CAPÍTULO 2

## **Definiendo constantes: Directiva .eqv**

En nuestros programas tendremos que usar **constantes** numéricas. Por ejemplo para establecer el valor inicial de un contador, o el de determinadas variables. Estas constantes siempre tienen el **mismo valor** durante toda la ejecución del programa

Es una **buena práctica** de programación el **definir** las constantes al **principio** del programa, de forma que se puedan cambiar fácilmente, sin tener que navegar por el código

Para **definir constantes** usaremos la **directiva .eqv**. Este es un **ejemplo** de uso

Se trata de una **directiva**, por lo que **NO** genera código máquina. Le dice al ensamblador que cada vez que vea el identificador INICIAL lo sustituya por el valor 5, en este caso

### Ejemplo: Contador con valor inicial

Modificaremos el programa del contador para que empiece su cuenta con un **valor inicial**, establecido mediate una **constate**. El programa es:

#-- Contador con valor inicial definido mediante

#-- un identificador

#-- Definir el valor inicial para el contador

.eqv INICIAL 20

.text

#-- Inicializar contador

#-- El ensamblador sustituye el identificador INICIAL

#-- por el numero 20

li x5, INICIAL

bucle:

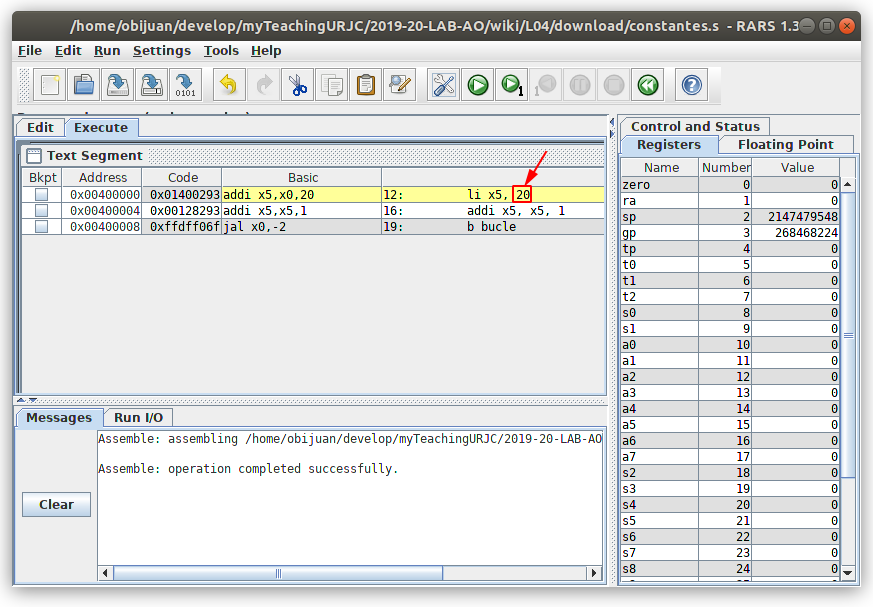
#-- Incrementar contador: x5 = x5 + 1

addi x5, x5, 1

#-- Repetir

b bucle

**Ensamblamos** el programa, y nos fijamos en la ventana del segmeno de código. Vemos que la instrucción de inicilización que aparece es **li x5, 20**



En el editor habíamos especificado **li x5, INICIAL**, pero el ensamblador ha sustituido el identificador INICIAL por su **valor 20**. Es el mismo comportamiento que ocurre con el **#define** del **lenguaje C**

