Universidad de la República
Facultad de Ingeniería
Instituto de Ingeniería Eléctrica

Introducción a los Microprocesadores

Laboratorio 2022

Práctica 2

Grupo:	***Mie 15:00 - 2***
Cédula de Identidad	Nombre
5.405.795-0	Joel Carabajal
5.569.102-0	Henola Cócaro
4.911.609-4	Carlos Gruss

Índice

1. Mapa entrada/salida

Dirección	Entrada	Salida
0xFF	-	-
...	-	-
0x87	-	CL_PSCLK
0x86	PSDATA	PSDATA
0x85	PSCLK	PSCLK
0x84	-	LEDG[7:0]
0x83	-	HEX3[7:0]
0x82	-	HEX2[7:0]
0x81	-	HEX1[7:0]
0x80	SW[7:0]	HEX0[7:0]
...	-	-
0x00	-	-

Cuadro 1: Mapa de entrada y salida con puertos diseñados.

2. Esquemáticos

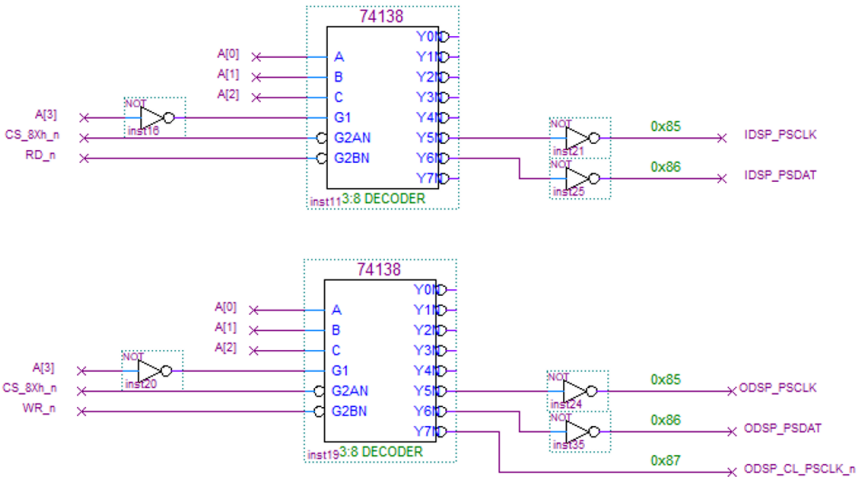

Figura 1: Decodificación utilizada para generar los pulsos de selección de puertos. La señal "CS_8Xh_n"venía dada en el hardware provisto.

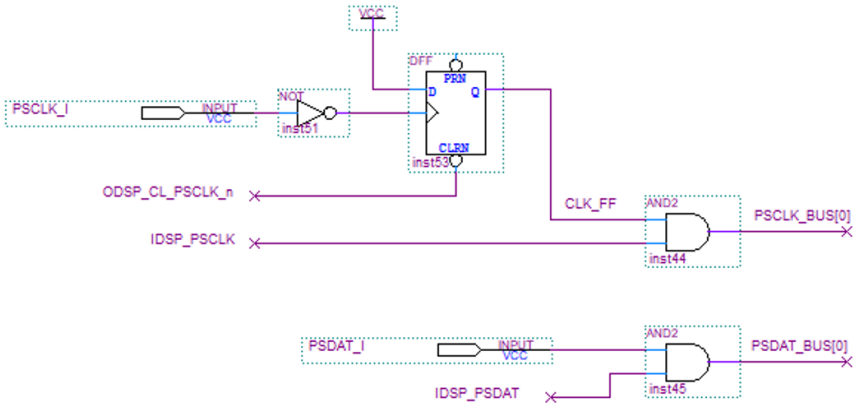

Figura 2: Puertos de entrada diseñados. Los bits 1 a 7 de PSCLK_BUS y PSDAT_BUS se ponen a cero.

