

- | | |
|--|--|
| <ul style="list-style-type: none">• Mínimo para aprobar: un problema• Nombre y CI en cada hoja• Numere las hojas, indique el total en la primera | <ul style="list-style-type: none">• Utilice sólo un lado de las hojas• Incluya un solo problema por hoja• Sea prolijo |
|--|--|

Si usa las directivas **.text** y **.data** indicar la dirección de comienzo de cada sección.

PROBLEMA 1

Un dispositivo a bordo de un automóvil transmite el valor de una magnitud física **S**, de 8 bits, en forma periódica a intervalos de 10ms.

El sistema debe leer **S** cada vez que el sensor solicita atención mediante un flanco de subida en la señal **new**, y almacenar el valor leído hasta su posterior transmisión cuando se complete el período de 10ms. **S** es válido únicamente en el instante del flanco de subida de **new**.

La transmisión debe hacerse en forma periódica a intervalos de 10ms de acuerdo a lo siguiente:

- Si en un intervalo se realiza más de una lectura de **S**, deberá transmitirse la última.
- Si no se realiza ninguna lectura de **S** en el intervalo, entonces se volverá a transmitir la última lectura realizada con un valor especial en el parámetro **data_age** definido más abajo.

El valor de **S** se transmite como un mensaje de tres bytes que se escriben por separado. La interfaz hacia la unidad de control tiene las siguientes entradas:

data_id[8]: Número fijo (**DATA_ID_VAL** = 0x5A) que identifica la magnitud medida.

data_age[8]: Tiempo entre la lectura de **S** y el momento en que se transmite el mensaje.

data_value[8]: Valor leído de la magnitud física.

stb: Un flanco de subida provoca que la unidad de control capture los valores presentes en las otras tres entradas y comience a transmitirlos.

El valor de **data_age** se mide en intervalos de 0.1ms. Cuando en un intervalo de 10ms no hay una lectura nueva, es decir, si la antigüedad de la última medida es mayor a $100 \times 0.1\text{ms}$, se transmite **data_age=255**.

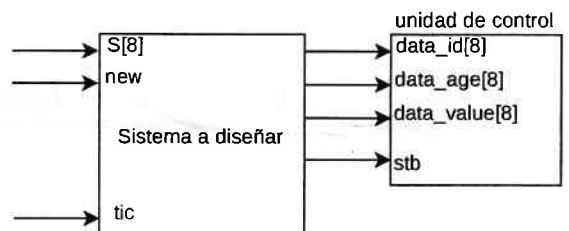
Luego de un reset se asumirá **data_value=0** hasta el primer flanco de subida de **new**.

Diseñar completamente el dispositivo con un sistema basado en el microprocesador **Z80**. Se dispone de una señal **tic** de período 0.1ms para interrumpir al procesador trabajando en modo 1. El programa principal se encarga de leer la señal **S** cuando corresponda.

Se pide:

- a) Diseñar el hardware del sistema incluyendo la memoria y los puertos necesarios para la funcionalidad descrita. Otras funcionalidades no descriptas ocupan la mitad más baja del espacio de E/S. Para la memoria se cuenta solamente con un chip de RAM de 8KB y un chip de ROM de 16KB, que deben decodificarse sin fantasmas.
- b) Rutina de atención a la interrupción periódica.
- c) Inicialización, loop de programa principal y directivas de reserva de memoria. Las funciones del sistema no descriptas en este problema no utilizan interrupciones. Para atenderlas debe invocarse la subrutina **init_otros()** durante la inicialización, y la subrutina **atiendo_otros()** dentro del loop del programa principal.

Nota: tener en cuenta que 0.1ms es un tiempo largo comparado con la duración de una instrucción. Se podrá despreciar la duración de trozos cortos de programa y de ciclo de reconocimiento de interrupción.



PROBLEMA 2

Utilizando un sistema basado en T80 se debe diseñar un instrumento para medir el ancho de los pulsos a cero en la entrada **pulso**, que está normalmente en nivel alto.

El resultado de la medida debe presentarse en la salida **medida** de 8 bits como múltiplos del intervalo **deltaT**.

