

- 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 debe diseñar un sistema basado en Z80 que implemente parte del módulo de control de un terminal telefónico conectado a una centralita telefónica. El módulo se encarga de supervisar el estado del terminal (REPOSO o ACTIVO) y de detectar la señal de *flash* que puede enviar el terminal a la central como se describe a continuación. La señal flash puede utilizarse por ejemplo para dejar en espera una llamada mientras se atiende una segunda llamada.

La entrada **LINEA_i** indica si el terminal está colgado (**LINEA_i** = 0) o descolgado (**LINEA_i** = 1). Inicialmente el terminal está en el estado REPOSO (**ESTADO_o** = 0) y colgado. Inmediatamente que **LINEA_i** sube a 1 el terminal se pasa al estado ACTIVO (**ESTADO_o** = 1).

En el estado ACTIVO, el cuelgue (**LINEA_i**=0) se interpreta de manera diferente según la duración:

- si **LINEA_i** vuelve a 1 antes de un tiempo $\Delta T_1 = 100\text{ms}$, entonces se interpreta que el terminal envía una señal *flash* y el módulo debe dar un pulso de corta duración en su salida **FLASH_o**,
- si **LINEA_i** permanece en 0 por un tiempo mayor a $\Delta T_2 = 800\text{ms}$, entonces se termina la llamada volviendo al estado REPOSO (**ESTADO_o** = 0),
- si el cuelgue tiene una duración comprendida entre los valores anteriores no se toma ninguna acción.

Para medir los tiempos se utilizará una interrupción periódica. Se cuenta para ello con una señal **TIC** que es una onda cuadrada con período 10ms (error aceptable para la medida de tiempo). La rutina de atención a la interrupción debe escribirse de acuerdo al pseudocódigo del recuadro. Observar que incrementa la cuenta de tiempo y detecta cuando el cuelgue tiene una duración mayor a ΔT_2 para volver al estado REPOSO.

```
Si ( estado == ACTIVO ) {
    si ( LINEA == 0 ) {
        cont++
        si cont >= DELTA2 {
            estado = REPOSO
        }
    }
}
```

Para simplificar el software y garantizar que pulsos de corta duración no son ignorados, el sistema debe incluir hardware para capturar los flancos de subida y bajada en **LINEA_i**.

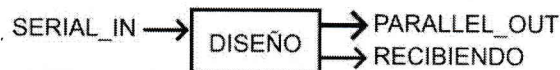
El programa principal será un loop infinito que en función de la ocurrencia de esos flancos y del estado deberá: reiniciar la cuenta de tiempo (cont) cuando sea necesario, detectar el cambio de estado REPOSO a ACTIVO y dar el pulso en la salida **FLASH_o** cuando corresponda. Además, el programa principal debe invocar la subrutina **atiendo_otros()** para atender otras tareas no descritas aquí.

Se pide:

- Hardware. Se usarán chips de ROM y RAM de 8KB mapeados en forma consecutiva en las direcciones más bajas de memoria. Los puertos para las funcionalidades descritas deben ubicarse en el rango 0x80-0x8F del espacio de E/S.
- Rutina de atención a la interrupción.
- Loop de programa principal, incluyendo la llamada a la subrutina **atiendo_otros()**.
- Inicialización del sistema, incluyendo una llamada a la rutina **init_otros()** y directivas de reserva de memoria.

PROBLEMA 2

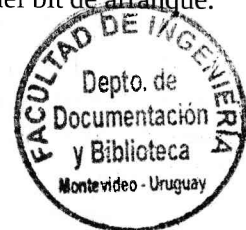
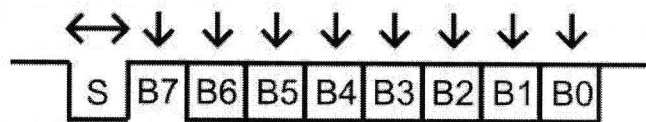
Se deberá diseñar un equipo basado en un T80 que recibe bytes en forma serie por la entrada **SERIAL_IN**, y una vez recibido lo escribe en la salida de 8 bits **PARALLEL_OUT**. El sistema deberá tener además una salida adicional **RECIBIENDO**.



PARALLEL_OUT: Se actualiza con el valor del byte recibido luego de muestrear el bit menos significativo, no se debe esperar el o los bits de parada.

RECIBIENDO: Vale 1 desde que se detecta el bit de arranque hasta que se actualiza **PARALLEL_OUT**, y 0 el resto del tiempo.