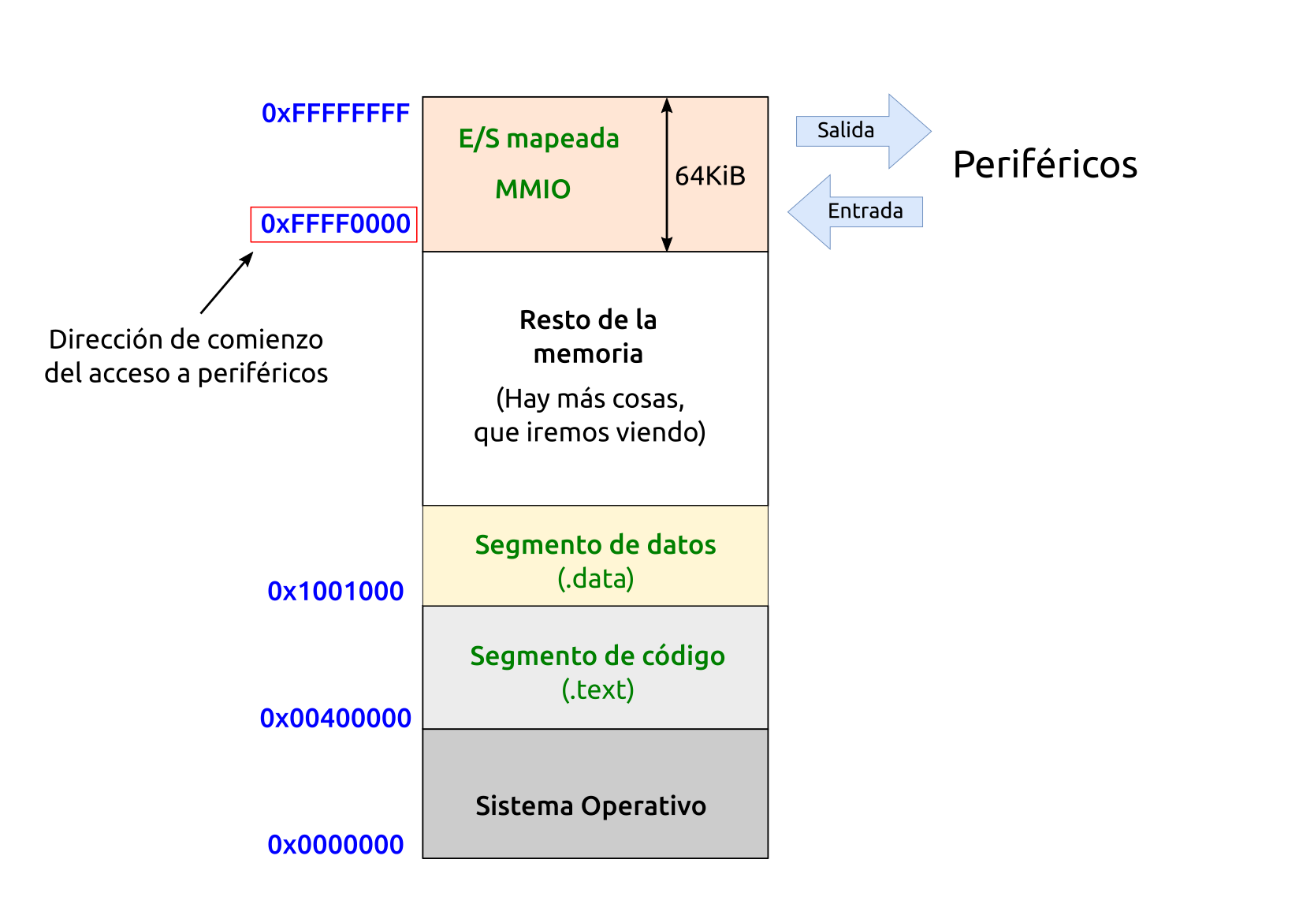
Entradas y salidas de periféricos

**Nuestros programas** se dividen en **código** y **datos**, ambos almacenados en la **memoria**. Cada uno en su segmento: de código o de datos. Las **instrucciones** del programa se almacenan en el **segmento de código**, y las **variables** en el **segmento de datos**. Nos falta una parte muy importante. ¿Cómo se comunica el procesador con el exterior?. Necesitamos una forma de realizar la **Entrada/Salida** de datos (E/S)

**Los periféricos se mapean en la memoria, es decir, tienen su entrada y salida conectadas a la memoria y desde el punto de vista del procesador se ve igual que entrar en memoria.**

