

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

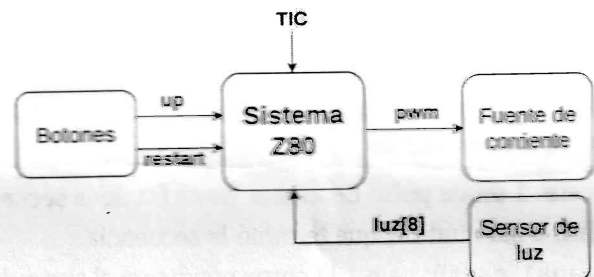


- Utilice sólo un lado de las hojas
- Incluya un solo problema por hoja
- Sea prolijo

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

PROBLEMA 1

Se desea diseñar un circuito de control para una luminaria LED, basado en un sistema con un Z80. El sistema cuenta con dos botones que permiten manejar la luz, un sensor de luz ambiental para diferenciar el funcionamiento entre el día y la noche, y una salida PWM que controla una fuente de corriente.



La señal **pwm** es una secuencia de pulsos de amplitud y período fijo **T_{pwm}**, donde se varía el tiempo que la señal está en alto (**T_{on}**) para controlar la corriente enviada a la luz LED. El ciclo de trabajo es el cociente entre **T_{on}** y **T_{pwm}**, y determina el valor medio de la corriente y por tanto la luminosidad de la lámpara. Por ejemplo, si está en alto durante medio período, el ciclo de trabajo es 50%.

El tiempo de encendido es **T_{on} = N_{on} x 5 us**, donde **N_{on}** toma valores entre 0 y 10. Por lo tanto se puede controlar **T_{on}** entre 0 us y 50 us en pasos de 5 us (ciclo de trabajo entre 0 y 100% en pasos de 10%). El período es **T_{pwm} = 50 us**.

El sistema tiene dos modos de funcionamiento según el nivel de luz ambiente. Cuando se ingresa al *modo noche* la luminaria se enciende con **T_{on} = N_{noche} x 5us**; cuando se ingresa al *modo día* la luminaria se apaga (**T_{on} = 0**).

Por otro lado, se cuenta con dos botones que permiten cambiar la intensidad de la luz. En cualquiera de los modos, con cada pulso del botón **up** se debe aumentar **T_{on}** en un paso (saturando en **T_{pwm}**). Al pulsar el botón **restart** se debe volver al **T_{on}** inicial que corresponda en cada modo.

El sensor de luz ambiental suministra la señal **luz[8]** que está siempre disponible. Si **luz[8]** está por debajo del valor **UMBRAL** el sistema funciona en *modo noche*, y en caso contrario en *modo día*.

El valor de luz **UMBRAL** y el valor **N_{noche}** se encuentran almacenados en ROM en las direcciones **BASE+0** y **BASE+1** respectivamente.

Como base de tiempo para la generación de la señal **pwm** se usará la señal periódica **TIC**, que tiene un flanco de bajada cada 5 us y deberá interrumpir al T80. La rutina de atención a interrupciones deberá encargarse de generar la salida **pwm** que corresponda en cada momento; los cambios en **T_{on}** se deben actualizar al fin de cada período de **pwm**. La rutina debe estructurarse de acuerdo al pseudocódigo del recuadro.

El programa principal deberá monitorear los botones y el valor del sensor para determinar el valor de **T_{on}** correcto y almacenar los valores que necesite la rutina de atención a interrupciones para su funcionamiento. Se pide:

- Hardware del sistema, 16K de ROM y 16K de RAM.
- Rutina de atención a interrupciones encargada de generar la señal **pwm**.
- Inicialización del sistema, reserva de memoria (incluyendo datos a partir de **BASE**) y programa principal.

```

si (cont == N_pwm) {
    cont=0
    Actualizar N_on
}
si (cont < N_on) {
    pwm=1
} else {
    pwm=0
}
cont++
  
