

- 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 implementar un **metrónomo programable** en una plataforma **RISC-V/T80**, capaz de generar una secuencia de pulsos con período configurable en la salida `pout`. El sistema debe poder iniciarse o detenerse mediante una señal externa de control (`start/stop = ss`) y detenerse automáticamente tras emitir una cantidad determinada de pulsos.

El sistema permanece inactivo hasta que se recibe un flanco de subida en la señal `ss`.

Al activarse, debe leer los dos valores de configuración que se indican a continuación y dar un primer pulso de corta duración en la salida `pout`.

- `tempo[8]`: período entre pulsos, en milisegundos.
- `n[8]`: cantidad de pulsos a emitir.

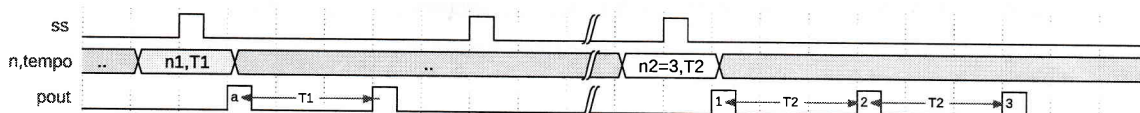
Una vez iniciado, el sistema debe seguir dando un pulso de corta duración en la salida `pout` cada `tempo` ms. El sistema debe contar los pulsos generados y detenerse automáticamente al completar `n` pulsos.

Si se recibe un nuevo flanco ascendente en `ss` antes de terminar, el sistema debe volver al estado inactivo inicial y quedar a la espera de un nuevo flanco.

Cada flanco ascendente en `ss` debe generar una **interrupción**. La rutina asociada debe o bien leer los valores de `tempo` y `n`, dar el primer pulso en `pout`, inicializar el conteo de pulsos y configurar el periférico **timer** para generar interrupciones periódicas, o bien detener el metrónomo, según corresponda. Se puede suponer que los valores de `tempo` y `n` se mantienen válidos después del flanco de `ss` por un tiempo suficiente para ser leídos dentro de la rutina de atención a la interrupción.

Por su parte, el **timer** debe generar interrupciones para activar la salida `pout` cada `tempo` ms. A su vez, debe llevar el conteo de pulsos emitidos y detener el sistema al alcanzar `n` pulsos emitidos. Se debe especificar claramente todas las palabras de control utilizadas (caso bloque Timer y T80) o el valor del prescaler PRE (caso bloque Timer y RISC-V).

La figura muestra dos casos: en el primero la secuencia se interrumpe por un nuevo pulso en `ss`; en el segundo caso la secuencia se completa luego de `n2=3` pulsos separados cada `T2` milisegundos en `pout`.



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.
- Rutinas de atención a las interrupciones de **timer** y `ss`.
- Inicialización del sistema y directivas de reserva de memoria. Llamar a `init_otros`, que inicializa componentes no especificados en este problema. Al final saltar a `main_loop`, un bucle infinito ya implementado. Considerar que las únicas interrupciones que pueden ocurrir son las definidas en este enunciado.

Nota: $F_{clk} = 32,768\text{MHz}$; $f_{clk} = 2^{15} \times 1\text{kHz}$; $1\text{ms} = 1 / 1\text{kHz}$

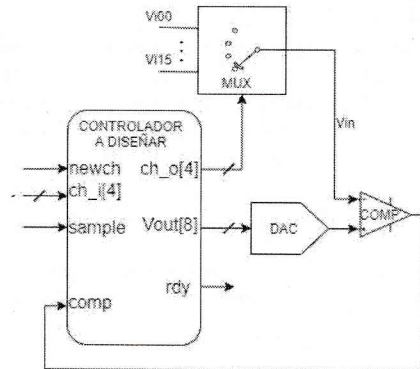


PROBLEMA 2

Un conversor analógico-digital de varios canales utiliza un conversor digital-analógico (DAC), un comparador analógico y un multiplexor analógico conectados como se indica en la figura. El conversor tiene 16 canales de entrada analógica (V_{i00} a V_{i15}) y una salida digital $V_{out}[8]$. La conversión se realiza ajustando el valor de V_{out} en función del valor de salida del comparador para acercarla al valor de la entrada analógica seleccionada como se explica más abajo.

El controlador a diseñar debe manejar las siguientes señales:

- $V_{out}[8]$ (out): maneja entrada de DAC y es el resultado de la conversión.
- rdy (out): indica que se completó una conversión.
- $ch_o[4]$ (out): selecciona la entrada del mux analógico que se conecta al comparador.
- $newch$ (in): con un flanco de subida ordena cambiar el canal seleccionado.
- $ch_i[4]$ (in): válido solamente en el flanco de subida de $newch$.
- $sample$ (in): señal periódica, con cada flanco de subida debe iniciarse una conversión del canal seleccionado.
- $comp$ (in): salida del comparador (1 si DAC es mayor que V_{in} , 0 en caso contrario).



