Universidad de la República
Facultad de Ingeniería
Instituto de Ingeniería Eléctrica

Introducción a los Microprocesadores

Laboratorio 2022

Práctica 1

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

Se definen los siguientes puertos que se usarán para entrada y salida:

```
HEX0 equ 0x80
HEX1 equ 0x81
HEX2 equ 0x82
HEX3 equ 0x83
SW equ 0x80
BTN equ 0x81
```

1. hexa7seg

1.1. Pseudocódigo subrutina

- Preservar registro HL.
- Cargar dirección de comienzo de la tabla en HL.
- Limpiar el contenido del acumulador para quedarnos con los cuatro bits menos significativos (que representan un numero entre 0 y 15).
- Cargar el contenido del acumulador en las posiciones menos significativas de HL.
- Cargar en el acumulador el contenido de la dirección apuntada por HL.
- Restaurar HL.
- Retornar.

1.2. Código subrutina

```
.data
tab_h7s:  ; _gfedcba
    db 11000000B ; 0
    db 11111001B ; 1
    db 10100100B ; 2
    db 10110000B ; 3
    db 10011001B ; 4
    db 10010010B ; 5
    db 10000010B ; 6
    db 11111000B ; 7
    db 10000000B ; 8
    db 10011000B ; 9
    db 10001000B ; a
    db 10000011B ; b
    db 10100111B ; c
    db 10100001B ; d
    db 10000110B ; e
```

```
18      db 10001110B ; F
19
20  hexa7seg:
21      push hl
22      ld hl, tab_h7s
23      and 0x0f
24      ld l, a
25      ld a, (hl)
26      pop hl
27      ret
```

1.3. Pseudocódigo de prueba

- Inicializar el Stack Pointer.
- Repetir infinitamente:
  - Cargar en el acumulador el contenido del puerto de entrada de los switch.
  - Llamar a la subrutina hexa7seg.
  - Cargar el contenido del acumulador al puerto de salida correspondiente al display 7 segmentos.

1.4. Código de prueba

```
1   .text
2
3   ld sp, 0x0000
4
5   loop_test:
6       in a, (SW)
7       call hexa7seg
8       out (HEX0), a
9       jr loop_test
10
11
12  .include "subrutinas.s"
13  .end
```

2. pbcda7seg

2.1. Pseudocódigo subrutina:

- Preservar el contenido del acumulador en un registro auxiliar.

- Llamar a hexa7seg.
- Cargar el contenido del acumulador en C.
- Restaurar el contenido original del acumulador.
- Rotar el acumulador cuatro veces a la derecha (ya que hexa7seg opera sobre los cuatro bits menos significativos del acumulador).
- Llamar a hexa7seg.
- Cargar el contenido del acumulador en B.

2.2. Código subrutina

```
1  pbcda7seg:
2      ld b, a                ; preservar el contenido del acumulador
3      call hexa7seg
4      ld c, a
5      ld a, b                ; restaurar el acumulador original
6      sra a                  ; rotar cuatro posiciones hacia la derecha
7      sra a
8      sra a
9      sra a
10     call hexa7seg
11     ld b, a
12     ret
```

2.3. Pseudocódigo de prueba:

- Inicializar el Stack Pointer.
- Generar un bucle infinito:
  - Cargar el contenido de los switch de entrada al acumulador.
  - Llamar a pbcd7seg.
  - Cargar el contenido de C en el acumulador.
  - Cargar el contenido del acumulador al display correspondiente.
  - Cargar el contenido de B en el acumulador.
  - Cargar el contenido del acumulador al display correspondiente.

2.4. Código de prueba:

```
.text

ld sp, 0x0000

loop_test:
    in a, (SW)
    call pbcda7seg
    ld a, c
    out (HEX0), a
    ld a, b
    out (HEX1), a
    jr loop_test


.include "subrutinas.s"
.end
```

3. binapbcd

3.1. Pseudocódigo subrutina:

- Inicializar un contador en 0.
- Mientras el resultado de (A-10) sea positivo:
  - Cargar en el acumulador el resultado de (A-10).
  - Incrementar el contador.
- En el contador se tiene el dígito de decenas y en el acumulador queda el dígito de las unidades.
- Rotar el contador 4 veces a la izquierda.
- Cargar en el acumulador el resultado de (A OR contador).

3.2. Código subrutina

```
binapbcd:
    ld b, 0d
    loop_binapbcd:
        cp 10d
        jp m, fin_binapbcd   ; if (A - 10d) > 0 then jump a fin_binapbcd
        sub 10d              ; else restar 10 e incrementar contador
        inc b
```

```
        jp loop_binapbcd
    fin_binapbcd:
        sla b               ; rotar cuatro posiciones a la izquierda
        sla b
        sla b
        sla b
        or  b
    ret
```

3.3. Pseudocódigo de prueba:

- Inicializar el Stack Pointer.
- Generar el bucle infinito:
  - Cargar los switch de entrada en acumulador.
  - Llamar binapbcd.
  - Llamar pbcd7seg.
  - Cargar C en el acumulador.
  - Cargar el acumulador a un display de salida.
  - Cargar B en el acumulador.
  - Cargar el acumulador a un display de salida.

