Simulación de E/S en ARM

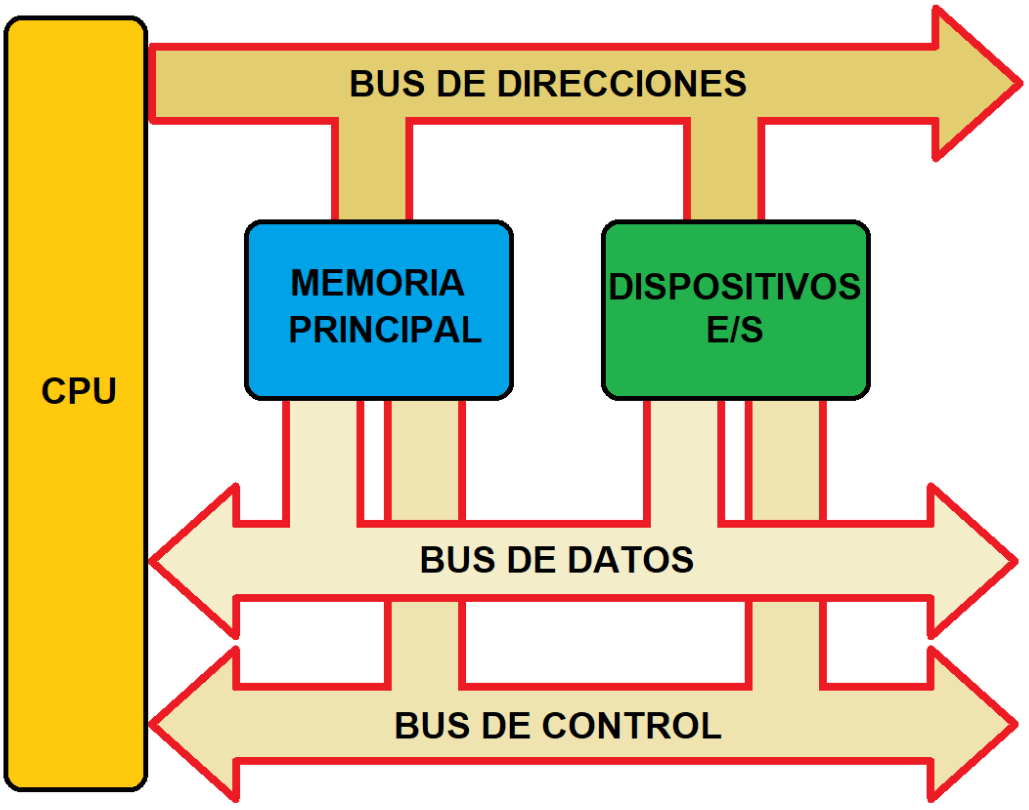
Dispositivos de Entrada / Salida

Si bien hay infinidad de dispositivos periféricos con características y particularidades diferentes, todos tienen algo en común:

1. El procesador los tiene que **identificar**.
2. El procesador tiene que **recibir y/o enviarles datos**.
3. Procesador y periférico deben estar **sincronizados para comunicarse**.

El hardware encargado de cumplir estas funciones es:

1. **Identificación**: Bus de direcciones.
2. **Comunicación de datos**: Bus de datos.
3. **Sincronización**: Bus de control.



