

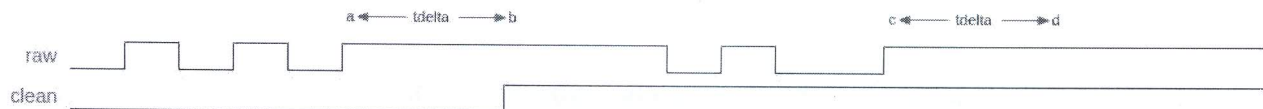
- Mínimo para aprobar: un problema
- Nombre y CI en cada hoja
- Numere las hojas, indique el total en la primera

- Utilice sólo un lado de las hojas
- Incluya un solo problema por hoja
- Sea prolijo

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

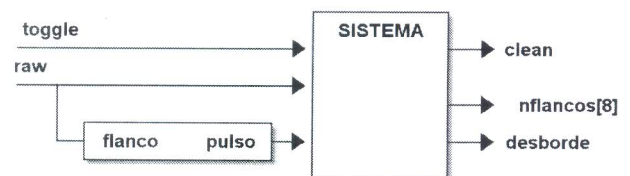
PROBLEMA 1

Un sistema basado en T80 recibe como entrada la señal **raw** (de un bit) que presenta rebotes y ruido, y debe generar como salida la señal **clean** libre de rebotes y pulsos espurios. Un cambio en la entrada se traslada a la salida **clean** recién cuando el nuevo valor se mantuvo sin cambiar durante un tiempo **tdelta**.



Asimismo, y a efectos de recabar información para ajustar el valor de **tdelta**, durante el proceso de validación de cada cambio debe registrarse la cantidad de transiciones en la entrada **raw**. Al final de cada validación debe desplegarse dicho valor en el puerto de salida de 8 bits **nflancos**, y en la salida **desborde** de un bit si hubo (=1) o no hubo (=0) desborde en el valor desplegado en **nflancos**.

Se dispone de un dispositivo que cada vez que hay una transición de subida o bajada en su entrada **flanco** genera en su salida **pulso** un pulso a 1 de un período de reloj. El resto del tiempo **pulso** vale 0. La señal **pulso** debe usarse para interrumpir al procesador.



Se usará un bloque Timer que debe reprogramarse cada vez que hay una transición para que genere una interrupción un tiempo **tdelta** después. Esta interrupción debe encargarse de trasladar el cambio de **raw** a la salida **clean** si corresponde.

Todo el mecanismo del eliminador de rebotes puede estar habilitado o deshabilitado. Cuando el mecanismo está deshabilitado la salida **clean** debe valer 0. Luego de un reset, comienza deshabilitado. Un pulso de corta duración en la entrada **toggle** debe habilitarlo si está deshabilitado y viceversa. El valor inicial de **clean** cuando se habilita debe copiarse del valor actual de **raw**.

La frecuencia del reloj del sistema es **fclk**.

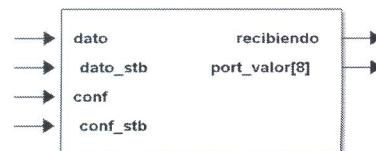
El tiempo $tdelta = 250 * (1/fclk * 2^{12})$. Se puede asumir que el tiempo entre transiciones debidas a rebote o ruido es mayor que la duración máxima de las interrupciones y menor que **tdelta** y que el tiempo transcurrido entre dos transiciones completas es mucho mayor que **tdelta** por lo que nunca llegan en forma simultánea las interrupciones de Timer y pulso.

Se pide todo lo necesario para agregar la funcionalidad del eliminador de rebotes a un sistema existente basado en T80. El sistema ya cuenta con 32KB de RAM y 32KB de ROM, otros puertos que ocupan la mitad inferior del espacio de E/S y otros dispositivos que interrumpen en modo 2 y ocupan las tres primeras entradas de la tabla de interrupciones. La interrupción de Timer debe ser la de más prioridad, seguida por la interrupción de pulso y finalmente las otras tres.

- Hardware para las interrupciones y puertos que se deben agregar.
- Rutinas de atención a interrupciones asociadas a una transición en la entrada **raw** (`isr_pulso`) y a la expiración del bloque Timer (`isr_timer`).
- Inicialización del sistema. Para inicializar las funcionalidades no descritas aquí se invocará la subrutina **init_otros** que se supone implementada.
- Loop de programa principal que en cada iteración habilite o deshabilite el mecanismo de eliminación de rebotes si corresponde e invoque a la subrutina **atiendo_otros** (que se supondrá implementada) para atender a otros dispositivos.

