EJERCICIO 1

a) Ecuaciones

Sea **tsu** el tiempo comprendido entre que los datos puestos por el procesador en la entrada D del registro son válidos y llega el flanco de subida a la entrada de reloj. El flanco de subida en la entrada de reloj se da sobre el final del ciclo de escritura a memoria, cuando el primero de {MREQ, WR, o A15_n} se desactive (pase a 1).

Entonces tsu es:

tsu = T1L + T2 + nTw + T3H + min{t12,t32} + tor_min - t53max

t12, t32 refieren a la suida de MREQ y WR respectivamente. No consideramos el camino por direcciones, ya que t45 (hold address a partir de subida de MREQ y WR) asegura que ese camino siempre tendrá más retardo.

tsu >= tsu_req

T1L + T2 + nTw + T3H + min{t12min,t32min} + tor_min - t53max >= tsu_req (A)

Además del requerimiento de tsu respecto al flanco de subida, también se debe asegurar que antes hay un flanco de bajada en la entrada de reloj consecuencia de la activación de MREQ, WR y direcciones. Sea tpw (pulse width) el ancho del pulso a cero en la entrada de reloj del registro. Se debe garantizar que el pulso exista, es decir que tpw sea mayor que 0.

Si la bajada a cero viene por MREQ, entonces:

tpw = T1L + T2 + nTw + T3H - t8max - tor_max + min{t32, t12} + tor_min >= 0 (B)

Si la bajada a cero viene por WR, entonces:

tpw = T2L + nTw + T3H - t30max -tor_max + min{t32, t12} + tor_min >= 0 (C)

No es necesario analizar el camino por direcciones ya que t7 garantiza que las bajadas de WR y MREQ siempre estarán más retardadas respecto a Address.

Sustituyendo los siguientes valores en inecuaciones A,B y C:

T=50ns TiL=TiH=T/2
tor_min = 5ns tor_max = 20ns
t8_max = 40ns
t30_max = 40ns
t12_min = 0ns
t32_min = 0ns
tsu_req = 50ns

