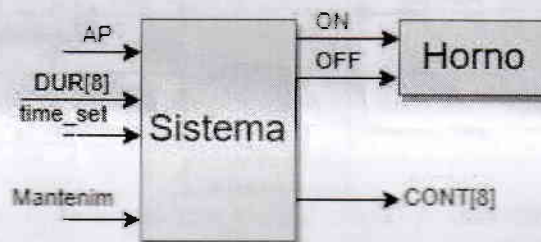


- | | |
|--|--|
| <ul style="list-style-type: none">• Mínimo para aprobar: un problema• Nombre y CI en cada hoja• Numere las hojas, indique el total en la primera | <ul style="list-style-type: none">• Utilice sólo un lado de las hojas• Incluya un solo problema por hoja• Sea prolijo |
|--|--|

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

PROBLEMA 1

Se desea diseñar un sistema basado en un T80 que pueda controlar un horno. El calentador del horno cuenta con una entrada ON que lo enciende con un flanco de subida y una entrada OFF que lo apaga cuando recibe un flanco de subida.



Se cuenta con un botón de entrada al sistema a diseñar, AP (por Arranque/Pausa), que cada vez que se presiona genera un pulso de corta duración a cero. También se cuenta con un selector de tiempo para poder elegir la duración de encendido del calentador del horno. Cuando una persona ingresa el tiempo deseado, se genera un pulso de corta duración a cero en la señal `time_sel` para indicar que el valor de la duración de encendido deseado (en segundos) está disponible en `DUR[8]` en el instante del flanco de subida de `time_sel`. Se cuenta con un bloque Timer para medir la duración de encendido del horno y que deberá interrumpir al sistema una vez que haya finalizado el tiempo.

La señal AP deberá interrumpir al sistema con máxima prioridad. La primera vez deberá iniciarse un ciclo programando el Timer para la duración más recientemente ingresada con el selector de tiempo. En las siguientes veces que se presione AP se deberá encender el calentador del horno si estaba apagado, y apagarlo si se encontraba encendido, de tal forma que la próxima vez que se presione el botón AP se retome la cuenta de tiempo desde donde se había dejado. Para eso antes de detener el Timer deberá almacenarse el valor pendiente de la cuenta para poder reprogramar el Timer en forma adecuada la vez siguiente. Cuando luego de uno o varios intervalos con el horno encendido se complete la duración deseada, debe apagarse el horno y dejar todo pronto para iniciar un nuevo ciclo en el próximo pulso de AP.

El programa principal deberá leer el valor vigente del selector de tiempo para que se pueda programar convenientemente el Timer cuando más adelante se deba encender el calentador al inicio de un ciclo. Además, en el programa principal deberá monitorearse la entrada **Mantenimiento**; cuando se genera un pulso a cero en Mantenimiento, se debe mostrar en la salida `CONT[8]` la cantidad de veces que se activó el pulso ON para encender el horno desde la última vez que se consultó.

Se pide:

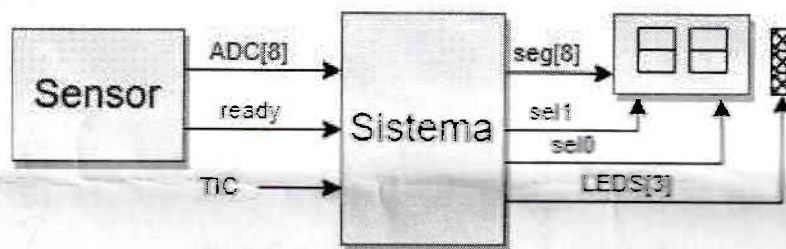
- Todo el hardware, excepto la memoria, que se supone ya implementada con 32 kB de ROM y 32 kB de RAM.
- Las rutinas de atención de las interrupciones.
- El programa principal.
- La inicialización del sistema incluyendo las reservas de memoria necesarias. Se deberá indicar la ubicación en memoria de todo el código, variables y constantes utilizadas.