PROBLEMA 2

Un sistema basado en Z80 recibe un bit de información por la señal **dato** con cada flanco de subida de la señal **dato_stb**. El valor de **dato** solamente es válido en el instante del flanco de subida de **dato_stb**. La información se recibe en palabras que representan un número con signo de acuerdo a la codificación que se describe a continuación.



	k		valor
1111	4	k-1	3
1110	3	k-1	2
110	2	k-1	1
10	1	k-1	0
01	1	-k	-1
001	2	-k	-2
0001	3	-k	-3
0000	4	-k	-4

Una secuencia de **k** bits consecutivos del mismo valor representa un valor igual a **-k** si los bits son 0 y **k-1** si los bits son 1. La cantidad de bits **k** puede variar entre 1 y **N** ($1 \leq k \leq N$). La palabra termina si se reciben **N** bits consecutivos iguales ($k=N$), o si se recibe un bit con polaridad opuesta a los anteriores (si $k < N$). Una vez completada una palabra, el próximo bit recibido será el comienzo de la siguiente palabra. Notar que la palabra completa puede tener entre 2 y **N** bits. La tabla muestra todos los valores posibles para $N=4$.

El sistema usa dos posibles anchos de palabra: $N=4$ o $N=8$. Se puede cambiar la configuración de ancho de palabra en cualquier momento a través de las señales **conf** y **conf_stb**. Un flanco de subida en **conf_stb** indica que debe leerse un nuevo valor en la señal **conf** (**conf** es 0 para indicar $N=4$ y 1 para indicar $N=8$). La nueva configuración no debe afectar una recepción que ya esté en curso. Luego de un reset el sistema

debe iniciar usando $N=4$.

El puerto de salida **recibiendo** debe valer 1 mientras se está recibiendo una palabra (desde que se recibe el primer bit hasta que se recibe el último) y 0 el resto del tiempo.

Se desea agregar al sistema la funcionalidad necesaria para recibir estos datos. Para eso cada flanco de subida de **dato_stb** solicitará una interrupción en modo 1 cuya rutina de atención debe estructurarse de acuerdo al pseudocódigo del recuadro. Observar que se almacena el valor del primer bit recibido, se lleva la cuenta de la cantidad de bits **k** y, cuando se detecta la condición de terminación (se llega a **N** bits o bit recibido diferente de primer bit), se calcula el valor **k-1** o **-k** según corresponda. Dicho valor debe escribirse en el puerto de salida **port_valor** y en la variable **var_valor** en formato complemento a dos de 8 bits. Además de lo indicado en el pseudocódigo, **isr_stb** debe manejar en forma adecuada el puerto de salida **recibiendo**.

Se pide:

- Hardware, incluyendo 16kB de ROM y 16kB de RAM (se debe dejar libre la mitad superior del espacio de memoria para futuras ampliaciones) y todo el hardware necesario para la funcionalidad descripta mapeado en el rango de direcciones 0x40-0x4F de E/S.
- Código assembler de la rutina **isr_stb** de acuerdo al pseudocódigo del recuadro que cada vez que se completa una palabra actualice en **port_valor** y **var_valor** el valor recibido.
- Inicialización del sistema incluyendo las reservas de memoria necesarias. Indicar la ubicación en memoria de todo el código, variables y constantes utilizadas. Se dispone de la subrutina **init_otros** para inicializar otras funciones del sistema.
- Loop de programa principal que en cada iteración actualiza la configuración si es necesario e invoca a la subrutina **atiendo_otros** para atender otras funciones del sistema.

