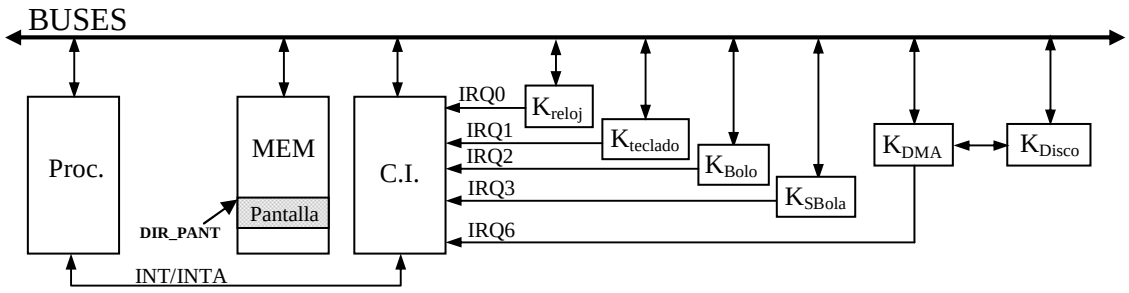
Universidad del País Vasco
Facultad de Informática

Departamento de Arquitectura y Tecnología de Computadores

Arquitectura de Computadores I

Subsistema de entrada/salida 2 (solución): Bolera

Queremos diseñar un sistema que controle las partidas de una bolera. Para ello disponemos de un sensor que detecta si la bola ha sido lanzada y un dispositivo capaz de contar y levantar los bolos derribados. La estructura hardware es la siguiente:



K_Bolo: Es el controlador del dispositivo utilizado para contar y levantar los bolos derribados. Tiene dos registros: un registro de control (R_CONT_KBOLO) y un registro de datos (R_DAT_KBOLO). Para poner el dispositivo en marcha debemos realizar una secuencia de *strobe* en el registro de control, tras lo cual el dispositivo levantará todos los bolos que hayan sido derribados. Una vez finalizado su trabajo, el dispositivo genera una petición de interrupción. En ese momento podremos leer en el registro de datos el número de bolos que ha tenido que levantar.

K_SBola: Es el controlador del sensor que detecta que la bola ha sido lanzada. La bola pasa junto a éste cuando está a punto de llegar a los bolos. En ese momento el controlador genera una petición de interrupción.

K_DMA: Es el controlador de DMA y se utiliza para hacer transferencias rápidas de memoria a disco. Posee los siguientes registros:

- **registro de control** (R_CONT_KDMA): cuando se escribe un 1 en él se inicia la transferencia (se pone a 0 automáticamente).
- **registro de dirección** (R_DIR_KDMA): dirección de comienzo del bloque a transferir.
- **registro de longitud** (R_LON_KDMA): contiene la longitud en bytes del bloque a transferir.
- **registro de estado** (R_EST_KDMA): al finalizar la transferencia, el contenido de este registro indica si la última operación realizada ha sido errónea (0) o si ha finalizado correctamente (1).

K_Disco: Es el controlador de disco. Para realizar la transferencia hay que inicializar este controlador. Supondremos que existe una rutina llamada *ProgramarKDisco()* que se encargará de la inicialización.

Los controladores de reloj, teclado e interrupciones son los mismos que los estudiados en la asignatura. En este sistema la sincronización del **teclado** se realiza **por encuesta**. La pantalla está mapeada en memoria a partir de la dirección DIR_PANT.

El **funcionamiento** del sistema es el siguiente. Mientras el encargado de la bolera no lo permita, no se podrá jugar; es por ello por lo que antes de la partida habrá una barrera delante de los bolos. Una vez el jugador haya abonado la partida, el encargado pulsará la tecla 'H' del teclado. De esta forma la barrera se levantará y podrá comenzar la partida. Para levantar la barrera podemos utilizar la rutina ya escrita *levantar_barrera()*.

Cada partida constará de 10 lanzamientos y después de cada uno de ellos, el dispositivo K_Bolo deberá recoger y contar los bolos que han sido derribados.

En cada lanzamiento, el sensor SBola detectará el momento en el que la bola se acerca a la posición de los bolos y generará una interrupción. Una vez detectada la bola, habrá que esperar 10 segundos a que los bolos caídos se estabilicen en el suelo y una vez transcurrido ese tiempo se deberá activar el dispositivo K_Bolo para que levante los bolos derribados.

Después de cada lanzamiento se deberá mostrar en pantalla la suma de todos los bolos derribados hasta ese momento. Para ello disponemos de la rutina ya escrita *mostrar_derribados(Num_Bolos, Num_Lanzamiento)*, que recibe dos parámetros: el número de bolos derribados hasta el momento, y el numero de lanzamientos realizados. De esta forma, todas las puntuaciones irán mostrándose en pantalla. Cada puntuación ocupará 4 bytes en memoria.

Una vez que se hayan realizado 10 lanzamientos la partida terminará, pero antes habrá que guardar todas las puntuaciones en disco. Para esta transferencia de memoria a disco se utilizará el DMA. Si la transferencia no se realiza correctamente, el sistema deberá intentarlo de nuevo. Si en 3 intentos no se consigue realizar correctamente la transferencia, se deberá dar por finalizada la partida dando cuenta al operario del fallo en la transferencia de datos mediante la rutina *error_de_transferencia()*.

Para que las partidas no se prolonguen demasiado, el jugador tiene un límite de tiempo para realizar cada lanzamiento. Si desde que se han levantado los bolos, pasan 5 minutos sin que la bola sea lanzada, la partida se terminará. En este caso la puntuación no será transferida al disco.

En cualquier caso, una vez finalizada la partida, la barrera deberá ser bajada otra vez, hasta que el encargado de la bolera vuelva a pulsar la tecla 'H'. Para ello utilizaremos la rutina *bajar_barrera()*.

Se pide: Escribe en lenguaje algorítmico todas las rutinas de atención que consideres necesarias, así como el programa principal. Comenta cualquier supuesto que hagas para la resolución del problema. Se valorará representar el comportamiento del sistema mediante un autómata de estados y transiciones.