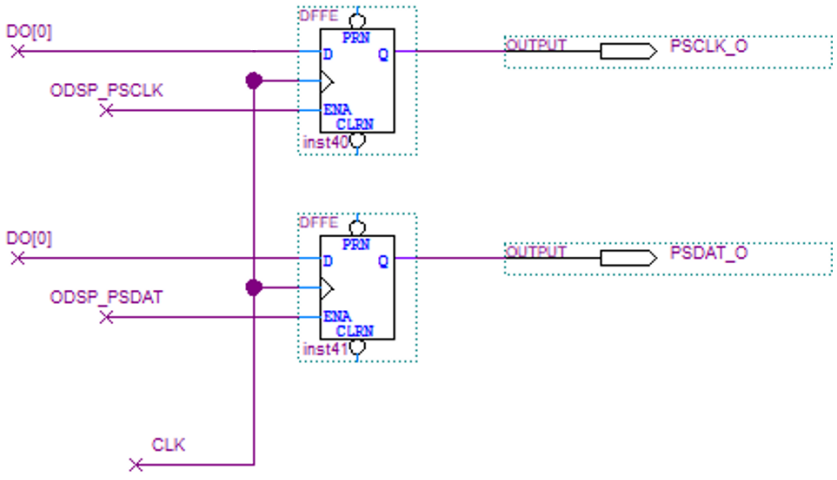

Figura 3: Puertos de salida diseñados.

3. Pseudocódigo

3.1. esperoflanco

3.1.1. Subrutina

- Poner a cero la salida del FF del puerto de entrada por si se encontraba en alto su previamente.
- Hasta detectar un flanco de bajada:
  - Cargar en acumulador contenido de puerto de entrada.
  - Testear el bit menos significativo.
  - Si es un uno → No llego el flanco de bajada → Saltar al inicio del bucle.
  - Si es un cero → Llego flanco de bajada → Salir del bucle.
- Poner a cero la salida del FF del puerto de entrada.
- Retornar.

3.1.2. Programa de prueba

- Inicializar un registro como contador.
- Repetir infinitamente:
  - Llamar a esperoflanco.
  - Incrementar contador.
  - Convertir número en contador a su representación binaria en display 7 segmentos.
  - Poner el resultado en el puerto de salida de los display 0 y 1.

3.2. get_ps2

3.2.1. Subrutina

- Preservar registros.
- Inicializar un registro en cero dónde se guardara el dato de 8 bits entrante.
- Esperar un flanco de bajada de reloj PS2 (Bit de Start).
- Repetir 8 veces:
  - Esperar un flanco de bajada de reloj PS2 (Datos).
  - Guardar dato entrante en el acumulador (Se recibe en el bit menos significativo del puerto de entrada).

    - Hacer un OR con el registro que se inicializo en cero.
    - Cargar el resultado en este registro.
    - Rotar los bits del registro un lugar hacia la derecha.
- Esperar un flanco de bajada de reloj PS2 (Bit de paridad).
- Recibir bit de paridad en el bit menos significativo del acumulador.
- Si es uno → Encender flag de Carry.
- Si es cero → Apagar flag de Carry.
- Esperar un flanco de bajada de reloj PS2 (Bit de Stop).
- Cargar el registro que recibió el dato al acumulador
- Restaurar registros.
- Retornar.

3.2.2. Programa de prueba

- Inicializar un registro de direcciones que apunte a un lugar reservado en RAM.
- Repetir infinitamente:
    - Llamar a get_ps2 para recibir un dato nuevo.
    - Guardar el antepenúltimo dato dos direcciones encima del lugar reservado.
    - Guardar el penúltimo dato una dirección encima del lugar reservado.
    - Guardar el último dato en la dirección reservada.
    - Mostrar el último dato en los display 0 y 1 de 7 segmentos.
    - Mostrar el penúltimo dato en los display 2 y 3 de 7 segmentos.
    - Mostrar el antepenúltimo dato en los leds.

3.3. get_tecla

3.3.1. Subrutina

- Leer un primer dato (siempre se manda el scan code al menos una vez antes de que llegue 0xF0).
- Repetir hasta recibir 0xF0:
    - Leer dato nuevo.
    - Compararlo con 0xF0.

    - Si no es, saltar al principio del bucle.
    - Si es, salir del bucle.
- Leer un dato más (el scan code de la tecla). Usando get_ps2, queda guardado en el acumulador.
- Retornar.

3.3.2. Programa de prueba

Es exactamente igual al programa de prueba de get_ps2, cambiando la llamada a dicha función por una llamada a get_tecla.