A priori no se conoce la velocidad de la comunicación serie, pero se sabe que el bit más significativo del byte a recibir vale siempre 1. Esto permite obtener el tiempo de bit midiendo la duración del bit de arranque. Luego se usa dicho tiempo para muestrear los datos en el centro de cada bit.



Se dispone de un bloque Timer para medir la duración del bit de arranque y luego programarlo para que interrumpa en el centro de cada bit a muestrear. Para minimizar el error en la medición del bit de arranque, el Timer debe arrancar con el flanco de bajada de ese bit.

Se pide:

a) Hardware del sistema incluyendo los puertos y periféricos. Las interrupciones del Timer deben gestionarse con un Controlador de Interrupciones. La memoria se supone implementada: ROM entre 0000h y 3FFFh y RAM entre 8000h y BFFFh.

b) Declaración mediante directivas al ensamblador de :

- Constante **CW_START_BIT**: Palabra de control a cargar en el Timer para que empiece a contar con la bajada de start.
- Constante **CW_DATA_BIT**: Palabra de control a cargar en el Timer para que interrumpa en el momento de leer los bits de datos y paridad.
- Reserva de espacio en memoria para variables que sean necesarias.

d) Rutina de atención a interrupción encargada de la recepción de los bits. Debe almacenar el dato recibido y notificar por algún mecanismo al programa principal para que éste maneje las salidas.

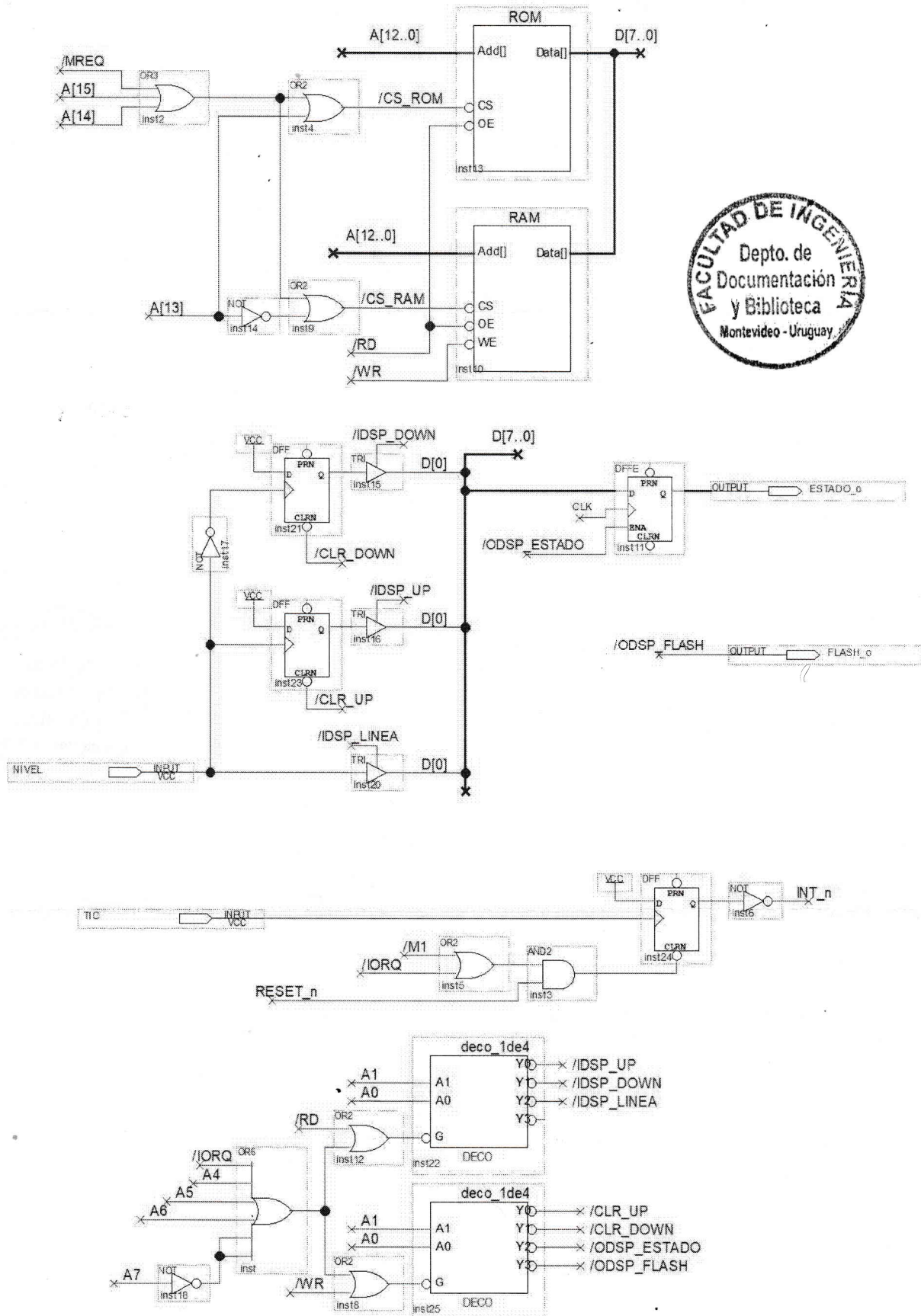
c) Código a ejecutar luego de un reset que inicialice todo lo que sea necesario (periféricos, puertos, variables, sistema de interrupciones). Programa principal encargado de determinar tiempo de bit y manejo de los puertos de salida.

