

# Introducción al concepto y uso del ensamblador del Computador Teórico

Área de Arquitectura y Tecnología de Computadores – Versión 1.0.899, 18/02/2022

## Objetivos

En esta sesión se pretende que el alumno se familiarice con el concepto y el uso del ensamblador del Computador Teórico, además de escribir programas en lenguaje ensamblador, ensamblarlos, corregir errores y cargar los programas resultantes en el simulador. Además, se analizará el objetivo de las directivas básicas del ensamblador y cómo construir estructuras de control.

## Conocimientos y materiales necesarios

Para poder realizar esta sesión el alumno debe:

- Haber asimilado la forma de implementar sentencias utilizadas en los lenguajes de alto nivel en el lenguaje del Computador Teórico.
- Haber asimilado los conceptos de codificación y ejecución de instrucciones desarrollados en prácticas anteriores.
- Estudiar la teoría relativa al ensamblador que se encuentra en el anexo A del libro de teoría para conocer el uso de las directivas del ensamblador y su estructura.
- Llevar a clase el libro de teoría.
- Durante la sesión se plantearán una serie de preguntas que puedes responder en el correspondiente [cuestionario](#) en el Campus Virtual. Puedes abrir el cuestionario en otra pestaña del navegador pinchando en el enlace mientras mantienes pulsada la tecla `<ctrl>`.

## 1. El proceso de ensamblado


En sesiones anteriores has aprendido a codificar instrucciones para el computador teórico y a preparar archivos de memoria que pueden contener programas. Como habrás comprobado, el proceso de codificación es tedioso y se cometen errores con facilidad. Para simplificar este proceso se introduce una herramienta, denominada *ensamblador*, que permite realizar la codificación de instrucciones. Además, esta herramienta puede interpretar ciertas directivas que le indican cómo debe codificar las instrucciones así como facilitar el trabajo con datos.

El ensamblador acepta archivos escritos en un lenguaje denominado lenguaje ensamblador. Habitualmente, estos archivos tendrán extensión `.ens` y se crearán con un editor de texto. El ensamblador transformará estos archivos en archivos `.eje` que podrán ser cargados en el simulador. Si durante el proceso de ensamblado se encuentran errores, el ensamblador generará un mensaje de error y habrá que solucionarlo volviendo a editar el archivo `.ens`.

Al leer un "archivo ejecutable", el simulador de CPU interpreta los números de 16 bits en hexadecimal que allí encuentra, de la siguiente forma:

- El primer número indica a partir de qué dirección de memoria se cargarán los datos del fichero.
- El segundo número es el valor con que se inicializará el registro `PC`.
- El tercer número es el valor con que se inicializará el registro `r7`.
- Los restantes números hasta el final del fichero son equivalentes a los de un "archivo de memoria", y se irán cargando a partir de la dirección especificada en el punto 1.

Vamos a utilizar un archivo de ejemplo con errores para que veas cómo funciona el ensamblador. Realiza las siguientes operaciones:



- Copia el archivo `4-4prog1.ens` a tu carpeta de trabajo.
- Abre el archivo con un editor de texto. Fijate en las directivas que aparecen en el código, si no entiendes su significado deberías repasar el anexo del libro de teoría donde se explican.
- Cuando un programa se carga en memoria se divide en tres secciones: datos, código y pila. Estas tres secciones se cargan de manera consecutiva a partir de la dirección indicada en la directiva `ORIGEN`. Partiendo de esta información y analizando el archivo, contesta a estas preguntas: ¿En qué dirección estará almacenado el dato etiquetado como `operando1`? Responde en el [cuestionario](#): pregunta 1. ¿En qué dirección estará la primera instrucción? Responde en el [cuestionario](#): pregunta 2. ¿Qué valor habrá en la dirección `0502h` al final de la ejecución del programa? Responde en el [cuestionario](#): pregunta 3.
- Para comprobar que tus respuestas a las preguntas anteriores son correctas, tendrás que ensamblar el archivo y cargarlo en el simulador. Abre la interfaz de comandos, sitúate en tu carpeta de trabajo e introduce la siguiente orden:  

```
ensambla 4-4prog1.ens
```