```



PROBLEMA 2

Se desea montar un sistema con T80 para registrar la llegada de una carrera entre dos competidores que llegan parejos a la meta. El sistema debe discernir cuál de los dos ganó y cuánto tiempo después el segundo cruzó la meta. Los sensores s_0 y s_1 generan un pulso a 1 de corta duración cuando el competidor correspondiente pasa por la línea de llegada. Con un pulso en la entrada **armar** el sistema debe quedar listo para registrar la llegada. Luego, con la llegada del primer competidor debe comenzar a medir el tiempo. La secuencia termina o bien con la llegada del segundo competidor, o bien porque transcurre un tiempo **TOUT**. En el segundo caso la medida del tiempo se realiza por otro medio menos preciso. El sistema debe manejar las siguientes salidas:

- **listo**: 1 desde pulso de **armar** hasta fin de la secuencia, 0 en otro caso.
- **fin**: 1 para indicar que terminó la secuencia.
- **gana1, gana0**: vale 1 la correspondiente al vencedor.
- **ventaja[8]**: centésimas de segundo entre la llegada de los dos competidores, 0xFF en caso de timeout.

Las valores de **fin**, **gana1**, **gana0** y **ventaja** se deben establecer al fin de cada secuencia para mostrar el resultado y volver a 0 con el armado de la siguiente.

Se dispone del circuito de la figura. Notar que si los FF empiezan en 0:

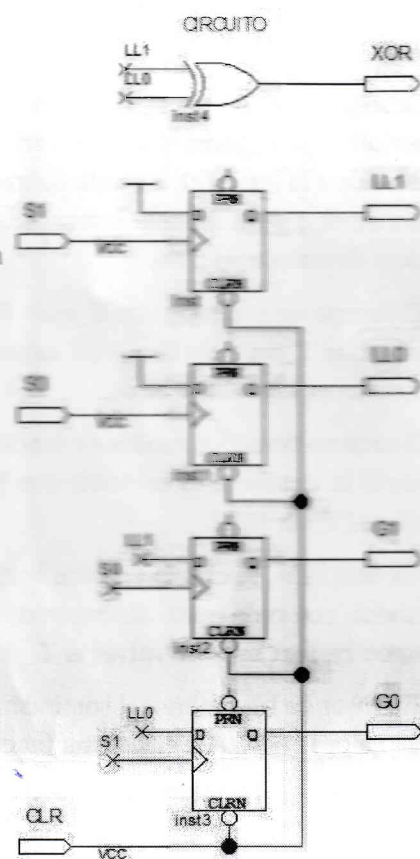
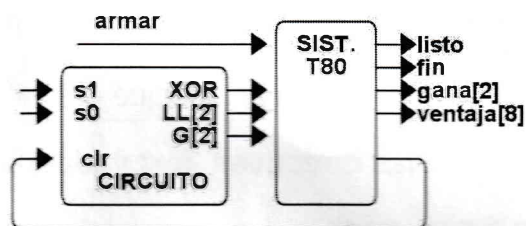
- la señal **XOR** sube a 1 cuando llega el primero de los competidores ($LL1..0 = 01$ o 10) y vuelve 0 cuando llega el segundo ($LL1..0 = 11$)
- **G1** captura el valor de **LL1** en el flanco de subida de s_0 . Luego que llegan ambos competidores se puede saber si el competidor 1 llegó antes ($G1=1$) o después que el 0.

Para esta función se utilizan 2 interrupciones: **Timer** y flanco de bajada de **XOR**. El bloque **Timer** se debe configurar para medir el tiempo a partir del flanco de subida de **XOR** (llegada del primero). El **Timer** debe interrumpir si transcurre un tiempo **TOUT** antes de la bajada de **XOR** (llegada del segundo competidor). En este caso la ventaja es 0xFF y el ganador se obtiene leyendo **LL1..0**. Si **XOR** interrumpe antes que el **Timer** (llegada pareja, diferencia < TOUT), la ventaja se deduce del valor del **Timer** y el ganador se identifica leyendo **G1..0**. En cualquiera de los dos casos una vez terminada la secuencia y hasta que comience una nueva, el **Timer** debe dejar de interrumpir y las interrupciones por flanco de bajada de **XOR** no deben producir ningún efecto.