## Periféricos mapeados en memoria (MMIO)

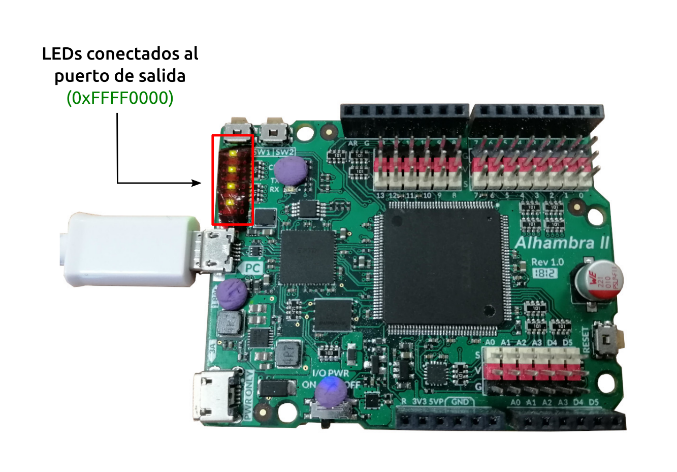
La **zona** del mapa de memoria situada en la parte superior se denomina **MMIO** (Memory mapped I/O): **Entrada/salida mapeada en memoria**. En esas direcciones se encuentran todos los **periféricos** de nuestro sistema. La **comunicación** con ellos se realiza igual que un **acceso a memoria**: Con la instrucción **store** se envía información a los periféricos y con la instrucción **load** se lee de ellos. Pero para el RISC-V es **exactamente igual que cualquier otro acceso a memoria**



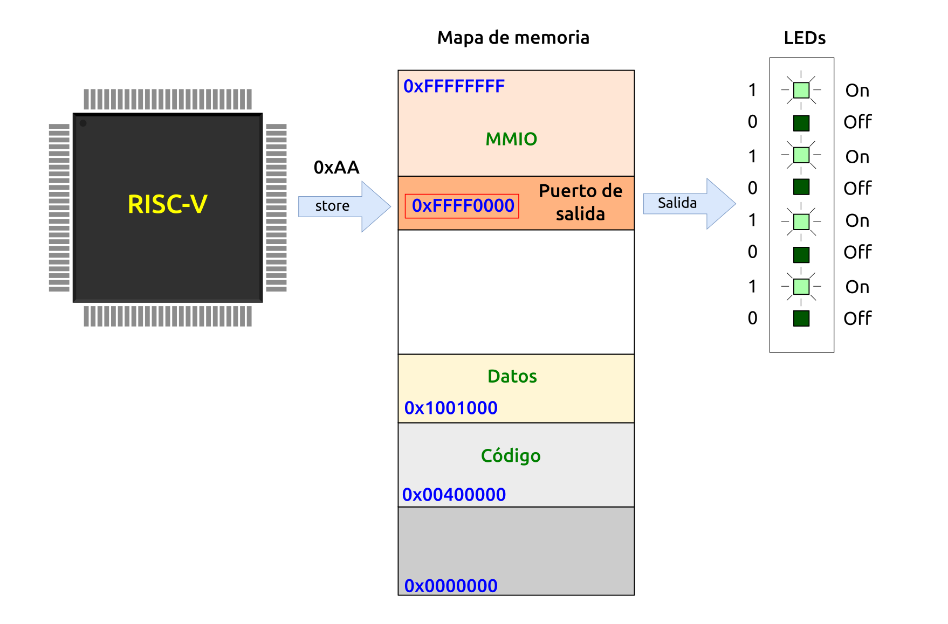
El **acceso** a estas posiciones de memoria es igual que al resto de la memoria: necesitamos colocar en un registro la **dirección** en donde se encuentra el periférico. Usaremos las instrucciones de **load** y **store** para leer o escribir en el periférico respectivamente

### Puerto de salida de 8 bits

Una manera muy utilizada para **enviar información** hacia el **exterior** es mediante los **puertos de salida** de 8 bits (aunque pueden ser de mayor cantidad de bits). Lo que se encuentra **mapeado** es un **registro**. Al hacer un **store** se escribe el valor en este registro, y sus **bits** salen al **exterior** a través de los pines



