Práctico 3
Subrutinas. Tiempos

Ejercicio 1

Primeramente se presenta un algoritmo para generar el retardo con un error muy pequeño y luego se indica para cada caso los valores que los parámetros en juego deben tomar.

Se debe tener presente que al invocar a la subrutina con un CALL, este CALL demora 17 estados T que están incluidos en el retardo total (aunque dependiendo del caso este valor puede ser despreciable).

```
Ajuste_fino     EQU
Ajuste_grueso   EQU                                          Estados T

                CALL DELAY                                   17
                ...

DELAY:  PUSH    BC          ; guarda B y C                   11        53
                PUSH    AF  ; guarda A y F                   11

                LD      B, Ajuste_grueso                     7
                LD      C, 0h       ; C se inicia con 256    7

LOOP1:  DEC     C                                            4
                JR      NZ, LOOP1                            7 (Z=0) / 12 (Z=1)    (*)
                DJNZ    LOOP1                                8 (B=0) / 13 (B<>0)

                LD      B, Ajuste_fino                       7
LOOP2:  DJNZ    LOOP2                                        8 (B=0) / 13 (B<>0)   (**)

                POP     AF          ; se restaura A y F      10
                POP     BC          ; se restaura B y C      10        30
                RET                                          10
```

(*) = ((4 + 12) x 256 – 5 + 13) x Ajuste_grueso – 5 = 4104 x Ajuste_grueso - 5
(**) = 7 + 13 x AJUSTE_FINO – 5 = 13 x Ajuste_fino + 2

Retardo_total (medido en estados T) = 53 + 4104 x Ajuste_grueso – 5 + 13 x Ajuste_fino + 2 + 30 =
= 80 + 4104 x Ajuste_grueso + 13 x Ajuste_fino

a) Z0840004

El período T para el microprocesador indicado es 250 ns.
Esto implica que el retardo debe ser de 400.000 T

Entonces: 80 + 4104 x Ajuste_grueso + 13 x Ajuste_fino = 400.000
4104 x Ajuste_grueso + 13 x Ajuste_fino = 399.920

La idea es hallar:

Ajuste_grueso = 399.920 / 4104 = 97 (valor truncado al entero inferior)
4104 x 97 + 13 x Ajuste_fino = 399.920
13 x Ajuste_fino = 399.920 – 398.088 = 1.832

Entonces:

Ajuste_fino = 1.832 / 13 = 141 (valor redondeado al entero más próximo).

Tomamos:

Ajuste_grueso = 97
Ajuste_fino = 141

Con lo que:

Retardo_total = = 400.001 * 250 ns = 0,100 000 250 s
Error absoluto = 250 ns
Error relativo = 0,00025 %

b) Z0840006

Ahora, el período T para el microprocesador indicado 162 ns.
Esto implica que el retardo debe ser de 617.283,95 T.
Redondeamos a 617.284 T pues no es posible medir tiempo menor a 1 T
Siguiendo el mismo razonamiento que antes:

Ajuste_grueso = 150
Ajuste_fino = 123
Retardo_total = 617.279 * 162 ns = 0,099 999 198 s
Error absoluto = 802 ns
Error relativo = 0,0008 %

Observaciones
- Es posible en ambos casos obtener una solución que coincida con la cantidad de períodos T requeridos; bastará jugar con "Ajuste_fino" e ir agregando instrucciones que no hagan nada (ej: NOP, y para este caso, LD A,FFh, LD A. (HL), etc) de la duración conveniente para lograr un "ajuste super fino".
- Cabe destacar que para este tipo de problemas existen infinidad de soluciones, basta con escribir código que se encargue de "perder el tiempo".
- La solución presentada solo pretende mostrar una forma simple de generar *loops* de espera para disponer de una herramienta y saber rápidamente como encarar un problema a partir del tiempo a medir y el error permitido.

