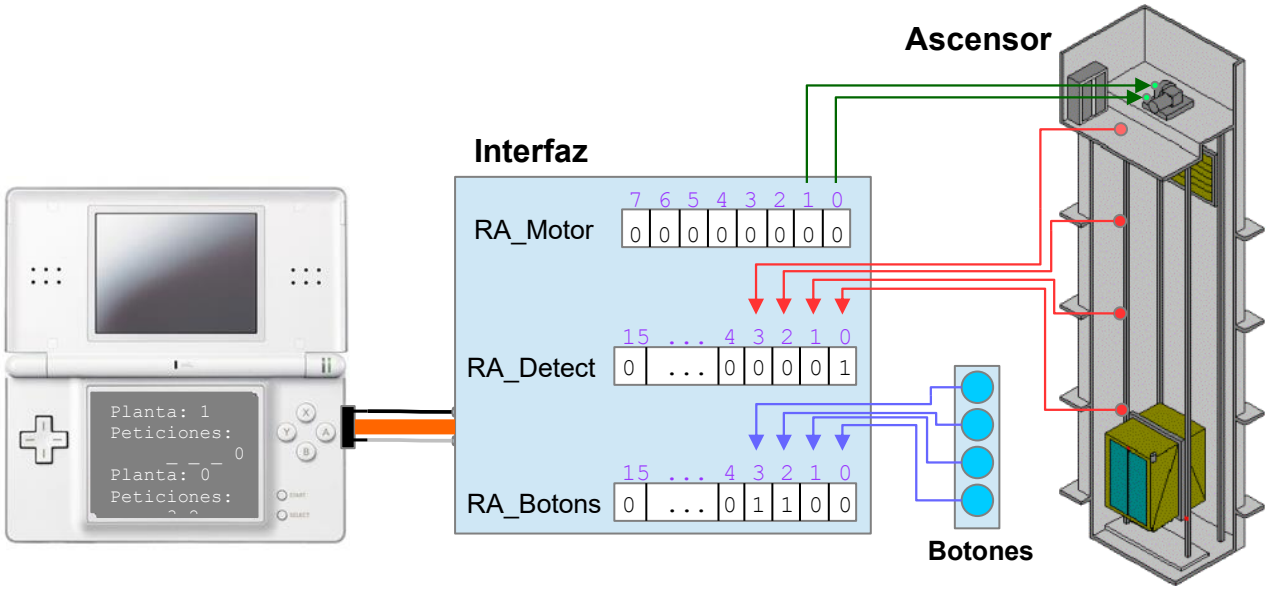
# Problema 22: Ascensor con memoria (Ex. 1ª Conv. 2017-18)

Se propone conectar una interfaz electrónica para controlar un ascensor con la NDS, suponiendo que no habrá plantas de sótano:




La interfaz permite acceder a tres registros de Entrada/Salida, que tienen las siguientes funcionalidades:

| *Registro* | *Bits* | *Función* |
|---|---|---|
| `RA_Motor` | 8 | Bit 0: activar (=1) / desactivar (=0) motor en subida<br>Bit 1: activar (=1) / desactivar (=0) motor en bajada |
| `RA_Detect` | 16 | Bits *n*-1..0: el bit *i*-ésimo se activa (=1) automáticamente cuando el ascensor está situado justo delante de la puerta de la planta *i*; el resto de bits estarán a 0; cuando el ascensor esté entre puertas, todos los bits estarán a 0 |
| `RA_Botons` | 16 | Bits *n*-1..0: el bit *i*-ésimo se activa (=1) automáticamente cuando algún usuario pulsa sobre el botón de la planta *i*; la petición queda memorizada en el propio registro hasta que, por programa, se escriba un 0 sobre ese bit; en todo momento pueden haber cero, una o más peticiones activadas (pendientes) |

En el esquema de ejemplo, el número de plantas *n* es 4, de modo que el valor de *i* oscilará entre 0 (planta baja) y 3 (piso más alto). Sin embargo, hay que realizar el programa de control configurable para estructuras de ascensor entre 2 y 16 plantas ($2 \leq n \leq 16$).

La interfaz generará una interrupción IRQ_CART cada vez que se active un bit del registro

`RA_Detect`, es decir, cuando el ascensor llegue (pase por delante) de la puerta de una determinada planta.

El programa a implementar deberá comprobar periódicamente si hay peticiones pendientes; en caso afirmativo y si el ascensor está parado, habrá que cerrar la puerta y activar el motor para mover el ascensor hacia las plantas solicitadas; si el ascensor está en marcha y llega a una nueva planta, habrá que escribir por pantalla información sobre el estado del ascensor (ver más adelante) y, si había una petición pendiente sobre esa planta, parar el ascensor y abrir la puerta.