El **puerto de salida** del [RISC-V que tenemos en la FPGA](https://github.com/Obijuan/RISC-V-FPGA) está en la dirección **0xFFFF0000**, y los **8 LEDs** están conectados a este puerto. De esta forma, al escribir un valor de 8 bits en esa dirección, aparece en binario en los LEDs. Cada **bit a 1** se correponde con un **LED encendido**, y cada **bit a 0** con un **LED apagado**

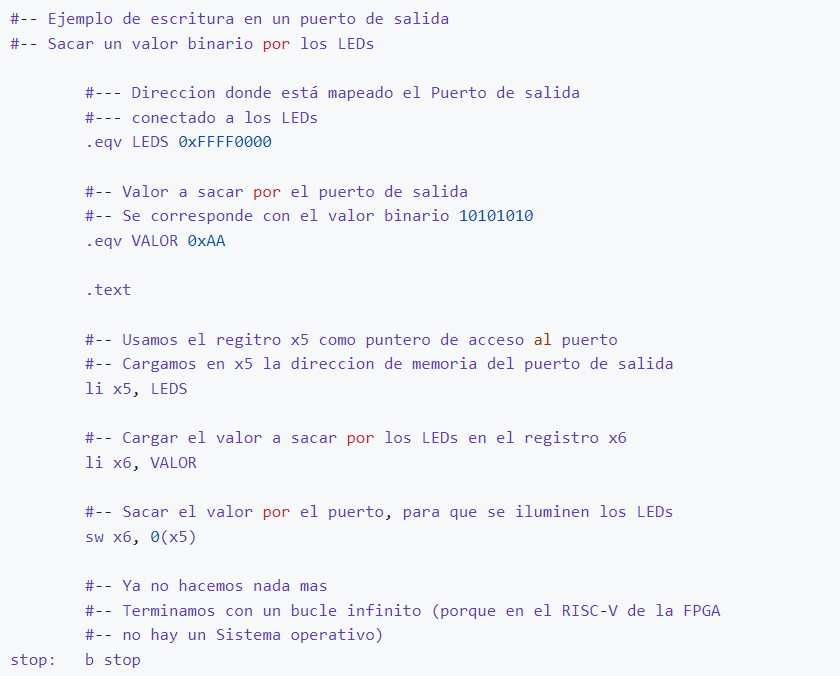


Así, si **escribimos** el valor **0xAA** (que en binario es 10101010) en la dirección **0xFFFF0000** se encenderán **4 leds**, correspondientes a los cuatro unos del valor 0xAA

### Escribiendo un valor en el puerto de salida

Vamos a hacer un programa para **escribir** nuestro primer dato en el puerto de salida. El valor que **típicamente** se utiliza es **0xAA** por tener los bits a 1 y 0 alternados. En el **simulador** simplemente veremos este valor **almacenado** en "la memoria", pero en el **sistema real** comprobaremos que se **encienden los LEDs**

Para **escribir** en este **puerto de salida** necesitamos almacenar en un registro **0xFFFF0000**, que es la **dirección** del puerto de salida. Luego cargamos 0xAA en otro registro y hacemos el **store**



## **Display de 7 segmentos**

A través del **puerto de salida** hemos controlado **8 LEDs** que estaban colocados en **línea recta**, uno detrás de otro. Un **display de 7 segmentos** está formado por **8 LEDs colocados en forma de dígito**. 7 de los leds se corresponden con los **segmentos** verticales y horizontales y el octavo es el **punto**. Esta es la pinta que tiene el componente

### 

### Nomenclatura

Para dibujar los dígitos necesitamos **nombrar** los diferentes **segmentos**, igual que hemos hecho en los ejemplos anteriores. La nomenclatura estándar es usar las letras **a**, **b**, **c**, **d**, **e**, **f** y **g** para los segmentos. El **punto** lo llamaremos **p**

Cada dígito lo controlaremos con un **bit independiente**. El **bit 0** se corresponde con el **segmento a**, el **bit 1** con el **b**,..., el **bit 6** con el **f** y finalmente el **bit 7** (de mayor peso) con el **punto**