El programa principal realiza además otras tareas, y cuando quiere iniciar una secuencia invoca a la subrutina **sub_armar()** que debe esperar un pulso en **armar**, configurar todo lo necesario para la medida (**timer**, **circuito**, etc.), subir la salida **listo** y retornar. El programa principal retoma sus tareas y el resto de la secuencia debe completarse mediante las interrupciones descritas arriba trabajando en modo 2. Las interrupciones de **Timer** y bajada de **XOR** serán las de mayor prioridad en ese orden, hay otras dos interrupciones ocupando los dos primeros lugares de la tabla. Se pide:

- Todo el hardware necesario para las funcionalidades descritas arriba. Se debe usar para eso el rango 0x40-0x7F del espacio de E/S. La memoria ya está implementada con 32K de ROM y 32K de RAM.
- Subrutina **sub_armar()**.
- Rutinas de atención a las interrupciones de **timer** y flanco bajada de **XOR**.
- Inicialización y directivas de reservas de memoria.

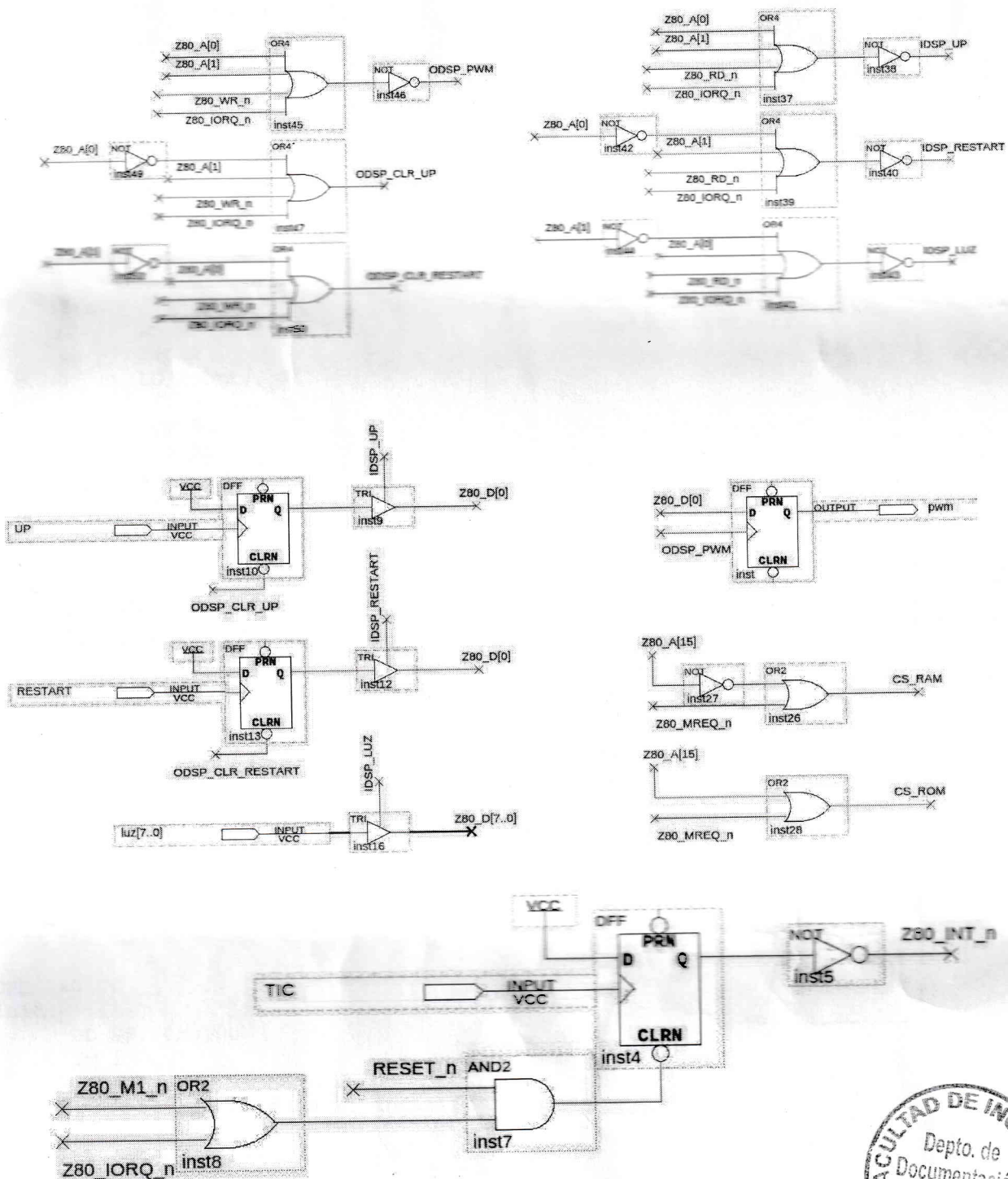
Nota: $TOUT = 100 * 0,01$ seg, $f_{clk} = 3,2768MHz$, $0,01$ seg = $2^{15} * 1/f_{clk}$



