

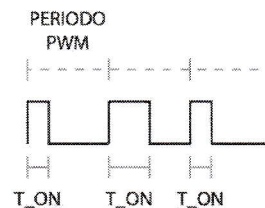
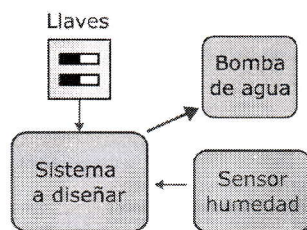
- Mínimo para aprobar: un problema
- Nombre y CI en cada hoja
- Numere las hojas, indique el total en la primera

- 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, basándose en un sistema con RISC-V/Z80, un circuito de control para una bomba de agua de un sistema de riego. El controlador cuenta con 2 llaves de entrada para seleccionar el caudal de agua, un sensor de humedad del suelo y una salida PWM que controla la bomba.



La señal PWM es una secuencia de pulsos de período constante cuyo tiempo en 0 y 1 puede variar para controlar el caudal de agua. A mayor tiempo en 1, mayor caudal. Dependiendo del valor de las llaves es la cantidad de agua que debe de bombear: 00: bomba apagada (salida PWM siempre 0), 01: caudal mínimo, 10: caudal medio, 11: caudal máximo.

El período de la señal PWM y el tiempo en 1 (T_{ON}) para cada caso se deben de almacenar en una tabla en ROM como se explica más adelante.

Para evitar que se sature de agua el suelo, se debe utilizar el sensor para conocer la humedad y ajustar el caudal. En caso de estar por encima de la humedad máxima h_{max} , se debe enviar la mitad del caudal seleccionado (disminuir T_{ON} a la mitad). La humedad es un valor de 8bits que está siempre disponible en el sensor.

Los parámetros de funcionamiento se deben almacenar en una tabla en ROM que debe ubicarse a partir de la dirección 3000h. El contenido de las direcciones será:

0x3000	Período del ciclo en ms = 100
0x3001	T_{ON} en ms para caudal mínimo = 20
0x3002	T_{ON} en ms para caudal medio = 40
0x3003	T_{ON} en ms para caudal máximo = 80
0x3004	Humedad máxima = 90



Como base de tiempo para la generación del PWM se debe utilizar una señal TIC que tiene un flanco de bajada cada 1ms y debe interrumpir el procesador.

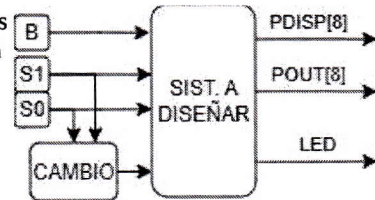
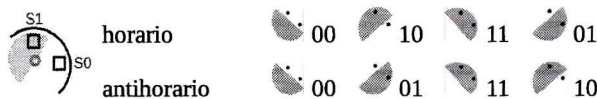
Se pide:

- Hardware del sistema. Suponer la memoria ya implementada, disponiendo de ROM entre 0x0000 y 0x4FFF y RAM entre 0x5000 y 0x7FFF. Para la decodificación de los puertos se dispone solo del rango 0x80 a 0x8F en el caso de Z80 y de 0x60080 a 0x6008F en el de RISC-V.
- Inicialización del sistema, declaración de variables en memoria y programa principal. Este último se encarga de monitorear el estado de las llaves y el sensor de humedad para reportar a la rutina de atención a interrupciones los valores necesarios para generar la señal PWM.
- Rutina de atención a interrupciones encargada de generar la señal PWM. Los cambios de período y tiempos de 1 y 0 se actualizan luego de finalizado cada período PWM.

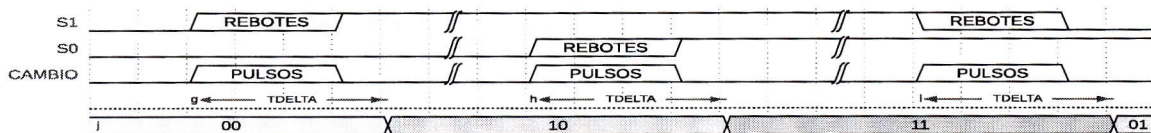
PROBLEMA 2

Se debe diseñar un sistema basado en un procesador Risc-v o T80 para permitir que un operador ajuste el valor de una salida **POUT[8]**. Para eso se dispone de un *encodificador* rotatorio, un pulsador **B** (0 presionado, 1 en caso contrario), un display que muestra el valor en su entrada **PDISPLAY[8]** (binario sin signo) y una salida **LED** (1 ON, 0 OFF).

El *encodificador* cuenta con una perilla que hace girar un semicírculo opaco. Dos sensores **S₁** y **S₀** separados un cuarto de vuelta detectan si frente al sensor **S_x** está el semicírculo (**S_x**=1) o no (**S_x**=0). De esa forma al girar la perilla en sentido horario los valores de los sensores **S₁S₀** recorrerán la secuencia 00, 10, 11, 01, 00..., y en sentido antihorario 00, 01, 11, 10, 00.



Se dispone de un bloque que genera en su salida **CAMBIO** un pulso a nivel alto de 1 Tclk de duración cada vez que hay una transición en **S₁** o **S₀**. Cuando cambia el valor de un sensor la señal presenta ruido y rebotes, como se muestra en el diagrama de tiempos para el caso de giro en sentido horario. Para evitar detectar cambios espurios, luego del primer pulso en **CAMBIO** se debe esperar un tiempo **TDELTA** antes de leer el valor de **S₁S₀**. Luego del primer pulso en **CAMBIO** llegarán nuevos pulsos que deben ser ignorados mientras transcurre el tiempo **TDELTA**.



Luego de reset deben comenzar **POUT** y **PDISPLAY** en cero, y **LED** apagado.

Pulsando **B** una vez se debe encender **LED** para indicar que se entra en modo edición en que se puede ajustar el valor de **PDISPLAY** girando la perilla del encodificador. **PDISPLAY** se debe incrementar (+1) con cada transición en sentido horario y decrementar (-1) con cada transición en sentido antihorario. El incremento/decremento es circular, pasa de 255 a 0 o viceversa.

