PRÁCTICA 2 -RESOLUCION

E/S mediante Consulta de Estado. Ejercicios Integradores.

Parte 1: Entrada/Salida con Luces y Llaves mediante el PIO

1) **Instrucciones de entrada y salida con Luces y Llaves** ★
Los siguientes programas interactúan mediante instrucciones IN y OUT con las luces y llaves a través del PIO. Completar las instrucciones faltantes, e indicar que hace el programa en cada caso

```
PB EQU 31h
CB EQU 33h

ORG 2000H
mov al, 0
out CB, al
mov al, 0Fh
out PB, al
int 0
end
```

```
PA EQU 30h
CA EQU 32h
ORG 1000h
msj db "Apagadas"
ORG 2000H
mov al, 0FFh
out CA, al
in al, PA
cmp al, 0
jnz fin
mov al, 8
mov bx , offset msj
int 7
fin: int 0
end
```

```
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h
ORG 2000h
mov al, 0FFh
out CA, al
mov al, 0
out CB, al
in al, PA
out PB, al
int 0
end
```

De izquierda a derecha:
- Enciende las 4 luces de más a la derecha y apaga el resto
- Verifica si todas las llaves están apagadas, de ser así muestra por pantalla el mensaje "Apagadas".
- Actualiza las luces en base al estado de las llaves.

2) **Uso de las luces y las llaves a través del PIO** ★★
Ejecutar los programas con el simulador VonSim utilizando los dispositivos "Llaves y Luces" que conectan las llaves al puerto PA del PIO y a las luces al puerto PB.

a) **Patrón de Luces Fijo** Escribir un programa que encienda las luces con el patrón 11000011, o sea, solo las primeras y las últimas dos luces deben prenderse, y el resto deben apagarse.

```
        CB EQU 33h
        PB EQU 31h

        ORG 1000h    ; Memoria de datos
patron  db 0C3h      ;1100 0011b

        ORG 2000h  ;Prog principal
        mov al, 0
        out CB, al
        mov al, patron
        out PB, al
        int 0
        end
```

b) **Verificar Llave** Escribir un programa que verifique si la llave de más a la izquierda está prendida. Si es así, mostrar en pantalla el mensaje "Llave prendida", y de lo contrario mostrar "Llave apagada". Solo importa el valor de la llave de más a la izquierda (bit más significativo). Recordar que las llaves se manejan con las teclas 0-7.

```
            CA EQU 32h
            PA EQU 30h

            ORG 1000h   ; Memoria de datos
prendida    db "Llave prendida"
apagada     db "Llave apagada"
fin_apagada db ?


            ORG 2000h   ; Prog principal
            mov al, 0ffh
            out CA, al

            in  al, PA  ; poner en 0 todos los bits menos el más sig
            and al, 80h ; 1000 0000
            cmp al, 0    ; si es 0
            jz  esta_apagada
                                ; esta prendida
            mov bx, offset prendida
            mov al, OFFSET apagada - OFFSET prendida
            jmp fin

esta_apagada: mov bx, offset apagada
            mov al, OFFSET fin_apagada - OFFSET apagada

       fin: int 7 ; imprimir
            int 0
            end
```

c) **Control de luces mediante llaves** Escribir un programa que permite encender y apagar las luces mediante las llaves. El programa no deberá terminar nunca, y continuamente revisar el estado de las llaves, y actualizar de forma consecuente el estado de las luces. La actualización se realiza simplemente prendiendo la luz **i**, si la llave **i** correspondiente está encendida (valor 1), y apagándola en caso contrario. Por ejemplo, si solo la primera llave está encendida, entonces solo la primera luz se debe quedar encendida.

```
        PA EQU 30h
        PB EQU 31h
        CA EQU 32h
        CB EQU 33h

        ORG 2000h
        mov al, 0FFh  ; PA entradas (Micro-commutadores)
        out CA, al
        mov al, 0
        out CB, al    ; PB salidas (Luces)
 poll:  in al, PA
        out PB, al
        jmp poll
        int 0
        end
```