Un flanco de subida en $sample$ ordena el comienzo de una conversión en el canal actualmente seleccionado. Para eso el controlador debe poner un valor inicial en la salida V_{out} y luego, dependiendo de la salida del comparador (entrada $comp$), incrementarlo (+1) o decrementarlo (-1) hasta que $comp$ cambie. En la primera conversión luego de un reset o luego de un pulso en $newch$, el valor inicial debe ser la mitad del rango (0x80). En las siguientes conversiones debe tomarse como valor inicial el resultado de la conversión anterior. P. ej., si el valor inicial de V_{out} es menor que V_{in} , entonces $comp$ valdrá 0. En ese caso el controlador debe iterar sumando 1 al valor de V_{out} hasta que el valor de $comp$ cambie a 1. El valor resultante de V_{out} será el resultado de la conversión.

La salida rdy debe subir cuando se completa una conversión y bajar al comienzo de la siguiente conversión o cuando se ordena cambiar el canal seleccionado.

Un flanco de subida en $newch$ indica que debe cambiarse el canal seleccionado al valor indicado por la entrada ch_i en el momento del flanco. Al inicio luego de un reset debe quedar seleccionado el canal 0.

La señal $sample$ debe usarse para interrumpir al sistema, la rutina de atención a la interrupción debe realizar la conversión como se describe más arriba.

El programa principal debe esperar los flancos en $newch$ y tomar las acciones que correspondan.

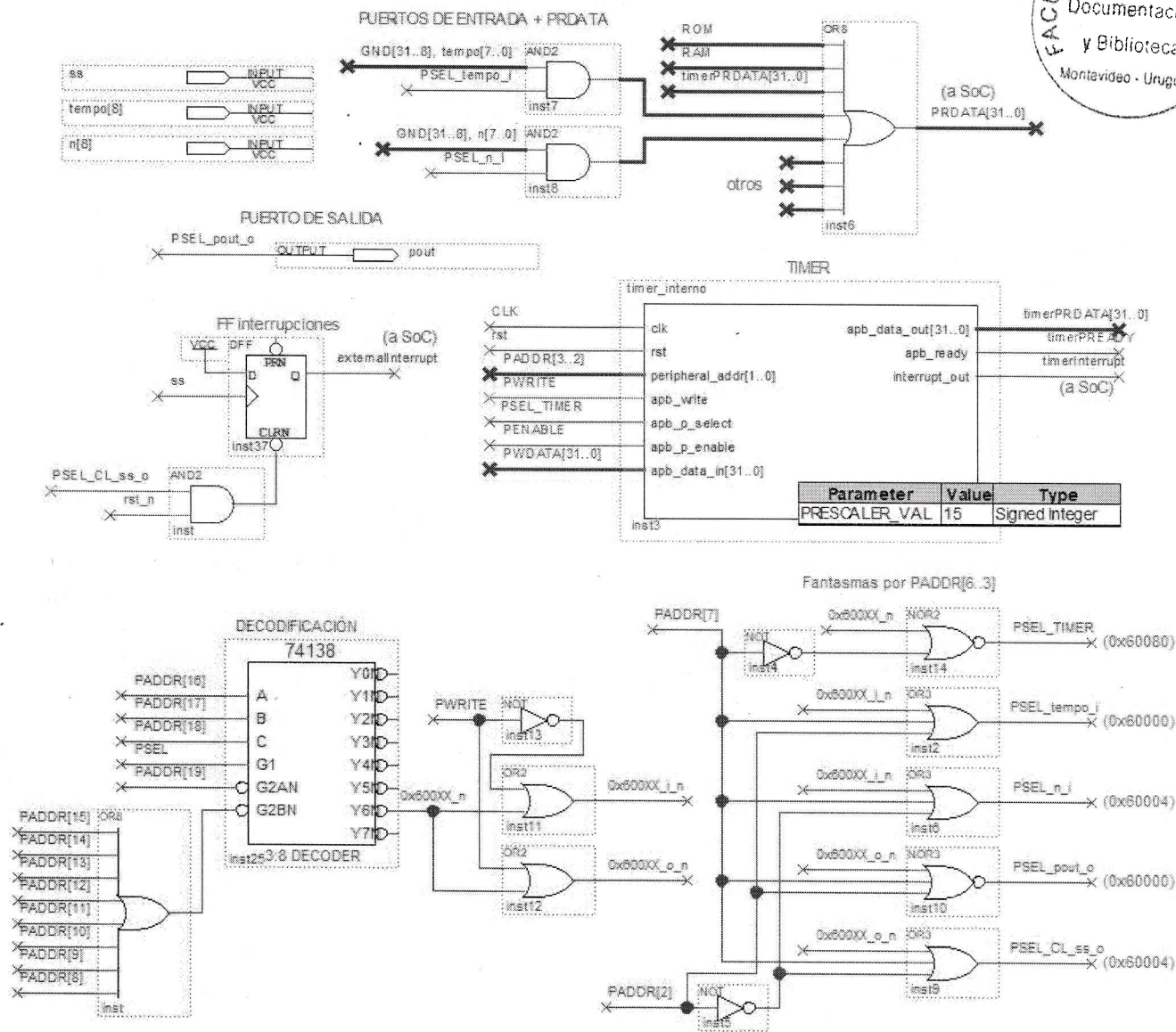
Diseñar completamente el controlador del conversor con un sistema basado o bien en el procesador **Z80** con buses triestado, o bien un procesador **RISC-V**. Se pide:

- Todo el hardware del sistema, incluyendo los puertos, la interrupción y 16KB de ROM y 16KB de RAM mapeados en direcciones consecutivas de memoria.
- Rutina de atención a interrupción provocada por $sample$.
- Loop de programa principal.
- Inicialización del sistema y directivas de reserva de memoria.



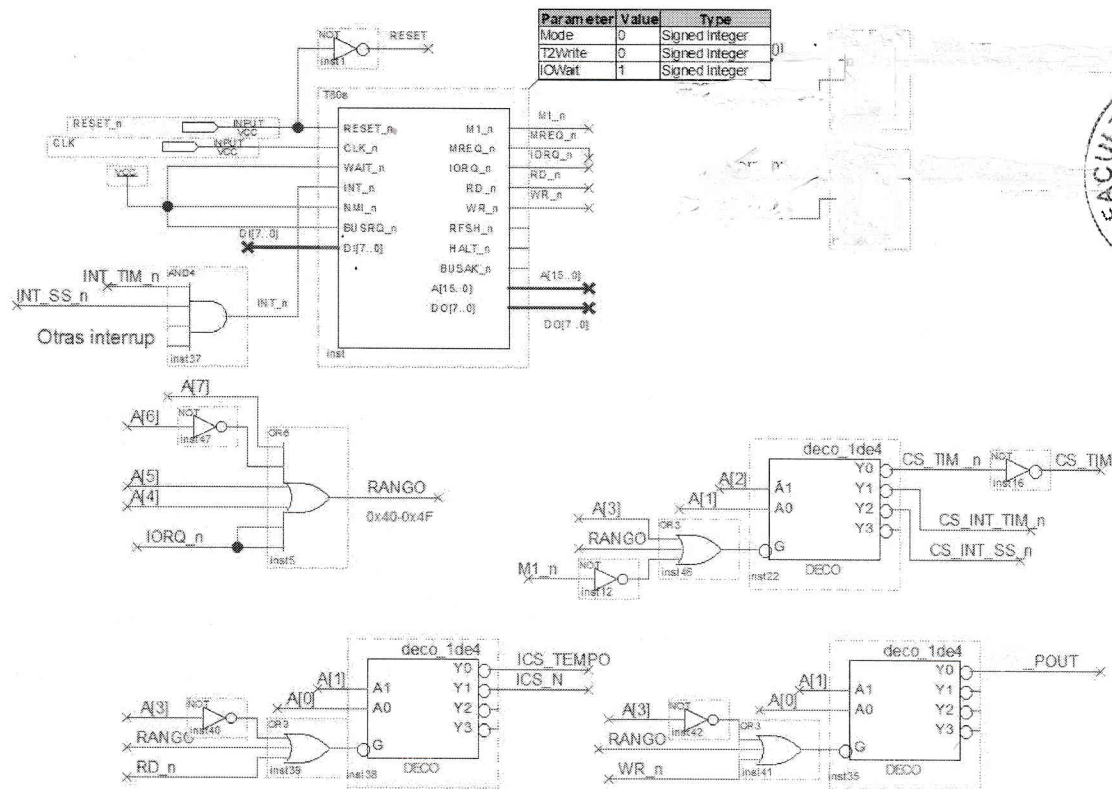
PROBLEMA 1 - RISC-V

a) Hardware



3/16





5/16

b) Rutinas de atención a la interrupción

```
# isr_ss{
#   preservo registros
#   CLR_FF_ss
#   if (next == START){
#       next = STOP
#       vtempo = tempo
#       vpendientes = n-1
#       primer pulso en pout
#       arranco timer
#       mtime_cmp = mtime + vtempo
#       bit_de_timer en mie = 1
#   }else{
#       next = START
#       deshabilitar timer en mie
#   }
#   recupero registros + mret
# }
isr_ss:
#   preservo registros
    addi sp, sp, 24
    sw t0, 0(sp)
    sw t1, 4(sp)
    sw t2, 8(sp)
    sw t3, 12(sp)
    sw t4, 16(sp)
    sw t5, 20(sp)
#   CLR_FF_ss
    li t0, CL_SS
    sw zero, 0(t0)
    li t0, next
    li t1, ESTADO_STOP
    beq t0, t1, stop
#   if (next == START){
start:
#       next = STOP
        sb t1, 0(t0)
#       vtempo = tempo
        li t2, TEMPO
        lbu t2, 0(t2)
        li t3, vtempo
        sb t2, 0(t3)
#       vpendientes = n
        li t2, N
        lb t2, 0(t2)
        addi t2, t2, -1
        li t3, vpendientes
        sb t2, 0(t3)
#       primer pulso en pout
        li t2, POUT
        sw zero, 0(t2)
#       arranco timer
#       mtime_cmp = mtime + vtempo
        li t1, vtempo
        lbu t1, 0(t1)
        li t2, BASE_TIMER
        lw t3, 0(t2)      # mtime_l
        lw t4, 4(t2)     # mtime_h
        addi t1, t3, t1   # mtime_l + vtempo
        blt t3, t1, no_overflow
overflow:
        addi t4, t4, 1
no_overflow:
        li t5, -1
        sw t5, 12(t2)
        sw t1, 8(t2)     # mtime_cmp_l