Nota: $f_{clk} = 32\,768\text{ Hz}$, 1 segundo = $1 / f_{clk} * 2^{15}$

Nota 2: Luego de Reset asumir Duración = 200 seg

PROBLEMA 2

Un sensor de temperatura codifica en su salida **ADC[8]** el valor medido en punto fijo de 8 bits: 6 para la parte entera en los bits más significativos (bit7 a bit2) y 2 para la parte fraccionaria en los menos significativos (bit1, bit0). Se debe desplegar la parte entera en un display 7 segmentos de 2 dígitos, y la parte fraccionaria en la salida **LEDS[3]**. Los dos bits de parte fraccionaria permiten una granularidad de $\frac{1}{4}$ de $^{\circ}\text{C}$. Si el valor es 00, se mantienen los 3 leds apagados; si el valor es 0,25 se debe encender el bit menos significativo de **LEDS**; con 0,5 se encienden los 2 bits menos significativos; y se encienden los 3 bits cuando es 0,75.



El sensor analógico tiene un conversor A/D para implementar la codificación antes descrita y da un pulso corto a 1 en su salida **ready** cuando tiene un nuevo dato de temperatura en su salida **ADC[8]**. El dato solo es válido con el flanco de subida de **ready**.

El display 7 segmentos es multiplexado: los dos dígitos comparten las líneas de segmento y cuentan con una línea de selección activa por nivel alto. Tiene una entrada **seg** de 8 bits para controlar cada uno de los segmentos (1 encendido, 0 apagado) de ambos displays, y otra entrada de selección **sel** de 2 bits. Un 1 en el bit menos significativo de la entrada de selección enciende el dígito menos significativo del display, un 1 en la entrada más significativa enciende el dígito más significativo del display.

Se debe diseñar un sistema con Z80 que en su programa principal haga el polling de la señal **ready**, y cuando corresponda lea el nuevo dato de temperatura, y lo guarde en tres variables en memoria: una para parte fraccionaria, y dos para la parte entera, separando en decenas y unidades. Para la separación en decenas y unidades de la parte entera se dispone de una tabla en ROM, a partir de la dirección **TAB_BCD** alineada a una página de 256. En esta tabla se guardan las unidades y decenas de los números del 0 al 63 codificados en BCD empaquetado, es decir el valor de unidades en los 4 bits menos significativos y el valor de decenas en los 4 bits más significativos.

Además, con el dato de la parte fraccionaria se deben encender los leds según corresponda. Para esto se sugiere utilizar una segunda tabla en ROM de cuatro lugares a partir de la dirección **TAB_LEDS** para convertir el valor de dos bits de la parte fraccionaria en un valor de tres bits para manejar los leds.