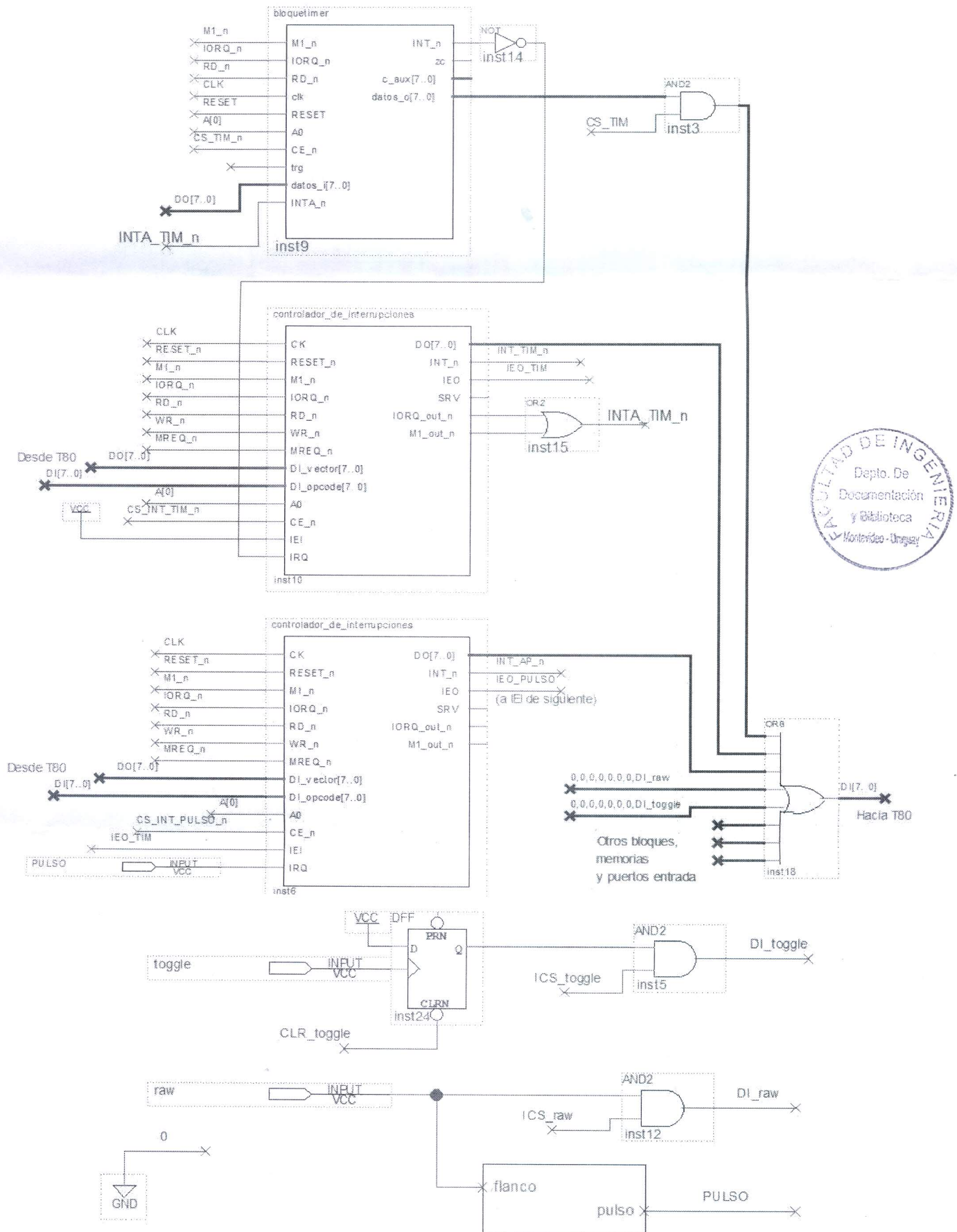
```

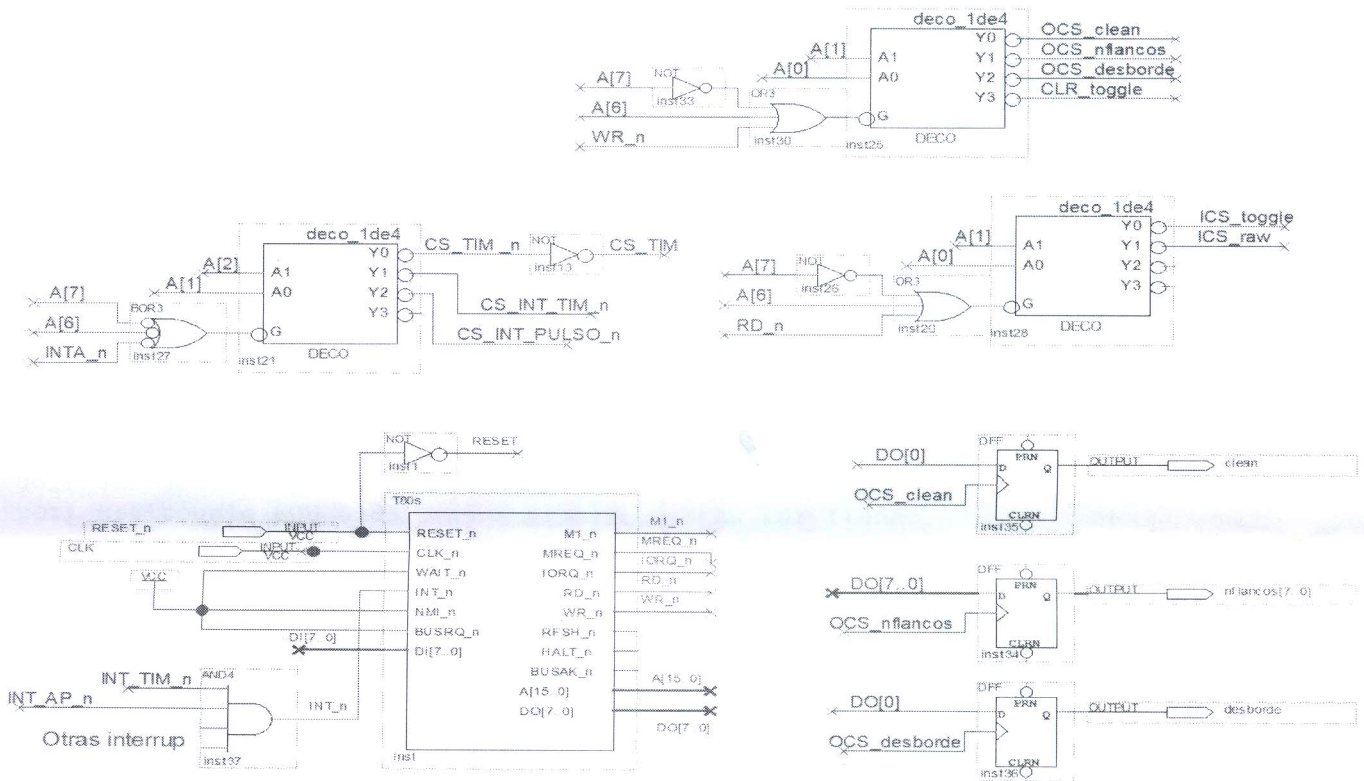
isr_stb:
si (k==0){
    primer_bit=dato
}
si (dato==primer_bit){
    k++
}
si (condic_termina){
    si (primer_bit==0){
        valor = -k
    }else{
        valor = k-1
    }
    k=0
}

