# SOLUCIONES COMENTADAS AL EXAMEN DE ESTRUCUTRAS DE LOS COMPUTADORES.
# SEPTIEMBRE DE 1.997

1º) Realizar un programa en ensamblador que lea una cadena de 80 caracteres y que como salida indique el número de veces que aparece la letra a, la e, la i , la o y la u que aparecen en el texto. Si no aparece un tipo de vocal, se debe escribir el literal *ninguna* en vez de poner 0.

Ejemplo:
**c:\>convoc**
Introduce una frase no mayor de 80 caracteres:
**La casa de Pedro está cerca del rio.**
Número de veces que aparece la letra a: 5
Número de veces que aparece la letra e: 3
Número de veces que aparece la letra i: 1
Número de veces que aparece la letra o: 2
Número de veces que aparece la letra u: ninguna.

**Nota**: Las frases en negrita se supone que es lo que se debería escribir por teclado.

```
;
; A continuación declaramos el segmento de datos.
;
DATOS SEGMENT
    Terminar EQU 4C00h          ; Función de la INT 21h para devolver control al DOS.
    LeerCadena EQU 0Ah          ; Función de la INT 21h para leer una cadena de caracteres.
    EscribirCadena EQU 09h      ; Función de la INT 21h para escribir una cadena de caracteres.
    EscribeCaracter EQU 02h     ; Función de la INT 21h para escribir un carácter.
    SaltaLinea EQU 0Dh; Código ASCII de salto de línea.
    PrincipioLinea EQU 0Ah      ; Cödigo ASCII para ir al principio de línea.
    BitInferiores EQU 0Fh       ; Máscara para quedarse con los 4 bits inferiores de un número.
    ;
    ; Mensajes de la interfaz con el usuario.
    ;
    MsgPeticionFrase DB 'Introduce una frase no mayor de 80 caracteres: ', SaltaLinea, PrincipioLinea, '$'
    MsgResultaA DB 'N£mero de veces que aparece la letra A: $'
    MsgResultaE DB 'N£mero de veces que aparece la letra E: $'
    MsgResultaI DB 'N£mero de veces que aparece la letra I: $'
    MsgResultaO DB 'N£mero de veces que aparece la letra O: $'
    MsgResultaU DB 'N£mero de veces que aparece la letra U: $'
    MsgNinguna DB 'ninguna', SaltaLinea, PrincipioLinea, '$'
    MsgEnter DB SaltaLinea, PrincipioLinea, '$'
    NumeroVocales DB 0,0,0,0,0 ; Estructura para dejar el número de vocales (a, e, ... u)
    Cadena DB 81,0
            DB 81 DUP (0) ; Estrucutra para dejar la cadena de 80 caracteres.
DATOS ENDS
;
; Definición de la pila.
;
PILA SEGMENT STACK
    DB 1024 DUP (" ")
PILA ENDS
```

```
;
; Una posible implementación para el código del programa es la siguiente:
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento Principal es procedimento que llama a los siguientes procedimientos:
;           LeerFrase: Lee una frase de 80 caracteres por teclado.
;           Enter: Escribe un salto de línea por pantalla.
;           ContarVocales: cuenta el número de vocales que aparecen en una frase.
;           ConvertirADecimal: convierte de hexadecimal a binario.
;           EscribeResultado: escribe el número de vocales de cada tipo.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
CODIGO SEGMENT
      ASSUME CS:CODIGO, SS:PILA, DS:DATOS ; Definición de los segmentos del programa
Principal PROC FAR
      MOV AX, DATOS   ; Iniciación del segmento de datos
      MOV DS, AX          ; del programa.
      CALL LeerFrase       ; Llamada al procedimiento que lee la frase.
      CALL Enter; Llamada a un procedimiento que realiza un salto de línea.
      CALL ContarVocales ; Llamada a un procedimiento que cuenta el número de vocales.
      CALL ConvertirADecimal ; Llamada a un procedimiento que convierte a decimal el número
                                    ; de vocales. Esto no se pediá en el examen.
      CALL EscribeResultado      ; Llamada al procedimiento que escribe el resultado.
      MOV AX, Terminar             ; Se devuelve el control
      INT 21h                           ; Al sistema operativo.
Principal ENDP
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    El procedimiento LeerFrase lee una frase de 80 caracteres por teclado. Requiere:
;           Cadena: estructura en la que se almacena la frase. Al ser leída mediante la función
;           del DOS 0Ah requiere 83 bytes de almacenamiento, distribuidos de la siguiente forma:
;                      Primer byte: número máximo de caracteres a almacenar más el ENTER.
;                      Segundo byte: posición en el que se almacena el número de caracteres que
;                                 se han leído realmente.
;                      Resto de bytes: deben ser el número de caracteres deseados más uno para
;                      almacenar el ENTER
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
LeerFrase PROC NEAR
      PUSH AX              ; Salvaguardamos los registros que vamos a
      PUSH DX              ; sobreescribir en el procedimiento.
      LEA DX, MsgPeticionFrase   ; Se escribe la solicitud de una frase por teclado.
      MOV AH, EscribirCadena
      INT 21h
      MOV AH, LeerCadena          ; Se lee una frase por teclado.
      LEA DX, Cadena
      INT 21h
      POP DX               ; Recuperamos los valores de los registros
      POP AX               ; antes de la llamada al procedimiento
      RET
LeerFrase ENDP
```

