   - **memoria.pdf (en cada entrega)**: Será un fichero en formato PDF que contenga la memoria del proyecto. En la entrega inicial, el contenido del fichero `memoria.pdf` deberá contener al menos la identificación de los miembros del grupo que realiza el proyecto.

   IMPORTANTE: Antes de efectuar cada entrega del proyecto se recomienda realizar el ensamblado del fichero `CDV.ens` asegurando que no genera ningún error, así como ejecutar el código del proyecto con varios casos de prueba. De este modo minimizará la probabilidad de malgastar alguna de las correcciones disponibles. Se recuerda también que **no debe borrar** los programas principales ni los datos de prueba incluidos en `CDV.ens` antes de la entrega, aunque sí debe comprobar que no los ha situado en las direcciones reservadas al DATSI.

FORMA DE ENTREGA DE LOS FICHEROS

Los ficheros del proyecto se entregan mediante un navegador web convencional conectándose a la dirección:

`http://www.datsi.fi.upm.es/Practicas`

Recuerde que la información de su proyecto no debe ser conocida por otros alumnos o grupos, ya que si por cualquier razón lo fuera, podría verse involucrado en un caso de COPIA cuyas consecuencias le perjudicarán (están descritas en las normas de la asignatura). Por ello le recomendamos que siempre que trabaje en equipos de la Escuela o, en general, en cualquier equipo al que puedan acceder directa o indirectamente otros alumnos, lo haga siempre manteniendo los ficheros del proyecto en dispositivos privados (extraibles), evitando así dejar copias temporales en lugares accesibles por otros grupos.