Interconexión entre el procesador y los dispositivos periféricos mediante buses

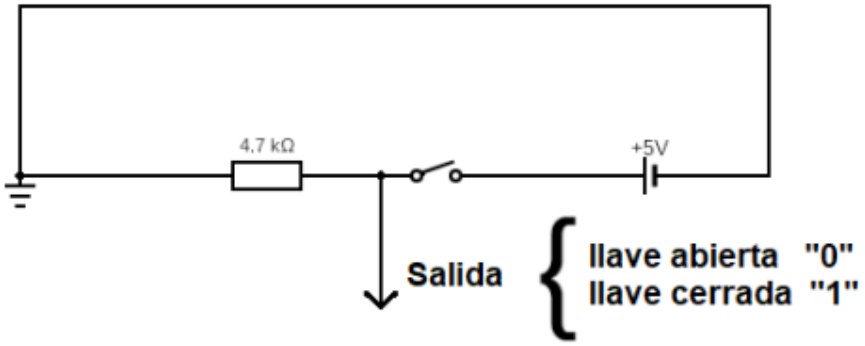
Sincronización

La sincronización coordina la diferencia de velocidad entre la CPU y los dispositivos periféricos. La sincronización puede ser programada o por interrupciones.
La sincronización programada consiste en un programa que genera una encuesta (poolling) a los dispositivos de E/S para ver si requieren atención. Dado que, al proceder con esta interrupción, el programa principal se detiene, haya o no una necesidad real de atender al periférico en ese momento, se genera una pérdida de tiempo. Una analogía puede ser la de un mozo en un restaurante recorriendo las mesas preguntando a cada uno si necesita algo.

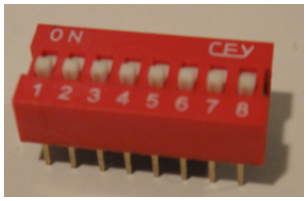
En la sincronización por interrupciones se genera una interrupción cuando el dispositivo periférico requiere ser atendido. Una analogía puede ser la de un comensal en un restaurante que llama al mozo.

Dos periféricos básicos

Un dispositivo de entrada básico para el procesador, de modo que pueda leer un 0 o un 1, es una llave (switch). Según esté abierta o cerrada, representa un 0 o un 1 lógicos.



Un DIP Switch es un componente que tiene un conjunto de llaves que pueden ser conectadas a un circuito.



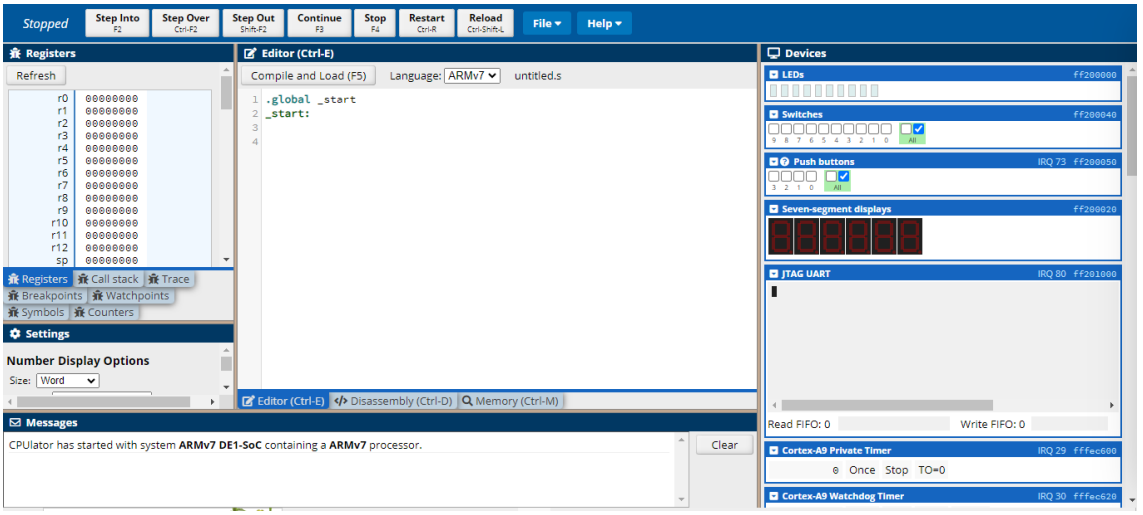
DIP Switch

Un dispositivo de salida básico que al recibir un “1” lógico por parte del procesador se enciende y al recibir un “0” lógico se apaga es un LED (diodo emisor de luz).