3.4. Código de prueba:

```
.text

ld sp, 0x0000

loop_test:
    in a, (SW)
    call binapbcd
    call pbcda7seg
    ld a, c
    out (HEX0), a
    ld a, b
    out (HEX1), a
    jr loop_test


.include "subrutinas.s"
.end
```

4. despreloj

4.1. Pseudocódigo subrutina:

- Cargar en el acumulador los segundos de la dirección apuntada por IX.
- Llamar a binapbcd.
- Llamar a pbcda7seg.
- Se obtienen en BC los segundos listos para representar en dos display de 7 segmentos.
- Cargar BC en las salidas correspondientes, preservando el valor de C en un registro auxiliar.
- Incrementar IX para que apunte a las centésimas.
- Cargar en el acumulador el valor correspondiente a las centésimas.
- Llamar a binapbcd.
- Llamar a pbcda7seg.
- Se obtienen en BC las centésimas listas para representar en dos display de 7 segmentos.
- Cargar BC en las salidas correspondientes.
- Incrementar IX para que apunte a la bandera.
- Cargar en el acumulador el valor de la bandera.
- Si (A-0 == 0):
  - Restaurar el valor de C que se respaldó previamente.
  - Encender el punto decimal sin modificar el valor numérico.
- Restaurar el valor de IX.

4.2. Código subrutina

```
1  despreloj:
2      ld a, (ix)        ; cargamos los segundos al acumulador
3      call binapbcd
4      call pbcda7seg    ; en bc tenemos los segundos codificados a 7seg
5      ld a, b
6      out (HEX3), a
7      ld a, c
8      ld e, a           ; preservamos lo que va en HEX2
9      out (HEX2), a     ; por si hay que agregarle punto decimal
```

```
    inc ix
    ld a, (ix)       ; cargamos las centesimas al acumulador
    call binapbcd
    call pbcda7seg   ; en bc tenemos las centesimas codificadas a 7seg
    ld a, b
    out (HEX1), a
    ld a, c
    out (HEX0), a
    inc ix
    ld a, (ix)
    cp 0d               ; evalua como cero solo si la flag es 0x00
    jp nz, fin_despreloj ; si no vale 0x00 no se debe agregar punto
    ld a, e             ; recuperamos lo que va en HEX2
    and 01111111B       ; mascara para prender el punto decimal
    out (HEX2), a
    fin_despreloj:
    dec ix              ; restauramos el registro ix
    dec ix
    ret
```

4.3. Pseudocódigo de prueba:

- Generar un bucle:
  - Llamar a despreloj.

4.4. Código de prueba:

```
.text

ld sp, 0x0000

loop_test:
    call despreloj
    break_test:
    jr loop_test

.include "subrutinas.s"

.end
```

Una forma de corroborar que el programa está funcionando como se espera es correr lo siguiente en el debugger:

```
set $ix=0xB400
set {char}0xB400=0x01
set {char}0xB401=0x63
set {char}0xB402=0xFF
cont
```

Se espera que se muestre **01 99** en los display de 7 segmentos (con el decimal apagado).

5. decreloj

5.1. Pseudocódigo subrutina:

- Cargar los segundos de IX.
- Cargar las centésimas de IX+1.
- Si (centésimas-10 ≥ 0):
  - Restarle 10 a centésimas.
- Si no:
  - Si (segundos-1 ≥ 0):
    - Las centésimas pasa a valer (90 + el resto que hay en IX+1).
    - Los segundos bajan en una unidad.
    - Se complementa la bandera.
  - Si no:
    - El contador satura a cero.

5.2. Código subrutina

```
1  decreloj:
2     ld b,(ix)         ; cargar los segundos
3     ld a,(ix+1)       ; cargar las centesimas
4     cp 10d
5     jp m, cent_neg    ; centesimas-10 >= 0? (underflow centesimas)
6     sub 10d           ; si no hay underflow, restamos 10
7     ld (ix+1), a
8     jp fin_decreloj   ; y saltamos al final
9     cent_neg:         ; si hay underflow en las centesimas:
10       ld c,(ix+1)
11       ld a,b
12       cp 1d
```

```
13          jp m , seg_neg ; segundos-1 >= 0? (underflow segundos)
14          sub 1d         ; si no hay, restamos 1
15          ld (ix), a
16          ld a, c
17          add a, 90d     ; las centesimas valen 90+resto previo
18          ld (ix+1), a
19          ld a, (ix+2)
20          cpl            ; complementar la bandera
21          ld (ix+2), a
22          jp fin_decreloj
23      seg_neg:           ; si hay underflow en los segundos:
24          ld (ix), 0d    ; satura a cero el contador
25          ld (ix+1), 0d
26      fin_decreloj:
27      ret
```

5.3. Pseudocódigo de prueba:

- Definir dos mascaras para acceder a los bits correspondientes a los botones de interés.
- Inicializar el Stack Pointer.
- Generar un bucle:
  - Cargar la entrada de los botones en el acumulador.
  - Pasar el acumulador por una máscara para quedarnos con el valor del botón 2. El resultado da cero si el botón está presionado.
  - Si el botón 2 no está presionado:
    - Saltar al comienzo del bucle.
  - Si no:
    - Llamar decreloj.
    - Llamar a despreloj.
    - Mientras el botón 1 no está presionado:
      - Cargar la entrada de los botones en el acumulador.
      - Enmascarar y chequear si está presionado.
    - Saltar al principio del bucle.