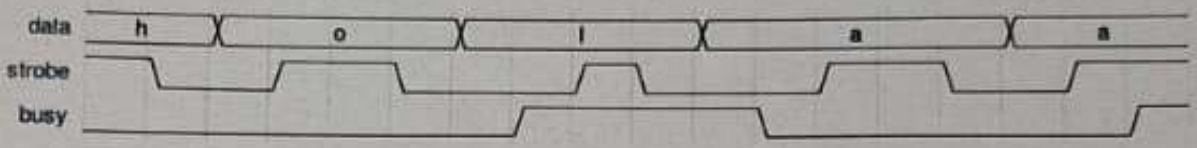
Parte 2: Entrada/Salida con la Impresora mediante el PIO y el HANDSHAKE

1) Protocolo de comunicación Centronics para la impresora

El protocolo Centronics sirve para indicarle a la impresora cuando se está enviando un carácter a imprimir. El protocolo utiliza 3 señales: **busy**, **strobe** y **data**. Para enviar un carácter a la impresora, se debe esperar que la misma esté libre (**busy = 0**), luego poner el carácter en la señal de **data**, y finalmente indicar que hay un carácter para imprimir mediante la generación de un **flanco ascendente** en la señal de **strobe**. Un flanco ascendente sucede cuando la señal cambia de 0 a 1.

La impresora sólo imprime cuando está disponible (**busy** es 0) y recibe una señal de flanco ascendente en la línea de **strobe**. Si no se cumple alguna de estas condiciones, la impresora ignora la línea de **data**.

El siguiente gráfico muestra un ejemplo de las señales:



a) Indica cuáles de las 3 señales son de entrada o de salida.
Rta: data es de salida, strobe es salida y busy de entrada.

b) Si bien la señal de **data** envía el string "holaa", las señales de **strobe** y **busy** que se envían no permiten que se impriman todos los caracteres. Indicar qué imprime la impresora. ¿Cuál es la razón por la cuál algunos carácteres no se imprimen?
Rta: La impresora solo lee los valores cuando busy está en 0 lógico y se genera el flanco ascendente en la línea de strobe. Por ello en el caso del ejemplo solo imprime las letras "oaa", la "h" no se imprime porque no se genera el flanco ascendente del strobe y la "l" no se imprime porque la impresora está ocupada y no ve el flanco ascendente del strobe.

c) El diseño de la impresora tiene 2 señales de sincronización. No obstante, se podría pensar que sólo con la señal **busy** alcanza. ¿Qué sucedería al querer imprimir "aa" si no existiera la señal de **strobe**?
Rta: cuando se tienen 2 o más letras iguales de manera consecutiva para imprimir, , en el ejemplo "aa", si no tenemos algún tipo de señalización para indicarle a la impresora que hay un nuevo valor, la impresora imprimiría solo una vez este valor y no se enteraría que el siguiente debe ser impreso. Por eso se necesita la señal de strobe que le indica a la impresora que el siguiente caracter "a" debe imprimirse.

Nota: para simplificar, la línea de **data** de la impresora se muestra directamente con el carácter a enviar. No obstante, en la realidad **data** consta de 8 líneas de 1 bit que representan el código ascii del carácter.

2) Verificación del bit busy

El siguiente programa tiene como objetivo verificar el bit de busy a través del PIO e imprimir "Ocupada" si la impresora está ocupada y "Libre" de lo contrario.

a) Complete el código para que el programa funcione correctamente.

```
PA EQU 30h
CA EQU 32h
ORG 1000h
si db "Ocupada"
no db "Libre"

ORG 2000H
mov al, 0FDh
out CA, al
in al, PA
```

```
          and al,1
          jnz ocupada
          mov bx, offset no
          mov al, 5
          int 7
          jmp fin
ocupada:  mov bx, offset si
          mov al, 7
          int 7
fin:      int 0
          end
```

b) Modifique el código para que el programa no imprima "Ocupada". En lugar de eso, el programa debe esperar a que el bit de busy sea 0 usando consulta de estado. Cuando eso suceda, imprimir "Libre" y terminar el programa.

```
PA EQU 30h
CA EQU 32h
ORG 1000h
no db "Libre"

ORG 2000H
mov al, 0FDh
out CA, al
```

```
poll: in al, PA
      and al,1
      jnz poll
      mov bx, offset no
      mov al, 5
      int 7
      int 0
      end
```