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento ContarVocales cuenta el número de vocales de cada tipo que aparecen
;  en la frase. Requiere:
;       Cadena: estructura en la que se encuentra almacenada la frase leída.
;       NumeroVocales: estructura en la que se almacena el número de vocales (en hexadecimal)
;       de cada tipo.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ContarVocales PROC NEAR
        PUSH AX                 ; Salvaguardamos los registros que vamos
        PUSH DX                 ; a emplear en este procedimiento.
        PUSH BX
        PUSH CX
        LEA BX, Cadena         ; Asignamos a BX la dirección de comienzo de la frase leída.
        INC BX
        XOR CX, CX
        MOV CL, [BX]            ; En el contador CL ponemos el número de caracteres leídos.

   BucleContar:
        INC BX                  ; Avanzamos el puntero sobre CADENA.
        MOV AL, [BX]            ; Si es A, E, I, O, U saltamos a sumar uno a la
        CMP AL, 'A'             ; posición correspondiente en VOCALES.
        JE SumarA1
        CMP AL, 'E'
        JE SumarE1
        CMP AL, 'I'
        JE SumarI1
        CMP AL, 'O'
        JE SumarO1
        CMP AL, 'U'
        JE SumarU1
        JMP FinBucleContar
    SumarA1:
        MOV AL, NumeroVocales[0]          ; NumeroVocales(0) es el número de aes.
        INC AL
        MOV NumeroVocales[0], AL
        JMP FinBucleContar
    SumarE1:
        MOV AL, NumeroVocales[1]           ; NumeroVocales(1) es el número de ees.
        INC AL
        MOV NumeroVocales[1], AL
        JMP FinBucleContar
    SumarI1:
        MOV AL, NumeroVocales[2]            ; NumeroVocales(2) es el número de ies.
        INC AL
        MOV NumeroVocales[2], AL
        JMP FinBucleContar
    SumarO1:
        MOV AL, NumeroVocales[3]             ; NumeroVocales(3) es el número de oes.
        INC AL
        MOV NumeroVocales[3], AL
        JMP FinBucleContar
    SumarU1:
        MOV AL, NumeroVocales[4]              ; NumeroVocales(0) es el número de ues.
        INC AL
        MOV NumeroVocales[4], AL
```