```

PROBLEMA 1

a) Hardware





b) Rutinas interrupción *****

```

isr_pulso:
; if(habilitado == 1)
;
;   reinicio timer
;   if(nflancos == 255)
;       desborde = 1
;   nflancos++

PUSH AF
PUSH BC
EI
LD A, (habilitado)
CP 1
JP NZ, no_habilitado

LD A, CW_TIMER
OUT (TIMER_A1), A
LD A, (nflancos)
CP 255
JP NZ, no_es_255
    ;;; bit 0 de 255 vale 1
LD (desborde), A
no_es_255:
INC A
LD (nflancos), A

no_habilitado:
POP BC
POP AF
RETI

```

```

isr_timer:
; preservo registros
; detengo timer
; salida = entrada
; muestro datos

```

; reinicio memorias

```

;
PUSH AF
PUSH BC
EI
LD A, CW_DI
OUT (TIMER_A1), A
IN A, (port_raw)
OUT (port_clean), A
LD A, (nflancos)
OUT (port_nflancos), A
LD A, (desborde)
OUT (port_desborde), A
LD A, 0
LD (nflancos), A
LD (desborde), A
POP BC
POP AF
RETI

```

c) inicializacion *****

```

; cte timer
; int hab, X, swreset, nottrigger, pre=12
CW_TIMER EQU 10101100B
; int apagada 0XXXXXXX
CW_DI EQU 00101100B
CTE_TIMER EQU 249

```

```

; vectores 6 y 8
; tres primeras entradas ocupadas
VI_FLANCO EQU 0x06
VI_TIM EQU 0x08

```

```

; cte timer
; int hab, X, sw reset 1, no trigger, pre=12)
CW_TIMER EQU 10101100B