3) **Subrutina para el envío del carácter y la señal de Strobe** ☆☆

El envío de la señal de strobe se puede modularizar en una subrutina para ser reutilizado en distintas ocasiones. Implementar una subrutina **flanco_ascendente** que envía el flanco ascendente (un 0 y luego un 1) a través del **strobe**. Asumir que el PIO ya está configurado correctamente para comunicarse con la impresora.

```
                PA EQU 30h
;Subrutina de generación de pulso 'Strobe'
                ORG 4000H
flanco_ascendente: in al, PA
                and al, 0FDh
                out PA, al
                in al, PA
                or al, 02h
                out PA, al
                ret
```

4) **Uso de la impresora a través de la PIO** ☆☆

Ejecutar los programas configurando el simulador VonSim con los dispositivos "Impresora (PIO)". En esta configuración, el puerto de datos de la impresora se conecta al puerto PB del PIO, y los bits de busy y strobe de la misma se conectan a los bits 0 y 1 respectivamente del puerto PA.

a) **Imprimir letra fija** Escribir un programa para imprimir la letra "A" utilizando la impresora a través de la PIO. Recordar que el programa deberá esperar hasta que el bit de busy esté apagado, luego enviar el carácter, y luego enviar el flanco ascendente a través de la señal de strobe. Modularizar el programa en 4 subrutinas:

ini_pio: Inicializa el PIO configurando los registros CA y CB según corresponde a strobe, busy y puerto de datos.

poll: Consulta el bit busy de la impresora, e itera hasta esté en 0. Cuando está en 0 la subrutina retorna sin devolver ningún valor

flanco_ascendente: Igual que la subrutina implementada en el ejercicio anterior

imprimir_caracter: Recibe un carácter a imprimir en AL, y utilizando **poll** y **flanco_ascendente**, realiza todos los pasos necesarios para enviar a la impresora el carácter.

```
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

    ORG 1000h; Memoria de datos
CHAR DB "A"

    ORG 3000h
ini_pio:  mov al, 0FDh   ; strobe salida (0), busy entrada (1)
          out CA, al
          mov al, 0      ; puerto de datos todo salida
          out CB, al
          ret

    ORG 3100h
poll: in al, PA
      and al,1
      jnz poll
      ret

          ORG 3200h
flanco_ascendente: in al, PA          ; primero envio 0, luego 1
          and al, 0FDh
          out PA, al
          in al, PA
          or al, 02h
          out PA, al
          ret

          ORG 3300h
imprimir_caracter: push ax
          call poll
          pop ax
          out PB, al
          call flanco_ascendente
          ret

          ORG 2000h      ; Prog principal
          call ini_pio
          mov al, CHAR
          call imprimir_caracter
          int 0
          end
```

b) **Imprimir mensaje fijo** Escribir un programa para imprimir el mensaje "ORGANIZACION Y ARQUITECTURA DE COMPUTADORAS" utilizando la impresora a través de la PIO. Utilizar subrutina **imprimir_caracter** del inciso anterior.

```
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

        ORG 1000h
msj DB "ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS"
fin DB ?

        ORG 3000h
ini_pio: mov al, 0FDh   ; strobe salida (0), busy entrada (1)
        out CA, al
        mov al, 0       ; puerto de datos todo salida
        out CB, al
        ret

        ORG 3100h
poll:   in al, PA
        and al,1
        jnz poll
        ret

        ORG 3200h
strobe0: in al, PA          ; envio 0
        and al, 0FDh
        out PA, al
        ret

        ORG 3300h
strobe1: in al, PA          ; envio 1
        or al, 02h
        out PA, al
        ret

        ORG 3500h
imprimir_caracter: push ax
        call poll
        pop ax
        out PB, al
        call strobe1
        call strobe0
        ret

        ORG 2000h       ; Prog principal
        call ini_pio
        call strobe0
        mov bx, offset msj
        mov cl, offset fin - offset msj
lazo:   mov al, [bx]
        call imprimir_caracter
        inc bx
        dec cl
        jnz lazo
        int 0
        end
```