Notas:

- Debe indicarse la ubicación en memoria de todo el código y variables mediante directivas al ensamblador adecuadas.
- Luego de reset la entrada **SERIAL_IN** está en 1 y el primer byte se recibirá luego de un tiempo suficiente para que se inicialice todo el sistema. La salida **PARALLEL_OUT** debe comenzar con valor 00h.
- La frecuencia de reloj es 4MHz. Si el Timer se programa para trabajar con un prescaler de 16, el período de cuenta es 4µs. Se sabe que el tiempo de bit estará entre 210µs y 50µs, por lo que la constante de cuenta será menor a 255.

PROBLEMA 1

a) Hardware



b, c y d) Inicialización, rutina de interrupción y Programa principal ***

;; constantes

FFup equ 0x80

FFdown equ 0x81

linea equ 0x82

clr_FFup equ 0x80

clr_FFdown equ 0x81

p_flash equ 0x82

p_estado_o equ 0x83

DELTA1 equ 10

DELTA2 equ 80

eREPOSO equ 0

eACTIVO equ 1

;; d) inicialización

.text ;; a partir de 0x0000

init:

im 1

;; RAM termina en 0x3FFF

ld SP, 0x4000

ld a, eREPOSO

ld (estado), a

out (p_estado_o), a

ld a, 0

out (clr_FFdown), a

out (clr_FFup), a

ld (cont), a

call init_otros

jp ppal

;; b) Rutina de atención a interrupción

.org 0x0038

; isr(){

; preservar

; si estado == eACTIVO {

; si (LINEA == 0){

; cont++

; si cont >= DELTA2 {

; estado = eREPOSO

; }

; }

; }

; restaurar

; }

isr:

push af

push bc

ld a, (estado)

cp eACTIVO

jr nz, fin_si_ACTIVO

;; si estado == eACTIVO {

in a, (linea)

bit 0, a

jr nz, fin_si_linea

;; si (LINEA == 0){{

;; cont++

;; si cont >= DELTA2 {

;; estado = eREPOSO

ld a, (cont)

inc a

ld b, a

ld (cont), a

ld a, DELTA2

cp b

jp P, fin_si_linea

ld a, eREPOSO

ld (estado), a

fin_si_linea:

fin_si_ACTIVO:

pop bc

pop af

ei

ret



```
;; c) programa principal
; ppal(){
;   forever {
;     if estado == eREPOSO {
;       si up {      ;; o si linea=1
;         estado = eACTIVO
;         cont=0    ;; no es necesario
;       }
;     }else{
;       if down {cont=0}
;       if (up and cont < DELTA1){
;         pulso en salida flash
;       }
;     }
;     call atiando_otros()
;   }
; }
```

```
ppal:
  ld a, (estado)
  cp eREPOSO
  jr nz, else_ppal
  in a, (FFup)
  bit 0, a
  jr nz, finisi_estado
  ;; si (estado == eREPOSO) and up
  out (clr_FFup), a
  ld a, eACTIVO
  ld (estado), a
  out (p_estado_o), a
  ld a, 0
  ld (cont), a
  jr finisi_estado
```

```
else_ppal:
  ;; eACTIVO
  in a, (FFdown)
  bit 0, a
  jr z, finisi_down
  ;; si flanco_bajada {
  out (clr_FFdown), a
  ld a, 0
  ld (cont), a
finisi_down:
  in a, (FFup)
  bit 0, a
  jr z, finisi_up
  ;; si flanco_subida {
  ld a, (cont)
  cp DELTA1
  jp nc, finisi_up
  ;; si cont < DELTA1
  out (clr_FFup), a
  out (p_flash), a
```

```
finisi_up:
finisi_estado:
  call atiando_otros
  jr ppal
```