```



FACULTAD DE INGENIERÍA UNIVERSIDAD DE LA REPÚBLICA

```
;;; int apagada 0XXXXXXX
CW_DI EQU 00101100B
CTE_TIMER EQU 249
```

```
;; vectores 6 y 8
;; tres primeras entradas ocupadas
VI_FLANCO EQU 0x06
VI_TIM EQU 0x08
```

```
;Puerto entrada
port_toggle EQU 0x80
port_raw EQU 0x81
```

```
;Puerto salida
port_clean EQU 0x80
port_nflancos EQU 0x81
port_desborde EQU 0x82
port_clr_toggle EQU 0x83
```

```
;Perifericos
TIMER_A0 equ 0xC0
TIMER_A1 equ 0xC1
CI_TIMER_A0 EQU 0xC2
CI_TIMER_A1 EQU 0xC3
CI_FLANCO_A0 EQU 0xC4
CI_FLANCO_A1 EQU 0xC5
```

```
;; inicializacion
.ORG 0x0000
```

```
LD SP, 0
IM 2
LD HL, tabla_int
LD L, A
LD I, A
```

```
; vector de timer
LD A, VI_TIM
OUT (CI_TIMER_A0), A
OUT (CI_TIMER_A1), A

;vector de flanco
LD A, VI_FLANCO
OUT (CI_FLANCO_A0), A
OUT (CI_FLANCO_A1), A
```

```
;carga constante timer, el control lo
carga en la int del flanco
LD A, CTE_TIMER
OUT (TIMER_A0), A
```

```
;borro puertos
IN A, (port_clr_toggle)
LD A, 0
OUT (port_nflancos), A
```

INTRODUCCIÓN A LOS MICROPROCESADORES

Diciembre 2023

```
OUT (port_desborde), A
```

```
; inicializo variables
LD A, 0
LD (nflancos), A
LD (desborde), A
LD (habilitado), A
CALL init_otros
EI
JP programa_ppal
```

```
;tabla int
.ORG 0x2000
tabla_int:
dw rutin_1
dw rutin_2
dw rutin_3
dw isr_flanco
dw isr_timer
```

```
;variables
.ORG 0x8000

nflancos: DB 0
desborde: DB 0
habilitado: DB 0
```

d) Programa principal

```
programa_ppal:
CALL atiendo_otros
leo_toggle:
IN A, (port_toggle)
bit 0,a
JP NZ, fin_principal

LD A, (habilitado)
CP 1
JP NZ, esta_habilitado
;; habilitado == 0
LD A, 0
LD (habilitado), A
IN A, (port_raw)
OUT (port_clean), A
JP fin_principal
```

