Práctico 2
Programas en lenguaje ensamblador

Ejercicio 1

i) Sin utilizar las instrucciones de transferencia y comparación de bloques.

```
INICIO:  CP    A, (HL)          ; A - (HL)
         INC   HL               ; HL = HL + 1. No se modifican las banderas.
         DEC   BC               ; BC = BC - 1. No se modifican las banderas.
         JR    Z, FIN           ; si Z prendido voy a FIN  (Z fue modificada por CP A,(HL))
         LD    E, A             ; voy a usar A, debo respaldar su valor.
         LD    A, C
         OR    B                ; modifico banderas para comprobar si BC = 0
         LD    A, E             ; recupero valor de A. No modifica banderas.
         JR    NZ, INICIO       ; si Z apagado voy a INICIO
         OR    A, 0FFh          ; apago Z (además de modificar A)
FIN:
```

ii) Utilizando las instrucciones de transferencia y comparación de bloques.

```
INICIO:  CPIR                   ; Eso es todo
```

Se sugiere tener presente las instrucciones de transferencia y comparación de bloques. Sin duda que para estas tareas en particular, ahorran mucho código (y tiempo).

Ejercicio 2

(La siguiente solución está planteada para hacer el producto HL ← E x C)

Para implementar la multiplicación existen varios caminos, algunos de los cuales son:

1) Sumando E veces el valor C y discutir el signo (el tiempo de respuesta depende de E y C).
2) Aplicar el método aprendido en la "escuela",
3) Aplicar el Algoritmo de Booth. Si bien este último es rápido y sencillo, no se implementa en esta solución. Por referencias, en la web hay mucha información al respecto.

Se implementa la solución para el caso 2) modificado para números con signo representados en complemento a 2.

Para comenzar a explicar el algoritmo, primero se consideran números sin signo.

Ejemplo: 1011_b x 1001_b = 11 x 9 0 = 99

```
          1 0 1 1       (multiplicando)
        x 1 0 0 1       (multiplicador)
          -------
          1 0 1 1
        0 0 0 0
      0 0 0 0
    1 0 1 1
    -------------
    1 1 0 0 0 1 1  = 64 + 32 + 2 + 1 = 99
```

Resultado = $1011_b \times 2^0 + 1011_b \times 2^3$ = 11 + 11 x 8 = 11 + 88 = 99

Rápidamente se puede observar que:
- si el bit del multiplicador es 0, no se suma nada

- si el bit del multiplicador es 1, se suma el multiplicando corrido "n" veces a la izquierda, donde "n", es el lugar del bit del multiplicador que es 1 (notar que 2^n es el peso del bit).

Observar que en el peor de los casos se necesitan tantos bits para el resultado como la suma de los bits del multiplicando y multiplicador.

Si ahora se considera el signo:
Opción (a):
- multiplicar los valores absolutos y ajustar el signo del resultado según el signo de los factores.

Opción (b):
- antes de comenzar se debe extender el signo del multiplicando al doble de bits.
- luego se opera igual que para números sin signo hasta el penúltimo bit
- si el bit más significativo es 1, se resta en lugar de sumarse (si es cero, no se hace nada).

Una forma de ver el algoritmo de la opción (b) es razonar en base al peso relativo de cada bit en un nro. en complemento a 2. P. ej. para 4 bits sería: -8, 4, 2, 1, es decir que el número $1011_b = -8 + 2 + 1 = -5$

Se implementa la opción (b).

Ejemplo: $1011_{2c} \times 1001_{2c} = 0010\ 0011_{2c} = -5 \times (-7) = 35$

```
    1 1 1 1 1 0 1 1       (multiplicando con signo extendido)
  x         1 0 0 1       (multiplicador)
    ---------------
    1 1 1 1 1 0 1 1
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0
  -  1 1 0 1 1             (en vez de sumar, se resta)
  --------------------
    0 0 1 0 0 0 1 1
```

Resultado = 1111 $1011_{2c} \times 2^0$ - 1 $1011_{2c} \times 2^3 = 0010\ 0011_{2c} = 35$

Pseudocódigo (Resultado = E x C)

```
Resultado ← 0
Extender el signo de E a 16 bits.
Para i = 0 hasta 6 hago
        Si C[i] = 1 entonces Resultado ← Resultado + E
        E ← E[6..0],0                   ; dejo E desplazado a la izq. para la proxima.
proximo i
Si C[7] = 1 entonces Resultado ← Resultado – E
```