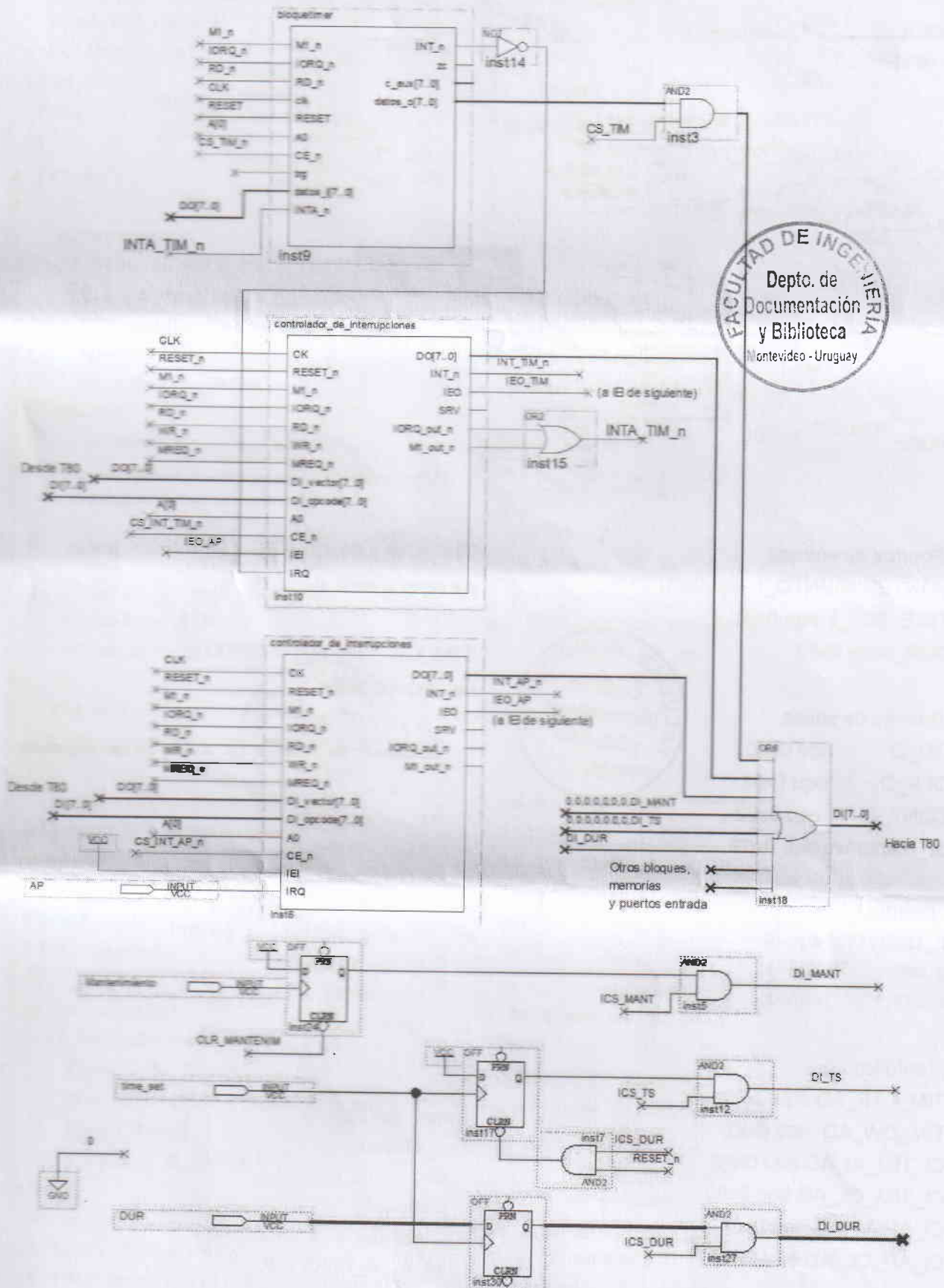
Se dispone de una señal **TIC**, de frecuencia adecuada para generar la ilusión óptica de que ambos dígitos del display están encendidos simultáneamente. Cada vez que se reciba una interrupción se deberá cambiar el display que se encuentra encendido, con el dato adecuado. Considerar ya implementada una rutina **bcd27seg**, que recibe en el acumulador un dígito decimal codificado en binario y devuelve en el acumulador el mismo dígito codificado en 7 segmentos.