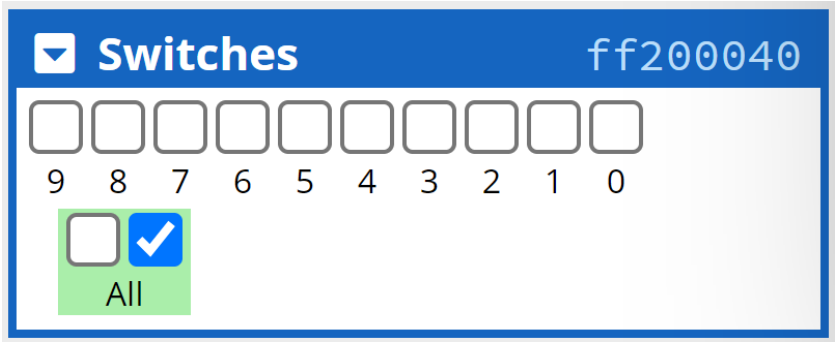
LED

Simulador

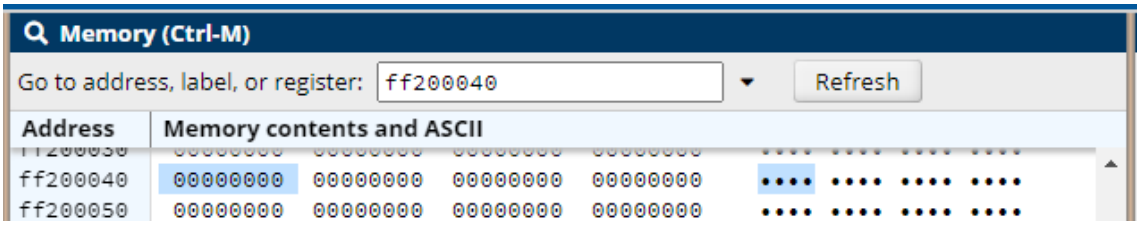
En el siguiente enlace se puede encontrar un simulador online para ARM con algunos dispositivos periféricos: https://cpulator.01xz.net/?sys=arm-de1soc



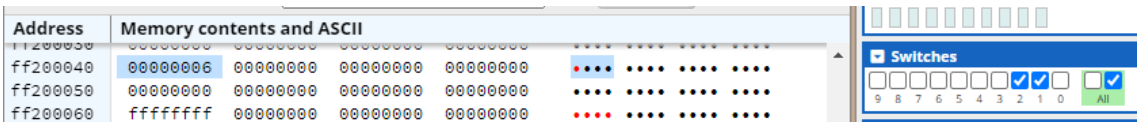
A la derecha puede observarse un conjunto de llaves similares a un DIP switch, que tiene asignada la dirección FF200040 del mapa de memoria.



Al hacer click con el mouse sobre la dirección, puede verse el contenido de la memoria para esa dirección.



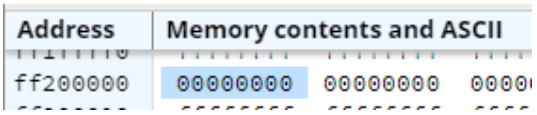
Cada bit almacenado en esa dirección representa un 0 o un 1 de acuerdo si la llave está abierta o cerrada.



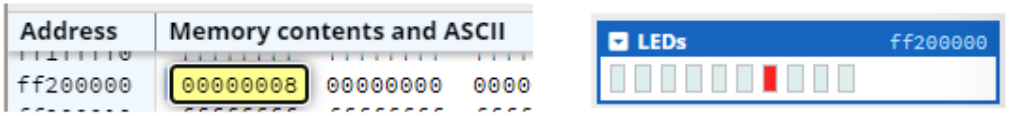
Los pesos de los bits que se corresponden con las llaves cerradas son 1 y 2, de lo que resulta en binario el número 110. En hexadecimal es el dato 6 de la posición de memoria asignada a la llave.

De manera similar, los leds tienen asignada la dirección FF200000

El contenido de la memoria en esa dirección es 0, dado que los LEDs están apagados:



Cada LED representa un bit de una palabra binaria. Si en la memoria se escribe, por ejemplo, el número 8 (0000001000 en binario), se encenderá el LED correspondiente al “1”:



Programa

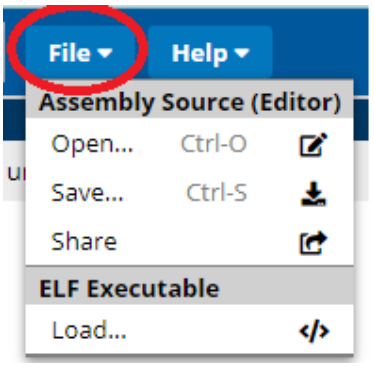
El siguiente programa va a encender un LED en cada posición correspondiente al SWITCH que esté en “1”.