c) **Imprimir mensaje leído** Escribir un programa que solicita el ingreso de cinco caracteres por teclado y los envía de a uno por vez a la impresora a través de la PIO a medida que se van ingresando. No es necesario mostrar los caracteres en la pantalla. Utilizar la subrutina **imprimir_caracter** del inciso anterior.

```
                PA EQU 30h
                PB EQU 31h
                CA EQU 32h
                CB EQU 33h
                   ORG 1000h
           num_car DB 5
               car DB ?

                  ORG 3000h
          ini_pio: mov al, 0FDh   ; strobe salida (0), busy entrada (1)
                   out CA, al
                   mov al, 0      ; puerto de datos todo salida
                   out CB, al
                   ret
                   ORG 3100h
             poll: in al, PA
                   and al,1
                   jnz poll
                   ret
                   ORG 3200h
          strobe0: in al, PA          ; envio 0
                   and al, 0FDh
                   out PA, al
                   ret
                   ORG 3300h
          strobe1: in al, PA          ; envio 1
                   or al, 02h
                   out PA, al
                   ret

                   ORG 3500h
imprimir_caracter: push ax
                   call poll
                   pop ax
                   out PB, al
                   call strobe1
                   call strobe0
                   ret

                  ORG 2000h        ; Prog principal
                   call ini_pio
                   call strobe0
                   mov bx, offset car
                   mov cl, num_car
             lazo: int 6
                   mov al, [bx]
                   call imprimir_caracter
                   inc bx
                   dec cl
                   jnz lazo
                   int 0
                   end
```

5) **Uso de la impresora a través del HAND-SHAKE** ★★

El HANDSHAKE es un dispositivo diseñado específicamente para interactuar con la impresora mediante el protocolo centronics. Por este motivo, no requiere enviar la señal de flanco ascendente manualmente; en lugar de eso, al escribir un valor en su registro **DATA**, el mismo HANDSHAKE manda el flanco ascendente. Ejecutar los programas configurando el simulador VonSim con los dispositivos "Impresora (Handshake)"

a) Escribir un programa que imprime "INGENIERIA E INFORMATICA" en la impresora a través del HAND-SHAKE. La comunicación se establece por **consulta de estado** (polling). Para ello, implementar la subrutina **imprimir_caracter_hand** que funcione de forma análoga a **imprimir_caracter** pero con el handshake en lugar del PIO.

```
DATO EQU 40h
ESTADO EQU 41h

    ORG 1000h ; Memoria de datos
msj DB "INGENIERIA E INFORMATICA"
fin DB ?

    ORG 3000h
poll: in al,ESTADO
    and al,1
    jnz poll
    ret
    ORG 3200h
imprimir_caracter_hand: push ax
    call poll
    pop ax
    out DATO, al
    ret

    ORG 2000H              ; Prog principal
    in al, ESTADO
    and al, 7Fh        ; modo consulta de estado
    out ESTADO, al
    mov bx, offset msj
    mov cl, offset fin - offset msj
lazo: mov al, [bx]
    call imprimir_caracter_hand
    inc bx
    dec cl
    jnz lazo
    int 0
    end
```

b) ¿Qué diferencias encuentra con el ejercicio anterior? ¿Cuál es la ventaja en utilizar el HAND-SHAKE con respecto PIO para comunicarse con la impresora? ¿Esta ventaja también valdría para otros dispositivos, como las luces?

Rta: El HAND-SHAKE fue diseñado específicamente para las impresoras centronics, tiene la "inteligencia" para hacer todas las tareas necesarias de comunicación con la impresora. Automatiza la señal de strobe; es decir, no se requiere generación de pulso de strobe, se encarga el propio HAND. Además trabaja en modo interrupción (lo veremos más adelante). El PIO puede conectarse a otros dispositivos genéricos. El HAND trabaja específicamente con la impresora.

