Soluciones comentadas de los problemas del examen de Estructuras de los Computadores (2 de Septiembre de 1.996)

1. Se tiene un ordenador con los siguientes formatos de representación:
    - Números enteros con 8 bits, representados en signo-magnitud.
    - Números en coma flotante con las siguientes características:
        - Mantisa expresada en signo-magnitud
        - El ancho de la mantisa es de 16 bits.
        - La mantisa es fraccionaria.
        - El exponente viene representado en exceso.
        - El ancho del exponente son 8 bits.

    Se pide:

    a) Calcular el rango de representación para los números enteros.

    b) Calcular el rango de representación para los números en coma flotante con bit implícito.

    c) Representar en el formato entero los números:
        - 1024.
        - -12.

    d) Representar en formato de coma flotante el número:
        - 12,5.

    e) Calcular el valor del número, suponiendo que no se ha utilizado la técnica del bit implícito en su representación:

1	10000110	011 1100 0000 0000

Solución:

a) Números enteros, quiere decirnos coma fija. El rango de los números en coma fija y signo-magnitud es el que podemos deducir de: $[-2^{n-1}+1, 2^{n-1}-1]$. El número n es el número de bits del que disponemos para poder representar el número en coma fija. En el enunciado nos dicen que contamos con 8 bits, representados en signo-magnitud. Esto es que en nuestro número n = 8. Por lo tanto, el rango de los números enteros en este sistema de representación será: $[-2^{8-1}+1, 2^{8-1}-1]= [-2^7+1, 2^7-1]$, es decir, [-127,127**]**, que como podemos comprobar es un rango simétrico.

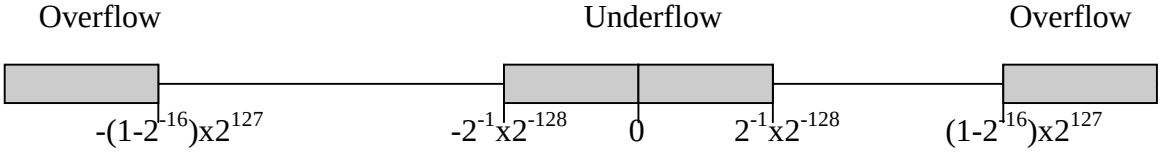
b) Números en coma flotante, nos quieren decir números reales. Los números reales se representan según el formato $V(x) = M \times 2^e$. Es decir, una mantisa, multiplicada por 2 elevado a un exponente. Nos dicen que la mantisa es fraccionaria, con lo que como ejemplo, tendremos que si queremos representar el 32,5 lo podemos representar como $0{,}325 \times 10^2$. Siendo 0,325 la mantisa M y 2 el exponente e. Además, si queremos emplear la técnica del bit implícito, deberemos de tener la mantisa normalizada.

   En el caso de la representación de signo-magnitud una mantisa fraccionaria estará normalizada si tenemos algo de la forma ,1xxxxxx.....xxxx. Es decir, una coma, un uno y después cualquier combinación de ceros y unos (se representa por una x).

   En signo-magnitud el rango es simétrico, con lo que tendremos el mismo número de valores positivos que de valores negativos. Entonces nos podemos ahorrar un cálculo, ya que podemos calcular el rango de los positivos y con sólo afectarles del signo tendremos el de los negativos.

   Empecemos por el menor número representable en mantisa normalizada fraccionaria en signo-magnitud y sin emplear la técnica del bit implícito. Tendremos una mantisa de la forma ,1xx.....xx. Como estamos en el caso de la mantisa más pequeña tendremos el valor ,100...00. Pero, ¿cuántos ceros? Nos dicen que la mantisa son 16 bits, pero uno se reserva para el signo; por tanto, tendremos 15 bits para representar

la mantisa, pero también se nos pide que empleemos la técnica del bit implícito, con lo que el número de bits que tenemos que tener en cuenta será de 16. Por tanto, la menor mantisa será: ,1000 0000 0000 0000. Y, ¿cuál es su valor? Tenemos un único uno, está afectado del peso -1, por lo que valdrá 2^{-1}, o lo que es igual 0,5.
En el caso de la mantisa más grande, tendremos ,111.....1. Es decir, en total 16 unos. Y valdrá: $1 - 2^{-n}$, siendo n el bit afectado del peso más negativo; en nuestro caso, -16. Por tanto, la mayor mantisa positiva valdrá $1-2^{-16}$, o lo que es igual, algo muy próximo a uno.
Resumiendo, el rango para la mantisa de los números positivos será $[2^{-1}, 1-2^{-16}]$, y al ser el rango simétrico, para los negativos será de $[-(1-2^{-16}), -2^{-1}]$