La secuencia de medida es como sigue:

- Un operador indica con un flanco de subida en **start** que debe medirse el próximo pulso. En respuesta a ese flanco el sistema debe leer **pulso** y, si está en nivel bajo debe activar la salida **fail** y abortar la secuencia.
- En caso contrario (**pulso** = 1) se debe activar **midiendo** y configurar un bloque Timer para que **arranque** con el próximo flanco de bajada de **pulso** y decremente a intervalos **deltaT**. El Timer se debe configurar con el mayor valor posible de constante inicial para poder medir pulsos de hasta $255 \cdot \text{deltaT}$.
- Si antes que expire el Timer llega un flanco de subida en **pulso**, se debe leer el Timer, calcular el ancho del pulso, escribir el resultado en **medida** y activar la salida **ok**.
- Si en cambio el Timer expira antes, si el pulso es mayor o igual a $256 \cdot \text{deltaT}$ se debe activar la salida **overrun** para indicar que la medida falló.

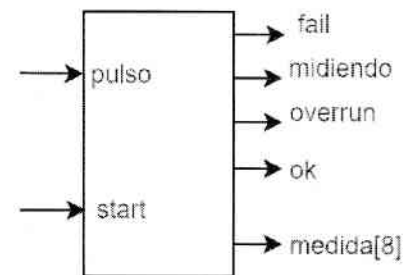
En cualquiera de los dos casos se debe apagar la salida **midiendo** y desactivar el Timer para evitar que siga generando interrupciones. Las salidas **fail**, **overrun** y **ok** deben mantenerse hasta que el instrumento sea armado nuevamente por el operador con un nuevo flanco en **start**. En ese momento estas salidas se deberán desactivar y se comenzará una nueva secuencia.

Se utilizarán dos interrupciones trabajando en modo 2: una asociada al flanco de subida de **pulso** y otra asociada al Timer. La interrupción del Timer tiene mayor prioridad, aunque a los efectos del problema se puede asumir que la subida de **pulso** y el fin del Timer llegan con suficiente diferencia para que nunca llegue una interrupción mientras se atiende la otra.

Se pide:

- a) HW del sistema: controladores de interrupción, Timer y puertos de entrada y salida. Asumir que ya está implementada la memoria con 32K de ROM y 32K de RAM.
- b) Programa principal que supervise la entrada **start** e inicie un ciclo de medida.
- c) Rutinas `isr_timer` e `isr_pulso` de atención a interrupciones de Timer y **pulso**.
- d) Código a ejecutar luego de un reset que inicialice procesador, periféricos, variables y puertos de salida. También directivas al ensamblador para reserva de espacio en memoria para variables. Indicar en el código dónde se incluye (usando `.include` o con un comentario) el programa principal de la parte b) y las subrutinas de atención a la interrupción de la parte c).

Nota: $\text{deltaT} = 2^{12} \cdot \text{Tclk}$



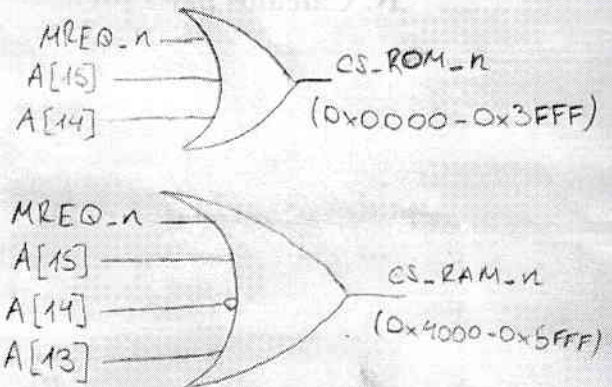
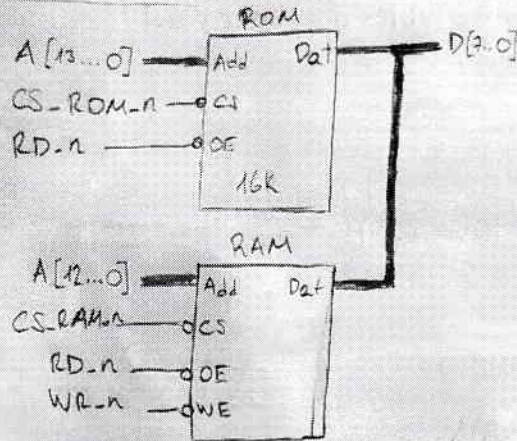
SOLUCIÓN PROBLEMA 1

