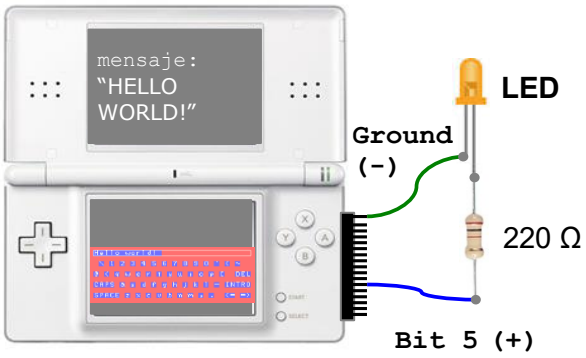
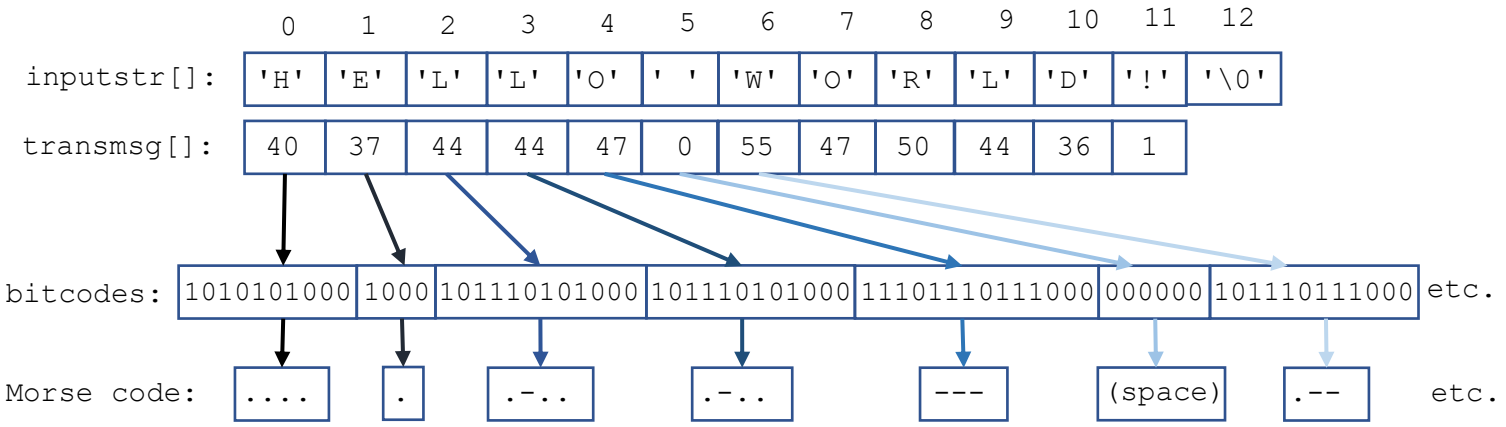
Problema 37: Escritura Morse (Ex. 2ª Conv. 2021-22)

Se propone realizar un programa para la NDS que permita visualizar un mensaje en código Morse utilizando un diodo led conectado al bit 5 de un registro de entrada/salida de nombre simbólico REG_DATA (16 bits). A continuación se muestra un esquema del *hardware*:



El programa incluirá una función (ya implementada) para leer el mensaje que el usuario quiera reproducir en Morse. Esta función mostrará un teclado virtual sobre la pantalla táctil de la NDS, más un buffer (de una línea) que mostrará las letras que se van introduciendo. Cuando el usuario ya tenga el mensaje listo, pulsará la tecla virtual RETURN y el mensaje se preparará para ser retransmitido como señales luminosas cortas (puntos) y largas (rallas) a través del led. Mientras se esté retransmitiendo dichas señales, el usuario podrá ir introduciendo el mensaje siguiente con el teclado virtual.