CÓDIGO:

```
        LD      HL, 00h         ; Inicializo el resultado
        LD      A, 00h
        BIT     7, E
        JR      Z, POS
NEG:    CPL
POS:    LD      D, A            ; Extiendo el signo de E a 16 bits (utilizo DE)
        LD      B, 7            ; Inicializo contador.
MULT:   SRL     C               ; 0,C[7..1] → C , C[0] → Cy
        JP      NC, DESP
        ADD     HL, DE          ; Si Ci = 1 entonces Resultado ← Resultado + DE
DESP:   SLA     E               ; desplazo E a la izquierda, queda MSB en Cy
        RL      D               ; roto D a la izq. tomando el Cy
        DJNZ    MULT            ; próximo i

        SRL     C               ; 0,C[7..1] → C , C[0] → Cy
        JP      NC, FIN
```

```
        CCF                 ; pues Cy = 1 y lo necesito = 0 porque no hay resta 16 sin Cy
        SBC   HL, DE        ; Si C7 = 1 entonces Resultado ← Resultado – DE
FIN:
```

Ejercicio 3

Lo que se tiene que hacer se puede resumir a los siguientes pasos:

1) escribir el bloque con AAh, 55h
2) chequear; si error → bit1 = 1 de puerto 01h y fin
3) escribir el bloque con 55h, AAh
4) chequeo; si error → bit1 = 1 de puerto 01h y fin
5) bit0 = 1 de puerto 01h

CÓDIGO:

```
inicio   EQU  400H          ; EQU no es una instrucción, sino una directiva para el
largo    EQU  256d          ; compilador. EQU  le indica al  compilador que cada vez
puerto   EQU  01h           ; que  aparezca el  "símbolo"  (palabra) que está  a la izq.
val_ok   EQU  01h           ; lo debe sustituir  por el valor que se indica a la derecha.
val_err  EQU  02h


; Escribo el bloque con 0AAh, 55h…..
         LD   IX, inicio    ; inicializo puntero
         LD   B, largo/2    ; inicializo largo a chequear
ESC_0:
         LD   (IX), 0AAh
         LD   (IX+1), 55h
         INC  IX
         INC  IX
         DJNZ ESC_0

; Chequeo lo que se escribió
         LD   IX, inicio
         LD   B, largo/2

LEO_0:   LD   A, 0AAh
         CP   (IX)
         JR   NZ,  ERROR    ; Si no hay coincidencia → hay un error.
         LD   A, 55h
         CP   (IX+1)
         JR   NZ, ERROR     ; Si no hay coincidencia → hay un error.
         INC  IX
         INC  IX
         DJNZ LEO_0


; Escribo el bloque con 0AAh, 55h…..
         LD   IX, inicio    ; inicializo puntero
         LD   B, largo/2    ; inicializo largo a chequear
ESC_1:
         LD   (IX), 55h
         LD   (IX+1), 0AAh
         INC  IX
         INC  IX
         DJNZ ESC_1

; Chequeo lo que se escribió
         LD   IX, inicio
         LD   B, largo/2
```

```
LEO_1:  LD    A, 055h
        CP    (IX)
        JR    NZ,  ERROR        ; Si no hay coincidencia → hay un error.
        LD    A, 0AAh
        CP    (IX+1)
        JR    NZ, ERROR         ; Si no hay coincidencia → hay un error.
        INC   IX
        INC   IX
        DJNZ LEO_1

OK:     LD    A, val_ok
        JR    FIN

ERROR:  LD    A, val_err

FIN:    OUT   (puerto), A       ; El registro A contiene el valor a escribir en el puerto.
```

Ejercicio 4

En ASCII los números 0 al 9 y las letras A a la F se codifican como se indica a continuación:

Caracter	Código	Caracter	Código	Caracter	Código	Caracter	Código
0	30h	4	34h	8	38h	D	43h
1	31h	5	35h	9	39h	D	44h
2	32h	6	36h	A	41h	E	45h
3	33h	7	37h	B	42h	F	46h

Esto implica que si R es un registro que contiene un dígito hexadecimal en código ASCII, entonces:

- Si R – 30 ≥ 10 entonces R es una letra cuya codificación en binario es R – 41h+ 10d
- SI R – 30 < 10 entonces R es un número cuya codificación en binario es R – 30h

CÓDIGO:

```
            digito  EQU 30h
            letra   EQU 41h

            LD    A, (HL)             ; A ← digito menos significativo
            SUB   digito              ; A ← A  - ASCII('0')
            CP    10d                 ;
            JP    M, ES_NUMERO        ; Si A < 10d, voy a ES_NUMERO
ES_LETRA:   SUB   A, letra - digito - 10 ; A ← A - ASCII('A') + ASCII('0') + 10
ES_NUMERO:  LD    B,A                 ; aquí A contiene el valor binario correcto
                                      ; B ← digito menos significativo
            INC   HL
            LD    A, (HL)
            SUB   digito              ; A ← A - ASCII('0')
            CP    10
            JP    M, ES_NUMERO_2      ; Si A < 10d, voy a ES_NUMERO_2
ES_LETRA_2: SUB   A, letra-digito-10  ; A ← A - ASCII('A') + ASCII('0') + 10
ES_NUMERO_2: SLA  A
            SLA   A
            SLA   A
            SLA   A                   ; Multiplico por 16 (corro 4 bit a la izq.)
            OR    B                   ; Agrago los cuatro bits menos sigificativos.
```

Nota:
Tener presente que EQU no es una instrucción, sino una directiva para el compilador. EQU le dice al compilador que cada vez que aparezca "letra" lo sustituya por 41h. Idem para "dígito".