Pulsando nuevamente **B** se debe salir de modo edición, apagar **LED** y copiar el valor final de **PDISPLAY** a **POUT**. Ambas salidas deben mantener ese valor hasta que se vuelva a pasar a modo edición.

Se utilizará el Timer interno o un bloque Timer para medir el tiempo **TDELTA**. El sistema contará con dos interrupciones: Timer y **CAMBIO**.

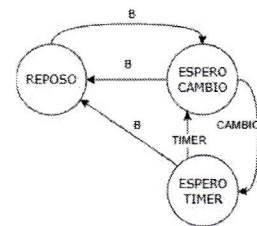
Se supone ya implementada la subrutina **calculo_inc()** que recibe los valores anterior (en registro a0 o A) y actual (en registro a1 o B) de **S₁S₀**, identifica el sentido de giro horario o antihorario y devuelve el valor del incremento correspondiente +1 o -1 (en registro a0 o A).

Se sugiere utilizar una variable **v_ESTADO** para que tanto el programa principal como las rutinas de atención sepan en qué estado está el sistema y actúen en consecuencia.

Las transiciones etiquetadas con B indican pulsaciones del botón detectadas en el programa principal, que debe ser un loop infinito. Las otras dos transiciones indican la llegada de la rutina de atención a interrupción correspondiente. Notar que en modo edición se espera o por un cambio en los sensores o por el Timer, pero nunca por ambos simultáneamente.

Se debe trabajar en modo vectorizado en caso de usar Risc-v o modo 2 en caso de usar Z80. Se pide:

- Hardware del sistema, incluyendo los puertos, el hardware asociado a las interrupciones y el periférico timer. Suponer memorias ROM y RAM de 16KB **ya implementadas** en direcciones consecutivas de memoria desde la dirección 0x0000. El rango de direcciones disponible para los puertos es de 0x60000 a 0x600FF del espacio de memoria para el caso RISC-V o de 0x40 a 0x4F del espacio de E/S para el caso T80. En el caso de RISC-V debe indicarse claramente el valor de prescaler utilizado.
- Programa principal.
- Rutinas de atención a interrupción de Timer y CAMBIO.
- Inicialización y directivas de reserva de memoria.