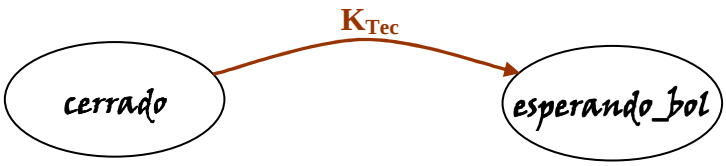
Solución

En primer lugar, intentaremos encontrar un autómata que refleje el comportamiento del sistema. Para ello es fundamental distinguir bien todas las situaciones o estados posibles del funcionamiento del sistema, así como identificar correctamente cuáles son las condiciones que se han de cumplir para que se produzcan las transiciones de unos estados a otros. En general, las transiciones entre estados serán consecuencia de sucesos o eventos externos al propio sistema, los cuales serán detectados por medio de los periféricos presentes en el sistema, sea cual sea el modo en que dichos periféricos se sincronicen con el procesador, tanto si es por encuesta como por interrupción.
En este ejercicio no se nos dice cuáles son los estados posibles del sistema, por lo que es necesario leer atentamente la información proporcionada acerca del funcionamiento del sistema, para poder deducir correctamente los estados del mismo. Veamos, pues, cuáles son las circunstancias y sucesos o eventos que pueden acontecer en este sistema y cuáles son las consecuencias que de ellos se derivan según el estado del sistema. Para ello, releeremos el enunciado atentamente.
Para empezar, según se nos indica, mientras no le abonemos la partida al encargado de la bolera, no nos dará permiso para poder jugar y habrá una barrera delante de los bolos para impedir que juguemos. Ese es, precisamente, el estado inicial del sistema, que denominaremos *cerrado*.

1. Estado *cerrado*:

- Características del estado:
  Como acabamos de decir, para evitar las partidas que no se hayan pagado, hay una barrera delante de los bolos. El sistema estará en ese estado hasta que el encargado pulse la tecla H. Es decir, es el teclado el periférico que genera el suceso o evento que propicia el cambio de estado (aunque la sincronización con el teclado se realiza por encuesta, este hecho no tiene relevancia en el autómata, ya que en él no distinguimos entre interrupciones o encuestas, sino únicamente eventos, y está claro que pulsar una tecla es un evento concreto). Pero únicamente se debe tomar en cuenta la tecla H, ninguna otra, y sólo el evento asociado al momento en que pulsamos la tecla, no cuando la liberamos —recordemos que el controlador del teclado distingue dos eventos para cada tecla: uno cuando se pulsa la tecla (MAKE), y otro cuando se libera la tecla (BREAK)—. En consecuencia, la posible transición de este estado al siguiente es condicional: únicamente cuando se pulsa la tecla H.
- Transiciones posibles:

a) Cuando el encargado de la bolera pulse la tecla H, empezará la partida, es decir, el jugador o jugadora podrá lanzar la bola por la pista, para lo que podrá tomarse el tiempo que necesite —aunque como máximo dispondrá de 5 minutos, tal como se indica en el enunciado, para no alargar la partida en exceso—. Por ello, a este segundo estado lo denominaremos *esperando_bola*.
En el autómata lo reflejaremos así: la flecha entre los dos estados indica el sentido de la transición —del estado *cerrado* al estado *esperando_bola*—, y la etiqueta que aparece sobre la flecha, por su parte, indica el periférico, suceso o evento que da lugar a dicha transición de estado.



Como queremos sicronizar el teclado por encuesta, los eventos a él asociados deberán ser tratados en el programa principal, no en una rutina de servicio a una interrupción, como se hace cuando la sicronización es por interrupciones. Así pues, el programa principal deberá ser capaz de detectar cuándo se pulsa o libera una tecla. Con periféricos corrientes, para realizar la encuesta el procesador lee el registro de estado del controlador asociado al periférico, hasta detectar el evento deseado. Pero en los ordenadores compatibles PC el controlador del teclado es un poco especial, ya que no dispone de registro de estado. En consecuencia, para poder realizar la encuesta del teclado el procesador debe leer el registro IRR del controlador de interrupciones, para detectar la interrupción que genera el controlador del teclado cuando se pulsa o libera una tecla, ya que la petición de interrupción se va a producir siempre, aunque queramos sincronizarlo por encuesta; por esta razón, cuando queremos sincronizarlo por encuesta es necesario inhibir las interrupciones del teclado (poniendo a 1 el bit 1 del registro máscara IMR del controlador de interrupciones), de manera que se impide que la petición de interrupción llegue al procesador. Así, el programa principal deberá leer el registro IRR y analizar el bit 1 del mismo: cuando ese bit toma el valor 1, eso indica que se ha pulsado/liberado una tecla, y entonces el programa principal deberá realizar las acciones que indicamos a continuación, para ejecutar lo indicado en el autómata.

- Acciones a realizar en la sección del programa encargada del tratamiento del teclado:

Además de la transición de estado que hemos indicado en el autómata, la sección del programa encargada del tratamiento de los eventos del teclado tiene que realizar también otras acciones (quede claro que el código correspondiente que veremos ahora se ejecutará después de haberse detectado en la encuesta que se ha pulsado o liberado una tecla; no escribiremos aquí la encuesta, porque es genérica y no depende del estado del autómata; al escribir el código del programa principal comentaremos en detalle cómo realizar la encuesta):

Tec.1) Para saber qué tecla ha sido pulsada (o liberada), se debe leer el registro de datos del controlador del teclado (`R_DAT_Ktec`). Se debe tener en cuenta que el valor leído en el registro de datos corresponde al código de posición o "*scan code*" de la tecla, no al carácter alfanumérico que la tecla tiene asignado en el teclado. Por esa razón, normalmente se suele traducir dicho código de posición, para saber qué carácter concreto le corresponde a la tecla pulsada (o liberada):

```
código_tecla = InPort(R_DAT_Ktec);
```

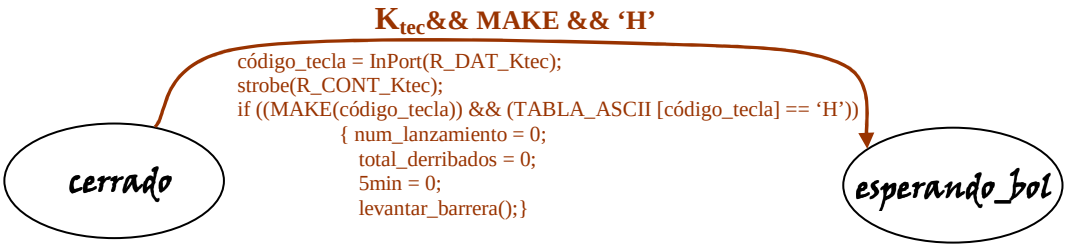
Tec.2) Es necesario realizar una secuencia de `strobe` sobre el registro de control (`R_CONT_Ktec`) del controlador del teclado:

```
strobe(R_CONT_Ktec);
```