        sw t4, 12(t2)   # mtime_cmp_h
#       bit_de_timer en mie = 1
        csrsi mie, 0x080
        j fin_isr_ss
#   }else{
stop:
#       next = START
        li t1, ESTADO_START
        sw t1, 0(t0)
#       deshabilitar timer en mie
        csrsi mie, 0x080
fin_isr_ss:
```



```
# recupero registros + mret
    lw t0, 0(sp)
    lw t1, 4(sp)
    lw t2, 8(sp)
    lw t3, 12(sp)
    lw t4, 16(sp)
    lw t5, 20(sp)
    addi sp, sp, -24
    mret

# isr_timer{
#   preservó registros
#   pulso en pout
#   vpendientes--
#   if (vpendientes != 0){
#       reprogramar timer
#       mtime_cmp = mtime_cmp + vtempo
#   }else{
#       deshabilitar timer en mie
#       next = START
#   }
#   recupero registros + mret
# }

isr_timer:
#   preservó registros
    addi sp, sp, 20
    sw t0, 0(sp)
    sw t1, 4(sp)
    sw t2, 8(sp)
    sw t3, 12(sp)
    sw t4, 16(sp)
#   pulso en pout
    li t0, POUT
    sw zero, 0(t0)
#   vpendientes--
    li t0, vpendientes
    lbu t1, 0(t0)
    addi t1, t1, -1
    sb t1, 0(t0)
```

```
beqz t1, fin_n
#   if (vpendientes != 0){
#       reprogramar timer
#       mtime_cmp = mtime_cmp + vtempo
    li t0, vtempo
    lbu t0, 0(t0)
    li t1, BASE_TIMER
    lw t2, 8(t1)      # mtime_cmp_l
    lw t3, 12(t1)     # mtime_cmp_h
    addi t0, t2, t0    # mtime_cmp_l + vtempo
    blt t2, t0, no_overflow_timer

overflow_timer:
    addi t3, t3, 1
no_overflow_timer:
    li t4, -1
    sw t4, 12(t1)
    sw t0, 8(t1)      # mtime_cmp_l
    sw t3, 12(t1)     # mtime_cmp_h
    j fin_isr_timer
#   }else{
fin_n:
#       deshabilitar timer en mie
    csrwi mie, 0x080
#       next = START
    li t0, next
    li t1, ESTADO_START
    lw t1, 0(t0)
fin_isr_timer:
#   recupero registros + mret
    lw t0, 0(sp)
    lw t1, 4(sp)
    lw t2, 8(sp)
    lw t3, 12(sp)
    lw t4, 16(sp)
    addi sp, sp, -20
    mret
```