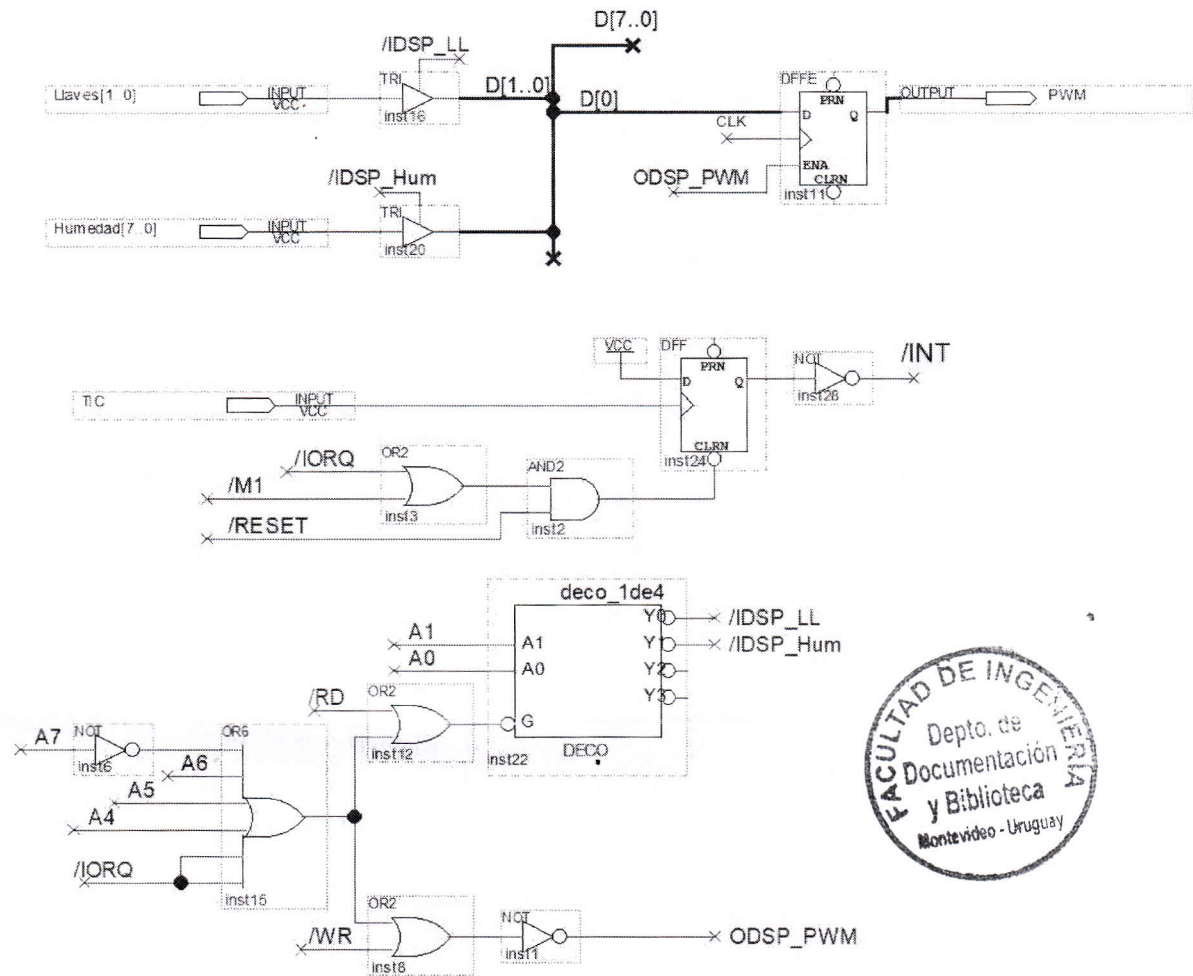


Nota: fclk = 6,4 MHz; TDELTA = 1 ms TDELTA = 100 * 2¹⁷ * 1/fclk



2/17

PROBLEMA 1 – SOLUCIÓN Z80



```

b)
;direcciones de puertos
LlavesIO    EQU    80h    ;puerto de entrada: llaves
HumedadIO    EQU    81h    ;puerto de entrada: humedad
PwmIO        EQU    80h    ;puerto de salida: pwm

;reserva de memoria para tabla en ROM
ORG 3000h
PERIODO      DB    100
TON_MIN      DB    20
TON_MED      DB    40
TON_MAX      DB    80
HMAX         DB    90

;reserva de memoria para variables en RAM
ORG 0x5000
PWM:         DB    ;proximo valor PWM a generar
PWMRUT:      DB    ;valor PWM a generar
CONT:        DB    ;contador de ciclos

;inicializacion del micro
ORG 0000h    ;codigo en ROM
ld SP, 0x8000 ; inicializo el stack pointer
im 1         ; interrupciones en modo 1

;inicializo el sistema y variables
ld A, 0x00
ld (CONT), A
ld (PWM), A
ld (PWMRUT), A
ld A, 0x00
out (PWMIO), A ;mando 1 a la bomba
ei             ;habilito las INT

; forever{
;   si (sw=00) entonces { pwm = 0 }
;   else {
;       pwmtmp = tabla[sw]
;       si ( t >= tmax )
;       entonces { pwmtmp = pwmtmp / 2 }
;       pwm = pwmtmp
;   }
; }

Loop:        ;loop principal
in A, (LlavesIO) ;leo los switches
and A, 03h
jp NZ, Else

ld A, 00h
ld(PWM), A
jp Loop      ;si (sw=00) entonces { pwm = 0 }

Else:        ;inicializo el puntero a la tabla
ld HL, TON_MIN
ld L, A
ld A, (HL)   ;leo T_ON de la tabla
ld C, A      ;guardo el valor en registro C

;control de la humedad
ContrHumedad: in A, (HumedadIO) ;leo la humedad
ld B, A
ld A, (HMAX)
cp B         ;hago (Hmax - HActual)
jp P, Fin    ;si >= 0 (es decir HActual <= Hmax) salto

H_Saturada:
srl C        ; dividido por dos el T_ON leído de la tabla

```



4/17



Fin: ld A, C
 ld (PWM), A
 jp Loop

c)
;rutina de atención a las interrupciones
 ORG 0038h

 push AF ;preservo AF
 push BC ;preservo BC

 ld A, (CONT)
 ld B, A
 ld A, (PWMRUT)
 cp B ;hago (PWMRUT - CONT)
 jp P, IncCont ;si > 0 (o sea CONT < PWMRUT) no transcurrio T_ON

TranscurrioTON:
 ld A, 00h
 out (PwmIO), A ;si <= 0 (o sea CONT >= PWMRUT), 0 a bomba

ContrCiclo: ld A, (PERIODO)
 ld B, A
 ld A, (CONT)
 cp B ;hago (CONT - Periodos_ms)
 jp M, IncCont ;si < 0 (o sea CONT < Periodos_ms) salto

TranscurrioCiclo:
 ld A, 01h
 out (PwmIO), A ;si >= 0 (o sea CONT >= Preiodo), 1 a la bomba

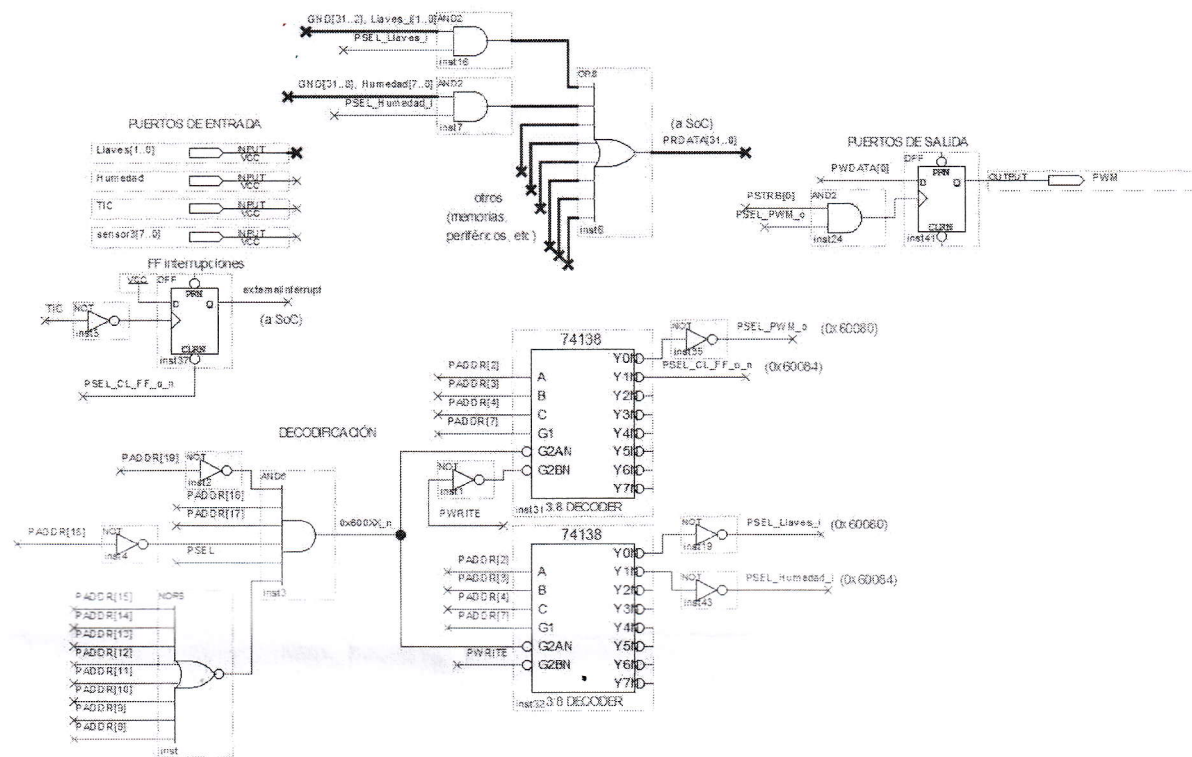
 ld A, (PWM)
 ld (PWMRUT), A ;leo el proximo PWM a generar y lo cargo en el usado por la rutina

