NOTA. Para realizar los ejercicios de laboratorio de E/S podéis utilizar las siguientes funciones y definiciones:

`void DisableInts();` Inhibir interrupciones: escribe un 0 en el flag IF (Interrupt Flag)

`void EnableInts();` Permitir interrupciones: escribe un 1 en el flag IF (Interrupt Flag)

`unsigned short Cs(name);` Devuelve la dirección base del segmento correspondiente a la rutina de servicio "name"

`unsigned short Ip(name);` Devuelve el desplazamiento dentro del segmento correspondiente a la rutina de servicio "name"

`unsigned char InPort(puerto);` Devuelve el contenido del registro cuya dirección en el espacio de entrada/salida es "puerto"

`void OutPort(puerto, valor);` Escribe "valor" en el registro cuya dirección en el espacio de entrada/salida es "puerto"

`unsigned char LeeByteFis(unsigned short Seg, unsigned short Desp);` Devuelve el contenido (1 byte) de la posición de memoria indicada por la dirección base "Seg" y desplazamiento "Desp"

`void EscribeByteFis (unsigned short Seg, unsigned short Desp, unsigned char c);` Escribe el valor "c" (1 byte) en la posición de memoria indicada por la dirección base "Seg" y desplazamiento "Desp"

`unsigned char LeePalFis(unsigned short Seg, unsigned short Desp);` Devuelve el contenido (2 bytes) de la posición de memoria indicada por la dirección base "Seg" y desplazamiento "Desp"

`void EscribePalFis (unsigned short Seg, unsigned short Desp, unsigned short p);` Escribe el valor "p" (2 bytes) en la posición de memoria indicada por la dirección base "Seg" y desplazamiento "Desp".

`void Eoi();` *End Of Interruption*. Indica que ha finalizado el tratamiento de la interrupción: cuando el controlador de interrupciones recibe esta señal, examina el registro ISR para desactivar el bit de más peso entre aquellos que estén activados. De esta forma, se pueden servir interrupciones de menor prioridad a la que acaba de finalizar

`#define DIR_PANT 0xB000` Dirección base de pantalla.

`#define ATRIB_NORMAL 0x07` Atributo normal: carácter blanco sobre fondo de pantalla negro.

`#define REG_IRR_CI8259 0x20` Dirección del registro IRR del controlador de interrupciones del PC en el espacio de E/S.

`#define REG_IMR_CI8259 0x21` Dirección del registro IMR del controlador de interrupciones del PC en el espacio de E/S.

`#define REG_DAT_TECLADO 0x60` Dirección del registro de datos del controlador del teclado del PC dentro del espacio de E/S.

`#define REG_CONTROL_TECLADO 0x61` Dirección del registro de control del controlador del teclado del PC dentro del espacio de E/S.

#define ASCII_RETURN 13 Código ASCII del carácter RETURN.

#define VAL_EOI 0x20 Valor a escribir en la dirección 0x20 para llevar a cabo la operación EOI

Igualmente supondremos también definido el vector TABLA_ASCII[], en el que podemos obtener el código ASCII correspondiente a la tecla pulsada. El índice de este vector es el código MAKE de la tecla pulsada, no el código BREAK, es decir, en el vector podremos obtener el código ASCII de una tecla cuando se pulsa la tecla, nunca al soltarla

Mediante estos ejercicios queremos estudiar, desde un punto de vista práctico, el funcionamiento de algunos periféricos del PC: pantalla, controlador de interrupciones, teclado, reloj, etc.

Ejercicio 1.
Desarrollar una aplicación que rellene la pantalla de forma continua con el carácter '*' hasta completar la pantalla. En este momento, la aplicación borrará la pantalla y comenzará nuevamente a rellenar la pantalla con asteriscos. Mientras el programa principal está en ejecución, también atenderá las interrupciones del teclado. Al pulsar una tecla, aparecerá en la esquina superior derecha de la pantalla el carácter tecleado con un atributo que permita distinguirlo de los asteriscos del resto de la pantalla. El programa finalizará al pulsar la tecla 'Q'.