3.4. scodeadigito

3.4.1. Subrutina

- Preservar registros.
- Inicializar un contador en cero.
- Inicializar un registro con la dirección de origen de la tabla.
- Inicializar un registro con el scan code a convertir a dígito
- Repetir mientras queden filas de la tabla por recorrer y no se haya encontrado el scan code en la tabla:
    - Comparar contenido de la fila actual en la tabla con el scan code que se tiene.
    - Si coincide:
        - Salir del bucle.
        - Cargar en acumulador el contenido del contador.
        - Restaurar registros.
        - Retornar.
    - Si no coincide:
        - Incrementar contador.
        - Incrementar dirección de la tabla (subir una fila).
    - Si el contador llega a 10 (no hay más filas por recorrer):
        - Salir del bucle.
        - Cargar en acumulador código de error.
        - Restaurar registros.
        - Retornar.

3.4.2. Programa de prueba

Análogo a programas de prueba anteriores, con la excepción de que si scodeadigito devuelve código de error (la tecla ingresada no es dígito) y por lo tanto no se guarda en memoria el nuevo dato.

4. Código assembler

Se definen los siguientes símbolos para todos los programas a continuación:

```
HEX0 equ 0x80
HEX1 equ 0x81
HEX2 equ 0x82
HEX3 equ 0x83
SW equ 0x80
BTN equ 0x81
LEDG equ 0x84
; Puertos agregados en Practica 2:
clearFF equ 0x87
salidaDATA equ 0x86
salidaCLK equ 0x85
entradaDATA equ 0x86
entradaCLK equ 0x85
```

4.1. esperoflanco

4.1.1. Subrutina

```
; espera un flanco de bajada de PSCLK y retorna
esperoflanco:
    out (clearFF), a ; se pone a cero el FF de entrada
    loop_esperoflanco:
        in a, (entradaCLK)
        bit 0, a      ; PSCLK esta en el bit 0 del puerto de entrada
        jr z , loop_esperoflanco
    out (clearFF), a ; pone a cero el FF de entrada
    ret
```

4.1.2. Programa de prueba

```
1  ld sp, 0x0000
2  ; se inicializan en nivel alto los puertos salidaCLK y salidaDATA:
3  ld a, 1d
4  out (salidaCLK), a
5  out (salidaDATA), a
6
7  ; cada vez que llega un flaco de bajada en CLK
8  ; incrementa un contador y lo muestra en HEX0 y HEX1
9  ld e, 0d
10 loop_test:
11    call esperoflanco ; espera a que llegue un flanco de bajada en CLK
12    inc e
13    ld a, e
14    call binapbcd
15    call pbcda7seg
16    ld a, c
17    out (HEX0), a
18    ld a, b
19    out (HEX1), a
20    jr loop_test
```

4.2. get_ps2

4.2.1. Subrutina

```
; recibe un dato ps2 y lo devuelve en acumulador
; devuelve el bit de paridad en el flag de carry
get_ps2:
    push bc    ; preservar registros
    ld b, 8d   ; inicializar contador para djnz
    ld c, 0d   ; inicializar registro de recepcion de datos
    call esperoflanco
    loop_get_ps2:
        call esperoflanco
        in a, (entradaDATA) ; todos los bits salvo el LSB son cero
        or c
        ld c, a ; pone bit recibido en LSB de c, sin modificar los demas
        rrc c  ; rotacion circular hacia la derecha
        djnz loop_get_ps2
    call esperoflanco
    in a, (entradaDATA) ; recepcion del bit de paridad
    cp 0d
    scf ; encender el flag de carry
    jr nz, paridad_uno
    ccf ; si el bit de paridad es cero, lo apago
    paridad_uno:
    call esperoflanco ; esperar al bit de stop
    ld a, c
    pop bc ; restaurar registros
    ret
```

4.2.2. Programa de prueba

```
1  ld sp, 0x0000
2  ; se inicializan en nivel alto los puertos salidaCLK y salidaDATA:
3  ld a, 1d
4  out (salidaCLK), a
5  out (salidaDATA), a
6
7  ld ix, datos_ps2
8
9  loop_get_ps2_test:
10    ; esperamos un dato nuevo y lo guardamos en registro a
11    call get_ps2
12
13    ; (ix+2) <-- (ix+1)
14    ld e, (ix+1)
15    ld (ix+2), e
16
17    ; (ix+1) <-- (ix)
18    ld e, (ix)
19    ld (ix+1), e
20
21    ; (ix) <-- a (ultimo dato)
22    ld (ix), a
23
24    ; mostrar ultimo dato
25    push af    ; preservamos el flag de carry
26    call pbcda7seg
27    pop af     ; recuperamos el flag de carry
28    ld a, c
29    jr nc, paridad_cero
30    and 01111111B
31    paridad_cero:
32    out (HEX0), a
33    ld a, b
34    out (HEX1), a
35    ; mostrar penultimo dato
36    ld a, (ix+1)
37    call pbcda7seg
38    ld a, c
39    out (HEX2), a
40    ld a, b
41    out (HEX3), a
42    ; mostramos antepenultimo dato
43    ld a, (ix+2)
44    out (LEDG), a
45
46    jp loop_get_ps2_test
```