 ld A, 00h
 ld (CONT), A ; CONT = 0
 jp FinIsr

IncCont: ld A, (CONT)
 inc A
 ld (CONT), A ;incremento el contador de INT

FinIsr:
 pop BC ;recupero BC
 pop AF ;recupero AF
 ei
 ret ;retorno

a)



b)

```

;direcciones de puertos
.equ BASE_IO,      60080h
.equ LlavesIO,     00h           ;puerto de entrada: llaves
.equ HumedadIO,    04h           ;puerto de entrada: humedad
.equ PwmIO,        00h           ;puerto de salida: pwm
.equ CL_FF_extint, 04h           ;puerto de salida: clear FF isr externa

```

```
.equ SP INI,      0x8000
```

```
.globl    main
```

```
.data # a partir de 0x05000
```

```
;reserva de memoria para variables en RAM
```

```

ORG          5000h
PWM:         .byte 0      ;proximo valor PWM a generar
PWMRUT:      .byte 0      ;valor PWM a generar
CONT:        .byte 0      ;contador de ciclos

```

```
.text # a partir de 0x000000
```

```
main:
```

```
;inicializacion del micro
li SP, SP INI      ; inicializo el stack pointer
```

```
;inicializo el sistema y variables
```

```
la t0, CONT
sb zero, 0(t0)
la t0, PWM
sb zero, 0(t0)
la t0, PWMRUT
```



6/17

```

sb zero, 0(t0)

li t1, 01h
li t0, BASE_IO
sb t1, PwmIO(t0) ;mando 1 a la bomba

;interrupciones
la t0, isr_tic
;Modo directo, mtvec = isr
csrw mtvec, t0

li t0, 0x800
;Habilitar externalInterrupt
csrs mie, t0
li t0, 0x8
;Habilitar interrupciones\
csrs mstatus, t0
j main_loop

; forever{
;   si (sw=00) entonces { pwm = 0 }
;   else {
;       pwmtmp = tabla[sw]
;       si ( t >= tmax )
;           entonces { pwmtmp = pwmtmp / 2 }
;       pwm = pwmtmp
;   }
; }

main_loop: ;loop principal
li s0, BASE_IO
lbu a0, LlavesIO(s0) ;leo los switches
andi a0, a0, 0x3
bnez a0, Else

la t0, PWM
sb zero, 0(t0)
j main_loop ;si (sw=00) entonces { pwm = 0 }

Else: la a2, tabla_datos ;base = 0x3000
add a2, a2, a0 ;base + sw (1..3)
lbu a2, 0(a2) ;a2 = TON_MIN/MED/MAX según sw

;control de la humedad
ContrHumedad: lbu a1, HumedadIO(s0) ;leo la humedad
la a0, HMAX
lbu a0, 0(a0)
bge a0, a1, Fin ;si Hmax >= HActual salto

H_Saturada: srli a2, a2, 1 ;divido por dos el T_ON leído de la tabla

Fin: la t0, PWM
sb a2, 0(t0)
j main_loop

c)
isr_tic: ;salvar registros usados
addi sp, sp, -16
sw ra, 12(sp)
sw t0, 8(sp)
sw t1, 4(sp)
sw t2, 0(sp)

```



```
;t1 = CONT
la  t0, CONT
lbu t1, 0(t0)

;t2 = PWMRUT
la  t0, PWMRUT
lbu t2, 0(t0)

;if (PWMRUT - CONT) >= 0 <=> PWMRUT >= CONT => no transcurrió T_ON
bge t2, t1, IncCont

TranscurrioTON:
li  t0, BASE_IO
sb  zero, PwmIO(t0)

ContrCiclo:
;t2 = PERIODO
la  t0, PERIODO
lbu t2, 0(t0)

;if (CONT - PERIODO) < 0 <=> CONT < PERIODO => no transcurrió ciclo
blt t1, t2, IncCont

TranscurrioCiclo:
li  t0, BASE_IO
li  t2, 1
sb  t2, PwmIO(t0)

;PWMRUT = PWM
la  t0, PWM
lbu t2, 0(t0)
la  t0, PWMRUT
sb  t2, 0(t0)

;CONT = 0 (reinicio de ciclo)
la  t0, CONT
sb  zero, 0(t0)
j   AckAndRet

IncCont:
;CONT++
la  t0, CONT
lbu t1, 0(t0)
addi t1, t1, 1
sb  t1, 0(t0)

AckAndRet:
;Ack/clear de la interrupción externa
li  t0, BASE_IO
sb  zero, CL_FF_extint(t0)

;restaurar registros y volver
lw  t2, 0(sp)
lw  t1, 4(sp)
lw  t0, 8(sp)
lw  ra, 12(sp)
addi sp, sp, 16
mret

;reserva de memoria para tabla en ROM
.org 0x3000
tabla_datos:
    PERIODO:      .byte 100
    TON_MIN:      .byte 20
```



FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

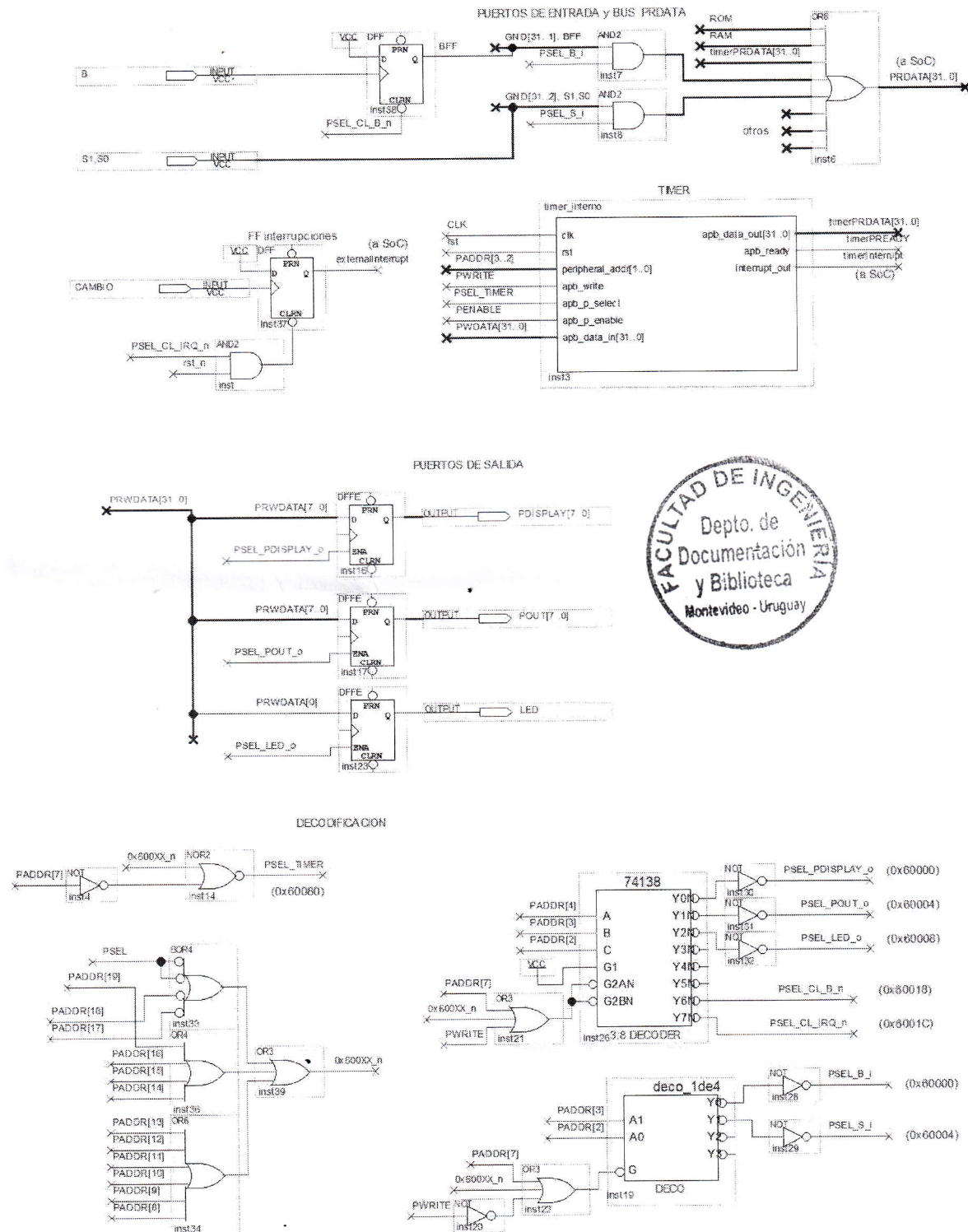
INTRODUCCIÓN A LOS MICROPROCESADORES
Febrero 2026

TON_MED: .byte 40
TON_MAX: .byte 80
HMAX: .byte 90



PROBLEMA 2 – SOLUCIÓN RISC-V

a) Hardware



PRESCALER Timer = 7 (divide frecuencia de reloj por 2^7)

10/77

ej. 2, risc-v, Software

```
.equ BASE_IO, 0x60000
.equ BUTTON, 0x00
.equ SENS, 0x04
.equ PDISPLAY, 0x00
.equ POUT, 0x04
.equ LED, 0x08
.equ CLR_BUTTON, 0x18
.equ CLR_CAMBIO, 0x1C

.equ E_REPOSO, 0
.equ E_WAIT_S, 1
.equ E_WAIT_T, 2

.equ mtime_off, 0x80 # 0x60080
.equ mtimecmp_off, 0x88 # 0x60088
.equ TDELTA, 100
```



b) prog. Ppal y d) inicialización

```
.text
main:
init:
# init_y_ppal{
# estado = E-REPOSO
# POUT, PDISPLAY, v_display = 0
# v_sens = SENS
#
# ## loop ppal
# forever{
#     espero B
#     borro FF B
#     si (estado == E_REPOSO) {
#         estado = E_WAIT_S
#         v_sens = puerto_sensores
#         borrar FF cambio sensores
#         habilito int sensores
#         LED = on
#     }else{
#         estado = E_REPOSO
#         deshabilito sensores y Timer
#         LED = off
#         POUT = PDISPLAY
#     }
# }
```

```
# }
# }
init_y_ppal:
# estado = E-REPOSO
la t0, v_estado
li t1, E_REPOSO
sb t1, 0(t0)
# POUT, PDISPLAY, v_display = 0
la t0, BASE_IO
sb zero, POUT(t0)
sb zero, PDISPLAY(t0)
la t1, v_display
sb zero, 0(t1)
# v_sens = SENS
la t1, v_sens
lw t2, SENS(t0)
sb t2, 0(t1)
```

```
forever:
li t1, BASE_IO
esperoB:
lb t2, BUTTON(t1)
andi t2, t2, 0x00000001
bne zero, t2, esperoB
sb t2, CLR_BUTTON(t1)
la t0, v_estado
lbu t2, 0(t0)
li t3, E_REPOSO
bne t2, t3, else_ppal
# si (estado == E_REPOSO)
# estado = E_WAIT_S
li t3, E_WAIT_S
sb t3, 0(t0)
# v_sens = puerto_sensores
la t0, v_sens
lw a0, SENS(t1)
sb a0, 0(t0)
# borrar FF cambio sensores
# LED = on
sb a0, CLR_CAMBIO(t1)
li t2, 1
sb t2, LED(t1)
# habilito int sensores (externa)
```