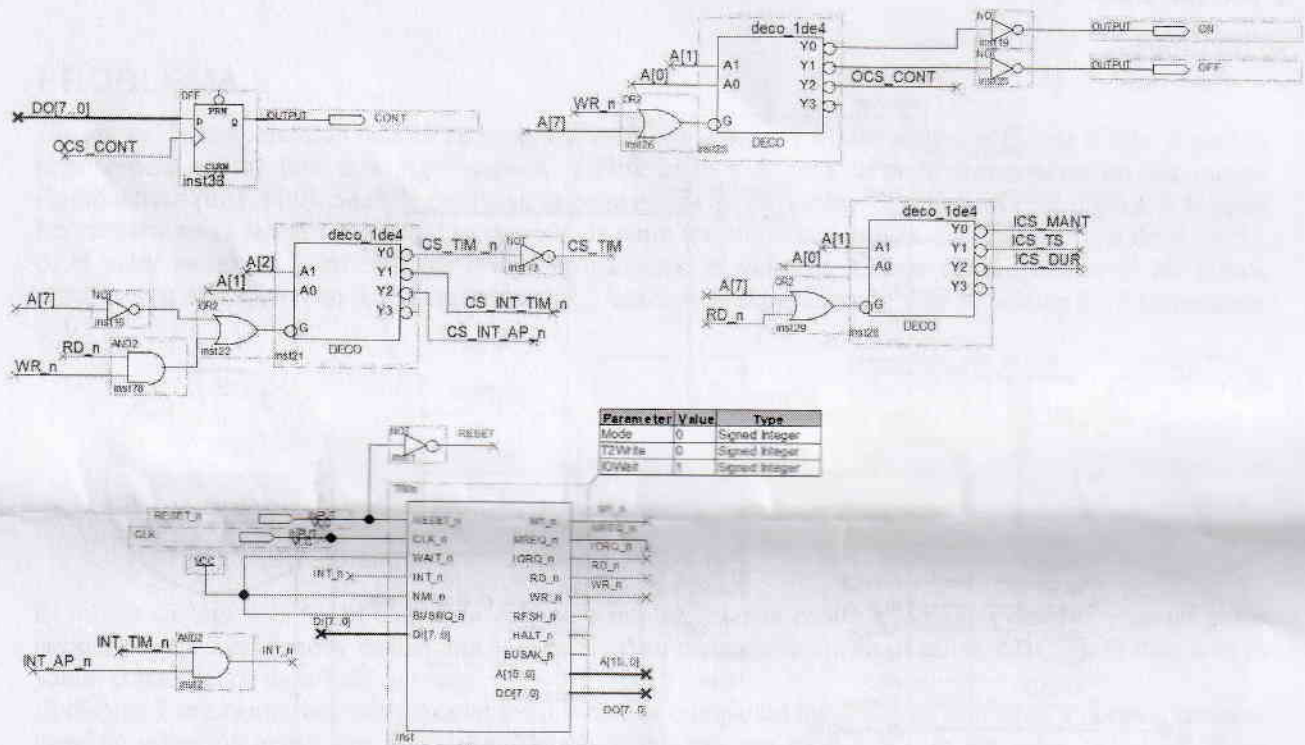
Se pide:

- Todo el hardware necesario para implementar el sistema. Se dispone de chips de memoria RAM y ROM de 32 kB.
- Rutina de atención a interrupción para controlar el display.
- Programa principal que hace el polling de **ready**, recibe el dato y lo guarda apropiadamente en memoria y actualiza el valor de los leds.
- Inicialización de todo el sistema, incluyendo reservas de memoria para todas las variables y tablas utilizadas y la ubicación en memoria de todo el código. Para la tabla **TAB_LEDS** indicar el contenido completo, para **TAB_BCD** indicar solamente los primeros lugares.

PROBLEMA 1

a) Hardware





;Puertos de entrada
MANTENIMIENTO_I equ 0x00
TIME_SET_I equ 0x01
DUR_I equ 0x02

;Puertos de salida
ON_O equ 0x00
OFF_O equ 0x01
CONT_O equ 0x02
clr_mantenim equ 0x03

; estados
E_INACTIVO equ 0
E_ON equ 1
E_OFF equ 2

;Periféricos
TIM_CTE_AD equ 0x80
TIM_CW_AD equ 0x81
CI_TIM_VI_AD equ 0x82
CI_TIM_CL_AD equ 0x83
CI_AP_VI_AD equ 0x84
CI_AP_CL_AD equ 0x85