Tec.3) Ahora se debe analizar si la tecla ha sido pulsada (MAKE) o liberada (BREAK) —supondremos que disponemos para ello de una función ya escrita: `MAKE(código_tecla)`, que devuelve un 1 si se trata de MAKE, o un 0 si se trata de BREAK—, y además debemos analizar si se trata de la tecla H —para ello, traduciremos el código de posición de la tecla al código ASCII que le corresponde, leyendo una tabla que tenemos almacenada en memoria, empleando el código de posición de la tecla como índice dentro de la tabla: `TABLA_ASCII [código_tecla]`—, y todo eso hay que hacerlo únicamente en el estado ***cerrado***, ya que en cualquier otro estado del sistema no hay que hacerle caso al teclado. Así pues, cuando se cumplen esas tres condiciones, además de realizar el cambio de estado, se debe levantar la barrera que está delante de los bolos, para lo que disponemos de una función específica ya escrita denominada `levantar_barrera()`. Además, como se trata del comienzo de la partida, habrá que inicializar algunas variables globales para poder controlar adecuadamente la partida. En concreto, estas variables: a) en cada partida se pueden realizar como máximo 10 lanzamientos, por lo que necesitaremos una variable que vaya contando cuántos lanzamientos se han realizado; la denominaremos `num_lanzamiento` y le asignaremos el valor 0 al principio de la partida; b) en cada lanzamiento se debe calcular y visualizar en pantalla la puntuación obtenida, para lo que se toma en cuenta el número total de bolos derribados hasta ese lanzamiento; así pues, deberemos disponer de otra variable global para ese menester: `total_derribados`, que inicializaremos con el valor 0; c) finalmente, sabemos que el jugador o jugadora dispone de 5 minutos como máximo para lanzar la bola en cada jugada y que, si no la lanza en ese intervalo de tiempo, la partida se dará por finalizada; por esa razón, se necesita una variable global que vaya controlando el transcurso del tiempo durante esos 5 minutos; la denominaremos `5min` y la inicializaremos con el valor 0 al principio de la partida. En consecuencia, el código correspondiente queda como sigue:

```
if ((MAKE(código_tecla))&&(TABLA_ASCII[código_tecla]== 'H')
&&(estado_automata == cerrado))
                {
                    num_lanzamiento = 0;
                    total_derribados = 0;
                    5min = 0;
                    levantar_barrera();
                    estado_automata = esperando_bola;
                }
```

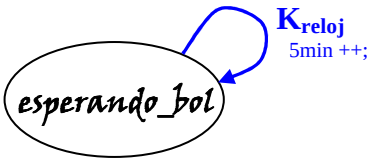
Es conveniente indicar todas esas acciones sobre el autómata, ya que ello resultará de gran ayuda a la hora de escribir el código del programa. Por tanto, bajo la flecha que indica la transición de estado, escribiremos todas esas acciones, así:



b) En principio, no hay más transiciones posibles que hagan que el sistema salga del estado *cerrado*.

2. Estado *esperando_bola*:

- Características del estado:
  Como el propio nombre indica, en este estado el sistema permanece a la espera de que el jugador o jugadora lance la bola. El sensor SBola es el encargado de detectar que la bola ha sido lanzada por la pista hacia los bolos, y cuando la detecte generará una petición de interrupción para indicárselo al procesador; como consecuencia de ello, el sistema pasará a otro estado. Pero mientras el sistema permanece en el estado *esperando_bola*, el jugador o jugadora dispone como máximo de 5 minutos para lanzar la bola, de manera que si transcurre ese tiempo sin que el sensor haya detectado la bola, el sistema volverá al estado inicial y dará por finalizada la partida. Así pues, en este estado es preciso controlar adecuadamente la variable `5min`, para saber si efectivamente pasan 5 minutos en este estado sin lanzar la bola.
  En consecuencia, se pueden dar varias posibilidades:
  - Por una parte, mientras el sistema está en el estado *esperando_bola*, se debe controlar la variable `5min`, actualizando su valor en cada una de las interrupciones del reloj, pero eso no tendrá influencia en el estado del sistema, es decir, el sistema se mantendrá en ese estado aunque el jugador o jugadora no lance la bola, mientras no transcurran los 5 minutos permitidos. Por tanto, en el autómata plasmaremos así esta posibilidad:



↳ Acciones a realizar en la rutina de servicio a la interrupción del reloj:

Reloj.1) Actualización de la variable `5min` cada vez que interrumpe el reloj:

```
if (estado_automata == esperando_bola)
   5min++;
```

De esta manera, sabremos que han transcurrido 5 minutos en este estado cuando la variable `5min` alcance el valor 5 × 60 × 18, ya que dicha variable cuenta el número de interrupciones del reloj (5 minutos × 60 segundos/minuto × 18 interrupciones/segundo).

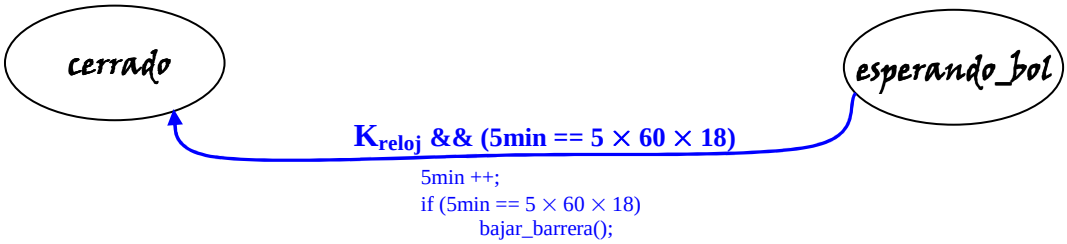
- Así pues, si mientras el sistema está en el estado *esperando_bola* el jugador o jugadora no lanza la bola antes de que transcurran 5 minutos, entonces el sistema volverá al estado *cerrado*, y quedará a la espera de la siguiente partida. Como se trata de una transición de estado, lo veremos en el subapartado siguiente.
- Finalmente, si el jugador o jugadora lanza la bola antes de transcurridos los 5 minutos, la bola será detectada por el sesor SBola, que generará una petición de interrupción para indicárselo al procesador; como consecuencia de ello, el sistema pasará a un nuevo estado. Como también en este caso se trata de una transición de estado, lo veremos en el subapartado siguiente.

- Transiciones posibles:

a) Si cuando interrumpe el reloj, una vez actualizada la variable `5min`, el nuevo valor resulta ser igual a $5 \times 60 \times 18$, entonces el sistema deberá volver al estado *cerrado*, ya que el llegar a ese valor supone que el jugador o jugadora no ha lanzado la bola. Además, se debe volver a bajar la barrera que hay delante de los bolos, lo que haremos mediante la función ya escrita `bajar_barrera()`. Indicaremos todo eso en el autómata mediante una flecha entre esos estados, y en la rutina de atención a la interrupción del reloj como sigue:

Reloj.2) Transición de estado, si es necesario:

```
if (5min == 5 × 60 × 18)
   {
        bajar_barrera();
        estado_automata = cerrado;
   }
```



b) Pero si el jugador o jugadora lanza la bola antes de que transcurran los 5 minutos, entonces el sensor SBola detectará la bola y generará una petición de interrupción, como consecuencia de la cual el sistema pasará a un nuevo estado, en el que esperará durante 10 segundos, para dar tiempo a que los bolos golpeados por la bola se estabilicen. Está claro, por tanto, que es la petición de interrupción realizada por el controlador K_SBola el evento que produce esa transición de estado. El nuevo estado lo denominaremos *estabilizando*, dado que eso es lo que está esperando el sistema.