7/16

c) Inicialización + directivas de
reserva de memoria

```
.equ BASE_TIMER,    0x60080
.equ TEMPO,         0x60000
.equ N,             0x60004
.equ POUT,          0x60000
.equ CL_SS,         0x60004

.equ ESTADO_START,  0
.equ ESTADO_STOP,   1
.equ SP_INI,        0x8000

.globl    main

.text # a partir de 0x00000
main:
## INICIALIZACIÓN
# SP
    li sp, SP_INI
# llamar a init_otros
    call init_otros
# variables: next = START
    la t0, next
    li t1, ESTADO_START
    sb t1, 0(t0)
# clr FF SS por hw (reset)
# mtvec
    la t0, tabla_isr
    ori t0, t0, 1
    csrwr mtvec, t0 # modo vectorizado

# ei sólo extInt
# timerInt luego cada vez
    li t0, 0x800
    csrs mie, t0
# habilito interrupciones
    li t0, 0x8
    csrs mstatus, t0
# salto a main_loop
    j main_loop
```

```
tabla_isr:
    j default_handler # 0
    j default_handler # 1
    j default_handler # 2
    j default_handler # 3
    j default_handler # 4
    j default_handler # 5
    j default_handler # 6
# 7: timerInterrupt
    j isr_timer
    j default_handler # 8
    j default_handler # 9
    j default_handler # 10
# 11: externalInterrupt
    j isr_ss
```

```
default_handler:
    j default_handler
```

```
.data # a partir de 0x04000
.p2align 2, 0
next:    .byte 0
vtempo:  .byte 0
vpendientes: .byte 0

.end
```



8/16

; b) Rutinas de atención a la interrupción

```
; isr_ss{
;   habilito interrupciones
;   preservo registros
;   CLR_FF_ss (no hace falta)
;   if (next == START){
;       next = STOP
;       vtempo = tempo
;       vpendientes = n-1
;       primer pulso en pout
;       arranco timer
;       cte = vtempo, cw con EI
;   }else{
;       next = START
;       deshabilitar interrupciones en
timer
;   }
;   restauro registros
; }
```

```
isr_ss:
;   habilito interrupciones
;   ei
;   preservo registros
push af
;   CLR_FF_ss - innecesario en T80
```

```
ld a, (next)
cp ESTADO_STOP
jr z, stop
;   if (next == START){
start:
;       next = STOP
ld a, ESTADO_STOP
ld (next), a
;       vtempo = tempo
in a, (TEMPO)
ld (vtempo), a
;       vpendientes = n-1
in a, (N)
dec a
ld (vpendientes), a
;       primer pulso en pout
out (POUT), a
;       arranco timer
;       mtime_cmp = mtime + vtempo
ld a, (vtempo)
out (BASE_TIMER+offset_CTE), a
ld a, CW_ON
out (BASE_TIMER+offset_CW), a
jr fin_isr_ss
;   }else{
stop:
;       next = START
ld a, ESTADO_START
ld (next), a
;       deshabilitar interrupciones timer
ld a, CW_OFF
```

out (BASE_TIMER+offset_CW), a

```
fin_isr_ss:
;   recupero registros + mret
pop af
reti

; isr_timer{
;   preservo registros
;   pulso en pout
;   vpendientes--
;   if (vpendientes == 0){
;       deshabilitar interrupciones
timer
;       next = START
;   }
;   restauro y retorno
; }
```

```
isr_timer:
;   ei, preservo registros
EI
push af
;   pulso en pout
out (POUT), a
;   vpendientes--
ld a, (vpendientes) → ld (vpendientes), a
dec a
jr nz, fin_isr_timer
;   if (vpendientes == 0){
;       deshab interrumpciones timer
ld a, CW_OFF
out (BASE_TIMER+offset_CW), a
;       next = START
ld a, ESTADO_START
ld (next), a
fin_isr_timer:
;   restauro y retorno
pop af
reti
```

; c) Inicialización + directivas

```
.equ offset_CTE, 0
.equ offset_CW, 1

; ei, x, sw_reset, auto, pre=15
.equ CW_ON, 10101111B
; di, resto no importa
.equ CW_OFF, 10101111B

.equ off_CIVECTOR, 0
.equ off_CIClr, 1

.equ vector_SS, 0
.equ vector_TIMER, 1

.equ BASE_TIMER, 0x40
.equ BASE_CI_TIMER, 0x42
.equ BASE_CI_SS, 0x44

.equ TEMPO, 0x48
```



9/16

```
.equ N,          0x49
.equ POUT,       0x48
```

```
.equ ESTADO_START, 0
.equ ESTADO_STOP,  1
.equ SP_INI,       0x8000
```

```
.text ; a partir de 0x0000
```

```
init:
```

```
;; INICIALIZACIÓN
```

```
; SP
```

```
    ld sp, SP_INI
```

```
; llamar a init_otros
```

```
    call init_otros
```

```
; variables: next = START
```

```
    ld a, ESTADO_START
```

```
    ld (next), a
```

```
; modo 2, IX
```

```
    im 2
```

```
    ld hl, tabla_isr
```

```
    ld a, h
```

```
    ld i, a
```

```
; controladores de interrupciones
```

```
    ld a, vector_SS
```

```
    out (BASE_CI_SS+off_CIVECTOR), a
```

```
    out (BASE_CI_SS+off_CIClr), a
```

```
    ld a, vector_TIMER
```

```
    out (BASE_CI_TIMER+off_CIVECTOR), a
```

```
    out (BASE_CI_TIMER+off_CIClr), a
```

```
; habilito interrupciones
```

```
    EI
```

```
; salto a main_loop
```

```
    jp main_loop
```

```
.org 0x0100 ;; alineado a página
```

```
tabla_isr:
```

```
    .dw isr_ss
```

```
    .dw isr_timer
```

```
init_otros:
```

```
main_loop:
```

```
    jr main_loop
```

```
.data ; a partir de 0x4000
```

```
next:      .db 0
```

```
vtempo:    .db 0
```

```
vpendientes: .db 0
```