```
  FinBucleContar:
    LOOP BucleContar
    POP CX                ; Recuperamos el valor que tenían los
    POP BX                ; registros antes de la llamada al procedimiento.
    POP DX
    POP AX
    RET
ContarVocales ENDP
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento EscribeResultado escribe el número de vocales de cada tipo que aparecen
;  en la frase. Requiere:
;      NumeroVocales: estructura en la que se almacena el número de vocales (en hexadecimal)  de cada tipo.
;      EscribeVocal: procedimiento que escribe el número de vocales de cada tipo.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
EscribeResultado PROC NEAR
    PUSH AX               ; Salvaguardamos los registros que vamos a
    PUSH DX               ; emplear en el procedimiento.
    LEA DX, MsgResultaA                 ; En DX dejamos la frase correspondiente a la A.
    MOV AL, NumeroVocales[0]            ; Llevamos a AL el número de veces que aparece la
    CALL EscribeVocal                   ; letra A y llamamos a EscribeVocal.
    LEA DX, MsgResultaE                 ; En DX dejamos la frase correspondiente a la E.
    MOV AL, NumeroVocales[1]            ; Llevamos a AL el número de veces que aparece la
    CALL EscribeVocal                   ; letra E y llamamos a EscribeVocal.
    LEA DX, MsgResultaI                 ; En DX dejamos la frase correspondiente a la I.
    MOV AL, NumeroVocales[2]            ; Llevamos a AL el número de veces que aparece la
    CALL EscribeVocal                   ; letra I y llamamos a EscribeVocal.
    LEA DX, MsgResultaO                 ; En DX dejamos la frase correspondiente a la O.
    MOV AL, NumeroVocales[3]            ; Llevamos a AL el número de veces que aparece la
    CALL EscribeVocal                   ; letra O y llamamos a EscribeVocal.
    LEA DX, MsgResultaU                 ; En DX dejamos la frase correspondiente a la U.
    MOV AL, NumeroVocales[4]            ; Llevamos a AL el número de veces que aparece la
    CALL EscribeVocal                   ; letra U y llamamos a EscribeVocal.
    POP DX                ; Recuperamos el valor del contenido de los registros
    POP AX                ; antes de la llamada al procedimiento.
    RET
EscribeResultado ENDP
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento EscribeNinguna escribe la palabra ninguna por pantalla
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
EscribeNinguna PROC NEAR
    PUSH AX               ; Salvaguardamos los registros que vamos a
    PUSH DX               ; emplear en el procedimiento.
    LEA DX, MsgNinguna
    MOV AH, EscribirCadena
    INT 21h
    POP DX                ; Recuperamos el contenido de los registros
    POP AX                ; antes de llamar a los procedimientos.
    RET
EscribeNinguna ENDP
```

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento EscribeVocal escribe el número de vocales del tipo deseado. Requiere:
;       NumeroVocales: estructura en la que se almacena el número de vocales (en hexadecimal)
;       de cada tipo.
;       EscribeVocal: procedimiento que escribe el número de vocales de cada tipo.
;       DX: contiene la dirección de memoria de la frase correspondiente.
;       AL: contiene el número de vocales del tipo deseado.
;       EscribeNinguna: llamada al procedimiento que escribe la palabra ninguna.
;       Enter: llamada al procedimiento que salta de línea.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
EscribeVocal PROC NEAR
        PUSH CX                 ; Salvaguardamos los registros que vamos
        PUSH DX                 ; a emplear en el procedimiento.
        PUSH AX
        XOR CX, CX
        MOV AH, EscribirCadena          ; Escribimos la frase correspondiente a la vocal
        INT 21h
        POP AX
        PUSH AX
        MOV AH, EscribeCaracter
        CMP AL, 00h                     ; Si hay cero vocales escribo ninguna.
        JZ EscribirNinguna
        MOV CL, 04h                     ; En otro caso escribo cada una de las dos cifras.
        SHR AL, CL
        JZ CifraBaja
        MOV DL, AL
        ADD DL, 30h
        INT 21h
   CifraBaja:
        POP AX
        PUSH AX
        MOV AH, EscribeCaracter
        AND AL, BitInferiores
        MOV DL, AL
        ADD DL, 30h
        INT 21h
        CALL Enter
        JMP Final
   EscribirNinguna:
        CALL EscribeNinguna
   Final:
        POP AX                  ; Recuperamos el contenido de los registros
        POP DX                  ; anterior a la llamada del procedimiento.
        POP CX
        RET
EscribeVocal ENDP
```

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento ConvertirADecimal convierte el número de vocales en hexadecimal, a
;  su correspondiente valor decimal. Requiere:
;       NumeroVocales: estructura en la que se almacena el número de vocales (en hexadecimal)
;       de cada tipo y en la que se deja el resultado decimal.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ConvertirADecimal PROC NEAR
       PUSH AX                ; Salvaguardamos los registros
       PUSH BX                ; que vamos a emplear en el procedimiento.
       PUSH CX
       PUSH DX
       XOR AX, AX
       XOR BX, BX
       XOR CX, CX
       XOR DX, DX
       MOV CL, 05h
       LEA BX, NumeroVocales          ; En BX almacenamos la dirección de la estructura de vocales.
   BucleVocales:
       MOV AL, [BX]                   ; Vamos dividiendo por 0Ah (10 decimal) hasta que
       CMP AL, 0Ah                    ; el resultado es menor que 0Ah (10 decimal) y dejando
       JB FinBucleVocales             ; el resultado en la posición de memoria apuntadapor BX.
       MOV DL, 0Ah
       DIV DL
       MOV DH, AL
       SHL DH, 1
       SHL DH, 1
       SHL DH, 1
       SHL DH, 1
       ADD DH, AH
       MOV [BX], DH
   FinBucleVocales:
       INC BX
       LOOP BucleVocales
       POP DX                 ; Recuperamos los contenidos de los registros antes
       POP CX                 ; de la llamada a los procedimientos.
       POP BX
       POP AX
       RET
ConvertirADecimal ENDP
```

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  El procedimiento Enter escribe un salto de línea por pantalla.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
Enter PROC NEAR
        PUSH AX                 ; Salvaguardamos el contenido de los registros
        PUSH DX                 ; que vamos a emplear en el procedimiento.
        LEA DX, MsgEnter
        MOV AH, EscribirCadena
        INT 21h
        POP DX                  ; Recuperamos el valor del contendio de los registros
        POP AX                  ; que tenían antes de llamar al procedimiento.
        RET
Enter ENDP
;
;  Fin del segmento de código.
;
CODIGO ENDS
END Principal
```

2º) Se tiene una máquina con la siguiente representación en coma flotante:

- Exponente 8 bits en signo-magnitud.
- Mantisa 8 bits en complemento a 1, sin emplear la técnica del bit implícito.

a) Calcular el rango para dicha representación.

b) Si se supone que recibimos el siguiente número en coma flotante, protegido mediante código Hamming, ver si el número llega correctamente o no.