- Como el archivo contiene errores, te deberá aparecer un mensaje de error con una posible explicación. Fijate en qué línea se encuentra el error. Edita el archivo y arréglalo. Tras guardar el archivo corregido y salir del editor, vuelve a intentar el ensamblado.
- Ahora no debería haber más errores. Si no es así, vuelve al paso anterior hasta que el ensamblador se ejecute sin mensajes de error. En este momento, comprueba que ha generado un archivo llamado `4-4prog1.eje` introduciendo la orden `dir` en la interfaz de comandos.
- Edita el archivo `4-4prog1.eje` para leerlo. ¿Cuál es la codificación de la instrucción `add r5, r3, r4` (recuerda que era la penúltima instrucción)? Responde en el [cuestionario](#): pregunta 4. Sal del editor.
- Abre el simulador y carga el archivo `4-4prog1.eje`. Comprueba con el desensamblador que las direcciones del `operando1` y de la primera instrucción coinciden con las que tú habías respondido anteriormente. Fijate que en esta primera instrucción, el ensamblador ha calculado la constante adecuada correspondiente al byte bajo de la dirección del operando. ¿Cuál es? Responde en el [cuestionario](#): pregunta 5.
- Ahora vas a ejecutar todas las instrucciones del programa. Fijate en que `PC` ya está inicializado a la dirección de la primera instrucción. Vete pulsando `F8` hasta que se ejecute la última instrucción. Comprueba entonces que el valor que hay en la dirección `0502h` coincide con el que tú habías previsto anteriormente.

Como has podido comprobar, el ensamblador simplifica mucho el desarrollo de programas.

## 2. Directivas

Vamos a ver ahora el uso de las directivas principales.


En primer lugar, nos vamos a fijar en la directiva `ORIGEN`. Al principio de la práctica apuntaste en qué dirección estaban el `operando1` y la primera instrucción. Además, acabas de ver que el ensamblador sustituye `BYTEBAJO DIRECCION operando1` por `00h`. Todas estas operaciones se hacen basándose en la directiva `ORIGEN`. Vamos a modificar ahora la dirección de carga del programa para que sea `ABCDh`. Sigue estos pasos:



- Edita el archivo. Modifica la directiva `ORIGEN` para que la dirección de carga sea `ABCDh`. Fijate que, para que el ensamblador sepa que `ABCDh` es un número y no una etiqueta, tienes que empezar el número con un cero.
- Guarda el archivo y ensámblalo.
- Carga el archivo `.eje` en el simulador. Comprueba con el desensamblador que la dirección del `operando1` es la esperada.
- Fijate en la primera instrucción. ¿Por qué constante ha cambiado el ensamblador la directiva `BYTEBAJO DIRECCION operando1`? Responde en el [cuestionario](#): pregunta 6.
- ¿Cuál es el valor inicial del registro `PC`? Responde en el [cuestionario](#): pregunta 7.

Al igual que estas dos últimas direcciones, ahora han cambiado todas las direcciones de los datos del programa, lo que ha modificado las instrucciones que tenían que ver con direcciones de memoria absolutas. Si hubieras tenido que realizar este proceso manualmente hubiera sido mucho más tedioso.


Fijate que el cambio en la dirección de carga (la que indica la directiva `ORIGEN`) ha afectado, lógicamente, al valor inicial de `PC`. El ensamblador calcula este valor como la dirección a la que apunta la etiqueta que siga a la directiva `INICIO`. Vamos a comprobar cómo funciona esta directiva:



- Edita el archivo y cambia la etiqueta `primera` del tal forma que apunte a la segunda instrucción. ¿Con qué valor se debería inicializar ahora el registro `PC`? Responde en el [cuestionario](#): pregunta 8.
- Guarda el archivo, ensámblalo y cárgalo en el simulador para comprobar si has respondido correctamente a la pregunta anterior.
- Vuelve a editar el archivo y vuelve a colocar la etiqueta `primera` apuntando a la primera instrucción de la sección de código.

¿Qué número tendrías que escribir tras la directiva `ORIGEN` para que al cargar en el simulador el archivo `.eje` el registro `PC` valiese `5B37h`? Responde en el [cuestionario](#): pregunta 9. Comprueba que tu respuesta es correcta poniendo ese valor en el archivo, ensamblándolo y cargándolo en el simulador.