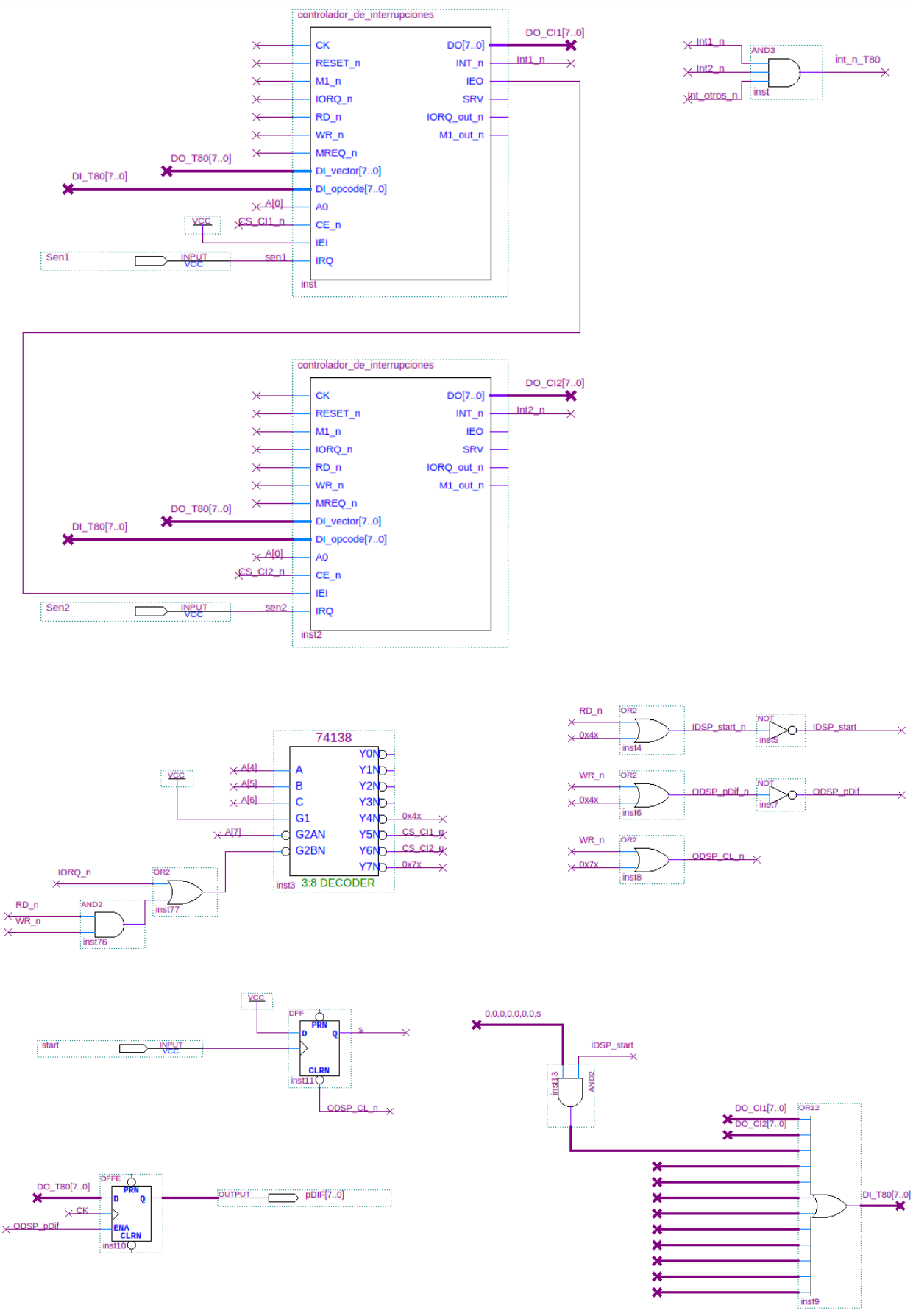
Resulta

A) n >= 0.4
B) n >= -0.9
C) n >= 0.1

Y el menor valor entero de n que cumple las tres es n=1, así que **es necesario insertar 1 Twait.**

EJERCICIO 2

a) Hardware



b)

```
CONT1       EQU 0x50
CONT2       EQU 0x60
VEC1        EQU 0x06
VEC2        EQU 0x08
CL_START    EQU 0x70
START       EQU 0X40
pDIF        EQU 0x40
UMBRAL      EQU 80

        ORG 0X0000

        LD SP, 0                    ; inicializo stack
        LD HL, TABLA_INT            ; inicializo vector I
        LD A, H
        LD I, A
        IM 2                        ; interrupciones modo 2
        LD A, VEC1
        OUT (CONT1),A               ; asigno VEC1 al controlador 1
        OUT (CONT1+1),A             ; borro posibles peticiones pendientes en el CI1
        LD A, VEC2                  ; igual controlador 2
        OUT (CONT2),A
        OUT (CONT2+1),A
        LD HL, RUTINT1              ; inicializo tabla de interrupciones porque se encuentra en RAM
        LD (TABLA_INT+VEC1), HL
        LD HL, RUTINT2
        LD (TABLA_INT+VEC2), HL
        OUT (CL_START), A           ; borro FF de start
        CALL INI_OTROS
        EI                          ; habilito interrupciones
        JP PPAL                     ; salto programa principal
```

c)

```
        ORG 0x0200
RUTINT1:
        EI                          ; habilito interrupciones al comienzo
        PUSH AF                     ; preservo registros
        PUSH HL
        LD HL, DIFERENCIA           ; incremento variable diferencia
        INC (HL)
        POP HL                      ; restauro registros
        POP AF
        RETI                        ; retorno

RUTINT2:
        EI                          ; habilito interrupciones
        PUSH AF                     ; preservo registros
        PUSH HL
        LD HL, DIFERENCIA           ; decremento variable diferencia
        DEC (HL)
        POP HL                      ; restauro registros
        POP AF
        RETI                        ; retorno
```