TIM_CW_ON equ 10101111; int habilitadas,
sw reset y prescaler = 15

TIM_CW_OFF equ 000000000;
int deshabilitadas

b) Rutinas de atención a la interrupción

```
; isr_timer{
;   ei
;   preservar
;   estado ← E_INACTIVO
;   Flanco de subida en OFF
;   Apago Timer
;   restaurar
;   reti
; }
```

```
org 0x1000
isr_timer:
    ei
    push AF
;   estado ← E_INACTIVO
    ld A, E_INACTIVO
```

```
ld (estado), A
; Flanco de subida en OFF
out (OFF_O), A
; Apago Timer
ld A, TIM_CW_OFF
out (TIM_CW_AD), A
pop AF
reti

; isr_ap{
; ei
; preservar
; if (estado == E_ON){
;   Flanco de subida en OFF
;   dur_pendiente ← Cuenta timer
;   Apago Timer
;   estado ← E_PAUSA
; }else{
;   if (estado == E_PAUSA){
;     Arranco Timer con dur_pendiente
;   }else{
;     Arranco Timer con dur_vigente
;   }
;   Flanco de subida en ON
;   estado ← E_ON
;   Cuenta += 1
; }
; restaurar
; reti
; }
```



```
org 0x1100
isr_ap:
ei
push AF
ld A, (estado)
cp E_ON
jr nz, encender
; if (estado == E_ON){
;   Flanco de subida en OFF
;   dur_pendiente ← Cuenta timer
;   Apago Timer
;   estado ← E_PAUSA
; }
apagar:
out (OFF_O), A
in A, (TIM_CTE_AD)
ld (dur_pendiente), A
```

```
ld A, TIM_CW_OFF
out (TIM_CW_AD), A
ld A, E_PAUSA
ld (estado), A
jr fin

encender:
; }else{
;   if (estado == E_PAUSA){
;     Arranco Timer con dur_pendiente
;   }else{
;     Arranco Timer con dur_vigente
;   }
; }

ld a, (estado)
cp E_PAUSA
jr nz, else_e_inactivo
ld a, (dur_pendiente)
jr fin_si

else_e_inactivo:
ld a, (dur_vigente)
fin_si:
out (), A
ld A, TIM_CW_ON
out (), A
; Flanco de subida en ON
; estado ← E_ON
; Cuenta += 1
out (ON_O), A
ld A, E_ON
ld (estado), A
ld A, (Cuenta)
inc A
ld (Cuenta), A
```

```
fin:
pop AF
reti

c) Programa principal
; ppal:
; en loop infinito{
;   if (time_set){
;     clr_FF_mantenim
;     dur_vigente ← DUR
;   }
;   if (Mantenim) then{
;     clr_FF_mantenim
```

```
;      CONT ← Cuenta
;      Cuenta ← 0
;      }
;      }
```

```
Cuenta: db 0
estado: db 0
dur_vigente: db 0
dur_pendiente: db 0
```

```
org 0x0100
ppal:
    in A, (TIME_SET_I)
    bit 0, a
    jr z, check_mantenimiento
;    if (time_set){
;        clr_FF_mantenim
;        dur_vigente ← DUR
    in A, (DUR_I) ; borra FF
    ld (dur_vigente), A
check_mantenimiento:
    in A, (MANTENIMIENTO_I)
    bit 0, a
    jr z, ppal
;    if (Mantenim) then{
;        clr_FF_mantenim
;        CONT ← Cuenta
;        Cuenta ← 0
;    }
    out (clr_mantenim), a
    ld A, (Cuenta)
    out (CONT_O), A
    ld A, 0
    ld (Cuenta), A
    jp ppal
```