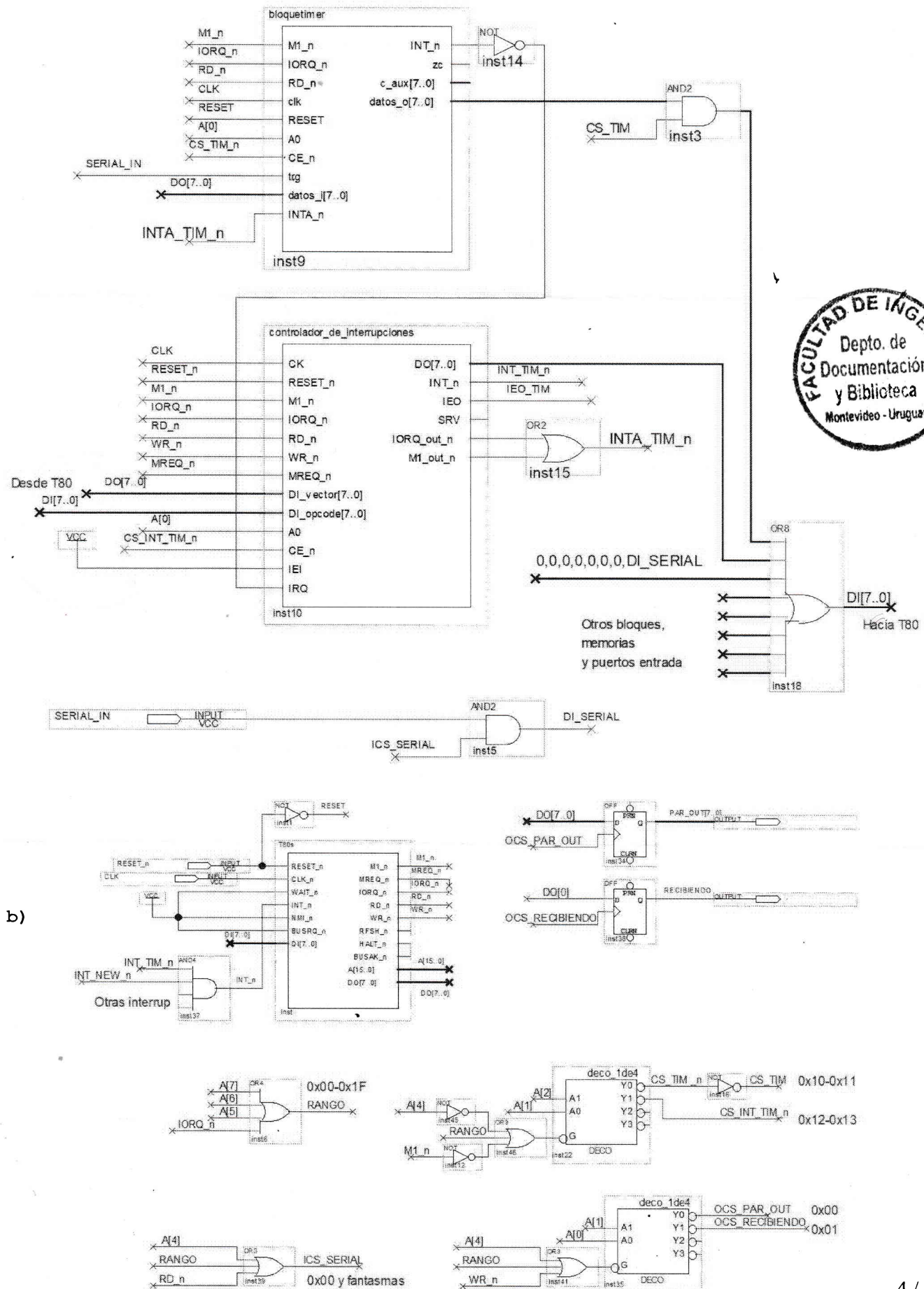
```
;; d) (cont) Reserva de memoria
.data    ;; a partir de 0x2000
cont: db
estado: db

.end
```