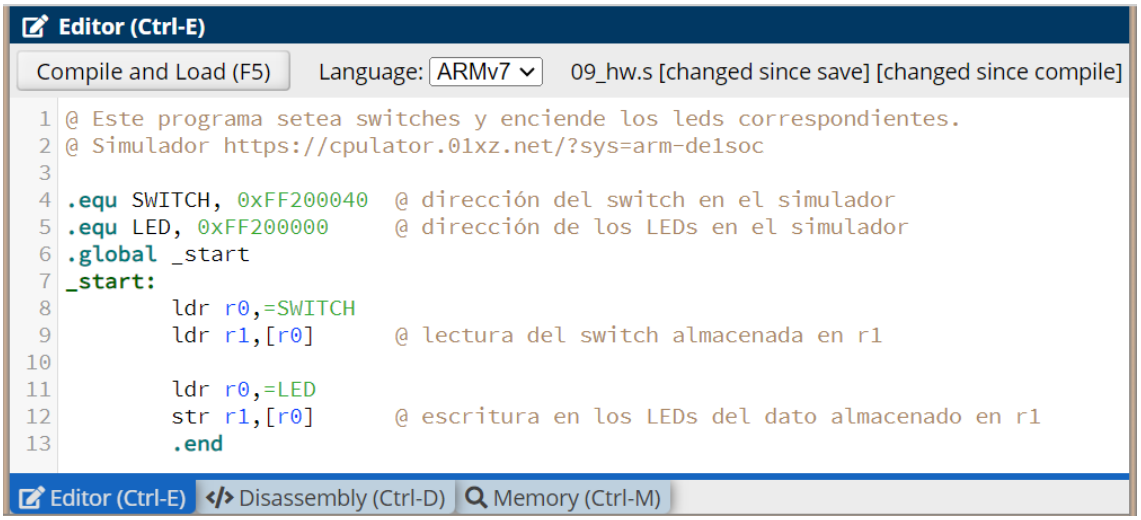
El algoritmo consiste en leer el periférico SWITCH, almacenar ese dato en un registro y escribirlo en el periférico LED.

.equ SWITCH, 0xFF200040 y .equ LED, 0xFF200000 permiten asociar las direcciones del switch y del LED que están en las ventanas del switch en el simulador a un nombre sencillo fácil de recordar.

El registro r0 almacena dichas direcciones y el registro r1 el dato que se lee del dispositivo de entrada SWITCH y se escribe en el dispositivo de salida LED.

El programa puede escribirse en el editor de texto del simulador o bien en el editor o IDE preferido y subirlo al simulador.