d)

```
        ORG 0X0100
PPAL:
        IN A, (START)           ; espero a que se genere un pulso en start
        CP 0
        JP Z, PPAL              ; si se genera sigo sino espero
        OUT (CL_START), A       ; borro el FF que captura el pulso
        LD A, 0                 ; pongo Diferencia en 0
        LD (DIFERENCIA), A
NO_UMBRAL:                      ; entro en el loop donde imprimo DIFERENCIA en
                                ; puerto hasta que supere umbral
        LD A, (DIFERENCIA)
        OUT (pDIF), A           ; escribo diferencia en puerto
        CP 0                    ; me fijo si es pos o neg para tomar el valor
                                ; absoluto
        JP P, POS               ; si es positivo no tengo que hacer nada
        NEG                     ; si es negativo tomo el opuesto
POS:
        CP UMBRAL               ; comparo con umbral
        JP M, NO_UMBRAL         ; si es menor me quedo en este loop
        JP PPAL                 ; si es mayor vuelvo a esperar un start

        ORG 0X8000
DIFERENCIA:
        db

        ORG 0x9000
TABLA_INT:
        dw
        dw
        dw
        dw
        dw
```

EJERCICIO 3

a) Hardware

Puertos entrada: BUSY
Salidas: DATA[16], PRINT. Tres posibles soluciones

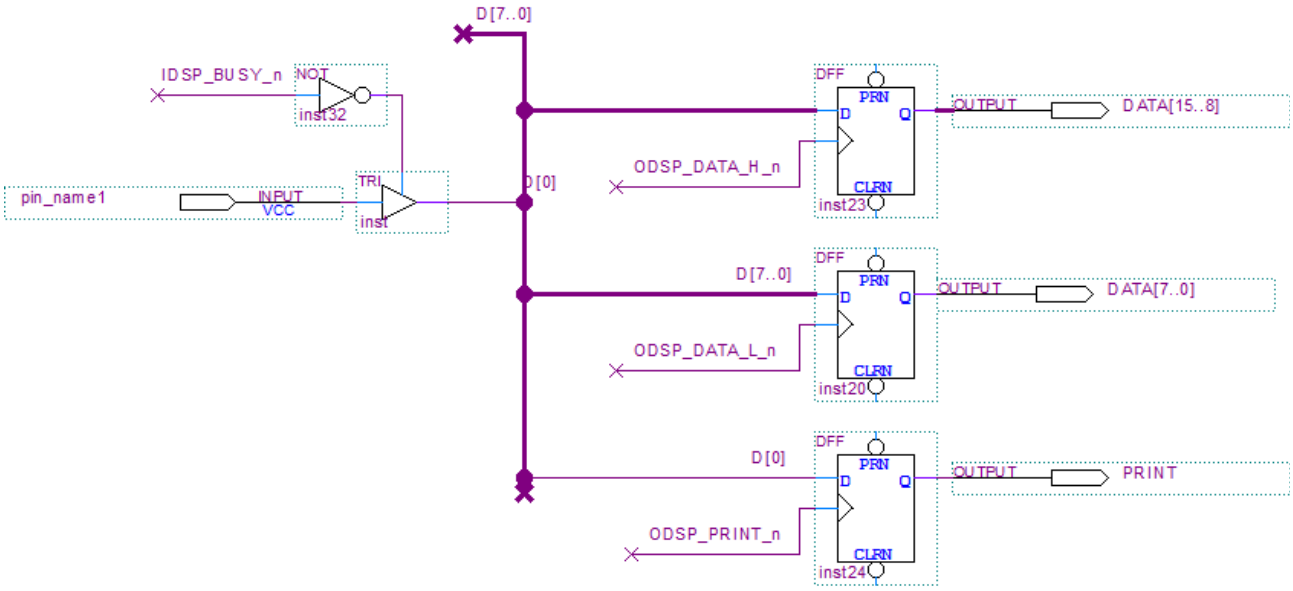
Alternativa 0: dos puertos 8 bits para byte alto y bajo de DATA + un puerto 1 bit para PRINT. Hacen falta dos OUT a PRINT (uno con 0 y otro con 1) para asegurar que hay un flanco. 4 OUTs en total.