Ejercicio 2

a) Por mayor detalle referirse a la solución del ejercicio 4 del práctico 2.

```
digito  EQU 30h
letra   EQU 41h

ASCII2HEX: LD    A, (HL)            ; A ← digito
           SUB   A,digito           ; A ← A - ASCII('0')
           CP    10                 ; ¿menor o mayor a10?
           JP    M, VOLVER          ; Si A < 10d, ya está y se puede "volver"
           SUB   A, letra-digito-10 ; sino:  A ← A  ASCII('A') + ASCII('0') + 10d
VOLVER:    RET                      ; retorna con el valor correcto en el acumulador
```

Nota: Por tratarse de una subrutina, finaliza con la instrucción RET. Notar que comienza con una etiqueta para independizarse de la dirección de la instrucción LD A,(HL).

b) Si bien el ejercicio planteado es similar al 2.4, se simplifica notoriamente al utilizar la parte a)

```
ASCII2BIN: PUSH  BC              ; Guarda B y C en el Stack
           PUSH  HL              ; Guarda H y L en el Stack
           CALL  ASCII2HEX       ; Se llama a la subrutina de la parte a).
                                 ; Queda en el acumulador el resultado.
           LD    B,A             ; B ← digito menos significativo
           INC   HL              ; HL queda apuntando al dígito más significativo.
           CALL  ASCII2HEX       ; Idem CALL anterior, A ← digito mas significativo
           SLA   A
           SLA   A
           SLA   A
           SLA   A               ; Multiplica A por 16 (corrimiento de 4 bits a la izq.)
           OR    A,B             ; Se agregan los cuatro bits menos sigificativos.
           POP   HL              ; Restaura H y L del Stack
           POP   BC              ; Restaura B y C del Stack
           RET
```

Notas:
- Si bien desde el punto de vista del código no cambia casi nada entre este ejercicio y el 2.4, esta segunda versión es más simple de escribir y comprender, además de ser más escalable.
- Si bien el registro C no se utiliza nunca, su valor es guardado en el stack y luego restaurado, dado que importa salvar el valor del registro B. Un error muy común es utilizar la instrucciones "inventadas" "PUSH B" y "POP B", pero esto no es correcto ya que instrucciones de stak de 8 bit no existen.

¿Que pasa si se pide que solo se puede modificar el registro A, sin modificar el registro F?
Para ello hay un truco simple que es conveniente tenerlo presente.
A continuación se muestra el mismo código, pero con las variantes en letra **negrita**.

```
ASCII2BIN: PUSH  BC
           PUSH  HL
           PUSH  AF          ; Guarda A y F en el Stack
           CALL  ASCII2HEX
           LD    B, A
           INC   HL
           CALL  ASCIIHEX
           SLA   A
           SLA   A
           SLA   A
           SLA   A
           OR    A,B
           LD    B, A        ; Guarda Resultado en B.
           POP   AF          ; Restaura A y F del Stack, aunque A no interesa.
           LD    A, B        ; Vuelve a poner el resultado en A.
           POP   HL
           POP   BC
           RET
```

c) Si bien es problema planteado es algo complicado de resolver, utilizando la subrrutuna de la parte b) que da nuevamente un código sencillo.

```
LD    HL, 8000h     ; HL apunta al primer número
CALL  NUMERO
LD    B, A          ; B ← primer número
INC   HL
INC   HL            ; HL apunta al segundo número
CALL  NUMERO        ; A ← segundo número
ADD   A, B
END
```

Ejercicio 3

Como indica la letra, inicialmente el bit 0 del puerto 08h es 0.

- A ← (puerto 20h) ; longitud en multiplos de 0.1 seg.
- (puerto 08h)[0] ← 1 ; no se debe modificar el resto de los bit del puerto.
- While A <> 0
  Delay (demorar 0.1 seg)
  A ← A - 1
- (puerto 08h)[0] ← 0 ; nuevamente, sin modificar el resto de los bits.

CODIGO

```
bit_out     EQU 0
puerto_in   EQU 20h
puerto_out  EQU 08h

IN    A, (puerto_in)    ; A ← long. del pulso
LD    B, A              ; Inicializo contador
IN    A, (puerto_out)
```

```
        SET   bit_out, A
        OUT   (puerto_out), A        ; (puerto 08H)[0]=1

LOOP:   CALL  DELAY
        DJNZ  LOOP
                                                    } (1)
        RES   bit_out,A
        OUT   (puerto_out),A         ; (puerto 08H)[0]=0

        END
```

Notas:
- La subrrutina DELAY es la realizada en el ejercicio 1.
- Si bien se agrega un error al no considerar las instrucciones DJNZ, RES y OUT en (1) se puede ver que el error relativo cometido es despreciable.