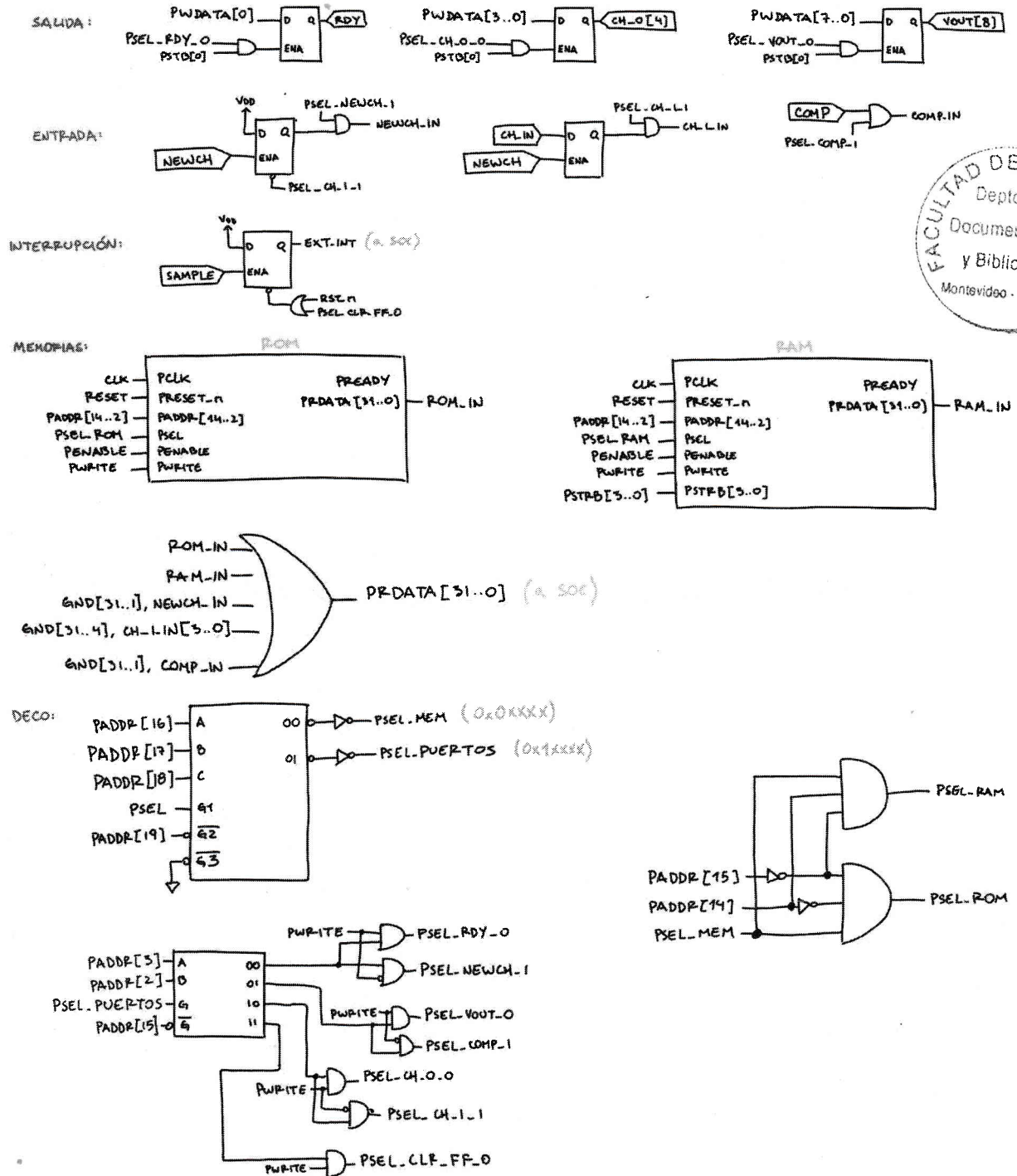
```
.end
```



10/16

PROBLEMA 2 - RISC-V

a) Hardware



11/16



b) Rutina de atención a interrupción

```
# Pseudocódigo de isr_sample
#   isr_sample:
#       preservar registros
#       borrar FF
#       out rdy = 0
#       out vout = vout_actual
#       leer comp
#       while (comp == comp_old):
#           if comp: vout--
#           else: vout++
#       actualizar vout_actual
#       out rdy = 1
#       restaurar registros
#       mret
```