Parte 3: Entrada/Salida con dispositivos genéricos a través del PIO

1) PIO con dispositivos genéricos ☆☆☆

Si bien el PIO solo permite conectarse a las luces, interruptores y la impresora en este simulador, al ser un dispositivo programable podríamos utilizarlo para interactuar con nuevos dispositivos que no cuenten con un hardware especializado para interactuar con ellos. En los siguientes incisos, te proponemos escribir programas que usen nuevos dispositivos ficticios, pero con un protocolo de comunicación definido. **Nota**: los dispositivos nuevos no están implementados en el simulador, pero podés probar el funcionamiento del programa utilizando las luces y las llaves haciendo de cuenta que sirven como entrada y salida del dispositivo.

a) Escribir un programa que, utilizando el puerto PB del PIO, envíe la cadena de caracteres "UNLP" a un dispositivo nuevo. Este dispositivo debe recibir la cadena de a un carácter a la vez. Para distinguir entre caracteres, el dispositivo necesita que antes de cada carácter nuevo se envíe el código ASCII 0. Además, para indicar que se finalizó el envío, debe mandarse el código ASCII 255.
Ejemplo: Para transmitir la cadena "UNLP", debe enviarse: 0, "U", 0, "N", 0, "L", 0, "P", 255.

```
    PB EQU 31h
    CB EQU 33h

       ORG 1000H
cadena db "UNLP"
   fin db ?

       ORG 2000H

       mov al, 0   ;config PIO
       out CB, al

       mov bx, offset cadena
       mov cl, offset fin - offset cadena

loop:  mov al, 0
       out PB, al

       mov al, [bx]
       out PB, al

       inc bx
       dec cl
       jnz loop

       mov al, 255
       out PB, al

       int 0
       end
```

b) Escribir un programa que reciba una cadena de caracteres de un dispositivo nuevo conectado a los puertos PA y PB. Este dispositivo envía la cadena de a un carácter a la vez. Para que el dispositivo sepa cuándo la CPU está lista para recibir un carácter, la CPU deberá enviar el valor FF al dispositivo a través del puerto PB. Luego, la CPU deberá leer desde el puerto PA, y volver a enviar el valor FF al dispositivo. La transmisión termina cuando se recibe el código ASCII 0.

Ejemplo para recibir la cadena "ASD": CPU envía el FF por PB → CPU recibe "A" por PA → CPU envía el FF por PB → CPU recibe "S" por PA → CPU envía el FF por PB → CPU recibe "D" por PA → CPU envía el FF por PB → CPU recibe 0 por PA

```
        PA EQU 30h
        PB EQU 31h
        CA EQU 32h
        CB EQU 33h

           ORG 3000h
config_pio:mov al, 0
           out CB, al
           mov al, 255
           out CA, al
           ret

         ORG 2000h
         call config_pio
   loop: mov al, 0FFh
         out PB, al
         in al, PA
         ; Proceso valor….
         cmp al, 0
         jnz loop
         int 0
         end
```

Parte 4: Ejercicios integradores o tipo parcial

1) Juego de Luces con Rotaciones ★★★

Escribir un programa que encienda una luz a la vez, de las ocho conectadas al puerto paralelo del microprocesador a través de la PIO, en el siguiente orden de bits: 0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-..., es decir, 00000001, 00000010, 00000100, etc. El programa nunca termina. Para ello, deberá utilizar las subrutinas **ROTAR_IZQ** y **ROTAR_DER_N**, que le permitirán rotar el bit de estado de las luces y generar el juego correspondiente.

Las rotaciones son operaciones que, aunque no parezca, tienen muchas utilidades, como dividir o multiplicar por dos de forma rápida, o manipular los bits de un byte, pero no hay una instrucción que las implemente directamente.

ROTARIZQ: Escribir una subrutina ROTARIZQ que aplique **una** rotación hacia la izquierda a los bits de un byte almacenado en la memoria. Dicho byte debe pasarse por valor desde el programa principal a la subrutina a través de registros y por referencia. No hay valor de retorno, se modifica directamente la celda de memoria referenciada.