**0 0010 0000 0111 0111 1111**

c) En caso de que llegue correctamente calcular su valor suponiendo el sistema de representación del enunciado

a) Números en coma flotante. Los números reales se representan según el formato $V(x) = M \times 2^e$. Es decir, una mantisa, multiplicada por 2 elevado a un exponente. Nos piden que calculemos el rango de dicha representación sin emplear la técnica del bit implícito.

<u>Rango para el exponente:</u>

Nos indican que el exponente viene expresado en signo-manitud y que tenemos un total de 8 bits para su representación.

El rango para un exponente en signo-magnitud es:

Positivos: $[\ 0,\ 2^{n-1}-1\ ] = [\ 0,\ 2^{8-1}-1\ ] = [\ 0,\ 2^{7}-1\ ] = [\ 0,\ 127\ ]$
Negativos: $[\ -0,\ -(2^{n-1}-1)\ ] = [\ -0,\ -(2^{8-1}-1)\ ] = [\ -0,\ (2^{7}-1)\ ] = [\ -0,\ -127\ ]$

<u>Rango para la mantisa:</u>

Nos indican que la mantisa es fraccionaria,está normalizada y no se emplea la técnica del bit implícito. Tenemos 8 bits para representarla, y se emplea complemento a 1.

Los números en complemento a 1 se encontrarán normalizados cuando:

Positivos: ,01 xxx …xx
Negativos: ,10 xxx …xx

En nuestro caso, si tenemos 8 bits, dos indicarán la normalización y seis se pueden poner con las combinaciones de ceros y de unos que deseemos.

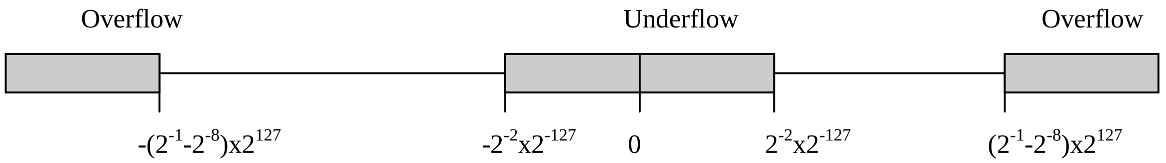
Para los positivos la menor y la mayor mantisa son, respectivamente:

Menor mantisa: ,01 000 000 = $2^{-2}$
Mayor mantisa: ,0111 111 = $2^{-1}-2^{-8}$

Y para los negativos, como el rango es simétrico en complemento a 1 tendre las mantisas:

$-2^{-2}$ y $-(2^{-1}-2^{-8})$

Con lo que la recta real queda de la forma:



b) Si se supone que recibimos el siguiente número en coma flotante, protegido mediante código Hamming, ver si el número llega correctamente o no.

**0 0010 0000 0111 0111 1111**

Nos indican que el número siguiente está codificado según el código de Hamming y se nos pide que comprobemos si el número ha sido recibido de forma correcta.

En el diagrama inferior se han resaltado en negrita los bits de paridad de Hamming. Que ocupan las posiciones correspondientes a las potencias de dos: 1, 2, 4, 8, 16 … A partir de las cuales se descomponen los demás números tal y como sigue:

1 = paridad
2 = paridad
3 = 2 + 1
4 = paridad
5 = 4 + 1
6 = 4 + 2
7 = 4 + 2 + 1

8 = paridad
9 = 8 + 1
10 = 8 + 2
11 = 8 + 2 + 1
12 = 8 + 4
13 = 8 + 4 + 1
14 = 8 + 4 + 2

15 = 8 + 4 +2 +1
16 = paridad
17 = 16 + 1
18 = 16 + 2
19 = 16 + 2 + 1
20 = 16 + 4
21 = 16 + 4 + 1

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | 0 | 1 | 0 | 0 | 0 | **0** | 0 | 0 | 1 | 1 | 1 | 0 | 1 | **1** | 1 | 1 | 1 | 1 | 1 |
| $\mathbf{b_1}$ | $\mathbf{b_2}$ | $b_3$ | $\mathbf{b_4}$ | $b_5$ | $b_6$ | $b_7$ | $\mathbf{b_8}$ | $b_9$ | $b_{10}$ | $b_{11}$ | $b_{12}$ | $b_{13}$ | $b_{14}$ | $b_{15}$ | $\mathbf{b_{16}}$ | $b_{17}$ | $b_{18}$ | $b_{19}$ | $b_{20}$ | $b_{21}$ |

El $\mathbf{b_1}$ protegerá a todos aquellos valores en cuya descomposición aparezca un 1. El $\mathbf{b_2}$ protege´ra todos aquellos valores en los que se encuentra un 2 en su descomposición y así sucesivamente.

