• 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

Un sistema basado en Z80 va a funcionar en campo sin atención de un operador, y debe cada tanto ejecutar un test para detectar posibles fallas.

Para evitar afectar al sistema durante intervalos largos de tiempo, el test se divide en **N** subrutinas (**sub_0** hasta **sub_n-1**, con **N** fijo menor que 100) que deberán ejecutarse con una separación de tiempo **deltaT** entre ellas. La duración de cada subrutina es mucho menor que deltaT y el sistema sigue su funcionamiento normal entre la ejecución de una subrutina y la siguiente. Se dispone de una señal **tic** periódica de período deltaT que se usará durante el test para interrumpir al sistema. No hay otra fuente de interrupción.

```
main:
 call check_trg
 call principal
 jp main
```

Luego de reset el test comienza detenido. Cada flanco de subida en la entrada **trg** debe recomenzar el test si estaba detenido o detenerlo si estaba habilitado. La primera interrupción después que se recomienza el test debe invocar a la subrutina **sub_0**. Las siguientes interrupciones invocarán a las siguientes subrutinas de la tabla en forma cíclica, recomenzando con **sub_0** luego de **sub_n-1**. Para detener el test cuando llegue un nuevo flanco de subida en **trg** simplemente se deshabilitan interrupciones.

```
tabla_sub:
  DW sub_0
  DW sub_1
  ...
  DW sub_n-1
fin_tabla_sub:
```

El programa principal consiste del loop infinito **main** dentro del cual se invoca a la subrutina **check_trg** que deberá detectar si hubo un flanco de subida en trg y tomar las acciones que corresponda (ver recuadro). Las direcciones de comienzo de las subrutinas están almacenadas en orden en una tabla en ROM indicada como **tabla_sub** en el segundo recuadro.

Cada vez que llega una interrupción, la rutina de servicio debe invocar a una subrutina de test diferente. Para eso se utilizará la subrutina **salto_hl** mostrada en el recuadro. Observar que haciendo **call salto_hl** habiendo cargando previamente la dirección de comienzo de la subrutina que se quiere invocar, el salto **jp (HL)** transfiere el control del programa a la subrutina deseada sin cambiar la dirección de retorno almacenada en el stack.

```
salto_hl:
  jp (HL)
```

Se pide:

a) Todo el hardware incluyendo memoria, interrupcion **tic**, puertos necesarios para la entrada **trg** y dos puertos de salida **pout1**[8], **pout0**[8] y dos puertos de entrada **pin1**[8], **pin0**[8] que serán utilizados por el programa principal. Para la memoria se dispone de chips de 16KB y se debe dejar libre la mitad superior del espacio de memoria para futuras ampliaciones. Dar un mapa de memoria de acuerdo a la decodificación utilizada.

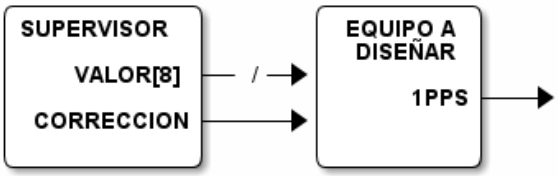
b) Subrutina **check_trigger**.

c) Rutina de atención a la interrupción.

d) Inicialización del sistema que incluya todo lo necesario para el funcionamiento correcto del sistema. Para inicializar las funciones no detalladas aquí se debe invocar la subrutina **init_principal** que se supondrá cargada en ROM. Incluir las directivas necesarias para determinar la ubicación en memoria de todo el código, variables, tablas y constantes usadas.

PROBLEMA 2

Se debe generar un sistema que genere una señal 1PPS, la cual consiste en un pulso a 1 de corta duración cuyo flanco de subida se da periódicamente cada 1 segundo.



Como base de tiempo para contar el segundo se debe utilizar un periférico Timer que deberá de interrumpir al sistema en modo 2. Cada vez que se da la interrupción el sistema debe dar el pulso 1PPS. La frecuencia del sistema es de 2Mhz y con un prescaler 1111 y constante 80 se logra una precisión suficiente.

Cambios de temperatura hacen que la frecuencia de reloj teórica de 4MHz pueda tener desviaciones, acelerándose o enlenteciéndose. Un sistema externo supervisa estas variaciones y en caso de llegar a determinado valor genera un pulso a 1 en su salida CORRECCION y en su salida VALOR de 8 bits, un número en Complemento a 2 con el valor a sumar para corregir la constante del Timer. Este valor se mantiene hasta el siguiente pulso en CORRECCION.

La señal CORRECCION debe generar otra interrupción modo 2 que se encargará de leer VALOR, calcular el nuevo valor a usar para la constante del Timer y guardarlo en la variable en memoria NUEVA_CTE_TIMER. Dado que las desviaciones de frecuencia están acotadas se puede suponer que no habrá overflow al calcular el nuevo valor de constante de Timer.

Como el cambio de período no puede ser brusco, la rutina de atención al timer deberá ir reprogramándolo con incrementos de a +/-1 cuenta hasta llegar al valor en NUEVA_CTE_TIMER. Se puede suponer que no llegará un nuevo pulso en CORRECCION hasta que se complete el incremento.

Se pide:

a) Hardware de I/O del sistema: periféricos, puertos y decodificación. Encadenar los controladores de interrupciones para dar más prioridad al Timer. El rango de direcciones 0x00-0x3F de E/S está utilizado por otros dispositivos no descriptos aquí. Se supone ya implementada la memoria con 32KB de ROM y 32KB de RAM.

b) Código a ejecutarse luego de un reset que inicialice variables, puertos, periféricos y el sistema de interrupciones, y luego salte a la etiqueta MAIN, donde se encuentra el código principal del problema que ejecuta otras tareas. Incluir las directivas necesarias para determinar la ubicación en memoria de todo el código, variables, tablas y constantes usadas.

c) Rutinas de atención a las interrupciones.