Ejercicio 4

- Inicializar contador
- Espera flanco ascendente
- Repeat
  Delay(1mseg)
  contador ← contador + 1
  Until flanco descendente.
- (puerto 06H) ← A

CODIGO

```
            bit_in      EQU 7
            puerto_in   EQU 03h
            puerto_out  EQU 06h

ESPERA_1:   IN    A, (puerto_in)
            BIT   bit_in, A
            JR    Z, ESPERA_1       ; si (puerto_in)[7] = 0 → se sigue esperando

            LD    B, 0              ; Inicializa contador.

CUENTA:     CALL  DELAY_1ms         ; retardo de 1mseg.
            INC   B                 ; B ← B + 1
            IN    A, (puerto_in)
            BIT   bit_in, A
            JR    NZ, CUENTA        ; si (puerto_in)[7] = 1 → se continúa contando

            LD    A, B              ; en B se obtiene el valor buscado.
            OUT   (puerto_out), A
```

Se debe entonces escribir la subrrutina DELAY_1ms para finalizar el ejercicio. La mecánica es la misma que la utilizada en el ejercicio 1. Se deja esta parte como ejercicio.

Ejercicio 5

Existen varios métodos para escribir el código requerido. Entre estos tenemos:

1. Realizar una comparación con varios IF anidados
2. Utilizar máscaras para ir “buscando” la solución.
3. Utilizar tablas en memoria.

1. En engorroso y fácil de equivocarse. Si se cambia la función lógica hay que rehacer todo el código.

2. Puede ser algo más simple, pero no deja de ser engorroso y fácil de equivocarse.
3. Es sencillo e intuitivo. Se guarda la tabla de verdad en memoria y luego solo basta con leer la dirección correspondiente.

```
          entrada   EQU 05h
          salida    EQU 34h

                    ORG 0000h

                LD  H, tabla/256    ; H ← 04h
loop:           IN  A, (entrada)    ;                                   11 T
                AND 00001111        ;                                    7 T
                LD  L, A            ; HL apunta a la respuesta           4 T
                LD  A, (HL)         ; Se carga la respuesta en A         7 T
                OUT (salida), A     ;                                   11 T
                JP  loop            ; Se vuelve a comenzar.             10 T

                    ORG 0400h       ;                  S3 S2 S1 S0  →  C2
tabla:      db 00h                  ; Dirección: 0400h  0  0  0  0  →   0
            db 00h                  ; Dirección: 0401h  0  0  0  1  →   0
            db FFh                  ; Dirección: 0402h  0  0  1  0  →   1
            db 00h                  ; Dirección: 0403h  0  0  1  1  →   0
            db 00h                  ; Dirección: 0404h  0  1  0  0  →   0
            db FFh                  ; Dirección: 0405h  0  1  0  1  →   1
            db 00h                  ; Dirección: 0406h  0  1  1  0  →   0
            db 00h                  ; Dirección: 0407h  0  1  1  1  →   0
            db 00h                  ; Dirección: 0408h  1  0  0  0  →   0
            db 00h                  ; Dirección: 0409h  1  0  0  1  →   0
            db FFh                  ; Dirección: 040Ah  1  0  1  0  →   1
            db 00h                  ; Dirección: 040Bh  1  0  1  1  →   0
            db 00h                  ; Dirección: 040Ch  1  1  0  0  →   0
            db FFh                  ; Dirección: 040Dh  1  1  0  1  →   1
            db 00h                  ; Dirección: 040Eh  1  1  1  0  →   0
            db FFh                  ; Dirección: 040Fh  1  1  1  1  →   1
```

En este caso la tabla de verdad se guarda en memoria a partir de la dirección 0400h. Tener presente que, de los 8 bits almacenados, el bit que interesa es el 2; el valor de los otros bits es indistinto.

Calculo de tiempos:

La lectura se da al final de la instrucción IN A, (n). Por simplicidad, se supone que es en el onceavo T.
La escritura se da al final de la instrucción OUT (n), A. Por simplicidad, se supone en el onceavo T.

El peor caso se da cuando el valor en la entrada varía un instante después de haber sido leída.

Total = 7T + 4T + 7T + 11T + 10T + 11T + 7T + 4T + 7T + 11T = 79T = 19,75us

Para el caso de utilizar compuertas, el peor caso se dará en un camino de 3 niveles (se incluye un posible “not”), por lo que si se hiciera con compuertas el tiempo máximo de respuesta es de 30 ns

Observar que la diferencia en los retardos es de 3 órdenes.