2/8

PROBLEMA 1

a) Hardware



b)

ORG 0x0038

isr_tic:

```
PUSH AF
PUSH BC
LD A, (cont)
CP N_pwm
JP NZ, no_es_igual
```

; cargo contador de interrupciones



3/8.

```
LD (cont), A
LD A, (N_on)
LD (N_on_isr), A
```

; actualizo N_on

```
no_es_igual:
LD A, (N_on_isr)
LD B, A
LD A, (cont)
CP B
JP M, es_menor
LD A, 0
OUT (p_pwm), A
JP fin
```

; si cont < N_on, salto

```
es_menor:
LD A, 0x01
OUT (p_pwm), A
```

```
fin:
LD A, (cont)
INC A
LD (cont), A
```

; incremento contador

```
POP BC
POP AF
EI
RETI
```

c)

```
; puertos
p_up EQU 0x00
p_restart EQU 0x01
p_luz EQU 0x02
p_pwm EQU 0x00
p_clr_up EQU 0x01
p_clr_restart EQU 0x02
```

```
N_pwm EQU 10
N_noche EQU 5 ; p. ej.
UMBRAL EQU 40
; para simplificar sw
modo_noche EQU N_noche
modo_dia EQU 0
```

ORG 0x0000

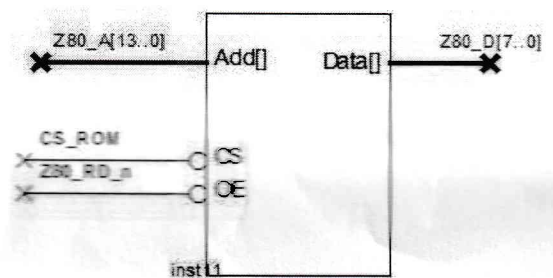
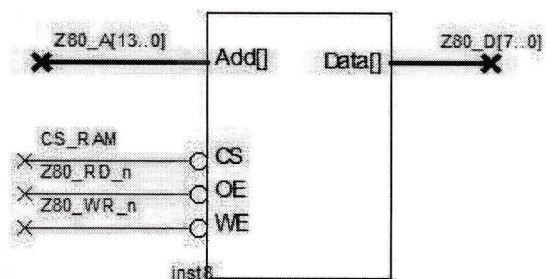
init:

```
LD SP, 0x0000
IM 1
```

```
; luminaria apagada al incializar
LD A, 0
out (p_pwm), A
```

```
; variables
LD A, 0
LD (cont), A
LD (N_on_isr), A
LD (N_on), A
LD (modo), A
EI
```

ppal:



4/8.

CP modo_dia

JP Z, en_modos_dia

en_modos_noches:

; veo si hay que cambiar de modo

IN A, (p_luz)

LD B, A

LD A, (dir_umbral)

CP B

JP NC, check_up

; no cambio, miro botones

LD A, modos_dia

LD (modo), A

LD (N_on), A

JP check_up

en_modos_dia:

; veo si hay que cambiar de modo

IN A, (p_luz)

LD B, A

LD A, (dir_umbral)

CP B

JP C, check_up

; no cambio, miro botones

LD A, modos_noches

LD (modo), A

LD (N_on), A

check_up:

; miro botón up

IN A, (p_up)