Estas son las acciones que hay que realizar en esta transición de estado:

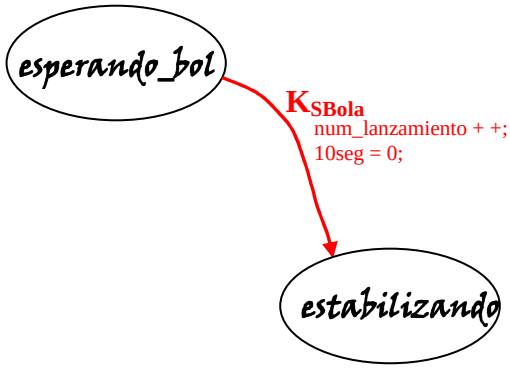
↳ Acciones a realizar en la rutina de servicio a la interrupción del sensor SBola:

SBola.1) Por una parte, el hecho de detectar la bola significa que el jugador o jugadora ha realizado un lanzamiento más, por lo que habrá que actualizar la variable `num_lanzamiento`:

```
num_lanzamiento ++;
```

SBola.2) Por otra parte, el sistema deberá permanecer en el nuevo estado durante 10 segundos, esperando a que se estabilicen los bolos. Por ello, para controlar ese intervalo de tiempo necesitamos una variable global, que denominaremos `10seg`, y que inicializaremos con el valor 0 en esta transición de estado, pues es en ese preciso instante cuando se deben empezar a contar los 10 segundos, no antes:

```
10seg = 0;
```



Pero las acciones anteriores únicamente deben realizarse si el sistema detecta la bola mientras está en el estado *esperando_bola*, ya que podría darse la circunstancia de que se detectara la bola en otras situaciones en las que no debería ser tenida en cuenta. Así pues, en la rutina de servicio al periférico K_SBola, por delante de las acciones anteriores, pondremos como condición que el sistema esté en el estado deseado, así:

```
if (estado_automata == esperando_bola)
  {
      num_lanzamiento++;
      10seg = 0;
      estado_automata = estabilizando;
  }
```

3. Estado *estabilizando*:

- Características del estado:
  En este estado el sistema está esperando a que los bolos se estabilicen, por lo que lo único que hay que hacer es esperar que vaya transcurriendo el tiempo, para lo que basta que la rutina de atención al reloj actualice adecuadamente la variable `10seg`:

↳ Acciones a realizar en la rutina de servicio a la interrupción del reloj:

Reloj.3) Actualización de la variable `10seg` en cada interrupción del reloj:

```
if (estado_automata == estabilizando)
      10seg++;
```

- Transiciones posibles:

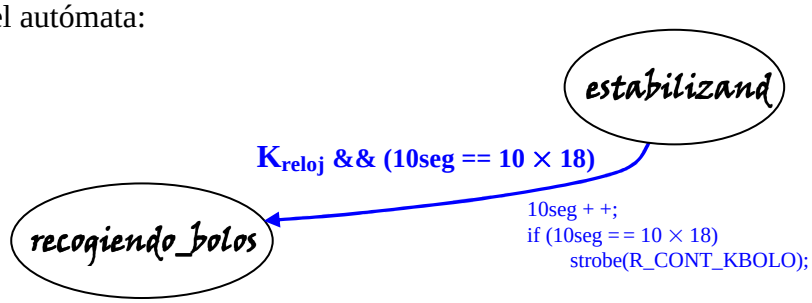
a) Una vez transcurridos los 10 segundos, se supone que los bolos ya se han estabilizado, por lo que se puede proceder a recogerlos, para saber cuántos han sido derribados y colocarlos en posición para la siguiente tirada. De todo ello se encarga el periférico K_Bolo. Por eso, al finalizar el intervalo de los 10 segundos, es necesario activar dicho periférico, para que se ponga en marcha y realice su cometido. Para ello, se debe realizar una secuencia de `strobe` sobre su registro de control, como consecuencia de la cual el dispositivo se pondrá en marcha. Así, el sistema pasará a otro estado, en el que permanecerá mientras K_Bolo esté recogiendo los bolos; designaremos este nuevo estado como *recogiendo_bolos*.

En lo que respecta a esta transición de estado, la rutina de servicio a la interrupción del reloj quedará como sigue:

Reloj.4) Transición de estado, si es necesario:

```
if (10seg == 10 × 18)
   {
      strobe(R_CONT_KBOLO);
      estado_automata = recogiendo_bolos;
   }
```

Y en el autómata:



4. Estado *recogiendo_bolos*:

- Características del estado:
  En este estado el sistema está a la espera de que K_Bolo finalice la recogida de los bolos, lo que indicará mediante una petición de interrupción. Así pues, es K_Bolo quien genera el evento que hace que el sistema pase a otro estado. Ahora bien, el nuevo estado dependerá del número de lanzamientos que se hayan realizado hasta ese momento, si ya son 10 y la partida debe finalizar, o si son menos y la partida debe continuar. En el siguiente apartado veremos esas dos transiciones posibles.
  Pero, además de la transición de estado, en el enunciado se indica que se debe presentar en pantalla la puntuación obtenida, para lo que disponemos de la función ya escrita `mostrar_derribados(total_derribados,num_lanzamiento)`. Para ello, deberemos saber con antelación cuántos bolos han sido derribados en el último lanzamiento realizado, es decir, deberemos leer el registro de datos del controlador K_Bolo, y luego sumarle el valor así obtenido (que denominaremos `derribados`) a la variable global `total_derribados`, que es la que computa cuántos bolos se han ido derribando en lanzamientos anteriores y es la que se debe visualizar en pantalla, junto con el número de lanzamiento. La presentación de los resultados en pantalla se debe realizar en las dos transiciones posibles indicadas.