isr_sample:

```
    addi sp, sp, -20
    sw t0, 0(sp)
    sw t1, 4(sp)
    sw t2, 8(sp)
    sw t3, 12(sp)
    sw t4, 16(sp)

    la t1, BASE_IO
    sb zero, CLR_FF(t1) # Borrar FF
    sb zero, RDY(t1)    # RDY = 0
```

```
# VOUT = vout_actual
la t0, vout_actual
lb t0, 0(t0)
sb t0, VOUT(t1)
```

lb t2, COMP(t1)

comp_loop:

```
    lb t4, COMP(t1)
    # En t2 está "comp_old"
    bneq t4, t2, end_conversion
    andi t4, t4, 1
    bnez t4, dec_vout
    addi t0, t0, 1
    j continue
```

dec_vout:

```
    addi t0, t0, -1
```

continue:

```
# Actualizar vout
sb t0, VOUT(t1)
# Actualizar "comp_old"
mv t2, t4
# Continuar hasta que comp cambie
j comp_loop
```

end_conversion:

```
# Conversión finalizada
# Guardar vout para la próxima
la t4, vout_actual
sb t0, 0(t4)
li t0; 1
sb t0, RDY(t1) # RDY = 1
```

Restaurar registros y retornar

```
lw t0, 0(sp)
lw t1, 4(sp)
lw t2, 8(sp)
lw t3, 12(sp)
lw t4, 16(sp)
addi sp, sp, 20
mret
```

c) Programa principal

```
# Pseudocódigo de main_loop
# while 1:
#   leer newch
#   if newch:
#       borrar FF de newch
#       leer ch_i
#       out(ch_o) = ch_i
#       dar vout_actual = DEFAULT_VOUT
#       dar rdy = 0
```

main_loop:

```
    la t0, BASE_IO
    lb t1, NEWCH_I(t0)
    andi t1, t1, 1
    beqz t1, main_loop
```

lb t1, CH_I(t0)

Esto borra el FF de NEWCH por HW

```
sb t1, CH_O(t0)
```

12/16

sb zero, RDY(t0)

li t1, DEFAULT_VOUT

la t0, vout_actual

sb t1, 0(t0)

j main_loop

d) Inicialización y directivas
de reserva de memoria

.equ BASE_IO, 0x1000

.equ NEWCH_I, 0

.equ COMP, 4

.equ CH_I, 8

.equ RDY, 0

.equ VOUT, 4

.equ CH_O, 8

.equ CLR_FF, 12

.equ SP_INI, 0x8000

.equ DEFAULT_VOUT, 0x80

A partir de 0x00000

.text

main: # Stack
li sp, SP_INI

Variables

la t0, vout_actual

li t1, DEFAULT_VOUT

sb t1, 0(t0)

Puertos

li t0, BASE_IO

sb zero, CH_O(t0)

sb zero, RDY(t0)

sb t1, VOUT(t0)

Interrupciones

sb zero, CLR_FF(t0)