Los bits de paridad (resaltados en negrita en la parte superior) protegen a los bits siguientes:
$\mathbf{b_1} = b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \oplus b_{13} \oplus b_{15} \oplus b_{17} \oplus b_{19} \oplus b_{21} = 0$ *correcto.*
$\mathbf{b_2} = b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} \oplus b_{14} \oplus b_{15} \oplus b_{18} \oplus b_{19} = 0$ *correcto.*
$\mathbf{b_4} = b_5 \oplus b_6 \oplus b_7 \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15} \oplus b_{20} \oplus b_{21} = 1$ *correcto.*
$\mathbf{b_8} = b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15} = 0$ *correcto.*
$\mathbf{b_{16}} = b_{17} \oplus b_{18} \oplus b_{19} \oplus b_{20} \oplus b_{21} = 1$ *correcto.*

Con lo que el número es correcto.

c) En caso de que llegue correctamente calcular su valor suponiendo el sistema de representación del enunciado

Como el número ha llegado de manera correcta debemos calcular su valor. Para ello, lo primero que debemos hacer es eliminar los bits de paridad del código Hamming, los cuales no forman parte del número.

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | | 0 | 0 | 0 | | 0 | 0 | 1 | 1 | 1 | 0 | 1 | | 1 | 1 | 1 | 1 | 1 |
| $\mathbf{b_1}$ | $\mathbf{b_2}$ | $b_3$ | $\mathbf{b_4}$ | $b_5$ | $b_6$ | $b_7$ | $\mathbf{b_8}$ | $b_9$ | $b_{10}$ | $b_{11}$ | $b_{12}$ | $b_{13}$ | $b_{14}$ | $b_{15}$ | $\mathbf{b_{16}}$ | $b_{17}$ | $b_{18}$ | $b_{19}$ | $b_{20}$ | $b_{21}$ |

De está manera nos quedan únicamente los dígitos que se corresponden con un número en coma flotante de las característcas del enunciado. Es decir, de la forma:

| | |
|---|---|
| <u>0</u> 0 0 0 0 0 1 1 | 1 0 1 1 1 1 1 1 1 |
| Exponente en Signo-Magnitud<br>( 8 bits) | Mantisa en Complemento a 1<br>( 8 bits) Sin bit implícito. |

Los números reales se representan según el formato $V(x) = M \times 2^{e}$

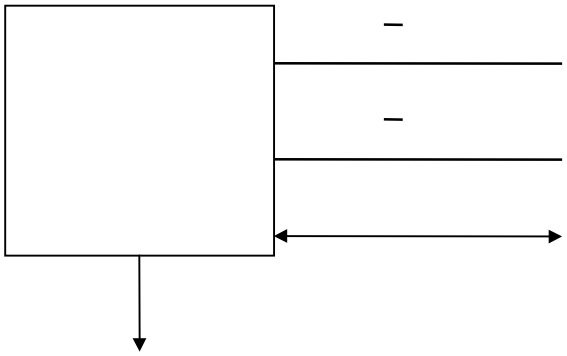
Si empezamos por el exponente, veremos que está representado en signo-magnitud. Mirando el bit de signo del exponente (dígito subrayado) veremos que indica que es positivo. Y su valor es 3. Luego el exponente e = 3.

La mantisa es fraccionaria, y está representada en complemento a 1. Al empezar por ,10 nos indica que se trata de un valor negativo. Para poder calcular su valor, volvemos a complementar el número, con lo que obtenemos el $,01000000 = 2^{-2}$. Luego si es negativo la mantisa valdrá $M = -2^{-2}$

Juntando todas las piezas obtendré:

$$V(x) = M \times 2^{e} = -2^{-2} \times 2^{3} = -2,0$$

3º) La CPU de la Figura 1 cuenta con un ancho de palabra de 16 bits. Se quiere dotar a esa CPU de una memoria con las siguientes características:
384 k x 16 de memoria ROM.
640 k x 16 de memoria RAM.
La ROM debe estar antes que la RAM.