Para realizar este ejercicio, además del programa principal, hay que programar las siguientes funciones:

```
void EscribeCar (int fila, int col, unsigned char car, unsigned char atrib)
      //Escribe el carácter "car" en la pantalla, en la fila y columna indicadas
      //como parámetros
void BorraPantalla ()
      //Borra toda la pantalla
void LeerCarPantalla (int fila, int col, unsigned char *car, unsigned char *atrib)
      //lee el carácter (y atributo) de la pantalla en la fila y columna indicadas
void CambiaVI (int entrada, unsigned short IPNueva, unsigned short CSNueva,
               unsigned short *IPAnt, unsigned short *CSAnt)
      //Cambia el vector de interrupción con la dirección de nuestra rutina
      //atención. Previamente guarda el contenido de esa entrada en el VI
void RecuperaVI (int entrada, unsigned short IPAnt, unsigned short CSAnt)
      //Recupera el valor original de la entrada del vector de interrupción
void Eoi ()
      //Eoi (End of Interruption)
unsigned char LeerRegDatosTeclado ()
      //Devuelve el contenido del registro de datos del controlador del teclado
void StrobeTeclado ()
      //Secuencia de strobe sobre el bit 7 del registro de control del teclado
void interrupt RutAtencionTec ()
      // Rutina de atención al teclado
```

Ejercicio 2.
Desarrollar una aplicación que lea un string desde el teclado por encuesta. La entrada del teclado finalizará cuando se tecleen 80 caracteres o el carácter de fin de línea (tecla RETURN). Para realizar este ejercicio se pueden utilizar las funciones ya programadas en el ejercicio 1. En este ejercicio, hay que programar el programa principal y las siguientes funciones:

```
unsigned char LeerIRR ()
      // Devuelve el contenido del registro IRR
void InhibirIntTeclado ()
      // Inhibir las interrupciones del teclado
      // Inhibir y desinhibir interrupciones
void DesinhibirIntTeclado ()
      // Permitir las interrupciones del teclado
      // Inhibir y desinhibir interrupciones
unsigned char LeerTecladoEncuesta ()
      // Devuelve el código ASCII de la tecla pulsada
            // Encuesta del teclado
            // Tratamiento de MAKE y código ASCII
            // Strobe del teclado
```

Ejercicio 3.
Desarrollar una aplicación que se ejecutará durante 10 minutos. En este intervalo de tiempo queremos emular un sencillo salvapantallas (se supone que la pantalla tiene algún texto escrito). Si no se pulsa una tecla durante 30 segundos, se almacenará en una variable auxiliar el contenido de la pantalla y se borrará la pantalla. La pantalla se recuperará nuevamente al pulsar cualquier tecla.

Para realizar este ejercicio se pueden utilizar las funciones ya programadas en el ejercicio 1. En este ejercicio, hay que programar el programa principal y las siguientes funciones:

```
void interrupt RutAtencionReloj ()
     // Rutina de atención al reloj
void interrupt RutAtencionTeclado ()
     // Rutina de atención al teclado
```

Ejercicio 4.
Modifica el ejercicio 3 para los dos supuestos siguientes:

4.1.- La pantalla sólo se recupera al pulsar la tecla 'A'.

4.2.- La pantalla se recupera al soltar cualquier tecla, después de pulsarla.

Ejercicio 5.
Queremos conectar un nuevo periférico a nuestro PC. Este periférico, que interrumpe por la línea IRQ6, dispone de los siguientes registros:

`Registro de datos: @0x279 (8 bits)  Registro de control: @0x281 (8 bits)`

Después de realizar cualquier operación sobre este dispositivo, se debe hacer una secuencia de STROBE en el bit 5 del registro de control (los bits se numeran de 0 a7).

Programa en lenguaje C las siguientes rutinas: **(a)** la rutina de `strobe`, **(b)** la rutina que cambia el vector de interrupciones, y **(c)** la llamada a la rutina que cambia el vector de interrupciones para atender las peticiones de este periférico.

Ejercicio 6.
Queremos programar el siguiente juego basado en la arquitectura del PC. Tenemos un pulsador que interrumpe por la línea IRQ4 cada vez que un jugador lo pulse. Este pulsador dispone de un registro de control en la dirección 0x281, sobre el que hay que hacer una secuencia de *strobe* en el bit 3 cada vez que se trata su interrupción. En este ejercicio, queremos que el pulsador funcione por encuesta. El funcionamiento del juego es muy sencillo: queremos ver quién pulsa más veces ese pulsador en 10 minutos.

Al iniciarse el programa sacaremos un mensaje indicando al jugador que comienza la cuenta, a fin de que empiece a pulsar. El programa finalizará indicando mediante otro mensaje cuántas veces ha pulsado el jugador ese pulsador. Para sacar los mensajes, suponed ya implementadas las funciones `EscribeTexto(texto)` y `EscribeNum(num)`.