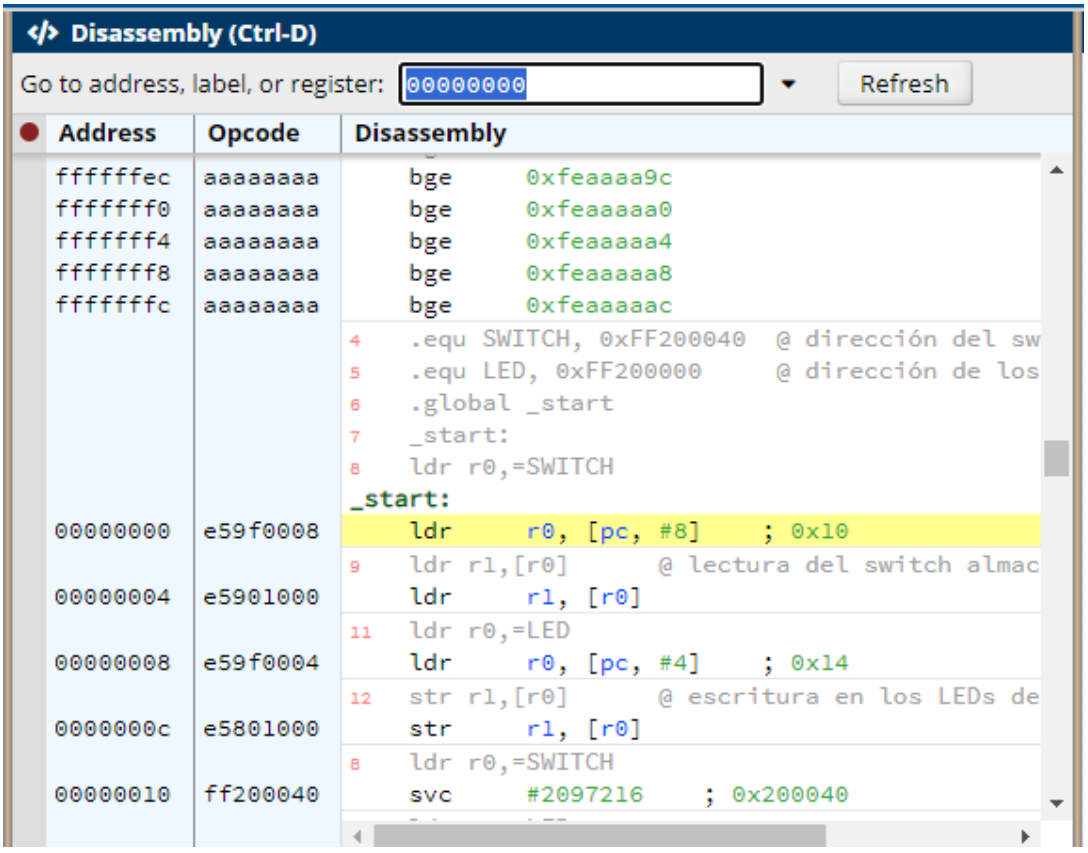



```
.equ SWITCH, 0xFF200040          @ la dirección está en la ventana del switch en el simulador
.equ LED, 0xFF200000             @ la dirección está en la ventana del LED en el simulador
.global _start
_start:
        ldr r0,=SWITCH
        ldr r1,[r0]              @ lectura del dispositivo de entrada almacenada en r1

        ldr r0,=LED
        str r1,[r0]              @ escritura del dispositivo de salida con el dato de r1
        .end
```

Simulación

- Ensamblar el programa ---> tecla F5
- Poner las llaves en la posición deseada.
- Abrir la ventana Disassembly para ver el programa en assembler y el código objeto.
- Poner la dirección 00000000 en el campo de texto “Go to address” para ir a la dirección del código.



- Observar cada línea de código, determinar que se va a almacenar en el registro en cada instrucción ldr y que se va a almacenar en la memoria en cada instrucción str. Ejecutar paso a paso con F2 esa línea de código y verificar si lo que ocurre coincide con lo analizado. Observar la ventana de registros y la ventana de memoria en la dirección de los periféricos.

- Finalmente verificar si los LEDs siguen el patrón de bits seleccionado en cada switch

- Repetir para otra combinación de los switches.

Preguntas

a. ¿Qué otro periférico puede representar un dispositivo de entrada?
b. ¿Qué otro periférico puede representar un dispositivo de salida?
c. ¿Sólo con observar la instrucción ldr r1,[r0] puede saberse si lee un periférico o la memoria principal?
d. ¿Sólo con observar la instrucción str r1,[r0] puede saberse si escribe en un periférico o en la memoria principal?
e. ¿En este ejercicio el procesador y los periféricos se sincronizan por pooling o por interrupción?
f. ¿Qué es una tabla de símbolos?
g. ¿Qué símbolos aparecen en la ventana de la tabla de símbolos y cuáles son sus valores?
h. ¿Qué dirección se ve en la memoria al hacer clic con el mouse sobre la lupa en el símbolo _end de tabla de símbolos? ¿Cuántos datos distintos se ven en esa fila? ¿Qué dirección tiene cada dato? ¿Por qué las direcciones van de 4 en 4? ¿Qué valor tienen los dos bits menos significativos de cada dirección y por qué?
i. ¿Cómo se escribe en lenguaje de máquina la instrucción ldr r1, [r0]?