**d) Inicialización + Reservas
de memorias**

VI_TIM equ 0x00

VI_AP equ 0x02

.data ; 0x8000, comienzo RAM

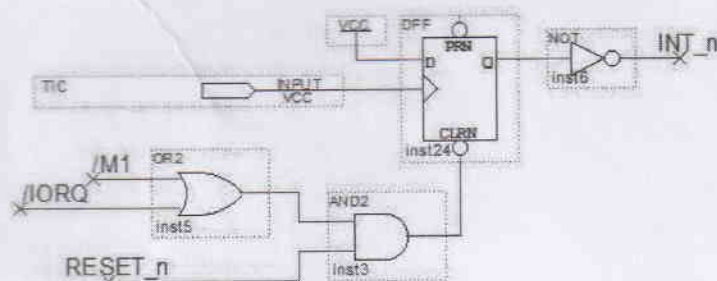
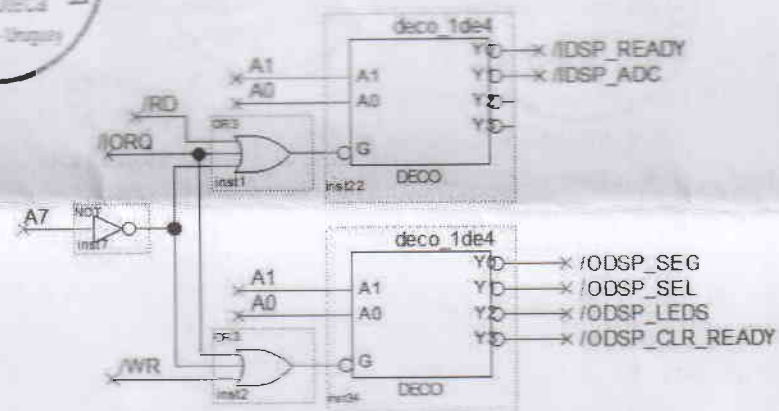
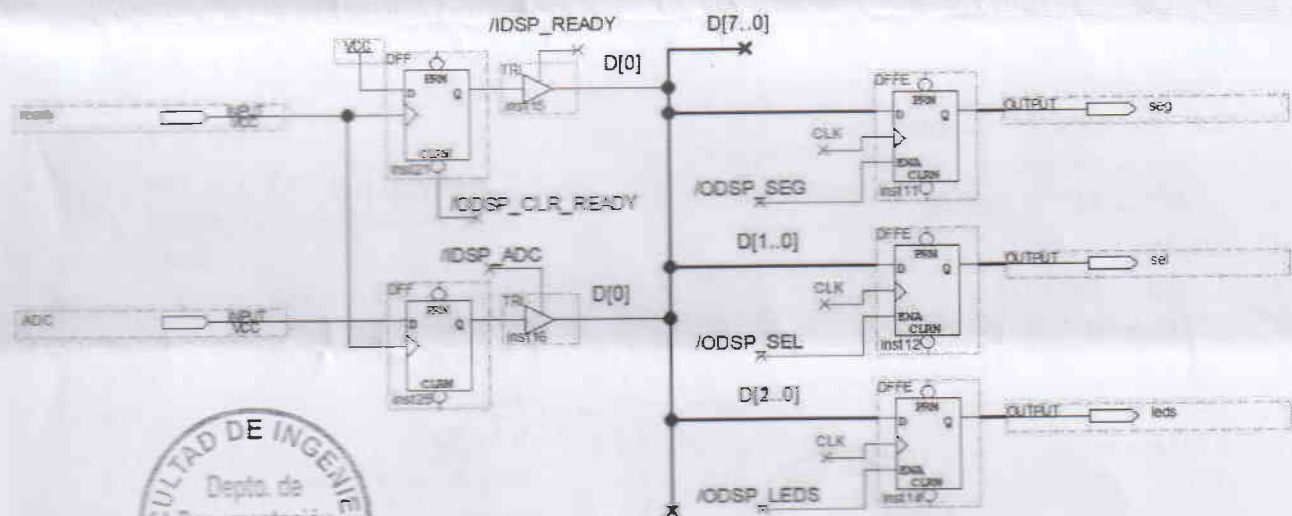
```
org 0x2000
TABLA_INT:
    dw isr_timer
    dw isr_ap

; Inicialización
    .text ; org 0x0000, comienzo ROM
; Stack Pointer
    ld SP, 0
; Modo 2
    im 2
    ld HL, TABLA_INT
    ld A, H
    ld I, A
    Variables
    ld A, 0
    ld (Cuenta), A
    ld A, E_INACTIVO
    ld (estado), A
    ld A, 200
    ld (dur_vigente), A

; Periféricos
    ld A, VI_TIM
    out(CI_TIM_VI_AD), A
    out(CI_TIM_CL_AD), A
    ld A, VI_AP
    out(CI_AP_VI_AD), A
    out(CI_AP_CL_AD), A

; Habilitar interrupciones
    ei
    jp ppal
```





```
;Puertos de entrada
READY_I equ 0x80
ADC_SET_I equ 0x81
```

```
;Puertos de salida
SEG_O equ 0x80
SEL_O equ 0x81
LEDS_O equ 0x82
CLR_READY_O equ 0x83
```