4.3. get_tecla

4.3.1. Subrutina

```
; devuelve en a el scode de una tecla presionada
get_tecla:
    call get_ps2  ; lee el primer dato (siempre hay uno al menos)
    loop_get_tecla:
        call get_ps2
        cp 0xf0
        jp nz, loop_get_tecla ; lee hasta que llega 0xF0
    call get_ps2  ; lee el scode de la tecla
    ret
```

4.3.2. Programa de prueba

```
ld sp, 0x0000
; se inicializan en nivel alto los puertos salidaCLK y salidaDATA:
ld a, 1d
out (salidaCLK), a
out (salidaDATA), a

ld ix, datos_ps2

loop_get_tecla_test:
    ; esperamos un dato nuevo y lo guardamos en a
    call get_tecla

    ; (ix+2) <-- (ix+1)
    ld e, (ix+1)
    ld (ix+2), e

    ; (ix+1) <-- (ix)
    ld e, (ix)
    ld (ix+1), e

    ; (ix) <-- a (ultimo dato)
    ld (ix), a

    ; mostramos ultimo dato
    push af    ; preservamos el flag de carry
    call pbcda7seg
    pop af     ; recuperamos el flag de carry
    ld a, c
    jr nc, paridad_cero
    and 01111111B
    paridad_cero:
    out (HEX0), a
```

```
33      ld a, b
34      out (HEX1), a
35      ; mostraemos penultimo dato
36      ld a, (ix+1)
37      call pbcda7seg
38      ld a, c
39      out (HEX2), a
40      ld a, b
41      out (HEX3), a
42      ; mostramos antepenultimo dato
43      ld a, (ix+2)
44      out (LEDG), a
45
46      jp loop_get_tecla_test
```

4.4. scodeadigito

4.4.1. Subrutina

```
1  ; recibe un scode en a y devuelve el digito correspondiente
2  ; o recibe 100d si la tecla presionada no es un digito
3  scodeadigito:
4      push bc
5      ld c, a ; cargar el scode en c
6      ld b, 0d  ; incializar b como contador
7      ld hl, tab_scodeadigito ; carga el origen de la tabla en hl
8      loop_tabla:
9          ld a, (hl)
10         cp c   ; compara el scode con el lugar de la table en hl
11         jr z, digito_encontrado ; si coincide salta
12         inc b  ; se incrementa el contador
13         ld a, b
14         cp 10d ; si se paso de 9 significa que no es digito
15         jr z, no_digito
16         inc hl ; se baja una fila en la tabla
17         jr loop_tabla
18     digito_encontrado:
19         ld a, b ; el numero en el contador es el digito correspondiente
20         pop bc
21         ret
22     no_digito:
23         ld a, 100d ; a no es digito, devuelve 100d
24         pop bc
25         ret
```

Se carga en memoria la siguiente tabla (correspondiente a los “scan code”de los dígitos) para esta parte:

```
tab_scodeadigito:
       db 0x70 ; 0
       db 0x69 ; 1
       db 0x72 ; 2
       db 0x7a ; 3
       db 0x6b ; 4
       db 0x73 ; 5
       db 0x74 ; 6
       db 0x6c ; 7
       db 0x75 ; 8
       db 0x7d ; 9
```

4.4.2. Programa de prueba

```
ld sp, 0x0000 ; incializaciones
ld a, 1d
out (salidaCLK), a
out (salidaDATA), a
ld ix, datos_ps2

loop_get_digito_test:
    ; esperamos un dato nuevo y lo guardamos en a
    call get_tecla
    call scodeadigito
    ; scodeadigito devuelve 100 si la tecla no es digito
    cp 100d
    jp z, loop_get_digito_test

    ; (ix+2) <-- (ix+1)
    ld e, (ix+1)
    ld (ix+2), e
    ; (ix+1) <-- (ix)
    ld e, (ix)
    ld (ix+1), e
    ; (ix) <-- a (ultimo dato)
    ld (ix), a

    ; mostramos ultimo dato
    push af    ; preservamos el flag de carry
    call hexa7seg
    ld b, a
    pop af     ; recuperamos el flag de carry
    ld a, b
    jr nc, paridad_cero
    and 01111111B
    paridad_cero:
    out (HEX0), a
```

```
34      ld a, 0xff
35      out (HEX1), a
36      ; mostraemos penultimo dato
37      ld a, (ix+1)
38      call hexa7seg
39      out (HEX2), a
40      ld a, 0xff
41      out (HEX3), a
42      ; mostramos antepenultimo dato
43      ld a, (ix+2)
44      out (LEDG), a
45
46      jp loop_get_digito_test
```