Ahora vamos a fijarnos en la directiva `VALOR`. Esta directiva sirve para introducir datos. Tienes tres ejemplos en el archivo `4-4prog1.ens`. La directiva también permite introducir varios valores en la misma línea separados por comas. En ese caso la etiqueta apuntará al primer valor de la lista. Vamos a modificar el programa para que se declaren en una sola línea los dos operandos:



- Abre el archivo `4-4prog1.ens` en el editor y guárdalo con el nombre `4-4prog2.ens`. No salgas del editor.
- Sustituye las dos líneas donde se definen los operandos por la siguiente línea:  


```
operandos VALOR 3, 0Ch
```
- Modifica ahora las instrucciones de carga de los operandos para que funcionen con el cambio anterior. Fijate en que ahora sólo tienes la etiqueta `operandos`, con lo que el primer operando estará en la dirección de `operandos` y el segundo operando estará una posición más allá.
- Guarda el archivo, ensámblalo, cárgalo en el simulador y ejecútalo para comprobar que funciona.

La directiva `VALOR` permite utilizar constantes en varios formatos. Vamos a ver algunos ejemplos:



- Edita el archivo `4-4prog2.ens`.
- Sustituye el valor del primer operando por la constante de carácter `'P'`.
- Guarda el programa y ensámblalo.
- Abre el archivo `4-4prog2.eje` que se ha generado. ¿Por qué número ha cambiado el ensamblador la constante de carácter `'P'`? Responde en el [cuestionario](#): pregunta 10.
- Vuelve a editar el archivo fuente `4-4prog2.ens`. Sustituye la constante `'P'` por la cadena `"PQRS"`. Tras este cambio, ¿qué valor tendrá inicialmente el registro `PC` cuando cargues el archivo ejecutable? Responde en el [cuestionario](#): pregunta 11. ¿Cuál será la dirección del resultado? Responde en el [cuestionario](#): pregunta 12. ¿Qué dos números se sumarán? Responde en el [cuestionario](#): pregunta 13.
- Guarda el programa y ensámblalo.
- Carga el archivo ejecutable generado en el simulador. Comprueba, ejecutando el programa, que tus respuestas a las preguntas anteriores son correctas.

Vamos a ver ahora el uso de la directiva `VECES`, que sirve para declarar varios datos con el mismo valor. Realiza las siguientes operaciones:



- Abre en el editor el archivo `4-4prog2.ens` y guárdalo con el nombre `4-4prog3.ens`.
- Sustituye la primera línea de la sección datos por esta:  

```
operandos VALOR 8 VECES 3
```

Esta línea hará que se declaren consecutivamente 8 datos con valor 3, al primero de los cuales apuntará la etiqueta `operandos`.
- Guarda el archivo, sal del editor y ensámblalo.
- Edita el archivo `4-4prog3.eje` generado. ¿En qué dirección está ahora el resultado? Responde en el [cuestionario](#): pregunta 14. ¿Qué dos números se sumarán? Responde en el [cuestionario](#): pregunta 15.
- Carga el archivo `4-4prog3.eje` en el simulador y comprueba si tu respuesta a las preguntas anteriores fue acertada.


## 3. Ejercicios adicionales

### 3.1. Archivos en la carpeta de trabajo

En tu carpeta de trabajo de prácticas deberás tener los archivos de programa `4-4prog1.ens`, `4-4prog2.ens` y `4-4prog3.ens`, además de sus correspondientes archivos `.eje`.

### 3.2. Ejercicios

✓ Copia el archivo `4-4prog1.ens` a otro llamado `4-4prog4.ens` y haz las siguientes modificaciones en este nuevo archivo:



- Añade una sección de datos en la que definas: un dato llamado `num1` que valga `1101b`, otro dato llamado `num2` que valga `-3` y otro dato sin nombre que vaya a continuación del anterior y valga `9`.
- Modifica el código del programa para que se cargue en el registro `r3` `num1` y en el registro `r4` `num2`.
- Comprueba que el programa funciona correctamente