c) Tenemos que representar los números 1.024 y -12. Empecemos por el primero. Como se trata de un número entero, emplearemos la representación de coma fija. Para ello teníamos 8 bits. Si miramos el rango que calculamos en el apartado a) vemos que el número es mucho mayor que el que podríamos representar. Por lo tanto no es posible representar el número 1.024.
El caso del -12 si que se puede representar. El número quedaría 1 000 1100.
Para representar el número -12,5 iremos por pasos. De los 16 bits de la mantisa uno será para el signo y el resto para la magnitud. El signo es negativo luego el valor del bit de signo será 1. La magnitud del número es de 12,5; que en binario vale 01100,10. Entonces, el -12,5 será un número de la forma 1xxx....xxxxx01100,10. Pero vemos que no se encuentra normalizado. Además tenemos esa x que no sabemos con que rellenarlas. Para que el número estuviese normalizado deberíamos tener 1,1100 10 xxxx ... xxx. Pero esas x deberían ser ceros. Si recordamos cuando representábamos el número en coma fija, indicábamos que la magnitud se representaba en binario puro ya que teníamos un bit de signo independiente para poder indicar si el número era positivo o negativo.
Pero como en signo-magnitud tendremos un número normalizado cuando se tenga una expresión s,1xxx ... xxxx, deberemos ajustar el exponente. Es decir es como si tuviésemos el número multiplicado por 2^4. Con ese valor del exponente el número ya es fraccionario y se encuentra normalizado. ¿Hemos terminado?
No hemos terminado, porque también nos dicen que el exponente se representa en exceso. Esta representación lo que hace es sumar el valor del número (o restar si el número es positivo) a una determinada cantidad, el exceso. Como nos dicen que el exponente son 8 bits, el exceso es $2^{n-1} = 2^{8-1} = 2^7 = 128$, que será el número al que tenemos que sumar el exponente ajustado. Es decir, en exceso el exponente 4 se representará como 128 + 4 = 132, que en binario es: 1000 0100.
En definitiva el número se representará como sigue:

Signo	Exponente	Mantisa fraccionaria normalizada
1	1000 0100	1100 1000 0000 000

e) Debemos de saber cual es el valor del número:

1	10000110	011 1100 0000 0000

Vemos claramente diferenciados el bit de signo, el exponente y la mantisa. El valor del número es de la forma V(x) = *signo*Mx2^e.
Empecemos por el signo, tiene un valor 1 lo que indica que el número es negativo.
En cuanto al exponente vemos que el número representado vale 134. Pero nuestro exponente estaba en exceso, con lo que si el exceso era 128, tenemos que:
Exceso + Número = Número en exceso => 128 + N = 134 => N = 6. Es decir, el exponente vale 6.
Finalmente veamos cuanto vale la mantisa.

$2^{-1}-2^{-5}=$ 0,46875

Si juntamos todas las piezas tendremos que el número vale - 0,46875 x 2^6, y si lo operamos será el número -30,00.

2. Realizar un programa en ensamblador que lea un número hexadecimal por teclado e imprima las cifras desde ese número hasta el 1. Se supone que el número introducido por teclado está comprendido entre 1h y Fh, y que el cero no se imprimirá.

Ejemplos:
c:\>**programa**
5
54321
c:\>**programa**
C
CBA987654321

NOTA: las letras en negrita indica lo que se debería introducir por teclado.

```
;
;
; A continuación declaramos el segmento de datos.
;
Datos SEGMENT
  Leer                  EQU 01h
  Escribir              EQU 02h
  He_Leido_Numero       EQU 3Ah
  Corrige_Numero        EQU 30h
  Corrige_Letra         EQU 07h
  Salto_De_Linea        EQU 0Ah
  Enter                 EQU 0Dh
  Codigo_ASCII_Leido     DB  00h
  Numero_Leido           DB  00h
Datos ENDS
;
Pila SEGMENT STACK
        DB   1024  DUP (0)
Pila ENDS
```