OUT (p_clr_up), A

CP 1

JP NZ, check_restart

; sumo 1 a N_on

LD A, (N_on)

CP N_pwm

JP Z, check_restart

; N_on satura en N_pwm

INC A

LD (N_on), A

check_restart:

; miro botón restart

IN A, (p_restart)

OUT (p_clr_restart), A

CP 1

JP NZ, ppal

; restauro T_on inicial

LD A, (modo)

; o if_then_else según modo

LD (N_on), A

JP ppal

ORG 0x3000

dir_umbral: DB UMBRAL

dir_n_noches: DB N_noches

ORG 0xC000

N_on: DB

N_on_isr: DB

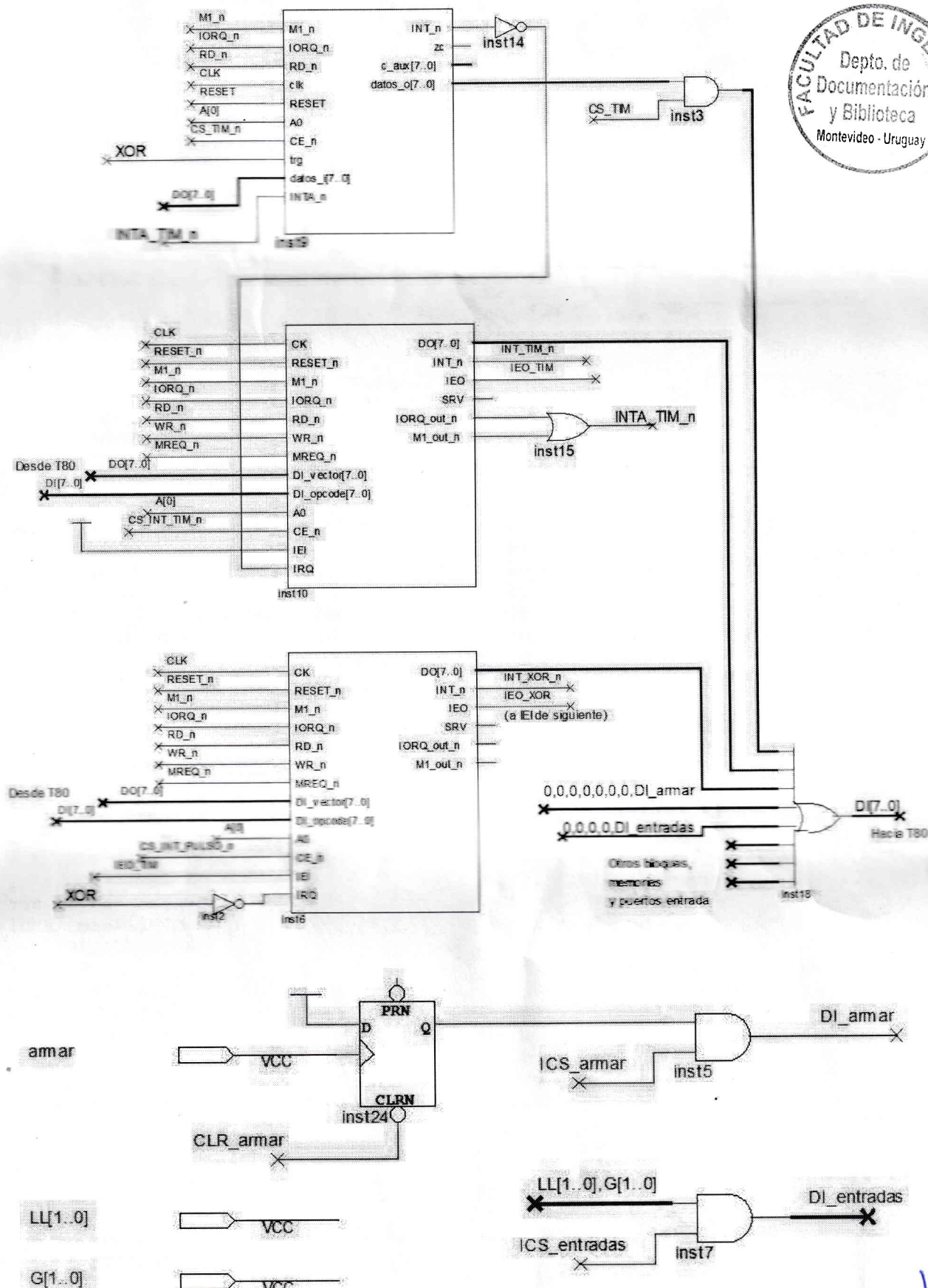
modo: DB



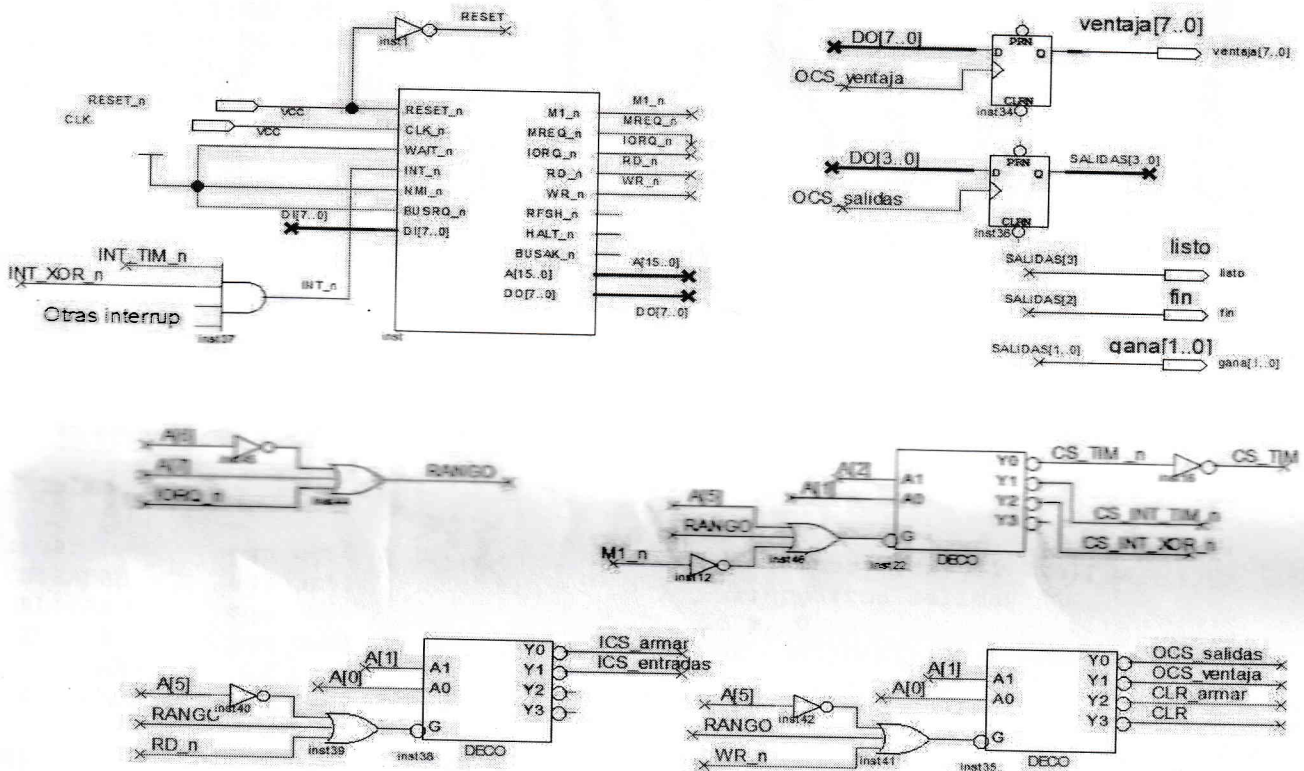
5/8.