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

INTRODUCCIÓN A LOS MICROPROCESADORES
Febrero 2026

```

li t2, 0x800 # MEIE (bit 11, ext int)
csrs mie, t2
j forever
else_ppal:
# else
# estado = E-REPOSO
li t3, E_REPOSO
sb t3, 0(t0)
# deshabilito int sensores y Timer
li t2, 0x880 # MEIE bit11 ext int,
MTIE bit7 timer int
csrc mie, t2
# LED = off
sb zero, LED(t1)
# POUT = PDISPLAY
la t0, v_display
lbu t2, 0(t0)
sb t2, POUT(t1)
j forever

```



c) rutinas atención interrupciones

```

# isr_sens{
#   preservar
#   si (estado == E_WAIT_S){
#       estado = E_WAIT_T
#       arranco_Timer(TDELTA)
#       hab Timer, deshab sensores
#   }
#   clr FF cambio
#   restaurar
# }
isr_sens:
# preservar
addi sp, sp, -12
sw t0, 8(sp)
sw t1, 4(sp)
sw t2, 0(sp)

la t0, v_estado
lbu t1, 0(t0)
li t2, E_WAIT_S
bne t1, t2, fin_si_isr_sens
# si (estado == E_WAIT_S){
#   estado = E_WAIT_T
li t2, E_WAIT_T

```

```

sw t2, (t0)
#   arranco_Timer(TDELTA)
#   (** suponiendo que no hay desborde
todavía **)
la t0, BASE_IO
lw t1, mtime_off(t0) # Leer
mtime
li t2, TDELTA
add t1, t1, t2
sw t1, mtimecmp_off(t0) #
Escribir en mtimecmp
# deshab sensores
li t1, 0x800 # MEIE (bit 11,
ext int)
csrc mie, t1
# hab Timer
li t1, 0x080 # MEIE (bit 11,
ext int)
csrs mie, t1
fin_si_isr_sens:
# clr FF cambio
sb a0, CLR_CAMBIO(t0)
# restaurar
lw t0, 8(sp)
lw t1, 4(sp)
lw t2, 0(sp)
addi sp, sp, 12
mret

# isr_timer{
#   preservar
#   estado = E_WAIT_S /* espero
sensores */
#   [ habilito sensores ]
#   deshabilito Timer
#   v_sens_old = v_sens
#   v_sens = puerto_sensores
#   v_inc = inc(v_sens_old, v_sens)
#   v_display = v_display + v_inc
#   PDISPLAY = v_display
#
#   restaurar
# }
isr_timer:
# preservar
addi sp, sp, -24
sw ra, 20(sp)

```

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

INTRODUCCIÓN A LOS MICROPROCESADORES
Febrero 2026

```

sw a0, 16(sp)
sw a1, 12(sp)
sw t0, 8(sp)
sw t1, 4(sp)
sw t2, 0(sp)
# estado = E_WAIT_S      /* espero
sensores */
la t0, v_estado
li t2, E_WAIT_S
sw t2, (t0)
# hab sensores
li t1, 0x800             # MEIE (bit 11,
ext int)
csrs mie, t1
# deshab Timer
li t1, 0x080             # MEIE (bit 11,
ext int)
csrc mie, t1
# a0/A = v_sens
# a1/B = puerto_sensores
li t0, BASE_IO
la t1, v_sens
lbu a0, 0(t1)
lbu a1, SENS(t0)
# v_sens = puerto_sensores
sb a1, 0(t1)
# v_inc = inc(v_sens_old, v_sens)
call calculo_inc

```

```

# v_display = v_display + v_inc
# PDISPLAY = v_display
la t2, v_display
lbu t1, 0(t2)
add t1, t1, a0
sb t1, 0(t2)
sb t1, PDISPLAY(t0)

```

```

# restaurar
lw ra, 20(sp)
lw a0, 16(sp)
lw a1, 12(sp)
lw t0, 8(sp)
lw t1, 4(sp)
lw t2, 0(sp)
addi sp, sp, 24
mret