Ejercicio 5

Pseudocódigo:

```
Leer "n" de la memoria                                  ; A ← (HL)
A ← A x 2
Si hubo acarreo en la operación anterior entonces FIN
Guardo el resultado parcial en B                        ; B ← A
A ← A x 2
Si hubo acarreo en la operación anterior entonces FIN
A ← A x 2                                               ; A contiene n x 8
Si hubo acarreo en la operación anterior entonces FIN
A ← A + B
FIN
```

CÓDIGO:

```
POR 10:  LD   A,(HL)      ; Leo "n".  A ← (HL)
         SLA  A           ; A ← A x 2
         JR   C, FIN      ; Si hay un acarreo, vuelve con Cy =1
         LD   B,A         ; B ← A .  B contiene n x 2
         SLA  A           ; A contiene n x 4
         JR   C, FIN      ; Si hay un acarreo, vuelve con Cy =1
         SLA  A           ; A contiene n x 8
         JR   C, FIN      ; Si hay un acarreo, vuelve con Cy =1
         ADD  A, B        ; A ← n x 8 + n x 2
FIN:
```

Ejercicio 6

a) Sin utilizar la instrucción DAA

Puede ser conveniente recordar como se suman 2 números A y B expresados BCD (no empaquetados):

- C ← A + B ; A, B y C registros de 8 bits.
- Si C ≥ 10 entonces C ← C + 6
- En C queda el resultado, en C[3..0] se indican las unidades y en C[7..4] las decenas (que para este caso a lo sumo será 1)

Pseudocódigo:

1) guardo D ← A y E ← B (pues voy a alterar los registros A y B)
2) A ← 0000,A[3..0]; B ← 0000,B[3..0]
3) A ← A + B ; L ← 0h
4) Si A ≥ Ah ➔ L ← 6h ;
5) A ← D + E + L (utilizo los valores guardados)
6) Si A ≥ A0h ➔ L ← L + 60h
7) A ← D + E + L

CÓDIGO:

```
            mascara  EQU  00001111b

SUMO_BCD_PCK: LD   D, A
              LD   E, B
              AND  mascara
              LD   C, A          ; C = 0000, A[3,0]
              LD   A, B
              AND  mascara       ; A = 0000, B[3,0]
              ADD  C             ; A ← A + C
              LD   L, 0h         ; inicializo L
              CP   0Ah
```

```
                    JP    M, SIGO1      ; SI A < Ah  entonces continuo
                    LD    L, 6h         ; sino L ← 6h

SIGO1:              LD    A, D          ; (D contiene el valor inicial de A)
                    ADD   L             ; A ← A + L (no hay carry pues D + 6 < FFh)
                    ADD   E             ; A ← A + E (E contiene el valor inicial de B)
                    RR    A             ; roto a la derecha con carry.
                                        ; Así comparo con el Cy incluido
                    CP    50h           ; 50h es igual a 0A0h corrido hacia la derecha.
                    LD    A, L
                    JP    M, SIGO 2     ; SI A < A0h  entonces A ← L y continuo
                    ADD   60h           ; sino  A ←  L + 60h
SIGO2:              ADD   D             ; A ← L + D
                    JR    C, CARRY      ; Si acarreo aseguro quede indicado
                    ADD   E             ; A ← A + E (E contiene el valor de B)
                    JR    FIN           ; Si da Cy = 1 queda solo
CARRY:              ADD   E             ; A ← A + E (teniendo en cuenta que hubo Cy)
                    SCF                 ; Cy ← 1
FIN:
```

b) Utilizando la instrucción DAA

```
SUMO_BCD_PCK:  ADD   A, B
               DAA
FIN:                                    ; Eso es todo
```

Ejercicio 7

La idea de este problema es utilizar las instrucciones de corrimiento en forma adecuada.

CÓDIGO:

```
INVERTIR:      LD    A, (IX)          ; A ← (IX)
               LD    B, 8             ; Inicializo contador (para contar los 8 bits)
LOOP:          RL    A                ; Cy ← A[7]; A ← A[6..0],Cy
               RR    C                ; C[7] ← Cy; C[6..0] ← C[7..1]
               DJNZ LOOP              ; Si lo hice 8 veces, entonces sigo
               LD (IX),C
```