> **Pista:** Una rotación a izquierda de un byte se obtiene moviendo cada bit a la izquierda, salvo por el último que se mueve a la primera posición. Por ejemplo al rotar a la izquierda el byte **1**0010100 se obtiene 00101001, y al rotar a la izquierda **0**1101011 se obtiene 1101011**0**.
>
> Para rotar a la izquierda un byte, se puede multiplicar el número por 2, o lo que es lo mismo sumarlo a sí mismo. Por ejemplo (verificar):
>
> ```
>   01101011
> + 01101011
>   11010110 (CARRY=0)
> ```
>
> Entonces, la instrucción add ah, ah permite hacer una rotación a izquierda. No obstante, también hay que tener en cuenta que si el bit más significativo es un 1, el carry debe llevarse al bit menos significativo, es decir, se le debe sumar 1 al resultado de la primera suma.
>
> ```
>   10010100
> + 10010100
>   00101000 (CARRY=1)
> + 00000001
>   00101001
> ```

ROTARIZQ_N: Usando la subrutina ROTARIZQ del ejercicio anterior, escriba una subrutina ROTARIZQ_N que realice N rotaciones a la izquierda de un byte. La forma de pasaje de parámetros es la misma, pero se agrega el parámetro N que se recibe por valor y registro. Por ejemplo, al rotar a la izquierda 2 veces el byte **10**010100, se obtiene el byte 01010010.

ROTARDER_N: • Usando la subrutina ROTARIZQ_N del ejercicio anterior, escriba una subrutina ROTARDER_N que sea similar pero que realice N rotaciones hacia la **derecha**.

> **Pista**: Una rotación a derecha de N posiciones, para un byte con 8 bits, se obtiene rotando a la izquierda 8 - *N* posiciones. Por ejemplo, al rotar a la derecha 6 veces el byte 10**010100** se obtiene el byte **01010010**, que es equivalente a la rotación a la izquierda de 2 posiciones del ejemplo anterior.

```
        PB EQU 31h
        CB EQU 33h

           ORG 3000h
 ; en al tengo la cadena de 8 bits a rotar
 ROTARIZQ: add al, al
           adc al, 0
           ret

 ; en cx tengo la cantidad de rotaciones a realizar cx=N
ROTARIZQ_N: push cx
           and cx, 7  ; hago N MOD 8, para no hacer rotaciones de más
sigo_rotizq:cmp cx, 0
           jz fin_rotizq
           call ROTARIZQ
           dec cx
           jmp sigo_rotizq
fin_rotizq: pop cx
           ret

 ; en cx tengo la cantidad de rotaciones a realizar cx=N
ROTARDER_N: push cx
           neg cx   ; truco para no usar otro registro
           add cx, 8 ; 8 - cx = -cx + 8
           call ROTARIZQ_N   ; hago 8 - N rotaciones
           pop cx
           ret

           ORG 2000H
           mov al, 0
           out cb, al
           mov al, 1 ; primera luz a mostrar
           mov cx, 1 ; cantidad de rotaciones a derecha
    loop: out PB, al
          call ROTARIZQ
          cmp al, 128
          jnz loop
loop_der: out PB, al
          call ROTARDER_N
          cmp al, 1
          jnz loop_der
          jmp loop
          int 0
          end
```

2) CriptoLlaves (Llaves, Luces): ☆☆☆

Escriba un programa de VonSim que permita jugar al CriptoLlaves. El usuario debe adivinar un patrón secreto de 8 bits que está almacenado en la memoria del programa. Para ello, debe manipular las llaves hasta que el patrón de bits de las llaves sea exactamente igual al del patrón secreto. Como ayuda para el usuario, si el estado de una llave acierta al bit correspondiente, el programa debe prender el led correspondiente. Por ejemplo, si el patrón es 0100 0101 y las llaves están en el estado 1110 0100, deben prenderse las luces de los bits 1, 2, 5, 6 y 6. Como no acertó a todos los bits, el usuario no ha adivinado el patrón y debe continuar jugando.