a) Hardware

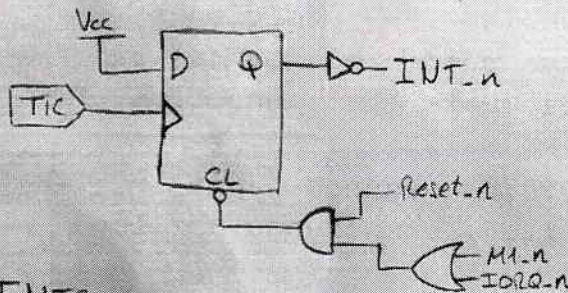


PROBLEMA 1

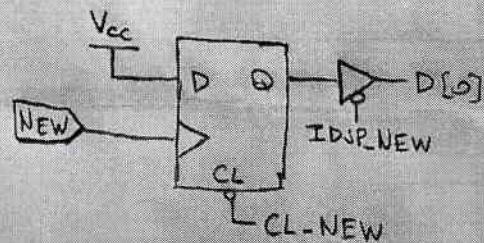
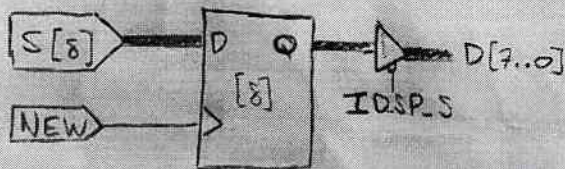
1. MEMORIA



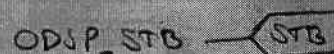
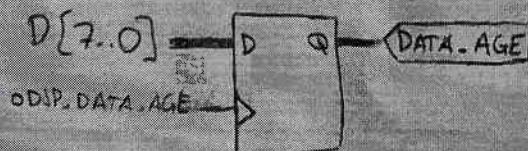
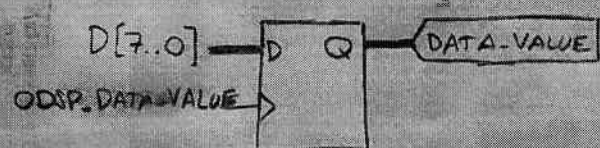
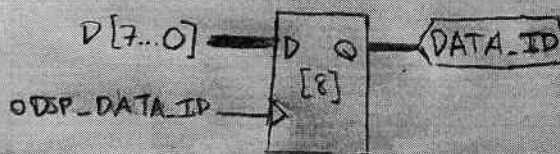
2. FF petición de interrupción