5. Diagrama de tiempos

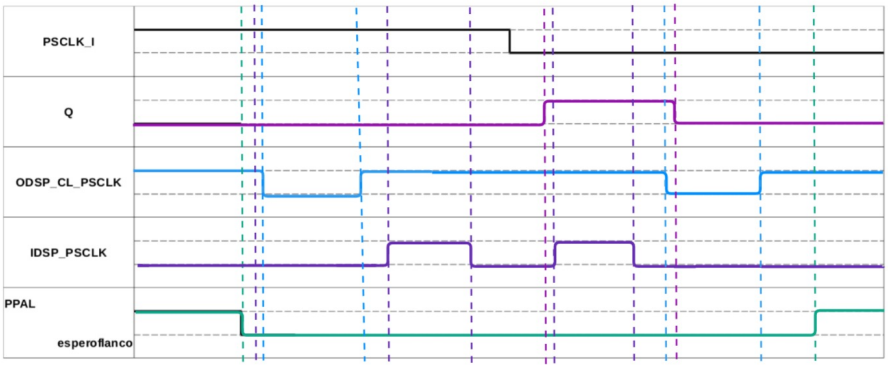

Figura 4: Diagrama planteado en parte b. Los retardos no están a escala.

Tener en cuenta que en la figura 4 cuando se dibuja "ODSP_CL_PSCLK"se está dibujando en realidad el negado de esta, que es la señal que se ve entrando al CLR del FF en la figura 2.

6. Mapa de memoria

Dirección	Bloque cuando se compila para ROM	Bloque cuando se compila para GDB
0xFFFF ...	Stack	Stack
... 0xB400	...	Variables (.data)
... 0xB300	...	tab_scodeadigito
... 0xB200	...	tab_h7s
... 0xB000	Variables (.data)	Programa principal, get_ps2, subrutinas auxiliares (.text)
... 0x8000	...	Datos monitor
... 0x0200	tab_scodeadigito	...
... 0x0100	tab_h7s	...
... 0x0000	Programa principal, get_ps2, subrutinas auxiliares (.text)	Monitor

Cuadro 2: Mapa de memoria del sistema cuando se compila para debugger y cuando se compila para ROM.

Bloque	**Stack**	**Variables**	**Tablas**	**Programa principal, subrutinas.**
¿Funcionaría en ROM?	No	No	Si	Si

Cuadro 3: Respuesta a pregunta planteada en parte c.

En el cuadro 2, vemos que en .text se ubica tanto el programa de prueba como la subrutinas, mientras que en .data están las variables datos_ps2 utilizadas para ir almacenando los datos recibidos. Cuando compilamos para debugger (columna 3) Monitor es el programa grabado en ROM para comunicación entre el debugger y el microprocesador. El Stack en este caso fue inicializado en 0x0000 para que comience a ocupar la dirección 0xFFFF de memoria y crezca hacia abajo dentro de la RAM a partir de ahí.

En cambio en la segunda columna, cuando se compila para ROM, no se carga el programa Monitor ya que no hay comunicación por medio del debugger.

En .text tenemos nuevamente los programas de prueba y las subrutinas implementadas, mientras que en .data solo tenemos las variables para almacenar los datos PS2 recibidos. El Stack se ubica igual que en el caso anterior.

Las tablas tab_h7s y tab_scodeadigito fueron inicializadas en el lugar de memoria 0x0100 y 0x0200 respectivamente (relativo a .text). La posición absoluta varía dependiendo de la ubicación de .text, que cuando se compila para GDB es la 0xB000 y para ROM la 0x0000.

Para que nuestro programa funcione cuando se compila de cualquiera de las dos maneras, es necesario explicitar que las tablas vayan en una dirección relativa a .text tal que queden en ROM cuando se compila sin debugger.