El programa termina cuando el usuario acierta todos los bits del patrón, mostrando en pantalla el mensaje "GANASTE".

```
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

        org 1000h
clave db 10101111b
ganaste db "GANASTE"

        org 3000h
config_pio: mov al, 0
        out CB, al
        mov al, 255
        out CA,al
        ret

        org 2000h
        call config_pio

loop: in al, PA
        xor al, clave
        not al
        out PB, al
        cmp al, 11111111b
        jnz loop

        mov bx, offset ganaste
        mov al, 7
        int 7
        int 0
        end
```

3) Llaves y mensajes ★★

a) Escribir un programa que continuamente verifique el estado de las llaves. Si están prendidas la primera y la última llave, y el resto están apagadas (patrón 10000001), se debe mostrar en pantalla el mensaje "ACTIVADO". En caso contrario, no se debe mostrar nada.

```
PA EQU 30h
CA EQU 32h
patron EQU 10000001b

      org 1000h
cadena db "activado"
fin_cadena db ?

      org 2000h
      mov al, 255  ; PA:entradas
      out CA, al
      mov bx, offset cadena
loop: in al, PA
      cmp al, patron  ;verifico que sea el mismo patrón
      jnz loop
      mov al, offset fin_cadena - offset cadena
      int 7
      jmp loop
      int 0
      end
```

b) Modificar a) para que el mensaje se imprima una sola vez cada vez que detecte ese patrón de bits. Por ejemplo, si el programa lee la siguiente secuencia de patrones:
00010101 → 10010000 → **10000001** → **10000001** → **10000001** → 10010001 → **10000001** → **10000001** → 10010101 → 01110001
Entonces solo deberá imprimir "ACTIVADO" dos veces

```
PA EQU 30h
CA EQU 32h
patron EQU 10000001b

        org 1000h
cadena db "activado"
fin_cadena db ?

        org 2000h
        mov al, 255  ; PA:entradas
        out CA, al
        mov bx, offset cadena
loop:   in al, PA
        cmp al, patron  ;verifico que sea el mismo patrón
        jnz loop
        mov al, offset fin_cadena - offset cadena
        int 7
loop2:  in al, PA
        cmp al, patron
        jz loop2
        jmp loop
        int 0
        end
```

4) Luces, llaves y opciones ★★★ Escribir un programa que deberá utilizar las luces y llaves. El programa revisa el estado de las llaves, y evalúa estos tres casos:

A. Verificar si todas llaves están apagadas. Si es así, mostrar en pantalla el mensaje "Fin de programa" y finalizar el mismo. Caso contrario, hacer tanto B como C.
B. Actualizar las luces a su estado opuesto. Por ejemplo, si las llaves están en el estado "00011010" las luces tendrán el estado "11100101".
C. Si la primera llave (la del bit menos significativo) está prendida, mostrar en pantalla el mensaje "Arquitectura de Computadoras: ACTIVADA".

El programa sólo termina en el caso A. En los casos B y C, debe volver a revisar el estado de las llaves y evaluar los 3 casos otra vez.

Las funciones "A", "B" y "C" deben implementarse utilizando tres subrutinas independientes. La subrutina A debe devolver 1 si hay que finalizar el programa y 0 de lo contrario.

```
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

   ORG 1000H
MSJ db "Arquitectura de Computadoras: ACTIVADA"
MSJ_FIN db "Fin de Programa"
FIN db 0

   ORG 3000H
A: cmp al, 0
   jnz seguir
   mov bx, offset MSJ_FIN
   mov al, offset FIN - offset MSJ_FIN
   int 7
   mov FIN,1
seguir: ret

   ORG 3200H
B: push ax
   not al
   out  PB, al
   pop ax
   ret

   ORG 3400H
C: push ax
   and al, 10000000b
   jz volver
   mov bx, offset MSJ
   mov al, offset MSJ_FIN - offset MSJ
   int 7
volver: pop ax
   ret

   ORG 2000H
   mov al, 0FFh
   out  CA, al
   mov al, 0
   out  CB, al

loop: in al, PA
   call A
   cmp FIN,1
   jz  terminar
   call  B
   call  C
   jmp loop
terminar: int 0
   end
```