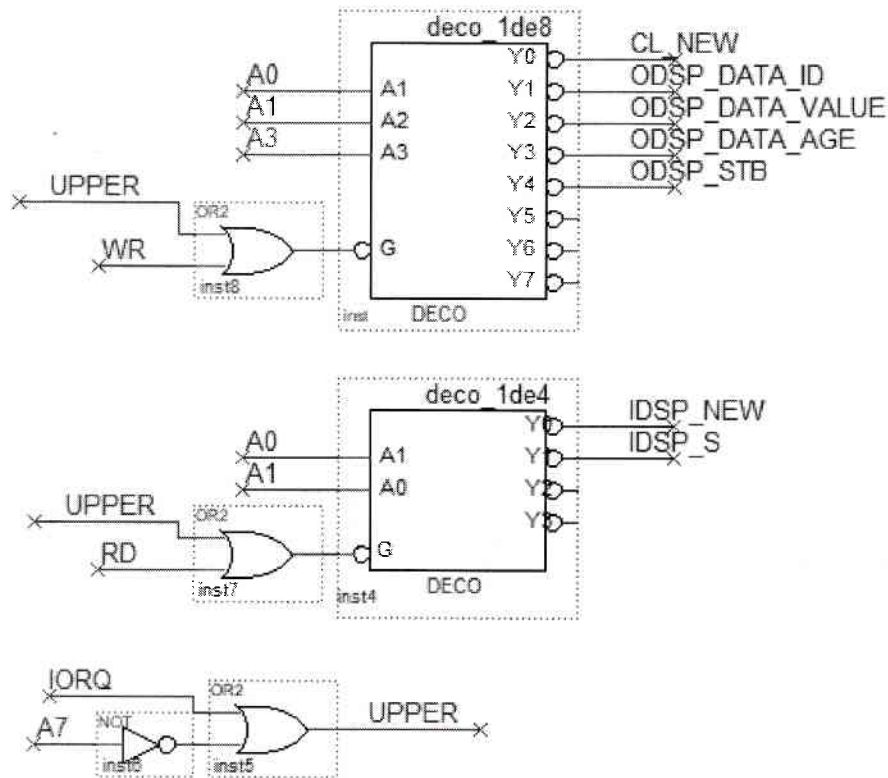


3. ENTRADAS



4. SALIDAS





b)

.org 0x0038

isr(){

 preservo

 cont_tics++

 si (cont_tics == 100){

 escribo puertos value y age

 doy pulso stb

 cont_tics = 0

 age = 0xFF

 }

 si (age != 0xFF){

 age++

 }

 restauro

}

isr_tic:

 EI

 PUSH AF

 LD A, (cont_tics)

 INC A

 LD (cont_tics), A

 CP 100 ; comparo con 100

 JR NZ, incremento_data_age

 ; si cumplí 10 ms

 LD A, (data_age)

 ; cargo las salidas

 OUT (DATA_AGE_OUT), A

 LD A, (data_value)

 OUT (DATA_VALUE_OUT), A

 OUT (STB), A ; pulso en stb

LD A, 0x00

LD (cont_tics), A ; reseteo contador de

datos

LD A, 0xFF

LD (data_age), A

incremento_data_age:

LD A, (data_age)

CP 0xFF ; sin valor en intervalo aún

JR Z, fin

INC A

LD (data_age), A

fin:

POP AF

RET

c)

NEW equ 0x80
CLEAR_NEW equ 0x80
S_PORT equ 0x81
DATA_ID_OUT equ 0x81
DATA_VALUE_OUT equ 0x82
DATA_AGE_OUT equ 0x83
STB equ 0x84

ID_cte equ 0x5A

.org 0x0000

; ROM más chica de lo habitual, fin en 0x5FFF

LD SP, 0x6000

IM 1

; inicializo variable

LD A, 0

LD (data_value), A

LD (data_age), A

LD (cont_tics), A

; inicializo puertos (no es necesario)

OUT (DATA_AGE_OUT), A

OUT (DATA_VALUE_OUT), A

LD A, ID_cte

OUT (DATA_ID), A

OUT (CLEAR_NEW), A *; borro*

handshake

CALL init_otros

EI

JP main

.org 0x0038

.include "isr_tic.s"

; main(){

si (new){

clear ff new

value = s_port

age = 0

};

call atiendo_otros

};

main:

IN A, (NEW)

BIT 0, A

JR Z, main_otros

OUT (CLEAR_NEW), A *; borro FF de*

handshake

IN A, (S_PORT)

LD (data_value), A

LD A, 0x00

LD (data_age), A

main_otros:

CALL atiendo_otros

JR main

.org 0x4000 *; primer dirección de RAM*

data_value: db

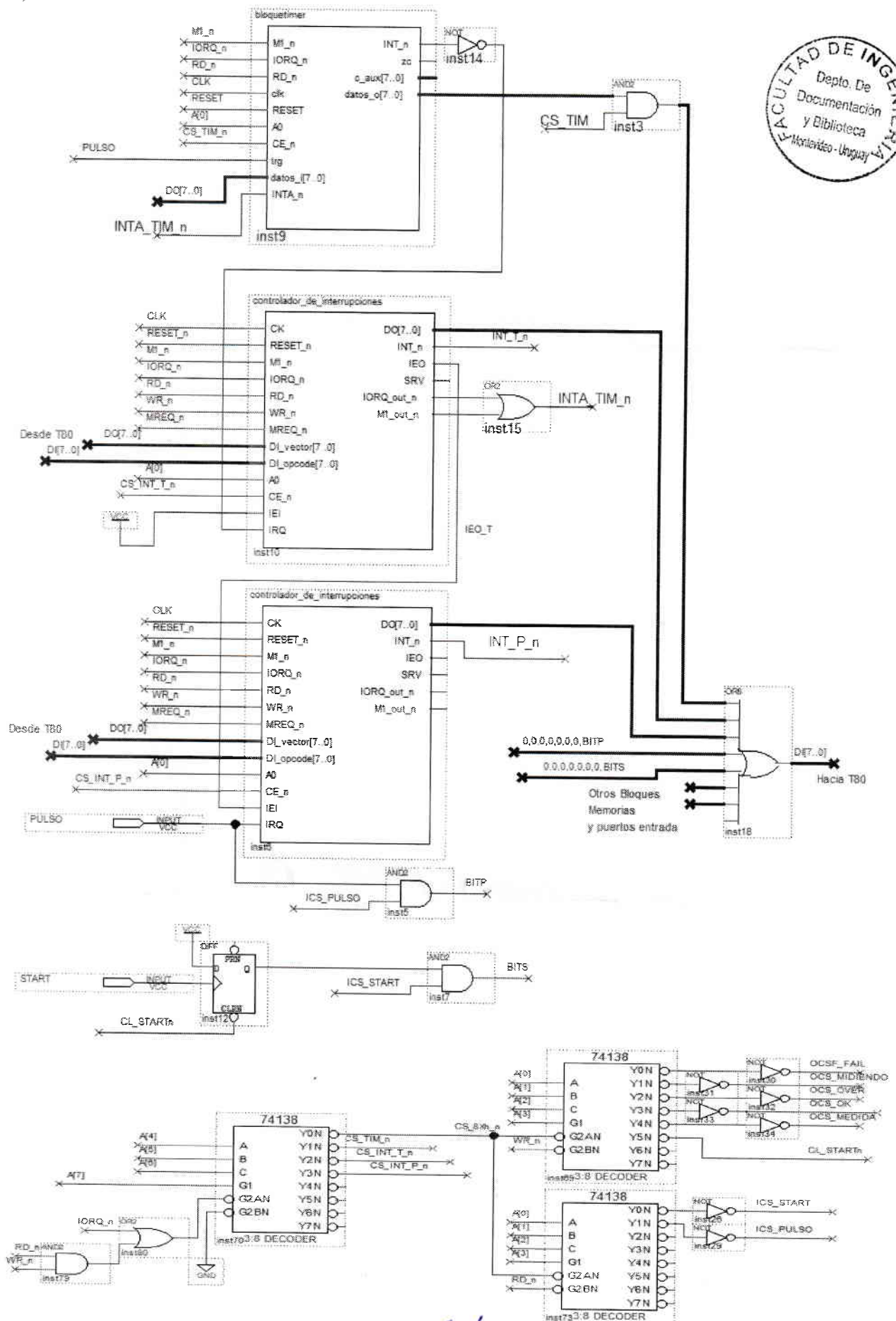
data_age: db

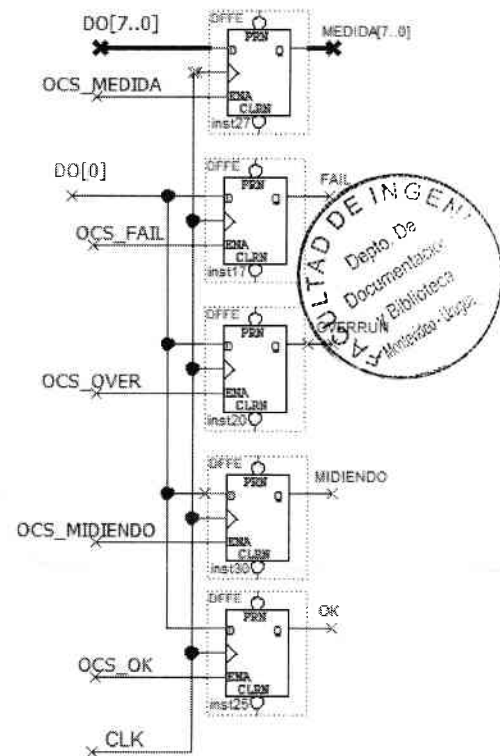
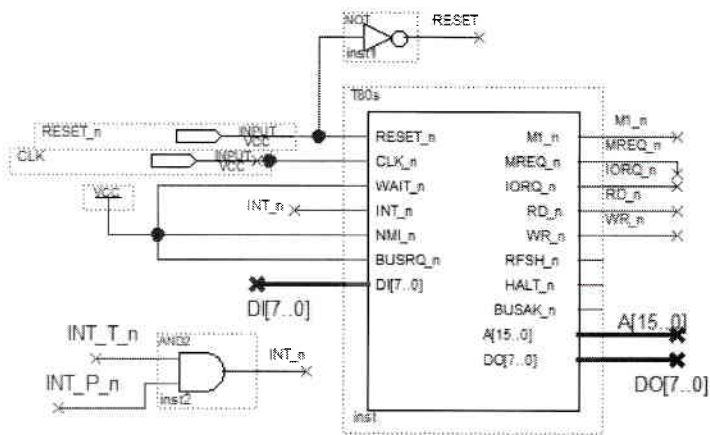
cont_tics: db