Alternativa 1: DATA idem + pulso para PRINT. Alcanza un solo OUT a la dirección asignada a PRINT para asegurar un flanco (3 en total)

Alternativa 2: se puede suprimir el registro de uno de los puertos DATA y conectar directo a PRINT el pulso ODSP correspondiente (por ej. si lo hacemos en el puerto del byte bajo conectamos ODSP_DATA_L_n). En este caso basta con dos OUT en total:

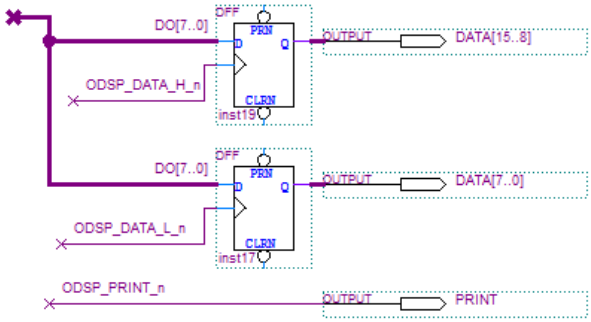
```
ld a, valor_h
out (data_h), a
ld a, valor_l
out (data_l), a
 ; el segundo out da el flanco en PRINT y copia dentro de la impresora el byte bajo desde el bus de datos y el byte alto desde el puerto data_h.
```