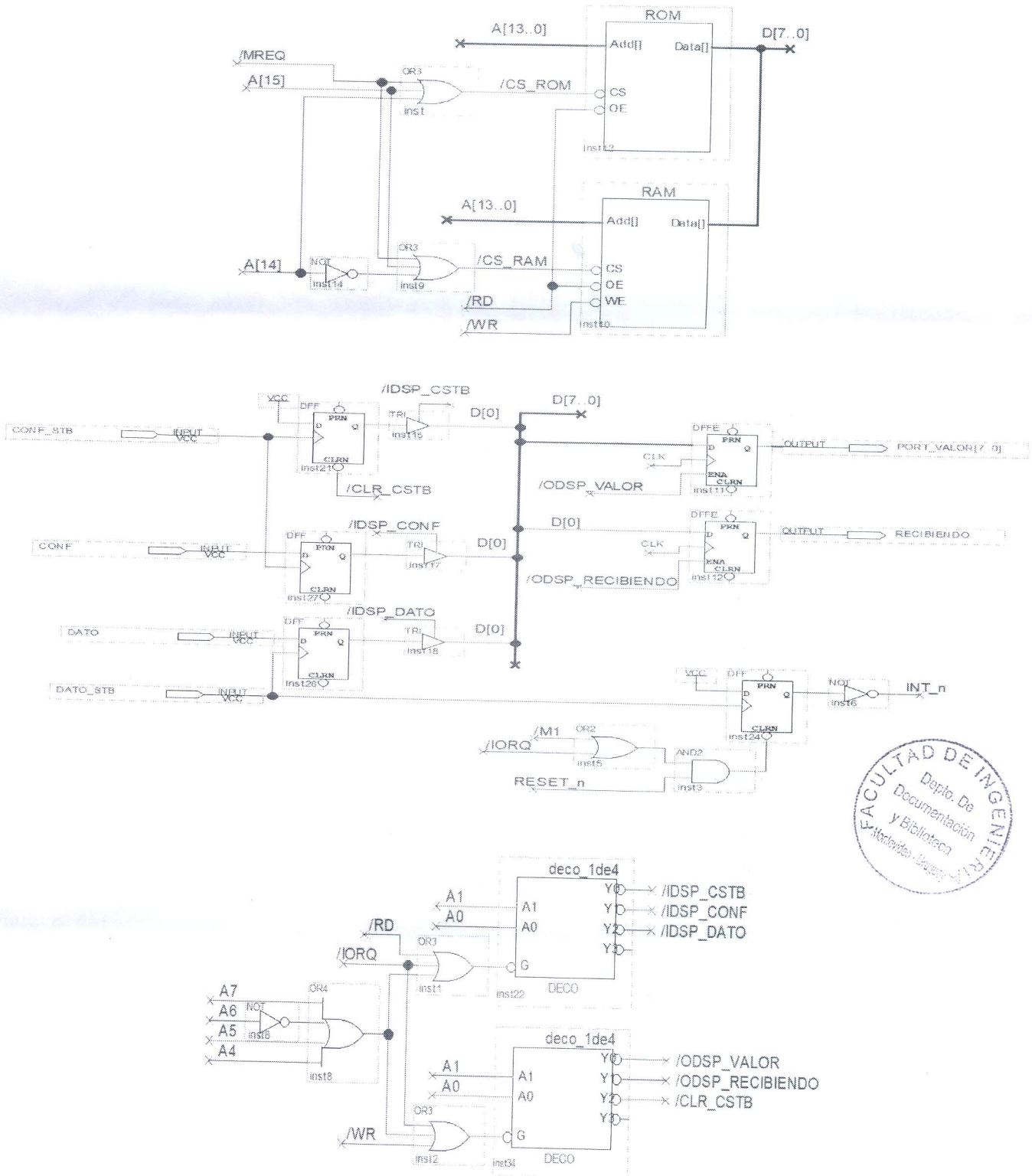
```
esta_habilitado:
LD A, 0
LD (habilitado), A
LD A, 0
OUT (port_clean), A
LD A, CW_DI
OUT (TIMER_A1), A
```

```
fin_principal:
out (port_clr_toggle), A
JP programa_ppal
```



PROBLEMA 2

Hardware



```
;; Etiquetas
conf_stb equ 0x40
conf      equ 0x41
dato      equ 0x42
port_valor equ 0x40
recibiendo equ 0x41
clr_conf  equ 0x42
```

b) Rutina de interrupción

```
org 0x0038
isr_stb:
    push af
    push bc
    ld a, (k)
    cp 0
    jr nz, cont
primera_vez:
    in a, (dato)
    ld (primer_bit), a
    ld a, 1
    out (recibiendo), a
    ;; Empieza la recepción
    ld a, (n)
    ;; Actualizo n_en_uso
    ld (n_en_uso), a
cont:
    ld a, (primer_bit)
    ld b, a
    in a, (dato)
    cp b
    jr nz, fin_trans
nuevo_dato:
    ld a, (k)
    inc a
    ld (k), a
    ld b, a
    ld a, (n_en_uso)
    cp b
    jr nz, fin_isr_1
fin_trans:
    ld a, (primer_bit)
    cp 0
    jr nz, es_1
es_0:
    ld a, (k)
    neg
```

```
jr fin_isr_2
es_1:
    ld a, (k)
    dec a
fin_isr_2:
    out (port_valor), a
    ld (var_valor), a
    ld a, 0
    ld (k), a
    ;; Termina la recepción
    out (recibiendo), a
fin_isr_1:
    pop bc
    pop af
    ei
    ret
```

c) ;; Inicialización, directivas

```
org 0x0000
;init_otros
    call init_otros
;modo 1
    im 1
    ld a, 0
;puertos
    out (recibiendo), a
    out (clr_conf), a
;variables
    ld (k), a
    ld a, 4
    ld (n), a
;habilito interrupciones
    ei
    jp ppal

;; Variables
    org 0x4000
k:      db 0
n:      db 0
n_en_uso: db 0
var_valor: db 0
primer_bit: db 0
```



d) ;; Programa principal

```
org 0x0100
ppal:
    call atiando_otros
    in a, (conf_stb)
    cp 1
    jr nz, ppal
    in a, (conf) ;; conf borra conf_stb
    cp 0
```

```
jr nz, n_es_8
n_es_4:
    ld a, 4
    ld (n), a
    jr ppal
n_es_8:
    ld a, 8
    ld (n), a
    jr ppal
```