SOLUCIÓN PROBLEMA 2

a) Hardware





b)

main:

IN A, (START)

BIT 0, A

; mientras no llegue start

; mantengo salidas anteriores

JP Z, main

OUT (CL_START), A ; handshake

; chequeo que pulso este en 1

IN A, (PULSO)

BIT 0, A

JR Z, fail

; borro salidas para iniciar una nueva secuencia

LD A, 0x00

OUT (FAIL), A

OUT (MEDIDAS_OUT), A

OUT (OVERRUN), A

OUT (OK), A

LD A, 0x01

OUT (MIDIENDO), A

; configuro timer arranque flanco bajada pulso

LD A, START_TIMER

OUT (TIMER+1), A

JR main

fail:

; apago todas las salidas menos fail

LD A, 0x00

OUT (MIDIENDO), A

OUT (MEDIDAS_OUT), A

OUT (OVERRUN), A

OUT (OK), A

LD A, 0x01

OUT (FAIL), A

LD A, STOP_TIMER

OUT (TIMER+1), A

JR main

c)

isr_timer:

```

EI
PUSH AF
; prendo overrun
LD A, 0x01
OUT (OVERRUN), A
; apago midiendo
LD A, 0x00
OUT (MIDIENDO), A
; apago timer
LD A, STOP_TIMER
OUT (TIMER+1), A
POP AF
RETI

```

isr_pulso:

```

EI
PUSH AF
PUSH BC
; leo cuentas restantes
IN A, (TIMER+1)
LD B, A
LD A, TIMER_CTE
SUB B
OUT (MEDIDA), A
LD A, 0x00
OUT (MIDIENDO), A
LD A, 0x01
OUT (OK), A
; apago timer
LD A, STOP_TIMER
OUT (TIMER+1), A
POP BC
POP AF
RETI

```

d)

```

FAIL equ 0x80
MIDIENDO equ 0x81
OVERRUN equ 0x82
OK equ 0x83
MEDIDA equ 0x84
CL_START equ 0x85
START equ 0x80
PULSO equ 0x81

```

```

TIMER equ 0x90
CTIMER equ 0xA0
CPULSO equ 0xB0

```

```

CTE_TIMER equ 0xFF
START_TIMER equ 10111100b
STOP_TIMER equ 00111100b

```

.org 0x0000

```

LD SP, 0x00
IM 2
; tabla interrupciones en 0x0100
LD A, 0x01
LD I, A
; CTIMER
LD A, 0x00
OUT (CTIMER), A
OUT (CTIMER+1), A
; CPULSO
LD A, 0x02
OUT (CTIMER), A
OUT (CTIMER+1), A
; SALIDAS
LD A, 0x00
OUT (FAIL), A
OUT (OVERRUN), A
OUT (MIDIENDO), A
OUT (MEDIDA), A
OUT (OK), A
; limpio handshake
OUT (CL_START), A
; TIMER
LD A, CTE_TIMER
OUT (TIMER), A
LD A, STOP_TIMER
OUT (TIMER+1), A
EI
JP main

```

.org 0x0100

tabla_int:

```

db isr_timer
db isr_pulso

```

A CONTINUACIÓN
VAN isr_timer,
isr_pulso y main.