ALTERNATIVA 0

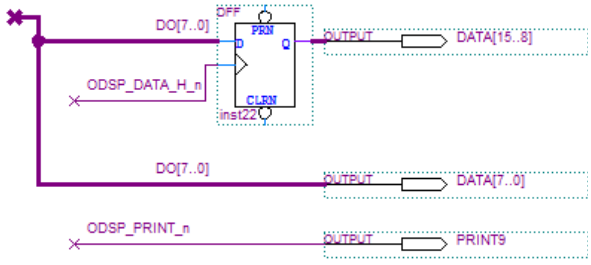


--

ALTERNATIVA 1

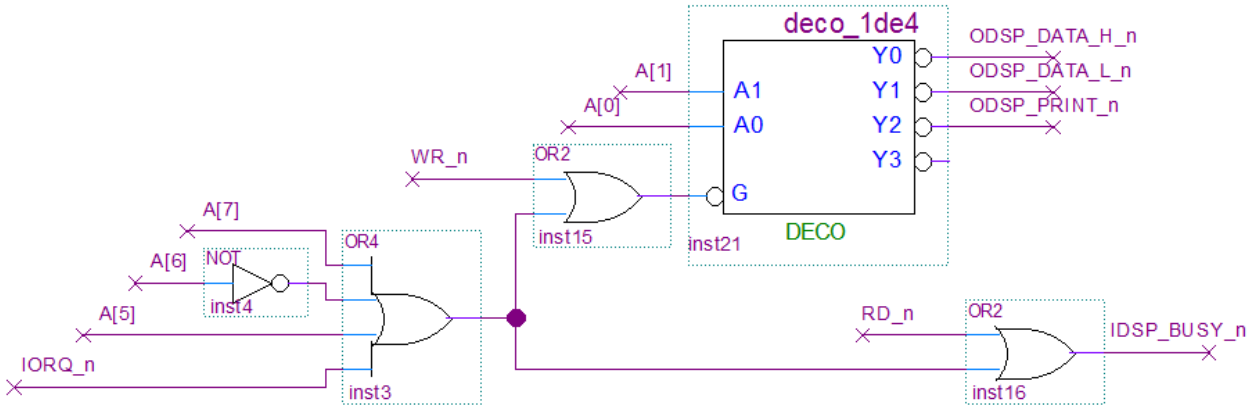


ALTERNATIVA 2

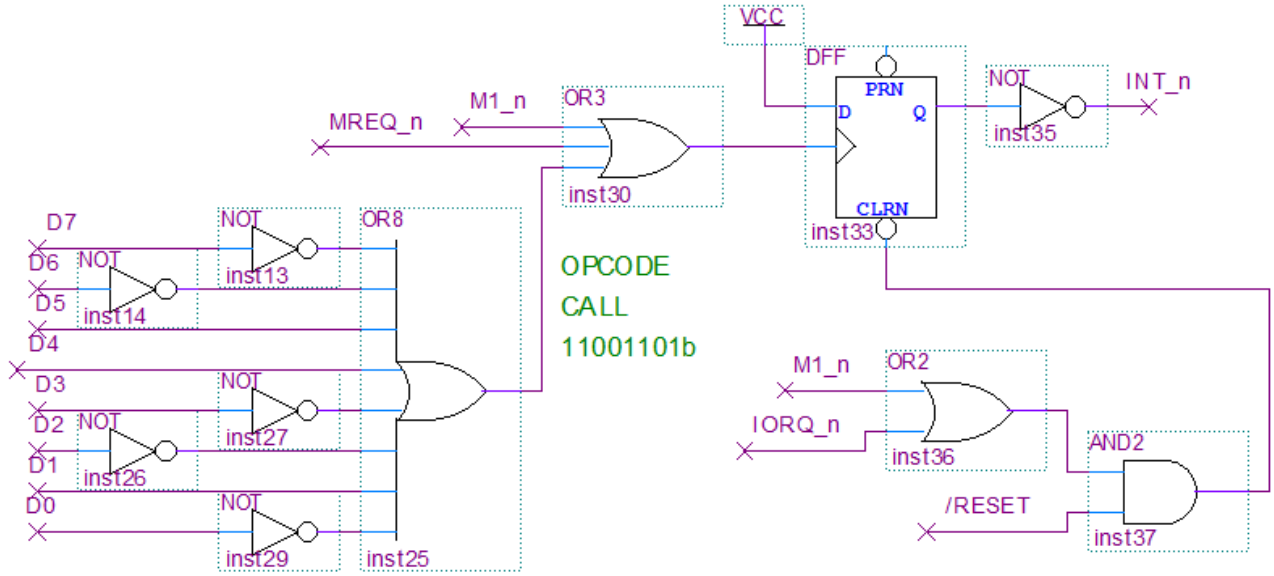


--

Decodificación



Interrupciones



b) Ciclos M

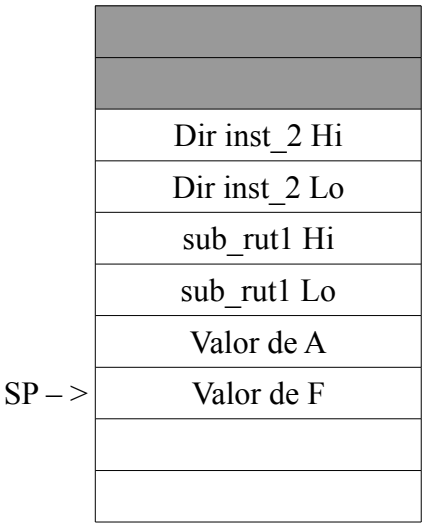
Ciclo	M1	RDm	RDm	WRm	WRm	INTA	WRm	WRm	M1	WRm	WRm
Al stack	--	--	--	Dir inst3 High	Dir inst3 Low		Dir inst2 High	Dir inst2 High		A	F
	CALL					Secuencia reconocimiento int			PUSH AF		

Call guarda en el stack la dirección de retorno, que es la de la instrucción siguiente al CALL.
En el ciclo M1 de CALL se activó la solicitud de interrupción (circuito de la parte a).

Una vez completada la ejecución de CALL el z80 está pronto para ejecutar inst_2, el contador de programa vale PC=sub_rut1. En ese momento se atiende la interrupción. Como se trabaja en modo 1 la secuencia de atención solamente guarda en el stack la dirección de retorno, que es subrut_1

Se pasa a ejecutar la instrucción cargada en la dirección 0x38, que suponemos que es PUSH AF

c) Contenido del stack



---

d) Software

```
data_h      equ   0x40
data_l      equ   0x41
busy        equ   0x40

ORG 0x38
isr:
        push af
        push ix         ; agrego IX respecto
                        ; a parte (c)
        ld ix, 0
        add ix, sp      ; valor de sp en IX
espero1:
        in a, (busy)    ; mientras busy espero
        bit 0, a
        jr nz espero1
        ld a, (IX+5)    ; byte alto de sub_rut
        out (data_h), a
        ld a, (IX+4)    ; byte bajo
        out (data_l), a : supongo Alternativa 2
espero2:
        in a, (busy)    ; mientras busy espero
        bit 0, a
        jr nz espero2
        ld a, (IX+7)    ; byte alto de dir int_2
        out (data_h), a
        ld a, (IX+6)    ; byte bajo
        out (data_l), a
        pop ix
        pop af
        ei
        ret
```