De acuerdo al funcionamiento del periférico, implementa en C las siguientes funciones:

- **a)** El programa principal. Indica qué variables globales son necesarias para el correcto funcionamiento del programa.
- **b)** La rutina de atención a la interrupción de reloj: `void interrupt Reloj()`
- **c)** La rutina de strobe necesaria: `void Strobe()`
- **d)** La función que inhibe las interrupciones para ese periférico: `void InhibirIntPulsador()`

Ejercicio 7.

Queremos programar un juego de dardos basado en la arquitectura del PC. Cada vez que un dardo impacte en la diana, el dispositivo interrumpe por la línea IRQ3 e indica en su registro de datos (dirección 0x281) la puntuación obtenida. Vamos a suponer que todos los dardos impactan en la diana.

El funcionamiento del juego es muy sencillo: queremos saber qué jugador obtiene la mayor puntuación en el lanzamiento de 6 dardos, teniendo en cuenta además el tiempo invertido. El tiempo empezará a contarse desde el impacto del primer dardo. Al iniciarse el programa sacaremos un mensaje indicando al jugador el comienzo del juego, a fin de que empiece a lanzar los dardos. El programa finalizará indicando mediante otro mensaje la puntuación obtenida y el tiempo empleado (en segundos). Para sacar los mensajes, suponed ya implementadas las funciones `EscribeTexto(texto)` y `EscribeNum(num)`.

De acuerdo al funcionamiento del periférico, implementa en C las siguientes funciones:

- **a)** El programa principal. Indica qué variables globales son necesarias para el correcto funcionamiento del programa.
- **b)** Las rutinas de atención que consideres oportunas.
- **c)** La función que cambia el vector de interrupciones para atender a un dispositivo:

Ejercicio 8.

Queremos conectar a nuestro PC un periférico que utiliza la línea IRQ5. El controlador de este periférico tiene 2 registros de 8 bits cuyas direcciones de E/S (independientes de memoria) son:

```
Registro de datos: @0x279        Registro de Control: @0x281
```

El funcionamiento del sistema es el siguiente. Cada vez que este periférico interrumpe hay que leer el contenido del registro de datos y, a continuación, hacer una secuencia de `strobe` sobre el bit 3 del registro de control. Si el bit 4 del registro de datos vale 1, sacará el carácter '1' por la pantalla. Si, por el contrario, ese bit vale 0, sacará el carácter '0' por la pantalla y acabará la ejecución del programa. La sincronización con este periférico se realiza por encuesta.

Para escribir en la pantalla utilizaremos la función `EscribeCar`. La pantalla está mapeada en memoria, tiene 60 filas y 132 columnas, y los caracteres se escriben sin atributo. La codificación de cada carácter a imprimir en la pantalla se realiza en 2 bytes.

De acuerdo al funcionamiento del periférico, implementa en C las siguientes funciones:

- **e)** La función que realiza la encuesta: *`unsigned char Encuesta().`* Esta función devuelve el contenido del registro de datos del controlador.
- **f)** El programa principal. Entre otras cosas, este programa llamará a la función `Encuesta()` para realizar la encuesta del periférico y cogerá el valor devuelto por la función. El programa realizará el tratamiento indicado para ese valor mientras no se produzca la condición de fin de programa (el bit 4 del registro de datos valga 0).
- **g)** La función que inhibe las interrupciones para ese periférico: `void Inhibir_Int_KPer()`
- **h)** La función de *strobe:* *`void Strobe()`*
- **i)** La función que escribe un carácter en la pantalla:

```
void EscribeCar (int fila, int col, unsigned short car)
```

Ejercicio 9.

Queremos conectar un nuevo periférico a nuestro PC. Este dispositivo, que interrumpe por la línea IRQ3, dispone de un registro de datos y de un registro de control, los dos de 8 bits y no mapeados en memoria. Las direcciones de estos registros son las siguientes:

```
Registros de datos: 0x279       Registro de control: 0x281
```

Cada vez que interrumpe, este dispositivo deja dos datos de 4 bits en el registro de datos, uno en los 4 bits de menos peso (Num1) y el otro en los 4 bits de más peso (Num2). Hay que obtener estos dos datos y guardarlos en las variables globales Num1 y Num2. Además hay que sacar su valor por pantalla: la variable Num1 en la fila 3 y en la columna 5, y la variable Num2 en la fila 4 y columna 5. Las características de la pantalla, que está mapeada en memoria a partir de la dirección DIRPANT, son las siguientes: consta de 30 filas y 60 columnas, los caracteres ocupan un byte y se escriben sin atributo. Cada vez que se lee el registro de datos es necesario realizar una secuencia de *strobe* en el bit 6 del registro de control (los bits se numeran de 0 a 7).