↳ Acciones a realizar en la rutina de servicio a la interrupción del periférico KBolo:

Bolo.1) Lectura del registro de datos:

```
derribados = InPort(R_DAT_KBolo);
```

Bolo.2) Actualización de la variable `total_derribados`:

```
total_derribados = total_derribados + derribados;
```

Bolo.3) Presentación en pantalla de la puntuación:

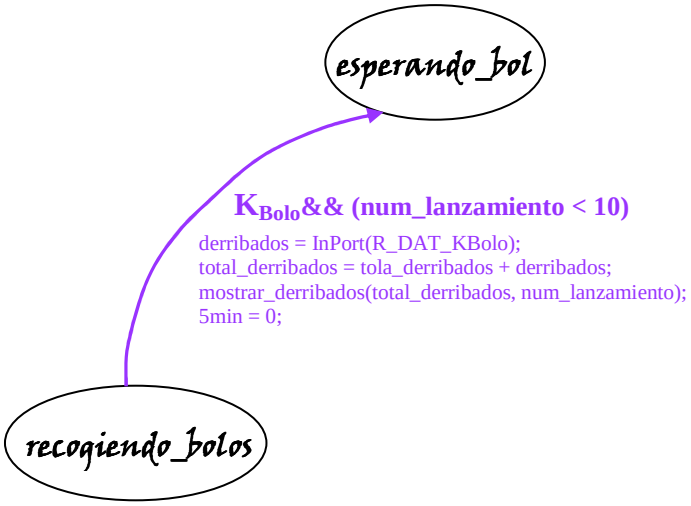
```
mostrar_derribados(total_derribados,num_lanzamiento);
```

En este caso no hemos puesto una condición inicial para la ejecución de esta rutina, ya que se supone que el controlador de K_Bolo únicamente realizará la petición de interrupción si ha sido activado previamente, lo que sólo ocurre cuando el sistema pasa al estado *recogiendo_bolos*, y no en ningún otro. No obstante, si se sospecha que pudiera darse un mal funcionamiento del periférico, entonces podríamos poner una condición previa a las acciones ya indicadas:

Bolo.0) `if (estado_automata == recogiendo_bolos)`

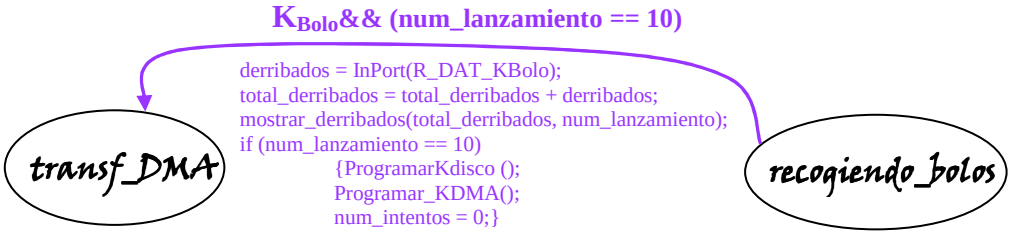
- Transiciones posibles:

a) Sabemos que una partida consta de 10 lanzamientos. Por eso, cuando K_Bolo interrumpe para indicar que ya ha recogido y recolocado los bolos, hay que analizar el valor de la variable `num_lanzamiento`: si es menor que 10, eso quiere decir que la partida puede continuar, y por eso el sistema volverá al estado *esperando_bola*, para quedar a la espera del siguiente lanzamiento.
En ese caso, se debe reinicializar de nuevo la variable `5min` a 0, para volver a controlar si transcurren 5 minutos antes de que se vuelva a producir el siguiente lanzamiento.



b) Por el contrario, si el valor de la variable `num_lanzamiento` no es menor que 10, es decir, si es igual a 10, entonces la partida se debe dar por finalizada, ya que el jugador ya ha realizado sus 10 lanzamientos. Por eso, además de presentar en pantalla la última puntuación obtenida, se debe programar el controlador de DMA,

para pasar a disco toda la información que se ha ido almacenando en memoria principal acerca de las puntuaciones parciales obtenidas en todos los lanzamientos. Para ello, es necesario programar también el controlador de disco, para lo que disponemos de la función ya escrita `ProgramarKdisco()`. Por esa razón, en este caso el sistema debe pasar a otro estado que no permita continuar con la partida (es decir, que no se quede esperando al siguiente lanzamiento de la bola), pero que sí permita esperar a que finalice la transferencia por DMA. A este nuevo estado lo denominaremos ***transf_DMA***. Según lo indicado en el enunciado, en la transferencia por DMA pueden ocurrir errores, en cuyo caso habrá que volver a realizar la transferencia, pero como máximo 3 veces. Por esa razón, necesitamos una variable global que vaya computando cuántas veces se producen errores en la transferencia por DMA o, lo que es lo mismo, cuántas veces hemos intentado realizar la transferencia sin éxito. A esa variable la denominaremos `num_intentos`, y la inicializaremos a 0 en la transición de estado que estamos considerando, ya que está claro que antes de comenzar la transferencia no se ha podido producir ningún error en la misma.
En el autómata, plasmaremos así esta transición de estado:



Las acciones correspondientes a estas dos transiciones se realizan en la rutina de servicio a la interrupción de K_Bolo:

↳ Acciones a realizar en la rutina de servicio a la interrupción del periférico KBolo:

Bolo.4) Transiciones de estado pertinentes según la condición:

```
if (num_lanzamiento < 10)
{
   5min = 0;
   estado_automata = esperando_bola;
}
else
{
   ProgramarKdisco();
   Programar_KDMA();
   num_intentos = 0;
   estado_automata = transf_DMA;
}
```

Y la función `Programar_KDMA()` utilizada queda como sigue:

```
OutPort(R_DIR_KDMA,DIR_PANT);
OutPort(R_LON_KDMA,4 × 10);
OutPort(R_CONT_KDMA,1);
```

Es decir, al controlador de DMA se le indica que la dirección a partir de la cual se encuentran los datos a transferir es la de comienzo de pantalla, DIR_PANT, ya que los datos a transferir se encuentran precisamente en la memoria de video; la longitud del bloque a transferir es 4 × 10, por tratarse de 10 lanzamientos y la puntuación almacenada en cada lanzamiento ocupa 4 bytes en memoria. Finalmente, al escribir un 1 en el registro de control, la transferencia empezará automáticamente.