```

```

calculo_inc:
ret

```

```

.data
v_display: .byte 0
v_sens:    .byte 0
v_estado:  .byte 0

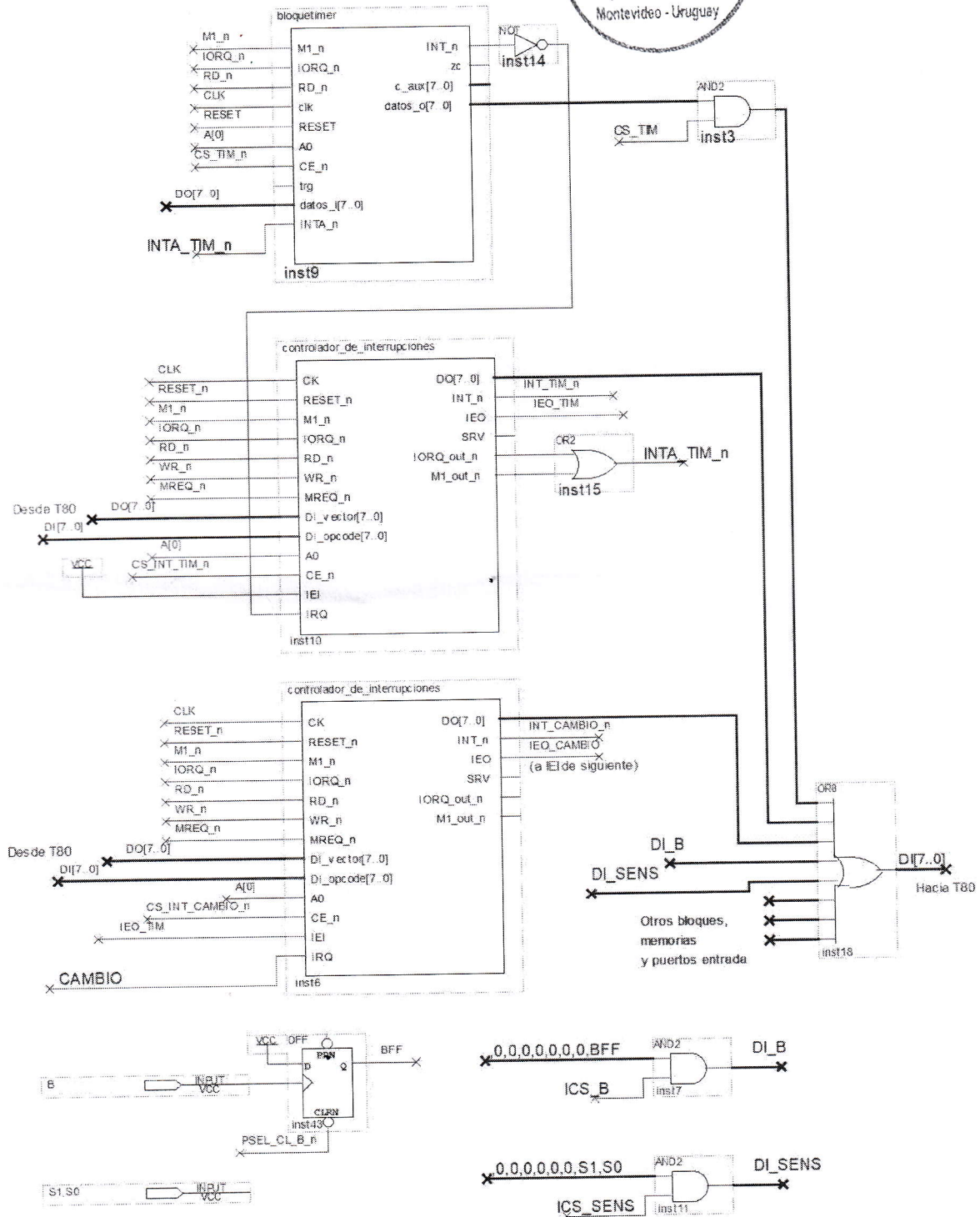
.end

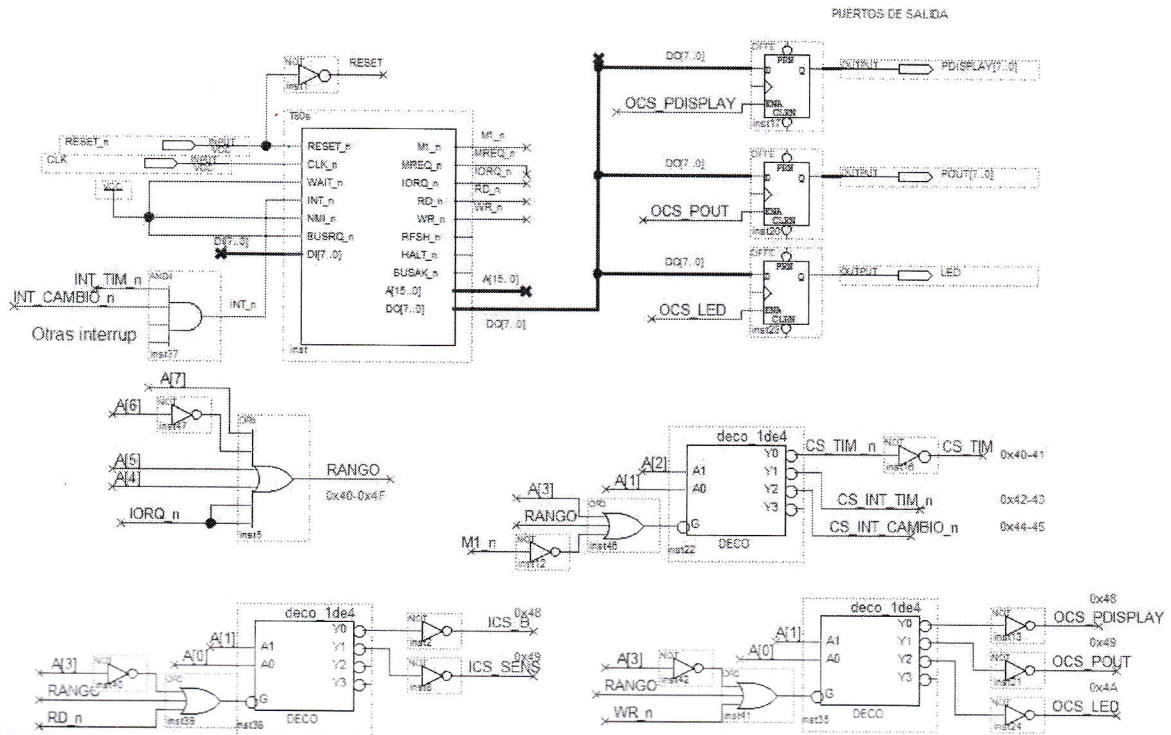
```



PROBLEMA 2 – SOLUCIÓN T80

a) Hardware



**ej. 2, Z80, Software**

```
.equ BUTTON,      0x48
.equ SENS,        0x49
.equ POUT,        0x49
.equ PDISPLAY,    0x48
.equ LED,         0x4A

.equ E_REPOSO,    0
.equ E_WAIT_S,    1
.equ E_WAIT_T,    2

.equ TIM,         0x40
.equ CINT_TIM,    0x42
.equ CINT_CAMBIO, 0x44

.equ TIM_CW_DI,   00000000B ; EI = 0
    # hab timer: EI, sw_reset, auto, pre=7
.equ TIM_CW_EI,   10100111B
.equ TIM_CTE,     100
.equ VEC_TIM,     0
.equ VEC_CAMBIO,  2
```