a) Diseñar la memoria con el menor número de pastillas sabiendo que disponemos de las siguientes pastillas:

| Pastillas de memoria ROM | Pastillas de memoria RAM |
|---|---|
| 32 k x 1 | 64 k x 1 |
| 64 k x 1 | 64 k x 8 |
| 64 k x 8 | 32 k x 2 |

b) Modificar el esquema anterior para que dicha memoria sea capaz de almacenar también un bit de paridad. Se supone que sí es posible.

c) Si suponemos que la CPU de la figura es un 8086/88, en qué posición de memoria se buscaría el operando o se dejaría el resultado en los casos siguientes:

| | |
|---|---|
| MOV CX, ETIQUETA | IP = 0100h<br>CS = 0100h<br>DS = 0500h<br>ETIQUETA = 1234h |
| MOV [BX]+ARTICULO, AL | IP = 0220h<br>CS = 0100h<br>DS = 0200h<br>ARTICULO = 5000h<br>BX = 1000h |

a) Diseñar la memoria con el menor número de pastillas sabiendo que disponemos de las siguientes pastillas:

| Pastillas de memoria ROM | Pastillas de memoria RAM |
|---|---|
| 32 k x 1 | 64 k x 1 |
| 64 k x 1 | 64 k x 8 |
| 64 k x 8 | 32 k x 2 |

**1. Comprobar que nos piden algo que realmente se pueda hacer.**

Para poder comprobarlo, debemos fijarnos en el número de bits que tenemos en el bus de direcciones, y ver que con ese número de bits, podemos direccionar la memoria que se nos pide.

El bus de direcciones tiene las líneas desde la $A_0$ hasta la $A_{19}$, en total 20 bits. Nosotros necesitamos direccionar 640k palabras de memoria RAM y 384k palabras de memoria ROM, luego debemos de ser capaces de direccionar 640k + 384k de memoria total, RAM y ROM.

Para que seamos capaces de poder direccionar 640k + 384k = 1024k, necesitaremos 20 bits ($2^n$ ### 1024k, de donde n = 20). Por lo tanto, vemos que con los 20 bits del bus de direcciones **si** que podemos direccionar la memoria que se nos pide.

Tenemos que comprobar que el valor del bus de datos también sea correcto. Como queremos que cada posición de memoria almacene una palabra (16 bits), necesitaremos que el bus de datos nos soporte el ancho de la palabra.

El bus de datos tiene las líneas desde $D_0$ hasta $D_{15}$, en total 16 bits. Por tanto, también podemos acceder a posiciones de memoria en las que se almacenen 16 bits.

**2. Calcular el menor número de módulos de memoria que nos harán falta.**

Para la memoria ROM.

Disponemos de los siguientes módulos de memoria.

a) 32k x 1
b) 64k x 1
c) 64k x 8

384k = 12 pastillas del tipo a) para obtener el mapa de memoria.
32k

384k=6 pastillas de los tipos b) y c)
64k

Pero además de direccionar 384k, debemos ser capaces de almacenar en cada posición de memoria 16 bits, es decir, una palabra.

Por consiguiente, para almacenar 16 bits con elementos de los tipos a) y b) - que solamente almacenan 1 bit- necesitaremos 16 módulos. Pero para almacenar 16 bits con elementos del tipo c) únicamente necesitaremos 2 módulos.

Finalmente, el mínimo número de módulos lo calcularemos a partir del número de módulos necesarios para direccionar el mapa de memoria que nos piden y del número de módulos necesarios para almacenar el número de bits solicitado.

Con los datos del enunciado necesitaremos:

Módulos del tipo 32k x 1.

- 12 módulos de 32k (para poder direccionar 384k) x 16 (módulos necesarios para almacenar dos bytes), en total 12 x 16 = 192 módulos del tipo a)

Módulos del tipo 64k x 1.

- 6 módulos de 64k (para poder direccionar 384k) x 16 (módulos necesario para almacenar dos bytes), en total 6 x 16 = 96 módulos del tipo b)

Módulos del tipo 64k x 8.

- 6 módulos de 64k (para poder direccionar 384k) x 2 (módulos necesario para almacenar dos bytes), en total 6 x 2 = 12 módulos del tipo c)

Para la memoria RAM.

Disponemos de los siguientes módulos de memoria.
a) 64k x 1.
b) 64k x 8.
c) 32k x 2.

$\frac{640k}{64k}$ = 10 pastillas de los tipos a) y b) para obtener el mapa de memoria.

$\frac{640k}{32k}$= 20 pastillas del tipo c)

Pero además de direccionar 640k, debemos ser capaces de almacenar en cada posición de memoria 16 bits, es decir, una palabra.