5.4. Código de prueba:

```
1  btn_1_mask equ 00000010B
2  btn_2_mask equ 00000100B
3
4  .text
```

```
5
6  ld sp, 0x0000
7
8  btn_2_loop:
9      in a, (BTN)
10     and btn_2_mask
11     jp nz, btn_2_loop
12     call decreloj
13     call despreloj
14     btn_1_loop:
15         in a, (BTN)
16         and btn_1_mask
17         jp nz, btn_1_loop
18         jp btn_2_loop
19
20 .include "subrutinas.s"
21
22 .end
```

6. Ensamblado, ciclos y tiempo de ejecución

6.1. Ensamblado

PC	Instrucción	OPCODE	# de Bytes	Ciclos M	Ciclos T
0xBA00	push AF	1111 0101	1	$M1$, WR^{MEM}, WR^{MEM}	11T
0xBA01	push DE	1101 0101	1	$M1$, WR^{MEM}, WR^{MEM}	11T
0xBA02	ld DE, CTE16bit	0001 0001 0010 1010 0010 1010	3	$M1$, RD^{MEM}, RD^{MEM}	10T
0xBA05	dec DE	XXXXXX	1	XXXXXXX	6T
0xBA06	ld A, D	XXXXXX	1	XXXXXXX	4T
0xBA07	or E	XXXXXX	1	XXXXXXX	4T
0xBA08	jr NZ, loop	0010 0000 0000 0101	2	$M1$, RD^{MEM} Inactivo	12T
0xBA0A	pop DE	1101 0001	1	$M1$, RD^{MEM}, RD^{MEM}	10T
0xBA0B	pop AF	XXXXXX	1	XXXXXXX	10T
0xBA0C	ret	XXXXXX	1	XXXXXXX	10T

Símbolo	Valor
CTE16bit	0x4242
espero	0xBA00
loop	0xBA05

6.2. Ciclos

Cuadro 1: Instrucción: push DE (ejemplo)

CICLO	$M1$	WR^{MEM}	WR^{MEM}
BUS DIRECCIONES	0xBA01	0xFFED	0xFFEC
BUS DATOS	11010101	0xAB	0xCD

Cuadro 2: Instrucción: ld DE, CTE16bit

CICLO	$M1$	RD^{MEM}	RD^{MEM}
BUS DIRECCIONES	0xBA02	0xBA03	0xBA04
BUS DATOS	00010001	0x42	0x42

Cuadro 3: Instrucción: jr NZ, loop

CICLO	$M1$	RD^{MEM}	Inactivo
BUS DIRECCIONES	0xBA08	0xBA09	-
BUS DATOS	00100000	0000 0101	-

Cuadro 4: Instrucción: pop AF

CICLO	$M1$	RD^{MEM}	RD^{MEM}
BUS DIRECCIONES	0xBA0B	0xFFEE	0xFFEF
BUS DATOS	11110001	0x34	0x12

6.3. Tiempos de ejecución

Pasando de ms a ciclos T:

$$10\text{ms} * 50\text{MHz} = 500000T$$

Una vez hecho esto se calcula el tiempo de ejecución del programa (desde la primera instrucción hasta llegar a **ret**, sin incluirlo). Se define una constante $\mathbb{C}$ como la cantidad de veces que se ejecuta el bucle.

$$11T + 11T + 10T + (6T + 4T + 4T + 12T)\mathbb{C} + 10T + 10T = 500000T$$

Se puede despejar $\mathbb{C}$ como:

$$\mathbb{C} = \frac{500000T - 52T}{26T} = 19228{,}8$$

Una vez obtenida la constante, se observar que la secuencia:

```
1 ...
2 ld a, d
3 or e
4 ...
```

Se puede sintetizar como **"d or e"** que solo evalúa como cero cuando tanto **d** cómo **e** valen cero. Por lo tanto, el valor que debe tener la constante para que el bucle se ejecute $\mathbb{C}$ veces es $\mathbb{C}$.

Como $\mathbb{C}$ no es un entero, se probó redondear para arriba y para abajo observando cual de las dos opciones aproxima mejor a los requerimientos. Si $\mathbb{C} = 19229$ se obtiene un error relativo de $0,0012\,\%$, mientras que si consideramos $\mathbb{C} = 19228$ se obtiene un error relativo de $0,004\,\%$. Por lo tanto se elegiría 19229 como el nuevo valor de CTE16bit, ya que es el que posee menor error relativo.