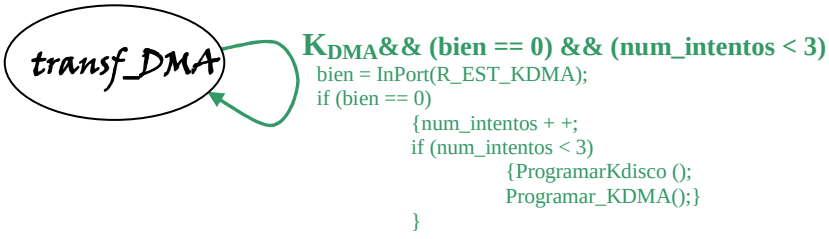
5. Estado *transf_DMA*:

- Características del estado:

  En este estado, el sistema está a la espera de que finalice la transferencia por DMA, lo que sabrá que ha ocurrido cuando el controlador de DMA realice una petición de interrupción. En ese momento, una vez finalizada la transferencia pueden darse tres circunstancias distintas: a) Que no haya ocurrido ningún error en la transferencia; en ese caso, se dará por finalizada la partida y el sistema volverá al estado ***cerrado***, debiéndose bajar la barrera y quedando el sistema dispuesto para poder comenzar una nueva partida (a la espera de que haya nuevos jugadores y que le abonen la partida al encargado de la bolera). b) Que ocurra un error en la transferencia, pero que se lleven realizados menos de tres intentos de hacer la transferencia; en ese caso, el sistema permanece en el mismo estado, pero se deben programar de nuevo los controladores de disco y de DMA (KDisco y KDMA) para volver a intenar de nuevo la transferencia. c) Que ocurra un error en la transferencia, pero que se lleven realizados ya tres intentos de hacer la transferencia; en ese caso, el sistema dará por finalizada la partida y volverá al estado ***cerrado***, pero le deberá hacer saber al encargado de la bolera que ha ocurrido un error en la transferencia, para lo que disponemos de la función `error_de_transferencia()`.

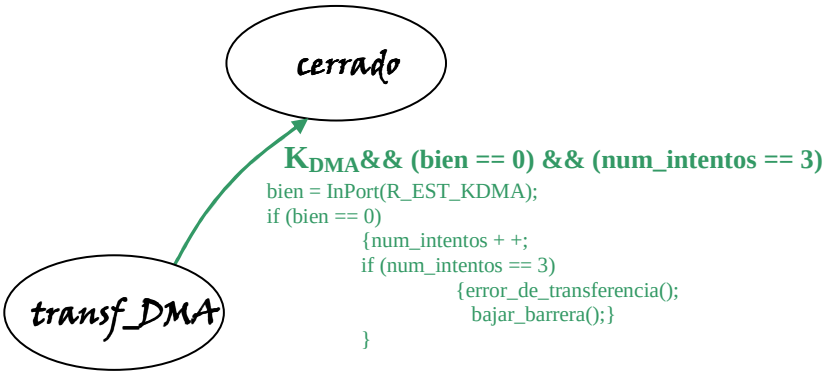
  En cualquier caso, para saber si ha ocurrido un error en la transferencia por DMA, se debe leer el registro de estado del controlador de DMA; si su valor es 1, la transferencia se ha realizado correctamente; si es 0, por el contrario, ha ocurrido un error.

  En el próximo subapartado veremos las dos transiciones de estado posibles. Ahora veamos cómo reflejar en el autómata el único evento que no supone transición de estado. El código que hay que programar en la rutina de atención a la interrupción del controlador de DMA lo veremos en el siguiente apartado, junto con el correspondiente a las dos transiciones de estado posibles.

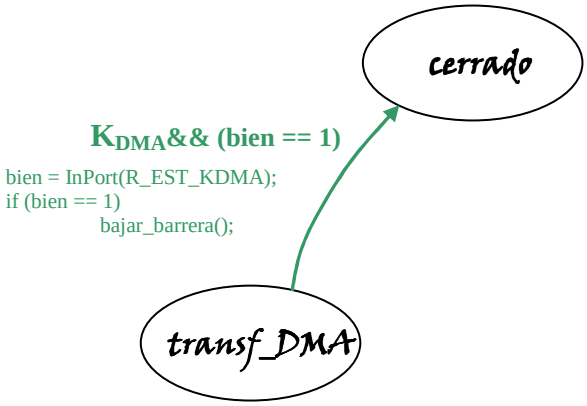


- Transiciones posibles:

a) Si en la transferencia ocurre un error por tercera vez:



b) Si en la transferencia no ocurre error:

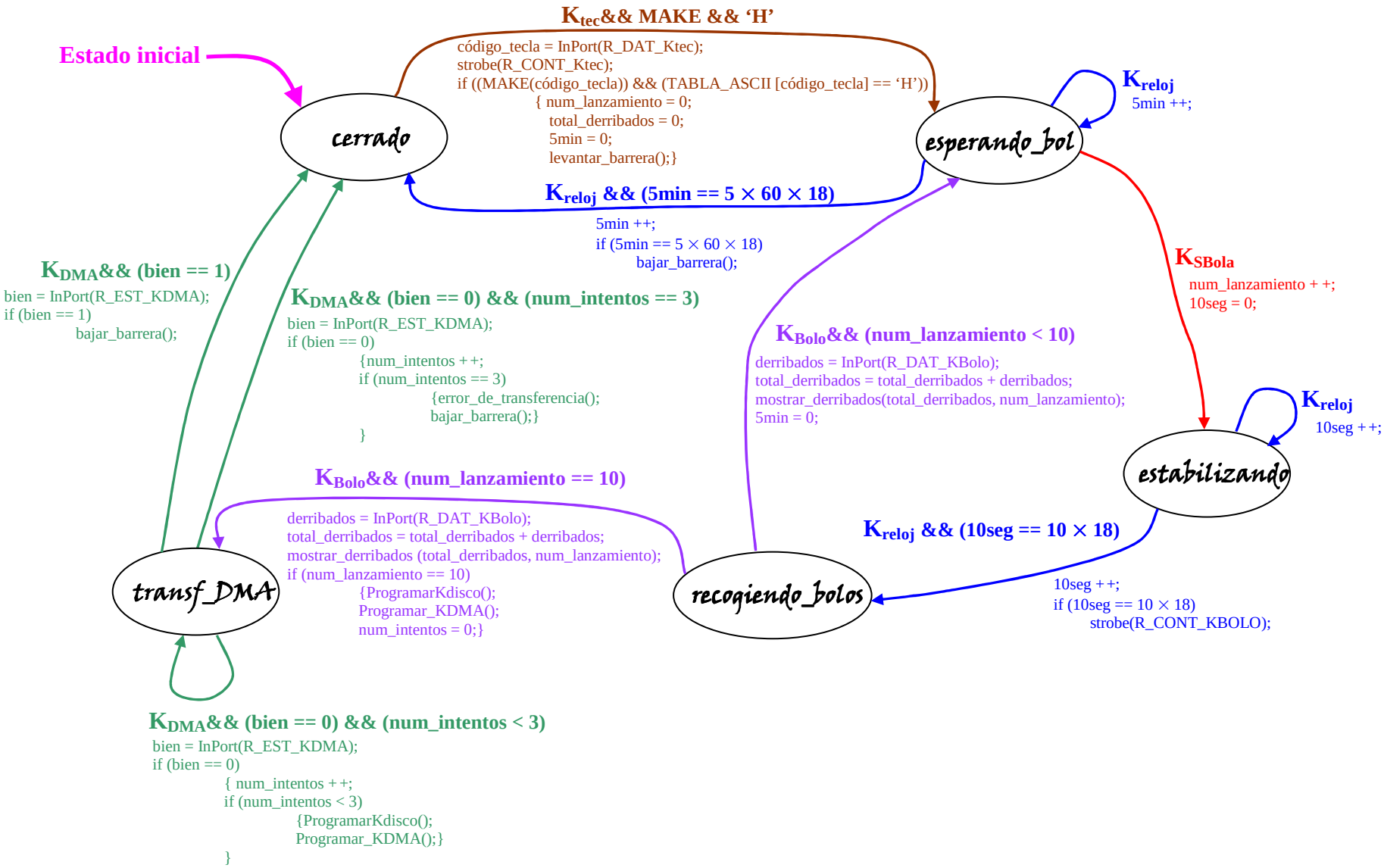


Y el código para los tres casos posibles es este:

↳ Acciones a realizar en la rutina de servicio a la interrupción del controlador KDMA:

```
bien = InPort(R_EST_KDMA);
if (estado_automata == transf_DMA)
    {
      if (bien == 0)
      {
            num_intentos + +;
            if (num_intentos < 3)
            {
                 ProgramarKdisco ();
                 Programar_KDMA();
            }
            else
            {
                 error_de_transferencia();
                 bajar_barrera();
                 estado_automata = cerrado;
            }
      }
      else
      {
        bajar_barrera();
        estado_automata = cerrado;
      }
    }
```

Teniendo en cuenta conjuntamente todos los casos posibles vistos, el autómata queda así:



Y ya sólo resta escribir el código. Tenemos que escribir el programa principal, en el que se ejecutará un bucle infinito y se realizará la encuesta del teclado, y las rutinas de atención a las interrupciones (RAI) de los periféricos: K_{SBola}, K_{Bolo}, K_{reloj} y K_{DMA}.

Programa principal.

En el programa principal se deben realizar determinada acciones, en concreto: inicialización de las variables (cuando sea necesario), modificación de algunos de los componentes del vector de interrupciones (al principio del programa) —para asegurar que se ejecutarán nuestras rutinas de servicio a las interrupciones, y no las de un hipotético sistema operativo— y recuperación de los valores iniciales de los mismos (al final del programa), y realización de un bucle infinito para que nuestro programa se ejecute de manera continua mientras no tenga que finalizar. Dentro de ese bucle se realizará la encuesta del teclado, para detectar cuándo se ha pulsado una tecla. Para poder sincronizar el teclado por encuesta se deben inhibir las interrupciones del teclado al principio del programa principal, ya que cada vez que se pulse o libere una tecla el controlador del teclado realizará una petición de interrupción que no debe ser tenida en cuenta si se desea sincronizarlo por encuesta. Al finalizar el programa, por otra parte, se deberán permitir de nuevo las interrupciones del teclado.

- Variables que hay que inicializar:
  Para empezar, supondremos que cuando el sistema se pone en marcha estará en el estado ***cerrado***. Debemos tener en cuenta, por tanto, que para asegurar el correcto funcionamiento del sistema es necesario saber en todo momento en qué estado se encuentra, para lo que utilizaremos una variable global, que denominaremos `estado_automata`, cuyo valor inicial será ***cerrado***. Además, supondremos también que la barrera está bajada. Así pues:

  ✓ `estado_automata = cerrado;`

  Conviene subrayar que no es necesario inicializar el resto de variables utilizadas, ya que se inicializan con el valor adecuado en el preciso momento en que van a ser utilizadas. Por ejemplo, la variable `10seg` se inicializa con el valor 0 en el instante en que el sistema pasa del estado ***esperando_bola*** al estado ***estabilizando***, ya que es en ese momento cuando se debe poner en marcha el "cronómetro".

- Componentes del vector de interrupciones que hay que modificar:
  Si se emplea un PC para controlar el sistema, es necesario modificar los componentes del vector de interrupciones correspondientes a los periféricos del sistema, para que el PC realice el control que nosotros deseamos, es decir, para que, cuando interrumpa uno de los periféricos, se ejecuten las rutinas de servicio que escribamos nosotros, no las que tiene programadas el sistema operativo.
  Así, en este caso concreto, los componentes que hay que modificar son los siguientes: el del reloj (entrada 0x1C del vector de interrupciones), el del dispositivo K_Bolo (entrada 0x0A), el del sensor SBola (entrada 0x0B) y el del controlador de DMA (entrada 0x0E). Para ello, supondremos que disponemos de una función ya escrita, que toma como parámetros las entradas del vector de interrupciones que hay que modificar:

  ✓ `CambiaVI(0x1C, 0x0A, 0x0B, 0x0E);`

De la misma manera, cuando finalice la ejecución de nuestro programa, será necesario recuperar los valores originales de esos componentes, para que el sistema operativo tome de nuevo el control del PC:

- ✓ `RecuperaVI(0x1C, 0x0A, 0x0B, 0x0E);`

- Inhibición y desinhibición de las interrupciones del controlador del teclado:

  Debemos asegurarnos de que las peticiones de interrupción que realice el controlador del teclado no lleguen al procesador, ya que en ese caso se sincronizaría por interrupción, y nosotros queremos sincronizarlo por encuesta. Para ello, basta con inhibir las interrupciones del teclado, pero no las del resto de los periféricos; por eso, utilizaremos el registro máscara del controlador de interrupciones, poniendo a 1 el bit 1 del mismo, que es el bit que corresponde al teclado. De esa manera, aunque el controlador del teclado genere una petición de interrupción cada vez que se pulse o libere una tecla, dicha petición de interrupción no llegará al procesador, por lo que éste deberá detectar esos eventos de otra manera.

  Así, para inhibir las interrupciones del teclado, supondremos que disponemos de una función ya escrita que pone a 1 el bit del registro máscara IMR cuya posición recibe como parámetro:

  - ✓ `InhibirKper(1);`

  Lógicamente, cuando finalice el programa se deberán habilitar de nuevo las interrupciones del teclado, para lo que también disponemos de otra función ya escrita que se encargará de poner a 0 el bit correspondiente del registro máscara IMR (el bit de la posición 1 en el caso concreto del teclado):

  - ✓ `DesinhibirKper(1);`

- Control del bucle:

  Debemos asegurarnos de que nuestro programa se ejecuta de manera continua, mientras no le indiquemos que debe finalizar su ejecución. Para ello, pondremos un bucle, que se repetirá una y otra vez mientras no haya que dar por finalizada la ejecución del programa. En este caso concreto, releyendo el enunciado, no está previsto que se pueda finalizar la ejecución de nuestro programa, por lo que emplearemos un bucle infinito:

  - ✓ `while (true)`

- Encuesta del teclado:

  Al estar inhibidas las interrupciones del teclado, el procesador deberá detectar de otra manera los eventos asociados al mismo. En general, con periféricos estándar se debe leer su registro de estado para saber si se ha producido un evento o no. Pero el teclado del PC es especial: no tiene registro de estado. En consecuencia, la única opción para saber si se ha pulsado o liberado una tecla es acceder al registro IRR del controlador de interrupciones, ya que en él quedan activadas las peticiones de interrupción aunque no sean atendidas por el procesador. Así, cuando se pulse una tecla, el controlador del teclado generará una petición de interrupción, con lo que el bit 1 del registro IRR pasará a valer 1; pero como las interrupciones del teclado están inhibidas, ese valor se quedará ahí, por lo que el procesador deberá leer el contenido de ese registro, para saber qué valor tiene ese

bit concreto: si vale 0, indica que no se ha pulsado/liberado ninguna tecla, por lo que debe leerlo una y otra vez, para realizar una encuesta continua del teclado, lo que haremos en un bucle condicional; si, por el contrario, el valor de ese bit es 1, eso significa que se ha pulsado/liberado una tecla, por lo que el programa saldrá del bucle anterior y ejecutará las acciones correspondientes al tratamiento deseado del teclado. La encuesta queda como sigue:

```
IRR = LeerIRR();
while (IRR1 == 0)
        IRR = LeerIRR();
```

Resumiendo, el código del programa principal queda así:

```
void main()
{
        //inicialización de variables
 estado_automata = cerrado;
        //modificación de los componentes del vector de interrupciones
 CambiaVI(0x1C, 0x0A, 0x0B, 0x0E);
        //deshabilitar interrupciones del teclado, para sincronizarlo por encuesta
 InhibirKper(1); // IMR1 = 1 en el controlador de interrupciones
        //bucle principal
 while (true)
  {
        //encuesta del teclado
   IRR = LeerIRR();
   while (IRR1 == 0)
        IRR = LeerIRR();
        //se ha pulsado o soltado una tecla (por eso ha salido del bucle anterior)
   código_tecla = InPort(R_DAT_Ktec);
   Strobe(R_CONT_Ktec);
   if ((estado_automata == cerrado) && MAKE(código_tecla) &&
   TABLA_ASCII[código_tecla] == 'H')
        {
        num_lanzamiento = 0;
        total_derribados = 0;
              //poner el cronómetro en marcha para contar 5 minutos
        5min = 0;
        levantar_barrera();
              // transición de estado pertinente
        estado_automata = esperando bola;
        }
  }
        //habilitar interrupciones del teclado
 DesinhibirKper(1);
        //recuperación de los componentes del vector de interrupciones al
        finalizar el programa
 RecuperaVI(0x1C, 0x0A, 0x0B, 0x0E);
}
```

➢ Rutina de atención a la interrupción (RAI) del controlador K_{SBola}.

Ya hemos visto cuáles son las acciones a realizar en esta rutina de servicio. Ahora sólo tenemos que ordenarlas:

```
void interrupt RAI_KSBola()
{
  if (estado_automata == esperando_bola)
  {
        //se ha lanzado la bola en el estado correcto
    num_lanzamiento++;
        //poner el cronómetro en marcha para contar 10 segundos
    10seg = 0;
        // transición de estado pertinente
    estado_automata = estabilizando;
  }
  Eoi();
  IRET;
}
```

➢ Rutina de atención a la interrupción (RAI) del controlador K_{Bolo}.

```
void interrupt RAI_KBolo()
{
      //leer la cantidad de bolos que se han derribado en esta tirada
  derribados = InPort(R_DAT_KBolo);
      //actualizar el número de bolos derribados en total en todas las tiradas
  total_derribados = total_derribados + derribados;
      //Mostar en pantalla el total de bolos derribados
  mostrar_derribados(total_derribados,num_lanzamiento);
      // transiciones de estado pertinentes
  if (num_lanzamiento < 10)
    {
      5min = 0;
      estado_automata = esperando_bola;
    }
     else
     {
        ProgramarKdisko();
        Programar_KDMA();
        num_intentos = 0;
        estado_automata = transf_DMA;
     }
  Eoi();
  IRET;
}
```

El código de la función `Programar_KDMA()` que se ha utilizado en la rutina anterior:

```
void Programar_KDMA()
{
        OutPort(R_DIR_KDMA, DIR_PANT);
        OutPort(R_LON_KDMA,4 × 10);
        OutPort(R_CONT_KDMA,1);
}
```

➢ Rutina de atención a la interrupción (RAI) del controlador K_{reloj}.

```
void interrupt RAI_Kreloj()
{
if (estado_automata == esperando_bola)
    {
            //actualizar variable 5min en cada interrupción del reloj
        5min++;
            // transición de estado pertinente
        if (5min == 5 × 60 × 18)
        {
            bajar_barrera();
            estado_automata = cerrado;
        }
    }
if (estado_automata == estabilizando)
    {
            // actualizar variable 10seg en cada interrupción del reloj
        10seg++;
            // transición de estado pertinente
        if (10seg == 10 × 18)
        {
            //secuencia "strobe" sobre el dispositivo KBolo para activarlo
           strobe(R_CONT_KBolo);
           estado_automata = recogiendo_bolos;
        }
    }

IRET;
}
```

Rutina de atención a la interrupción (RAI) del controlador K_{DMA}.

Hay que tener muy claro que esta rutina de servicio sólo se ejecutará al finalizar la transferencia por DMA, ya que es en ese momento cuando el controlador de DMA genera la petición de interrupción, nunca antes. En consecuencia, se debe leer el registro de estado del controlador de DMA, para saber si la transferencia ha finalizado correctamente, o si, por el contrario, ha surgido algún problema que ha dado lugar a una transferencia errónea. En este último caso, se debe volver a reintentar la transferencia, pero sólo se permiten 3 intentos como máximo, una vez realizados los cuales el sistema no lo intentará nuevamente, sino que volverá al estado inicial, para permanecer a la espera y preparado para comenzar una nueva partida. También si la transferencia finaliza correctamente (es decir, sin errores) el sistema volverá al estado inicial, dando por finalizada la partida actual y quedando a la espera de la siguiente.

```
void interrupt RAI_KDMA()
{
 bien = InPort(R_EST_KDMA);
 if (estado_automata == transf_DMA)
    {
      if (bien == 0)
      {
            num_intentos++;
            if (num_intentos < 3)
            {
                 ProgramarKdisco();
                 Programar_KDMA();
            }
            else
            {
                 error_de_transferencia();
                 bajar_barrera();
                 estado_automata = cerrado;
            }
      }
      else
      {
        bajar_barrera();
        estado_automata = cerrado;
      }
 Eoi();
 IRET;
}
```