Se pide implementar las siguientes rutinas: a) la rutina de atención a la interrupción que implemente el tratamiento descrito `void interrupt RutPer ()`, b) la rutina de strobe, `void Strobe()`, c) la rutina que inhibe las interrupciones de este dispositivo `void Inhibir_KPer()`.

Ejercicio 10.

Queremos controlar un sistema que nos diga si los asientos de un cine están ocupados o no. Los asientos forman una matriz de 256 filas y 256 columnas. El sistema funciona sobre la base de un controlador *Kasiento* con un registro de datos de 16 bits y un registro de control de 8 bits. Cuando alguien se sienta o se levanta de un asiento, este controlador genera una interrupción por la línea IRQ7 y codifica en el registro de datos el asiento correspondiente: de los 16 bits del registro, los 8 bits de más peso indican la fila y los 8 bits de menos peso la columna. Además, si alguien se levanta, el bit 3 (contado desde el bit 0) del registro de control estará a 0 y, si alguien se sienta, este bit estará a 1.

Por otra parte, cuando alguien se levante o se siente en un asiento, el sistema debe actualizar en memoria la situación de ese asiento. La matriz de asientos está mapeada en memoria a partir de la dirección MEMDIR. Los elementos de esta matriz son de 1 byte. El valor 1 en una posición indica que ese asiento está ocupado, mientras que el valor 0 indica que está libre.

Aunque la sincronización con el controlador *Kasiento* puede ser por interrupción, se ha decidido hacerla por encuesta. Programa en lenguaje C las siguientes rutinas: (a) la función que inhibe las interrupciones del periférico (`void Inhibir_Kasiento()`); (b) la función que realiza la encuesta al periférico (`void Encuesta_Kasiento()`) y almacena en las variables globales *ASIENTO* y *ESTADO* el código y el estado del asiento (libre/ocupado); y (c) el programa principal que, entre otras cosas, realiza la encuesta y actualiza la información del asiento en la matriz de memoria.

Ejercicio 11.

En una empresa han instalado un sistema de detección automática de incendios. Para ello han utilizado un PC compatible con el Intel 8086. Para detectar fuego se utiliza un sensor cuyo controlador es KFU. Este sensor interrumpe por la línea IRQ3 cuando detecta un incendio, dejando en su registro de datos información acerca del fuego detectado (en concreto, esta información está codificada en un carácter). A pesar de que la sincronización de este dispositivo puede ser por interrupción, lo vamos a controlar por encuesta. Como sabes, para ello es necesario inhibir las interrupciones de este controlador. Te pedimos que programes en lenguaje C la rutina que inhibe las interrupciones de este controlador: `void InhibirIntKFU();`

Una vez inhibidas las interrupciones, programa en C la rutina que realiza la encuesta de este periférico. Esta rutina deberá devolver el código almacenado en el registro de datos del controlador KFU. La gestión de este periférico requiere además una secuencia de *strobe*. Programa la rutina `unsigned char EncuestaKFU ();`

Hay que sacar por la pantalla la información referente al incendio leída en la rutina anterior. La pantalla está mapeada en memoria a partir de la dirección C000h y es una pantalla de 48 filas y 160 columnas. Programa en C la siguiente rutina que escribe un carácter en la fila y columna especificadas como parámetros.

```
void EscribeCar(int fila, int columna, unsigned char car, unsigned char atrib);
```

Ejercicio 12.

Queremos conectar un nuevo periférico a nuestro PC. Este dispositivo, que interrumpe por la línea IRQ5, dispone de dos registros de datos y de un registro de control, todos ellos de 8 bits no mapeados en memoria. Las direcciones de estos registros son las siguientes:

```
Registros de datos, DAT1 y DAT2: @0x479 y @0x480
Registro de control: @0x481
```

Tras realizar cualquier operación con este dispositivo, se requiere una secuencia de STROBE sobre el bit 2 del registro de control (los bits de los registros se numeran del 0 al 7).

Programa en lenguaje C el siguiente código: (a) la rutina de STROBE (`void Strobe()`); (b) la rutina que cambia el vector de interrupción para atender a este periférico; (c) escribe la llamada que hace el programa principal para cambiar el vector de interrupción para ese periférico; y (d) la rutina de servicio a la interrupción de este periférico (`void interrupt RutPer()`) que almacena en la variable global RESULTADO un número de 16 bits: el contenido de DAT1 en los 8 bits de menos peso y el contenido de DAT2 en los 8 bits de más peso.