PROBLEMA 2

a) Hardware



6/8.



b) Subrutina sub_armar() *****

```
;; sub_armar(){
;;   esperar pulso armar
;;   clr FF_armar
;;   clr FFs circuito detección
;;   arrancar Timer
;;   listo,fin,gana1,gana0 = 1,0,0,0
;;   ventaja = 0
;;   estado = E_LISTO
;; }
```

sub_armar:

```
in a, (armar)
bit 0, a
jr z, sub_armar
out (clr_ARMAR), a
out (clr_CIRCUITO), a
ld a, CW_TIMER_EI
out (timer_base+off_cw), a
ld a, maskLISTO
out (SALIDAS), a
ld a, 0
out (VENTAJA), a
ld a, E_LISTO
ld (estado), a
ret
```

```
push af
push bc
ld a, (estado)
cp E_LISTO
jr nz, fin_xor
ld a, E_IDLE
in a, (timer_base+off_cuenta)
ld b, a
;; frenar Timer
ld a, CW_TIMER_DI
out (timer_base+off_cw), a
;; ventaja = (TOUT-1) - cuenta
ld a, TOUT-1
sub b
out (VENTAJA), a
;; listo,fin,gana1,gana0 = 0,1,G1,G0;
in a, (ENTRADAS)
and maskG1G0
srl
srl ; gana1,gana0 = G1,G0
or maskFIN ; fin=1
out (SALIDAS), a
```

fin_xor:

```
pop bc
pop af
reti
```

c) Rutinas interrupción *****

```
;; isr_bajadaXOR(){
;;   si (estado == E_LISTO){
;;       estado = E_IDLE
;;       leer cuenta
;;       frenar Timer
;;       ventaja = (TOUT-1) - cuenta
;;       listo,fin,gana1,gana0 = 0,1,G1,G0;
;;   }
```

```
;; isr_timer(){
;;   frenar Timer
;;   estado = E_IDLE
;;   ventaja = 0xFF
;;   listo,fin,gana1,gana0 = 0,1,LL1,LL0
```

isr timer:



FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

INTRODUCCIÓN A LOS MICROPROCESADORES

Julio 2024

```
;; frenar Timer
ld a, CW_TIMER_DI
out (timer_base+off_cw), a
ld a, E_IDLE
ld (estado), a
ld a, 0xFF
out (VENTAJA), a
;; fin, gana1, gana0 = 1, LL1, LL0
in a, (ENTRADAS)
and maskLL1LL0
or maskFIN
out (SALIDAS), a
pop af
reti
```

d) inicializacion *****

E_LISTO equ 0
E_IDLE equ 1

armar equ 0x60
ENTRADAS equ 0x61
SALIDAS equ 0x60
VENTAJA equ 0x61
clr_ARMAR equ 0x62
clr_CIRCUITO equ 0x63

maskLISTO equ 00001000B ; bit3
maskFIN equ 00000100B ; bit2
maskG1G0 equ 00001100B ; bit3..2
maskLL1LL0 equ 00000011B ; bit1..0

timer_base equ 0x40
off_cte equ 0
off_cw equ 1
off_cuenta equ 1
;; -, up, sw rst, trg, PRE=15
CW_TIMER_DI equ 01111111B
CW_TIMER_EI equ 11111111B
TOUT equ 100

CI_TIMER_BASE equ 0x42
CI_XOR_BASE equ 0x44
off_vector equ 0
off_clr_pend equ 1
VECTOR_TIMER equ 6

VECTOR_XOR equ 4

.text ;; a partir de direcc 0

```
ini:
ld sp, 0
;; interrupciones
im 2
ld hl, tabla_int
ld a, h
ld i, a
ld a, VECTOR_TIMER
out (CI_TIMER_BASE+off_vector), a
out (CI_TIMER_BASE+off_clr_pend), a
ld a, VECTOR_XOR
out (CI_XOR_BASE+off_vector), a
out (CI_XOR_BASE+off_clr_pend), a
;; cte recarga timer
ld a, TOUT-1
out (timer_base+off_cte), a
;; puertos salida
ld a, 0
out (SALIDAS), a
out (VENTAJA), a
;; variables
ld a, E_IDLE
ld (estado), a
;; otros
call init_otros
ei
jp ppal
```

; tabla interrupciones

org 0x100

tabla_int:

dw isr_0
dw isr_2
dw isr_bajadaXOR
dw isr_timer

; variables

.data ;; a partir de 0x8000
estado: db 0



8/8