Por consiguiente, para almacenar 16 bits con elementos del tipo a) - que solamente almacenan 1 bit- necesitaremos 16 módulos. Para almacenar 16 bits con elementos del tipo b) únicamente necesitaremos 2 módulos. Y con elementos del tipo c) necesitaré 8 módulo para poder direccionar 16 bits.

Finalmente, el mínimo número de módulos lo calcularemos a partir del número de módulos necesarios para direccionar el mapa de memoria que nos piden y del número de módulos necesarios para almacenar el número de bits solicitado.

Con los datos del enunciado necesitaremos:
Módulos del tipo 64k x 1.

- 10 módulo de 64k x 16 (módulos necesarios para almacenar un byte), en total 10 x 16 = 160 módulos del tipo a)

Módulos del tipo 64k x 8.

- 10 módulos de 64k  x 2 (módulos necesario para almacenar dos bytes), en total 10 x 2 = 20 módulos del tipo b)

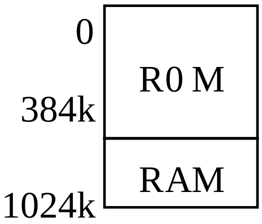
Módulos del tipo 32k x 2.

- 20 módulo de 32k  x 8 (módulos necesario para almacenar dos bytes), en total 20 x 8 = 160 módulo del tipo c)

Entonces necesitaremos 12 módulos de 64k x 8 para la memoria ROM  y 20 móduloa de 64k x 8 para la memoria RAM.

**3. Diseñar el mapa de memoria.**

Para diseñar el mapa de memoria, aunque no nos indican nada, haremos que la ROM esté antes que la RAM, tal y como reflejamos en la siguiente figura.



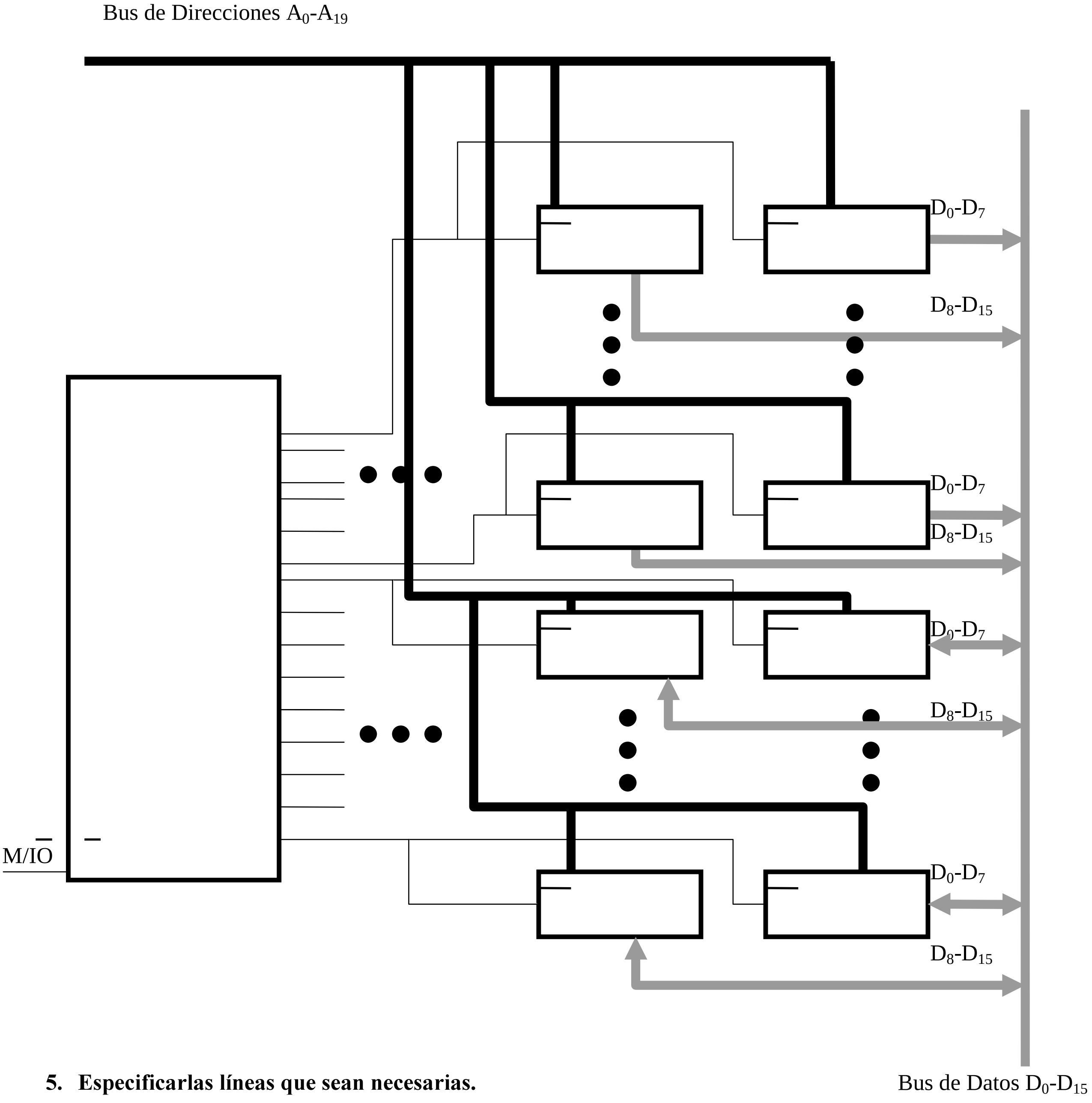
Además, para direccionar la memoria que se nos pide, necesitábamos 20 bits.