PROBLEMA 2

Hardware



```
CW_START_BIT = 0011 0100
; DI, FLANCO_BAJADA, SW_RESET,
; ARRANQUE_TRIGGER, Prescaler 16
```

```
CTE_START_BIT = 250d ;
```

```
CW_DATA_BIT = 1010 0100
; EI, X, SW_RESET, ARRANQUE AUTO,
; Prescaler 16
```

```
org 8000h
N_BIT db 0
BYTE_RECIBIDO db 0
DURACION_BIT db 0
```

```
org 3000h
tabla_int:
    DW rut_int
```

c) Código luego de reset

```
:: constantes
I_SERIAL_IN equ 00h
O_PARALLEL_OUT equ 00h
O_RECIBIENDO equ 01h
```

```
TIMER_CTE equ 10h
TIMER_CW equ 11h
CI_VI equ 12h
CI_BORRA_PET equ 13h
```

```
org 0000h
; Inicializo puertos de salida,
; las variables de memoria de
; inicializan en main
ld A, 0
out (O_PARALLEL_OUT), A
out (O_RECIBIENDO), A
```

```
; CI, modo
im2
ld A, tabla_int / 256
ld I, A
; VI
ld A, 0
```

```
out (CI_VI), A
out (CI_BORRAR_PET), A
```

```
; SP, habilitación de interrupciones
ld SP, C0000h
ei
jp main
```

```
org 0100h
main:
; Armo timer para que empiece
; a contar desde constante conocida
; con flanco de bajada de trigger
ld A, CTE_START_BIT
out (TIMER_CTE), A
ld A, CW_START_BIT
out (TIMER_CW), A
```

```
; resteo la cuenta de bits recibidos
ld A, 0
ld (N_BIT), A
```

```
main_loop:
; espero a que empiece start.
; se que el contador va a empezar
; a contar con el flanco
in A, (I_SERIAL_IN)
bit 0, A
jp NZ, main_loop
```

```
; pongo a 1 salida RECIBIENDO.
ld A, 1
out (O_RECIBIENDO), A
```

```
espero_fin_start:
; el primer bit luego de Start es 1
in A, (I_SERIAL_IN)
bit 0, A
jp Z, espero_fin_start
```

```
llego_fin_start:
; leo timer y resto a la constante
; conocida el valor leído.
; Ese es el tiempo de bit
in A, (TIMER_CTE);
```



```
sub CTE_START_BIT ; A=duración de bit
ld (BIT_DURACION), A
```

```
; la primera vez debo de esperar
```

```
; 1/2 tiempo de bit
```

```
; reprogramo timer
```

```
SRL A ; A=duración de 1/2 bit
```

```
out (TIMER_CTE), A
```

```
ld A, CW_DATA_BIT
```

```
out (TIMER_CW), A
```

```
; espero que el contador valga 8
```

```
; que es cuando se leyó el bit 0
```

```
espero_ultimo_bit:
```

```
ld (N_BIT), A
```

```
cp 8
```

```
jp NZ, espera_ultimo_bit
```

```
; leo byte recibido
```

```
; y lo saco por el puerto
```

```
ld A, (BYTE_RECIBIDO)
```

```
out (PARALEL_OUT), A
```

```
; apago bit de RECEPCION
```

```
ld A, 0
```

```
out (O_RECIBIENDO), A
```

```
; lo primero que hace main es
```

```
; reprogramar timer para que empiece a
```

```
; contar con bajada de trigger
```

```
jp main
```

d) rutina atención interrupción

```
org 0200h
```

```
rut_int:
```

```
ei
```

```
push AF
```

```
; incremento contador
```

```
; de numero de bit que recibo
```

```
; N_BIT = 1 leo bit7
```

```
; N_BIT = 2 leo bit6
```

```
; ...
```

```
; N_BIT = 7 leo bit1
```

```
; N_BIT = 8 leo bit0
```

```
ld A, (N_BIT)
```

```
inc A
```

```
ld (N_BIT), A
```

```
cp 1
```

```
jp Z, primer_bit
```

```
jp dato
```

```
primer_bit;
```

```
; reprogramo timer ahora para que
```

```
; interrumpa en 1 tiempo bit
```

```
ld A, (BIT_DURACION)
```

```
out (TIMER_CTE), A
```

```
ld A, CW_DATA_BIT
```

```
out (TIMER_CW), A
```

```
dato:
```

```
in A, (I_SERIAL_IN)
```

```
srl A ; bit recibido a carry
```

```
ld A, (BYTE_RECIBIDO)
```

```
rl A ; roto izq. resto y carry a bit  
menos significativo
```

```
ld (BYTE_RECIBIDO), A
```

```
fin_rutint:
```

```
pop AF
```

```
reti
```