b) Rutina de atención a la interrupción

```
; isr_ctrl_display{
; ei
; Preservar
; if (Bandera == disp0) then{
;     seg ← Unidades
;     sel ← bit 0 (0x01)
; }else{
;     seg ← Decenas
;     sel ← bit 1 (0x02)
; }
; Complementar Bandera
; Restaurar
; ret
```

```
org 0x0038
isr_ctrl_display:
    ei
    push AF
    ld A, (Bandera)
    cp 0
    jr NZ, disp1
; if (Bandera == disp0) then{
;     seg ← Unidades
;     sel ← bit 0 (0x01)
; }
disp0:
    ld A, (Unidades)
    call bcd27seg
    out (SEG_O), A
    ld A, 0x01
    OUT (SEL_O), A
    jp fin_isr
; else{
```

```
;     seg ← Decenas
;     sel ← 1
; }
disp1:
    ld A, (Decenas)
    call bcd27seg
    out (SEG_O), A
    ld A, 0x02
    OUT (SEL_O), A
fin_isr:
; Complementar Bandera
    ld A, (Bandera)
    cpl
    ld (Bandera), A
    pop AF
    ret
```

c) Programa principal

```
; ppal{
; en loop infinito{
;     if (ready == 1){
;         ready ← 0
;         A ← ADC
;         SRL A x2 ; Parte entera en A
;         Decenas ← (TAB_BCD + A)_high
;         Unidades ← (TAB_BCD + A)_low
;         A ← ADC
;         AND 3 ; Parte fraccionaria en A
;         Parte_frac ← (TAB_LEDS + A)
;         Actualizar leds
;     }
; }
; }
```

```
org 0x0100
ppal:
    in A, (READY_I)
    cp 0
    jr Z, ppal
;     ready ← 0
    out (CLR_READY_O), A
parte_entera:
;     A ← ADC
;     SRL A x2 ; Parte entera en A
```



```

IN A, (ADC)                db 00000000
SRL A                      db 00000001
SRL A ; Parte entera en A
ld HL, TAB_BCD
ld L, A
; Decenas ← (TAB_BCD + A)_high    org 0x2100
ld A, (HL)
SRL A
SRL A
SRL A
SRL A
ld (Decenas), A
; Unidades ← (TAB_BCD + A)_low    org 0x2200
ld A, (HL)
AND F
ld (Unidades), A
parte_fraccionaria:
; A ← ADC
; AND 3 ; Parte fraccionaria en A
; Parte_frac ← (TAB_LEDS + A)
in A, (ADC)
AND 3 ; Parte fraccionaria en A
ld HL, TAB_LEDS
ld L, A
ld A, (HL)
ld (Parte_frac), A
; Actualizar leds
out (LEDS_O), A
jp ppal

; Inicialización
; Stack Pointer
ld SP, 0

; Modo 1
im 1

; Variables
ld A, 0
ld (Bandera), A
; empiezo mostrando 00
ld (Unidades), A
ld (Decenas), A

; Handshake ready
out (CLR_READY_O), A

; Habilitar interrupciones
ei

org 0x2000
TAB_BCD:

```

