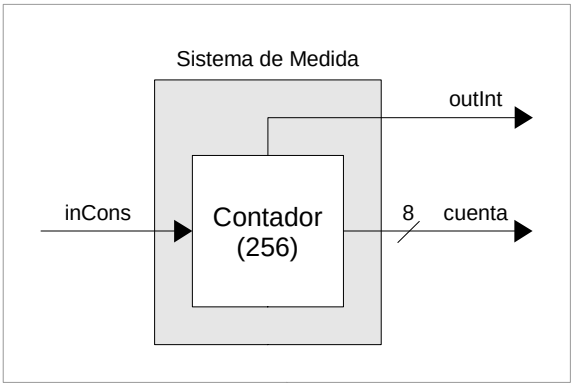
• Nombre y CI en cada hoja • Numere las hojas, indique el total en la primera • Utilice sólo un lado de las hojas	• Incluya un solo problema por hoja • **Sea prolijo** • **Aprobación:** mínimo UN problema

PROBLEMA 1

Se desea integrar a un sistema ya existente basado en Z80 y alimentado a baterías, un sistema de medida de consumo que permita medir la energía consumida de la batería durante la ejecución de ciertas partes de código.



Al sistema de medida con que se cuenta, ingresa una señal **inCons** en la que cada flanco equivale a una unidad de consumo. Estos flancos pasan por un bloque contador que genera un flanco de subida en la señal **outInt** cada 256 flancos de la señal **inCons**. El bloque contador tiene disponible todo el tiempo una señal **cuenta** de 8 bits con la cuenta parcial de flancos de la señal **inCons**.
La señal outInt se utiliza para interrumpir al procesador y así llevar la cuenta de la cantidad de interrupciones en una variable **num_int** de 8 bits. Esta es la única señal que interrumpe al procesador.

La subrutina **consAcum** debe devolver en el par de registros BC el consumo acumulado desde que se encendió el sistema ([cant de flancos en outInt] * 256 + cuenta). Notar que el resultado es un nro de 16 bits donde el byte alto es **num_int** y el byte bajo es **cuenta**. Si la cantidad de interrupciones superó el valor 255 la subrutina consAcum debe devolver el valor 0xFFFF.

Para indicar cuándo es necesario cambiar las baterías se cuenta con una señal **alarma** activa por nivel alto mediante la cual se deberá indicar cuando la cantidad de interrupciones supere el valor de la variable **umbral** de 8 bits. Inicialmente el umbral está dado por la constante **UMBRAL_INI** y para dar flexibilidad a la solución el umbral puede ser cambiado con el sistema en funcionamiento.

El nuevo umbral se recibe mediante la entrada **datoUmbral**, que tiene un valor válido en el momento del flanco de subida en la entrada **reqUmbral**. Se debe indicar que se realizó la lectura mediante un flanco de subida en la salida **ackUmbral**.

El programa principal invoca periódicamente la subrutina **verifUmbral**, que si hay un nuevo valor lo lee según el protocolo descripto y actualiza la variable umbral. En caso contrario retorna sin hacer nada.

```
Loop:
  CALL otros
  CALL verifUmbral
  JP loop
```

Se tiene disponibles los puertos de entrada/salida a partir de la dirección 0x80 y debido a futuras expansiones se debe ocupar la mínima cantidad de puertos de E/S. El sistema debe equiparse con 32Kb de RAM y 32Kb de ROM con el mapeo habitual.

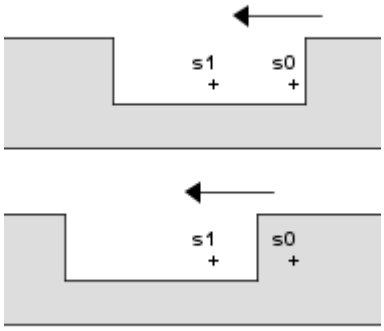
Se pide:
a) Hardware completo indicando: conexión del sistema de medida, decodificación de todos los puertos necesarios.

b) Subrutina de atención a la interrupción, que acumule el consumo desde que se encendió el sistema, indique a la subrutina consAcum si hubo desborde y maneje la señal de alarma.

c) Subrutina **consAcum**. que devuelve el consumo acumulado. A los efectos del problema se podrá suponer que no se produce un flanco en outInt durante la ejecución de esta subrutina.

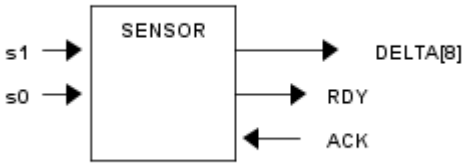
d) Inicialización del sistema y código de la subrutina **verifUmbral**.

PROBLEMA 2



Se quiere diseñar un sensor que reporte a un sistema central los cambios en la posición angular de una perilla. Para sensar la posición de la perilla se utiliza un disco ranurado adosado a su eje y dos sensores **s1** y **s0**. La salida de cada sensor toma el valor uno cuando el sensor está frente a una ranura y cero en caso contrario. Los sensores están ubicados a una distancia igual a media ranura de modo que el valor de s1 en el momento de un flanco de subida de s0 indica el sentido de giro de la perilla. Si la perilla gira en sentido positivo s1 valdrá 0 (cambio de s1s0=00 a s1s0=01 como en la figura), y si gira en sentido contrario (negativo) valdrá 1 (cambio de s1s0=10 a s1s0=11). Con cada flanco de subida de s0 se actualiza la posición angular de la perilla en +/- 1 ranura.

El sensor se comunica con el sistema central poniendo un dato en la señal **DELTA** (en complemento a 2 de 8 bits) y subiendo a 1 su salida **RDY** para indicar que hay un nuevo dato disponible. El sistema central informa que leyó el dato con un pulso a 0 de corta duración en **ACK**, que debe provocar la bajada de RDY.



Cada reporte indica el incremento en la posición de la perilla desde el reporte anterior, o desde el reset en el caso del primer reporte. El sensor debe generar un nuevo reporte solamente si se cumplen las tres condiciones siguientes:

- El reporte anterior ya fue leído por el sistema central.
- Transcurrió al menos un tiempo **TMIN** desde el reporte anterior.
- La posición de la perilla difiere de la del reporte anterior.

Dicho de otra manera, el sensor no debe sobreescribir datos que no han sido leídos, no debe enviar datos separados por menos de un tiempo TMIN y no debe informar si no ha cambiado nada desde el reporte anterior. A los efectos del problema se podrá suponer que el valor a reportar es siempre representable en complemento a 2 de 8 bits (no hay overflow).

La solución debe organizarse en un programa principal y dos interrupciones usando modo 2:

- Interrupción por el flanco de subida de s0 que lleve la cuenta de la posición angular.
- Interrupción que informe al programa principal cuando ya transcurrió TMIN desde el reporte anterior.
- El programa principal debe escribir los reportes respetando las restricciones definidas más arriba.

Se dispone o bien de un Z80 tradicional con bus triestado y un CTC para generar las interrupciones; o bien de un T80 con buses de entrada y salida separados y bloques Timer y Contador como los usados en el laboratorio.

Se pide:
a) Todo el hardware excepto la memoria (32K de RAM y 32K de ROM) que se supone implementada.
b) Todo el software que incluye:

i. Bucle infinito del programa principal.
ii. Rutinas de atención a interrupción.
iii. Inicialización del sistema. Debe indicarse la ubicación en memoria de todo el código, tablas y variables utilizadas mediante directivas al ensamblador.

Nota: **fck** = 4MHz, **TMIN** = 8ms, 8ms = (1 / 4MHz) * 256 * 125