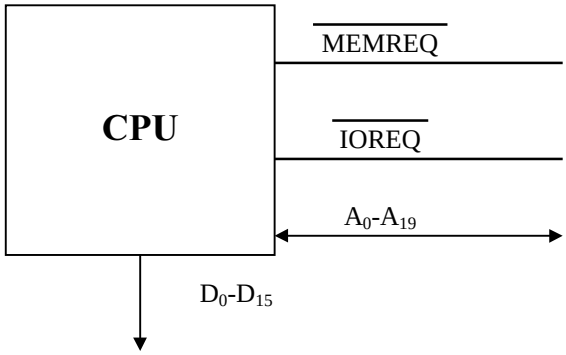
```
;
; Una posible implementación del segmento de código es la siguiente:
;
Codigo SEGMENT
   ASSUME CS:Codigo, DS:Datos   ;Inicialización de   los
   MOV AX, Datos                ;segmentos del programa.
   MOV DS, AX
;
; Leemos un número que se supone comprendido entre 01h y 0Fh por teclado.
;
   XOR AX, AX
   MOV AH, Leer
   INT 21h
;
; Salvaguardamos el número leído por teclado en la variable que tenemos
; para ese fin. Realmente  lo que hemos  leído no es  el número en sí, sino
; su código ASCII.
;
   MOV Codigo_ASCII_Leido, AL
;
; Convertimos el  código  ASCII  en el número  de veces que se debe mostrar
; en pantalla. Para  ello, debemos de  comprobar  si el número se corresponde
; con  el  código ASCII  de  una  letra (y  le   restaremos 37h para  obtener  su
; valor numérico)o un número (en cuyo caso se  le restara 30h). El  valor  con
; el  que   debemos   comparar  es  3Ah, ya  que   el  código  ASCII  que   se
; corresponde  con los números va desde el 30h hasta el 39h. Y almacenamos
; el valor corregido en la variable Numero_Leido.
;
   CMP AL, He_Leido_Numero
   JLE Era_Numero
   SUB AL, Corrige_Letra
Era_Numero:
   SUB AL, Corrige_Numero
   MOV Numero_Leido, AL
;
; Ahora saltamos de línea para que el resultado lo escriba en la línea
; siguiente. Para ello deberemos de escribir un 0Ah y después un 0Dh.
;
   XOR AX, AX
   XOR DX, DX
   MOV AH, Escribir
   MOV DL, Salto_De_Linea
   INT 21h
   MOV DL, Enter
   INT 21h
;
; Tenemos ahora en AL el verdadero  valor del  número que  se ha leído por
; teclado. Como tenemos que  escribir el número  ese  tantas veces  como el
; valor  que   hemos  leído, debemos  emplear un  bucle. El  valor  inicial del
```

```
; bucle será precisamente el contenido de AL. Y el valor que deberemos
; imprimir es el número que leímos antes pero almacenado en DL. También
; deberemos de indicar que deseamos escribir y no leer.
;
  XOR AX, AX
  XOR CX, CX
  MOV AH, Escribir
  MOV DL, Codigo_ASCII_Leido
  MOV CL, Numero_Leido
Bucle_De_Escribir:
   INT 21h
   DEC DL
   CMP DL, 40h
   JNE Seguir
   SUB DL, Corrige_Letra
   Seguir:
  LOOP Bucle_De_Escribir
;
; Finalmente deberemos de indicar al sistema operativo que se ha terminado
; el programa. Para ello se emplea la interrupción 21h y se almacena en AH
; el valor 4Ch.
;
  XOR AX, AX
  MOV AH, 4Ch
  INT 21h
;
; Con todo lo anterior ya hemos terminado el segmento de código.
;
Codigo ENDS
END
```

3) La CPU de la figura cuenta con un ancho de palabra de 16 bits. Se quiere dotar a esa CPU de una memoria con las siguientes características:
   - 256 k palabras (256 k x 16) de memoria ROM.
   - 512 k palabras (512 k x 16) de memoria RAM.

   La ROM debe estar antes que la RAM.
   Diseñar la memoria con el menor número de pastillas sabiendo que disponemos de las siguientes pastillas:

Pastillas de memoria ROM	Pastillas de memoria RAM
128 k x 1	128 k x 1
64 k x 8	256 k x 8
128 k x 16	256 k x 16



Solución

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 512k palabras de memoria RAM y 256k palabras de memoria ROM, luego debemos de ser capaces de direccionar 512k + 256k de memoria total, RAM y ROM.

Para que seamos capaces de poder direccionar 512k + 256k = 768k, 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 RAM.

Disponemos de los siguientes módulos de memoria.

a) 128k x 1
b) 256k x 8
c) 256k x 16

512k de RAM es una potencia de dos, así que:

512k = 4 pastillas del tipo a) para obtener el mapa de memoria.
128k

512k=2 pastillas de los tipos b) y c)
256k

Pero además de direccionar 512k, 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. Pero para almacenar 16 bits con elementos del tipo b) únicamente necesitaremos 2 módulos. Y para almacenar esa cantidad con módulos de tipo c) necesitaré 1 módulo.

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 128k x 1.

- 4 módulos de 128k (para poder direccionar 512k) x 16 (módulos necesarios para almacenar un byte), en total 4 x 16 = 64 módulos del tipo a)

Módulos del tipo 256k x 8.

- 2 módulos de 256k (para poder direccionar 512k) x 2 (módulos necesario para almacenar dos bytes), en total 2 x 2 = 4 módulos del tipo b)

Módulos del tipo 256k x 16.

- 2 módulos de 256k (para poder direccionar 512k) x 1 (módulos necesario para almacenar dos bytes), en total 2 x 1 = 2 módulos del tipo c)

Para la memoria ROM.