| $A_{19}$ | $A_{18}$ | $A_{17}$ | $A_{16}$ | A15 | ......... | $A_1$ | $A_0$ | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 1ª  fila de pastillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (0k-64k) |
| 0 | 0 | 0 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 2ª fila de pastillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (64k-128k) |
| 0 | 0 | 1 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 3ª fila de pastillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (128k-192k) |
| 0 | 0 | 1 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 4ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (192k-256k) |
| 0 | 1 | 0 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 5ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (256k-320k) |
| 0 | 1 | 0 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 6ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | ROM (320k-384k) |
| 0 | 1 | 1 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 1ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (384k-448k) |
| 0 | 1 | 1 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 2ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (448k-512k) |
| 1 | 0 | 0 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 3ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (512k-576k) |
| 1 | 0 | 0 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 4ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (576k-640k) |
| 1 | 0 | 1 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 5ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (640k-704k) |
| 1 | 0 | 1 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 6ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (704k-768k) |
| 1 | 1 | 0 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 7ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (768k-832k) |
| 1 | 1 | 0 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 8ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (832k-896k) |
| 1 | 1 | 1 | 0 | 0 | ......... | 0 | 0 | Mínima dirección. | 9ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (896k-960k) |
| 1 | 1 | 1 | 1 | 0 | ......... | 0 | 0 | Mínima dirección. | 10ª fila de patillas: |
| | | | | 1 | | 1 | 1 | Máxima dirección. | RAM (960k-1024k) |

Para poder direccionar 64k necesitaremos 16 bits, que se corresponden con las líneas desde la $A_0$ hasta la $A_{15}$ del bus de direcciones ($64k = 2^{16}$)

Si miramos la tabla del mapa de memoria que estamos creando, observamos que solamente nos quedan las líneas $A_{16}$, $A_{17}$, $A_{18}$ y $A_{19}$ del bus de direcciones para poder distinguir entre las 16 filas de pastillas y que con 4 bits podemos distinguir $2^4 = 16$ elementos.

## 4. Dibujar el esquema.



## 5. Especificarlas líneas que sean necesarias.

Las líneas que debemos escoger son:

- L/E lectura o escritura en las pastillas RAM.
- Lectura en las pastillas ROM.

b) Modificar el esquema anterior para que dicha memoria sea capaz de almacenar también un bit de paridad. Se supone que sí es posible.

Bastaría con que se añadieran 6 módulos de 64k x 1 de ROM y 10 módulos de 64k x 1 de RAM, en las filas correspondientes. Activándose con las mismas señales del decodificador.

c) Si suponemos que la CPU de la figura es un 8086/88, en qué posición de memoria se buscaría el operando o se dejaría el resultado en los casos siguientes:

| | |
|---|---|
| MOV CX, ETIQUETA | IP = 0100h<br>CS = 0100h<br>DS = 0500h<br>ETIQUETA = 1234h |
| MOV [BX]+ARTICULO, AL | IP = 0220h<br>CS = 0100h<br>DS = 0200h<br>ARTICULO = 5000h<br>BX = 1000h |

**El esquema del 8088/8086 divide la memoria en segmentos de 64k. Con lo que se debe saber en qué segmento y en que posición dentro del segmento se encuentran los operandos.**

**La forma de calcularlo es REGISTRO x 16 + Desplazamiento.**

**Así en la primera instrucción MOV CX, ETIQUETA. La posición de memoria del operando será:**

| | | |
|---|---|---|
| **REGISTRO DS x 16** | | **5000h** |
| **ETIQUETA** | + | **1234h** |
| **Posición** | | **6234h** |

**En la segunda instrucción** MOV [BX]+ARTICULO, AL La posición del operando es:

| | | |
|---|---|---|
| **REGISTRO DS x 16** | | **2000h** |
| **[BX]** | | **1000h** |
| **ARTICULO** | + | **5000h** |
| **Posición** | | **8000h** |