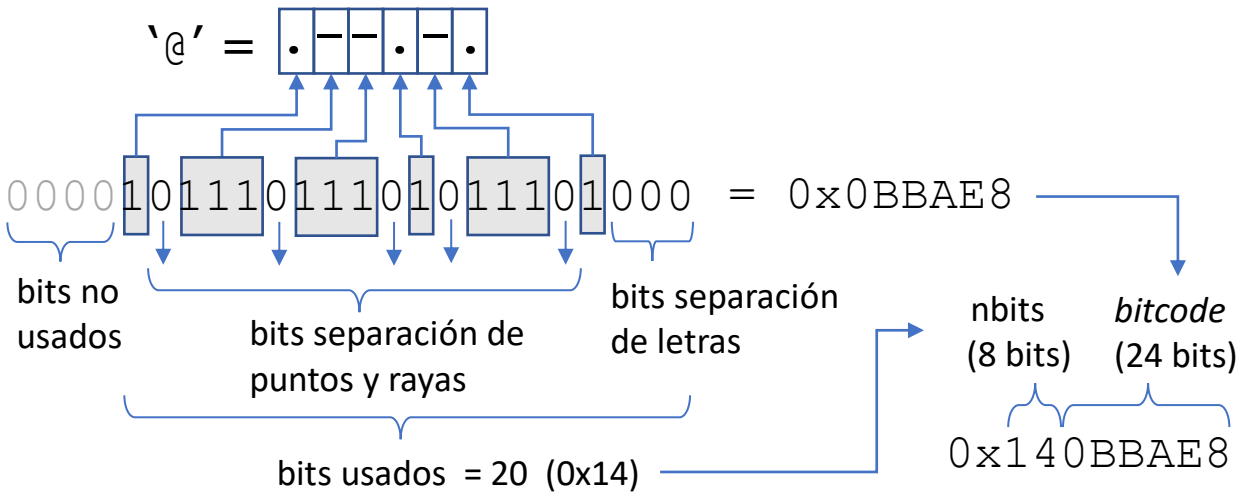
Para reproducir el código Morse correspondiente, los códigos ASCII del mensaje, contenidos en un *string* de nombre `inputstr[]`, se transformarán en un vector de códigos numéricos, de nombre `transmsg[]`. Cada código numérico sirve para indexar otro vector, que se explicará más adelante, que contiene un *bitcode* para cada letra. Este *bitcode* consiste en patrones de bits que representan un punto con 1 uno y una raya con 3 unos seguidos, más 1 cero de separación entre puntos y rayas, 3 ceros de separación entre letras y 6 ceros de separación entre palabras. El siguiente esquema muestra un ejemplo de generación de una secuencia de *bitcodes* para el mensaje "HELLO WORLD!":



El vector `nbcode[]` contiene los *bitcode* correspondientes a cada letra. Este vector ya está inicializado con 59 *words*. A continuación se muestra una tabla que representa el contenido de algunas posiciones de dicho vector. Para cada posición se muestra el índice (`ind`), la letra correspondiente (`char`), el código de puntos y rayas (`Morse`), el número de bits útiles del *bitcode* (`nbits`), el código binario (*`bitcode`*) y el valor específico de 32 bits que está almacenado finalmente en el vector (`nbcode[ind]`), expresado en hexadecimal:

ind	char	Morse	nbits	*bitcode*	nbcode[ind]
0	' '	(space)	6	00000000000000000000000000000000	0x06000000
1	'!'	-.-.--	22	00111010111010111011100	0x163AEBB8
2	'"'	.-..-.	18	00000010111010101110100	0x1202EAE8
⋮					
16	'0'	-----	22	00111011101110111011100	0x163BBBB8
17	'1'	.----	20	00001011101110111011100	0x140BBBB8
18	'2'	..---	18	00000010101110111011100	0x1202BBB8
19	'3'	...--	16	00000000101010111011100	0x1000ABB8
⋮					
32	'@'	.--.-.	20	00001011101110101110100	0x140BBAE8
33	'A'	.-	8	00000000000000001011100	0x080000B8
34	'B'	-...	12	00000000000011101010100	0x0C000EA8
35	'C'	-.-.	14	00000000001110101110100	0x0E003AE8
36	'D'	-..	10	00000000000001110101000	0x0A0003A8
37	'E'	.	4	00000000000000000001000	0x04000008
⋮					
56	'X'	-..-	14	00000000001110101011100	0x0E003AB8
57	'Y'	-.--	16	00000000111010111011100	0x1000EBB8
58	'Z'	--..	14	00000000001110111010100	0x0E003BA8

Los *bitcode* tienen longitud variable, oscilando entre 4 y 22 bits, de modo que es necesario registrar la longitud de cada *bitcode* (`nbits`). Cada *word* almacenado en `nbcode[]` empaqueta dicha longitud en los 8 bits altos y el propio *bitcode* en los 24 bits bajos. El siguiente esquema muestra un ejemplo de codificación para la letra '`@`':



Se dispone de las siguientes rutinas ya implementadas:

Rutina	*Descripción*
`inicializaciones()`	Inicializa el *hardware* (pantalla, interrupciones, etc.)
`activar_timer( int ntim, int nfreqin, short divfreq)`	Activa el funcionamiento e interrupciones del *timer* `ntim`, con el identificador de la frecuencia de entrada `nfreqin` (0, 1, 2 o 3) y con el divisor de frecuencia `divfreq`
`desactivar_timer(int ntim)`	Desactiva el funcionamiento del *timer* `ntim`
`leer_mensaje(char *str, int maxlength)`	Gestiona la interfaz que permite al usuario introducir un *string* mediante un teclado virtual sobre la pantalla táctil; el *string* se devuelve por referencia sobre el parámetro `str`, limitado a la longitud indicada en `maxlength` (centinela excluido); la rutina no retorna hasta que el usuario valida el *string* pulsando la tecla virtual RETURN
`transformar_mensaje( char *str, unsigned char vect[])`	Realiza la conversión de códigos ASCII del *string* de entrada `str` en índices para el vector `nbcode[]`, que se guardan sobre el vector `vect[]`; devuelve el número de letras transformadas, excluyendo el centinela del *string* (el resultado es de tipo `unsigned char`)
`swiWaitForVBlank()`	Espera hasta el próximo retroceso vertical

El programa a realizar debe permitir entrelazar la ejecución de la rutina `leer_mensaje()` con la retransmisión del mensaje anterior. Para conseguir esta concurrencia de tareas, la retransmisión de los *bitcode* se realizará mediante la RSI del *timer* 0. Hay que tener en cuenta que el usuario podría introducir un nuevo mensaje antes de que la RSI haya terminado de retransmitir el mensaje anterior. En este caso, será necesario esperar al final de dicha retransmisión.

Se propone el siguiente programa principal:

```
#define FREQ_ENT0   33513982          // posibles frecuencias de entrada
#define FREQ_ENT1   523656            // de un timer (en Hz)
#define FREQ_ENT2   130914
#define FREQ_ENT3   32728
#define MAX_LON 50                    // longitud máxima del mensaje

unsigned int nbcode[59];              // vector códigos (ya inicializado)
char inputstr[MAX_LON+1];             // mensaje introducido por usuario
unsigned char transmsg[MAX_LON];      // mensaje transformado
unsigned char lontr = 0;              // longitud mensaje transformado
unsigned char curr_ind = 0,           // índice de letra actual
unsigned char curr_bit;               // número de bit actual
```

```
int main()
{
  inicializaciones();
  do
  {
     leer_mensaje(inputstr, MAX_LON);
     while (curr_ind < lontr)   // esperar el final de la reproducción
     {                          // del mensaje anterior (si la hay)
          swiWaitForVBlank();
     }
     lontr = transformar_mensaje(inputstr, transmsg);
     if (lontr > 0)
     {
          curr_ind = 0;
          curr_bit = nbcode[transmsg[0]] >> 24;
          activar_timer(0, ¿_a_?, ¿_b_?);
     }
  } while (1);
  return(0);
}
```

Después de leer mensaje, espera el final de una posible retransmisión anterior, transforma el mensaje y comprueba que haya alguna letra a retransmitir. En caso afirmativo, el cuerpo del `if` inicializa las variables globales `curr_ind` y `curr_bit` para establecer el índice de la letra a retransmitir (inicialmente cero) y el número de bit actual, inicializado con el número de bits útiles del *bitcode* de la primera letra del mensaje, que se extrae de los 8 bits de más peso del valor almacenado en `nbcode[transmsg[0]]`.

A continuación, se activa el *timer* 0. Se han dejado unas marcas `¿_a_?` y `¿_b_?` para indicar el número de frecuencia de entrada (0, 1, 2 o 3) y el cálculo del divisor de frecuencia adecuado, que se piden como parte de la solución. Teniendo en cuenta que la RSI deberá transmitir un bit del *bitcode* a cada interrupción, hay que determinar los valores de `¿_a_?` y `¿_b_?` para que la retransmisión de la letra `'S'`, cuyo código Morse es `'...'` (3 puntos), junto con el espacio entre letras, se realice en un segundo.

En general, la RSI del *timer* 0 se debe encargar de las siguientes tareas:

- decrementar `curr_bit`,
- obtener el *bitcode* de la letra que se está retransmitiendo actualmente (según `curr_ind`),
- obtener el valor del bit actual del *bitcode* que se está retransmitiendo actualmente, y fijar el bit 5 de REG_DATA a ese valor (sin modificar el resto de bits de REG_DATA),
- si `curr_bit` ha llegado a `0` , pasar a la siguiente letra y reinicializar `curr_bit`,
- si ya se ha llegado al final del mensaje, detener el *timer* 0.

Se pide:

RSI del *timer* 0 en ensamblador, expresión de los parámetros `_a_` y `_b_` en lenguaje C.