Disponemos de los siguientes módulos de memoria.

a) 128k x 1.
b) 64k x 8.
c) 128k x 16.

256k = 2 pastillas de los tipos a) y c) para obtener el mapa de memoria.
128k

256k= 4 pastillas del tipo b)
64k

Pero además de direccionar 256k, 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 1 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 128k x 1.

· 2 módulos de 128k (para poder direccionar 256k) x 16 (módulos necesarios para almacenar un byte), en total 2 x 16 = 32 módulos del tipo a)

Módulos del tipo 64k x 8.

· 4 módulos de 64k (para poder direccionar 256k) x 2 (módulos necesario para almacenar dos bytes), en total 4 x 2 = 8 módulos del tipo b)

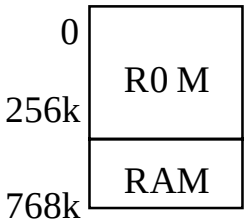
Módulos del tipo 128k x 16.

· 2 módulos de 128k (para poder direccionar 256k) x 2 (módulos necesario para almacenar dos bytes), en total 2 x 1 = 2 módulos del tipo c)

Entonces necesitaremos 2 módulos de 256k x 16 para la memoria RAM y otros 2 módulos de 128k x 16 para la memoria ROM.

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.

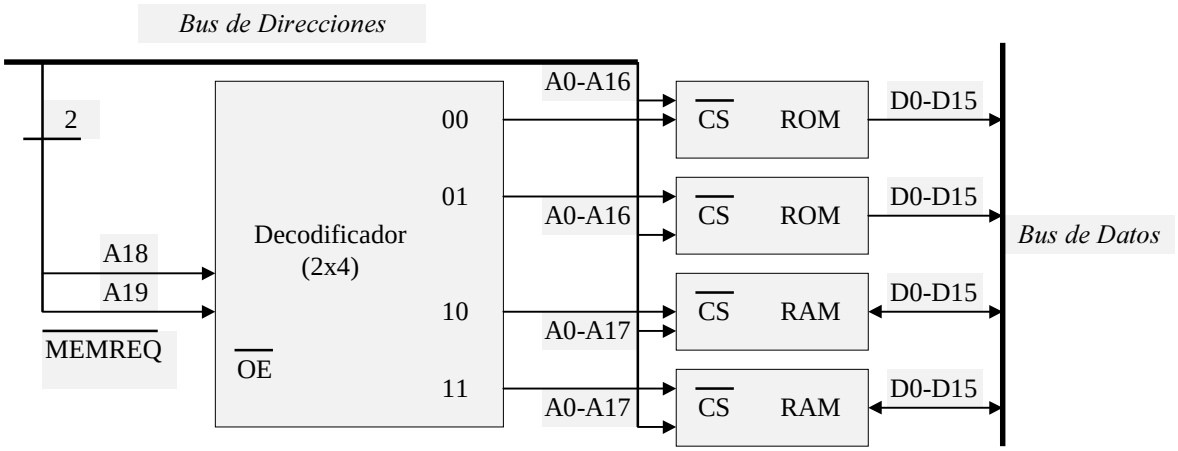
A_{19}	A_{18}	A_{17}	A_{16}	.....	A_1	A_0		
0	0	X	0 1	.....	0 1	0 1	Mínima dirección. Máxima dirección.	1ª fila de pastillas: ROM (0k-128k)
0	1	X	0 1	.....	0 1	0 1	Mínima dirección. Máxima dirección.	2ª fila de pastillas: ROM (128k-256k)
1	0	0 1	0 1	.....	0 1	0 1	Mínima dirección. Máxima dirección.	3ª fila de pastillas: RAM (256k-512k)
1	1	0 1	0 1	.....	0 1	0 1	Mínima dirección. Máxima dirección.	4ª fila de patillas: RAM (512k-768k)

Para poder direccionar 256k necesitaremos 18 bits, que se corresponden con las líneas desde la A_0 hasta la A_{17} del bus de direcciones ($256k = 2^{18}$)

Mientras que para poder acceder a 128k necesitaremos 17 bits, son las líneas desde la A_0 hasta la A_{16} del bus de direcciones ($128k = 2^{17}$)

Si miramos la tabla del mapa de memoria que estamos creando, observamos que solamente nos quedan las líneas A_{18} y A_{19} del bus de direcciones para poder distinguir entre las 4 filas de pastillas y que con 2 bits podemos distinguir $2^2 = 4$ elementos. Si nos fijamos en la parte de la tabla remarcada en gris, existen algunas combinaciones de la línea A_{17} que no se emplean.
Las combinaciones de la línea A_{17} que corresponden a la memoria ROM no se emplean. Ya que hemos visto que con las líneas A_{18} y A_{19} ya nos sirve.

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.