Todas estas tareas se deberán realizar concurrentemente con determinadas tareas independientes. Se podrán realizar múltiples peticiones, con el ascensor en marcha o parado, y se deberán ir atendiendo todas las peticiones en un sentido (de subida o de bajada) y, cuando se hayan terminado las peticiones en ese sentido, pasar a atenderlas en el otro sentido. Al iniciar el programa supondremos que el ascensor está situado en la planta baja.

Se dispone de las siguientes rutinas, ya implementadas:

| *Rutina* | *Descripción* |
|---|---|
| `inicializaciones()` | Inicializa el *hardware* (pantalla, interrupciones, etc.) |
| `tareas_independientes()` | Tareas que no dependen del movimiento del ascensor (ej. calcular el peso de los ocupantes del ascensor); tiempo de ejecución < 0,1 s |
| `abrir_puerta()` | Abre la puerta del ascensor y espera un cierto tiempo antes de retornar (10 segundos, aproximadamente) |
| `cerrar_puerta()` | Cierra la puerta del ascensor comprobando que no haya obstáculos, en cuyo caso la volverá a abrir y después repetirá el proceso de cerrarla (5 segundos, mínimo) |
| `swiWaitForVBlank()` | Espera hasta el próximo retroceso vertical |
| `printf(char *format,...)` | Escribe un mensaje en la pantalla inferior de la NDS |

Además de gestionar el control general del ascensor, el programa principal deberá escribir por la pantalla inferior de la NDS el número de planta actual y las peticiones pendientes, cada vez que el ascensor llegue o pase por delante de la puerta de una planta. El formato del texto deberá ser el siguiente (sin los comentarios entre paréntesis):

```
Planta: i                   (0 ≤ i ≤ n-1)
Peticiones:                 (si planta i está solicitada, pi = "i ")
(tabulador) pn-1 pn-2 … p2 p1 p0                (sino, pi = "_ ")
```

Por ejemplo, para una configuración con 8 plantas ($n = 8$), si el ascensor llega a la planta 5 ($i = 5$) y están solicitadas las plantas 6, 2 y 1, la salida será la siguiente:

```
Planta: 5
Peticiones:
     _ 6 _ _ _ 2 1 _
```

Se sugiere el uso de las siguientes variables globales y definiciones:

```
#define NUM_PLANT   4              // número de plantas a gestionar

unsigned char planta_actual = 0;   // número de planta actual
unsigned char sentido = 1;         // sentido de movimiento actual
                                   //   = 1 → subida,
                                   //   = 2 → bajada
```

El `#define` permitirá configurar el número de plantas a gestionar para una determinada estructura del ascensor (*n*). La variable `planta_actual` deberá registrar en todo momento el número de planta actual (*i*), cuyo valor siempre estará entre 0 y `NUM_PLANT`-1. La variable `sentido` deberá registrar el sentido actual del movimiento, ya sea de subida (=1) o de bajada (=2).

Por su parte, la RSI del ascensor deberá actualizar la variable global `planta_actual` para reflejar la planta donde se encuentra el ascensor en todo momento. Además, si la planta a la que acaba de llegar el ascensor estaba solicitada, deberá parar el motor inmediatamente, así como desactivar el bit correspondiente de petición de planta.

Por último, será necesario programar una rutina de soporte con la siguiente cabecera:

```
unsigned char siguiente_movimiento(unsigned short peticiones,
                                   unsigned char p_actual,
                                   unsigned char *s_actual);
```

Esta rutina calcula cual es el siguiente sentido del movimiento del motor, según el estado actual de los bits de petición (`peticiones`), la planta actual (`p_actual`) y el sentido actual del movimiento (`s_actual`); el tercer parámetro se pasa por referencia para poder ser modificado directamente en memoria desde la propia rutina.

Para determinar el nuevo sentido, la rutina tiene que recorrer los bits de petición de planta a partir del bit correspondiente a la planta actual, comprobando si existe alguna petición pendiente en el sentido de movimiento actual; si la encuentra, no habrá cambio del sentido actual; en caso contrario, se cambiará el sentido del movimiento.

Además de cambiar directamente el contenido de la variable del sentido actual, su valor final se devuelve también como resultado de la rutina. Para simplificar el funcionamiento de esta rutina, se puede suponer que solo se llamará si existe alguna petición pendiente, aunque habrá que asegurarse de esta condición antes de llamar a la rutina. Además, vamos a suponer que la interfaz no registrará peticiones sobre la planta en la que esté situado el ascensor cuando esté parado.

## Se pide:

Programa principal y variables globales en C,

RSI del ascensor y rutina `siguiente_movimiento()` en ensamblador.