**## b) prog. Ppal y d) inicialización**

```
.text
init:
# init_y_ppal{
# estado = E_REPOSO
# POUT, PDISPLAY, v_display = 0
# v_sens = SENS
# controladores int (t80)
# Cte TIMER (t80)
#
# ## loop ppal
# forever{
# espero B
# si (estado == E_REPOSO) {
# estado = E_WAIT_S
# v_sens = puerto_sensores
# ## (sólo risc-v)
# ## borrar FF cambio sensores
# ## habilito int sensores
# LED = on
```

15/17

```

#     }else{
#         estado = E_REPOSO
#         deshab. sensores (sólo risc-v) y
Timer
#         LED = off,
#         POUT = PDISPLAY
#     }
# }
# }

# estado = E_REPOSO
ld a, E_REPOSO
ld (v_estado), a
# POUT, PDISPLAY, v_display = 0
ld a, 0
out (POUT), a
out (PDISPLAY), a
ld (v_display), a
# v_sens
in a, (SENS)
ld (v_sens), a
# controladores int (t80)
ld a, VEC_TIM
out (CINT_TIM), a
out (CINT_TIM+1), a
ld a, VEC_CAMBIO
out (CINT_CAMBIO), a
out (CINT_CAMBIO+1), a
# Cte TIMER (t80)
ld a, TIM_CTE
out (TIM), a

forever:
esperoB:
    in a, (BUTTON)
    bit 0, a
    jr z, esperoB
    ld a, (v_estado)
    CP E_REPOSO
    jr nz, else_ppal
    # si (estado == E_REPOSO)
    #     estado = E_WAIT_S
    ld a, E_WAIT_S
    ld (v_estado), a
    # v_sens = puerto_sensores
    in a, (SENS)
    ld (v_sens), a
    # LED = on
    ld a, 0xFF
    out (LED), a
    jp forever
else_ppal:
    #else
    #     estado = E_REPOSO
    ld a, E_REPOSO
    ld (v_estado), a
    # deshabilito int sensores y Timer
    #     (solamente Timer en T80)
    ld a, TIM_CW_DI
    out (TIM+1), a
    # LED = off
    ld a, 0
    out (LED), a
    # POUT = PDISPLAY
    ld a, (v_display)
    out (POUT), a
    jp forever

## c) rutinas atención interrupciones
# isr_sens{
#     preservar
#     si (estado == E_WAIT_S){
#         estado = E_WAIT_T
#         arranco_Timer(TDELTA)
#         hab Timer, deshab sensores
#     }
#     restaurar
# }
isr_sens:
    # preservar
    ei
    push af
    ld a, (v_estado)
    cp E_WAIT_S
    jr nz, fin_si_isr_sens
    # si (estado == E_WAIT_S){
    #     estado = E_WAIT_T
    ld a, E_WAIT_T
    ld (v_estado), a
    #     arranco_Timer(TDELTA)

```



FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

INTRODUCCIÓN A LOS MICROPROCESADORES
Febrero 2026

```

ld a, TIM_CW_EI
out (TIM+1), a
# deshab sensores
# no se puede en contr int T80
fin_si_isr_sens:
# restaurar
pop af
reti

# isr_timer{
# preservar
# estado = E_WAIT_S
# deshabilito Timer
# v_sens_old = v_sens
# v_sens = puerto_sensores
# v_inc = inc(v_sens_old, v_sens)
# v_display = v_display + v_inc
# PDISPLAY = v_display
# restaurar
# }

isr_timer:
# preservar
ei
push af
push bc

# estado = E_WAIT_S /* espero
sensores */
ld a, E_WAIT_S
ld (v_estado), a
# deshab Timer
ld a, TIM_CW_DI
out (TIM+1), a

# a0/A = v_sens

```



```

# a1/B = puerto_sensores
ld a, (v_sens) ; v_sens old
ld c, a
in a, (SENS)
ld b, a ; B=port_sens
ld (v_sens), a ; v_sens = port
ld a, c ; a = v_sens_old

# a = inc(v_sens_old, v_sens)
call calculo_inc
# v_display = v_display + a
# PDISPLAY = v_display
ld b, a
ld a, (v_display)
add a, b
ld (v_display), a
out (PDISPLAY), a
# restaurar
pop bc
pop af
reti

```

```

.org 0x100 # comienzo de pág. en .text
tabla_isr:
.dw isr_timer
.dw isr_sens

.data
v_display: .byte 0
v_sens: .byte 0
v_estado: .byte 0

.end

```

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

INTRODUCCIÓN A LOS MICROPROCESADORES
Febrero 2026

```

ld a, TIM_CW_EI
out (TIM+1), a
# deshab sensores
# no se puede en contr int T80
fin_si_isr_sens:
# restaurar
pop af
reti

# isr_timer{
# preservar
# estado = E_WAIT_S
# deshabilito Timer
# v_sens_old = v_sens
# v_sens = puerto_sensores
# v_inc = inc(v_sens_old, v_sens)
# v_display = v_display + v_inc
# PDISPLAY = v_display
# restaurar
# }

isr_timer:
# preservar
ei
push af
push bc

# estado = E_WAIT_S /* espero
sensores */
ld a, E_WAIT_S
ld (v_estado), a
# deshab Timer
ld a, TIM_CW_DI
out (TIM+1), a

# a0/A = v_sens

```



```

# a1/B = puerto_sensores
ld a, (v_sens) ; v_sens old
ld c, a
in a, (SENS)
ld b, a ; B=port_sens
ld (v_sens), a ; v_sens = port
ld a, c ; a = v_sens_old

# a = inc(v_sens_old, v_sens)
call calculo_inc
# v_display = v_display + a
# PDISPLAY = v_display
ld b, a
ld a, (v_display)
add a, b
ld (v_display), a
out (PDISPLAY), a
# restaurar
pop bc
pop af
reti

```

```

.org 0x100 # comienzo de pág. en .text
tabla_isr:
.dw isr_timer
.dw isr_sens

.data
v_display: .byte 0
v_sens: .byte 0
v_estado: .byte 0

.end

```