la t0, isr_sample

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

A partir de 0x04000

.data

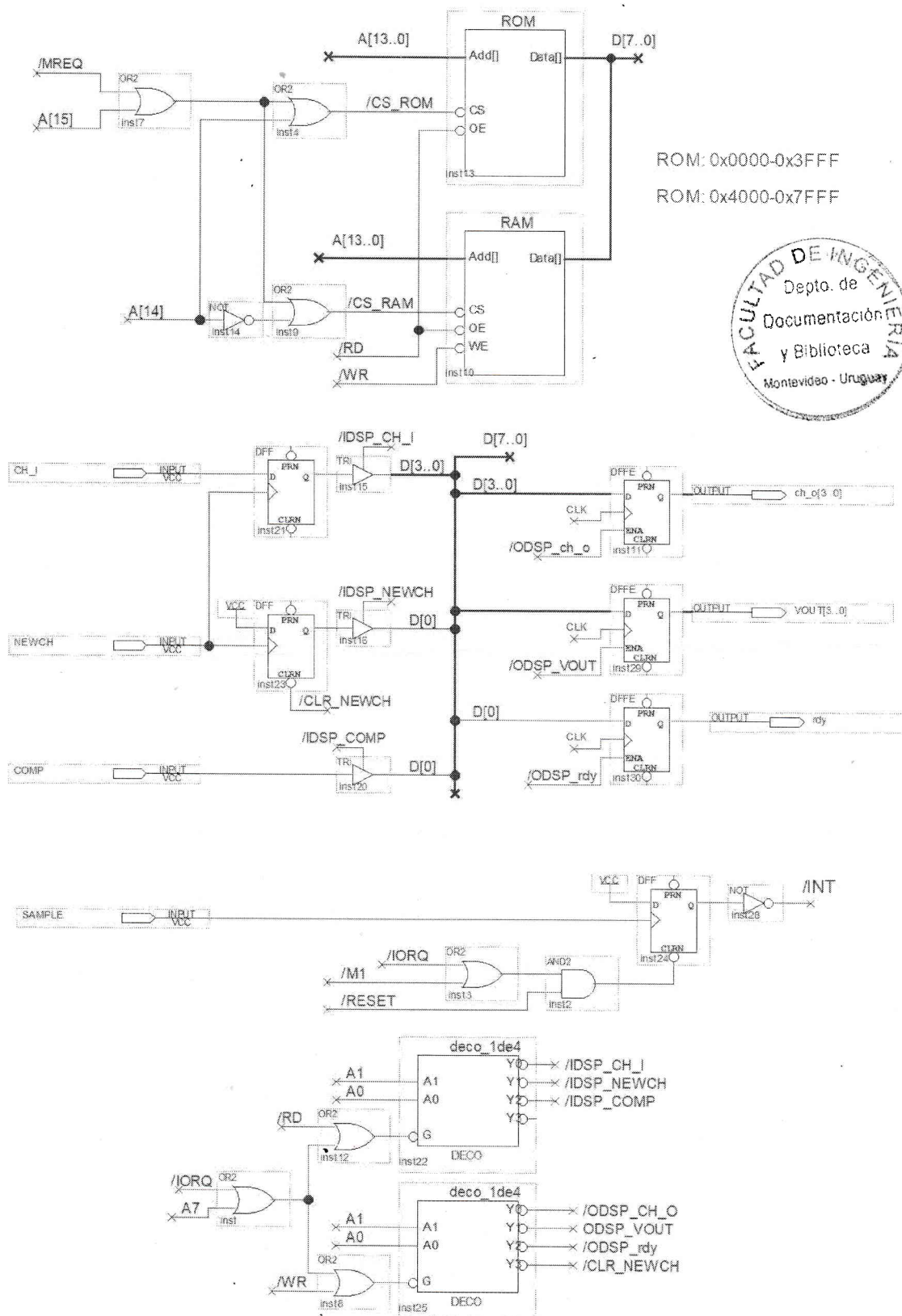
vout_actual: .byte 0x80

.end



13/16

a) Hardware



74/16

b) Rutina de atención a interrupción

```
# Pseudocódigo de isr_sample
#   preservar registros
#   out rdy = 0
#   out vout = vout_actual
#   leer comp
#   si (vout>vin){ ## comp==1
#       repetir vout-- hasta comp==0
#   }else{
#       repetir vout++ hasta comp==1
#   }finsi
#   actualizar vout_actual
#   out rdy = 1
#   restaurar registros
#   mret
```

ORG 0x0038

isr:

```
push af
push hl
push bc
    ;; out rdy = 0
ld a, 0
out (RDY), a
ld hl, vout_actual
ld a, (hl)
out (VOUT), a
ld b, a
in a, (COMP)
bit 0, a
jr z, esmenor
```

esmayor:

```
ld a, b
dec a
ld b, a
out (VOUT), a
in a, (COMP)
bit 0, a
jr nz, esmayor
jr termine
```

esmenor:

```
ld a, b
inc a
ld b, a
out (VOUT), a
in a, (COMP)
```

```
bit 0, a
jr z, esmayor
```

termine:

```
ld (hl), a
ld a, 1
out (RDY), a
pop bc
pop hl
pop af
ei
ret
```

c) Programa principal

```
# while 1:
#   leer newch
#   if newch:
#       borrar FF de newch
#       leer ch_i
#       out(ch_o) = ch_i
#       dar vout_actual = DEFAULT_VOUT
#       dar rdy = 0
```

ppal:

```
in a, (NEWCH)
bit 0, a
jr z, ppal
ld a, 0
out (RDY), a
in a, (CH_I)
out (CH_O), a
ld a, DEFAULT_VOUT
ld (vout_actual), a
jr ppal
```

d) Inicialización y directivas de reserva de memoria

```
.equ CH_I,      0
.equ NEWCH,     1
.equ COMP,      2
```

```
.equ CH_O,      0
.equ VOUT,      1
.equ RDY,       2
.equ CLR_FF,    3
```

```
.equ SP_INI,    0x8000
.equ DEFAULT_VOUT, 0x80
```



15/16

ORG 0x0000

ld sp, SP_INI

IM1

ld a, 0

out (RDY), a

out (CH_O), a

out (CLR_FF), a

ld a, 0x80

ld (valor_actual), a

ei

jr ppal



ORG 0x4000

vout_actual: .db 0x80

.end

16/16