Problemas
Estructura y Tecnología de Computadores I
Informática de Gestión

PROBLEMA N°1

Estamos observando un sistema de computación y queremos saber:

-1- Cómo representar el número binario 1010 1100 0100 0111 en hexadecimal:
a) AC47 b) 47AC c) ACB4 d) 4CA7

-2- Cómo representar el número decimal 1227 en 16 bits complemento a 2:
a) CB04 b) F4CB c) 04CB d) ninguna de las anteriores

-3- Cómo representar el número decimal -1227 en 16 bits complemento a 2:
a) FB35 b) 8B35 c) 84CB d) F4CB

-4- Cómo representar el número decimal -1227 en 20 bits complemento a 2:
a) FF4CB b) 804CB c) 8FB35 d) FFB35

-5- C19E0000 es un número en formato IEEE754 (32 bits) que representa el número decimal:
a) 19,75 b) -19,75 c) -3,75 d) 4,25

-6- Cómo se representa el número decimal 1540 en formato IEEE754 (32 bits):
a) 8240 8000 b) 0240 8000 c) C4C0 8000 d) 44C0 8000

-7- Cuál es el error, en valor absoluto, que se comete al representar el número decimal 291,072 con el número en formato IEEE754 (16 bits) 4391:
a) 1,072 b) 0,928 c) 0,464 d) 0,536

PROBLEMA N°2

-1- Represente el número 554 (en base decimal) en 16 bits complemento a 2:
a) 0285 b) 0151 c) 0105 d) 022A

-2- Represente el número -554 (en base decimal) en 16 bits complemento a 2:
a) FEFB b) FD7B c) FEAF d) FDD6

-3- Represente el número -554 (en base decimal) en 20 bits complemento a 2:
a) FFDD6 b) FFEAF c) FFD7B d) FFEFB

-4- Represente el número C282 0000 (en formato IEEE754 32 bits) en decimal:
a) -132 b) -65 c) -41 d) -44

-5- Represente el número 41 (en base decimal) en formato IEEE754 16 bits:
a) 4224 b) 4304 c) 4282 d) 4230

-6- Represente el número -41 (en base decimal) en formato IEEE754 16 bits:
a) C230 b) C282 c) C304 d) C224

-7- Represente el número 0'40625 (en base decimal) en formato IEEE754 16 bits:
a) 3E50 b) 3ED0 c) 3ED8 d) 3F98

PROBLEMA N°3

-1- Represente el número 1012 (en base decimal) en 16 bits complemento a 2:
a) 03F4 b) 0AF1 c) 03CB d) 1F48

-2- Represente el número -1012 (en base decimal) en 16 bits complemento a 2:
a) FC0C b) E0B8 c) F50F d) FC35

-3- Represente el número -971 (en base decimal) en 20 bits complemento a 2:
a) FE0B8 b) FF50F c) FFC0C d) FFC35

-4- Represente el número 3ED0 0000 (en formato IEEE754 32 bits) en decimal:
a) 0'421875 b) 0'203125 c) 0'40625 d) 1'1875

-5- Represente el número 480 (en base decimal) en formato IEEE754 16 bits:
a) 43B0 b) 43F0 c) 43A0 d) 43E0

-6- Represente el número -480 (en base decimal) en formato IEEE754 16 bits:
a) C3E0 b) C3A0 c) C3F0 d) C3B0

-7- Diga cuál de los siguientes números (en formato IEEE754 16 bits) aproxima mejor el número 0'003387 (en base decimal):
a) 3B51 b) 3B79 c) 3B5E d) 3B6B

PROBLEMA N°4

Se quiere diseñar un sistema de supervisión de la temperatura en una caldera. Este sistema funciona midiendo la temperatura de la caldera y codificándola en digital según se recoge en la siguiente tabla. El sistema encenderá unos diodos de colores en función del modo de operación en que se encuentre la caldera. Si el código hexadecimal se sitúa en un registro, se pide:
Diseñar unos circuitos combinacionales que tengan como entradas los cuatro bits (abcd) situados en el registro y como salida el valor de los diodos. La función para el diodo verde se llamará L1. La función para el diodo naranja se llamará L2. La función para el diodo rojo se llamará L3.

Temperatura (°C)	Cód. Hex (abcd)	Modo de operación	Diodo
440	0	Opera correctamente	L1 (VERDE)
450	1		
460	2		
470	3	Opera incorrectamente Sin peligro	L1 (VERDE) Y L2 (NARANJA)
480	4		
490	5		
500	6		
510	7	Zona límite	L2 (NARANJA)
520	8		
530	9	Zona peligrosa	L2 (NARANJA) Y L3 (ROJO)
540	A		
550	B		
560	C		
570	D	Riesgo de explosión	L3 (ROJO)
580	E		
590	F		

-1- Expresión en minterms de L1
a) L1=m0+m2+m4+m6+m8
b) L1=m0+m1+m2+m3+m4+m5+m6
c) L1=m2+m3+m4+m5+m6+m7+m8
d) L1=m2+m3+m4+m5+m6

-2- Expresión en minterms de L2
a) L2=m3+m4+m5+m6+m7+m8
b) L2=m3+m4+m5+m6+m7+m8+ m10+m11+m12
c) L2=m3+m4+m5+m6
d) L2=m3+m4+m5+m6+m7+m8+m9+ m10+m11+m12

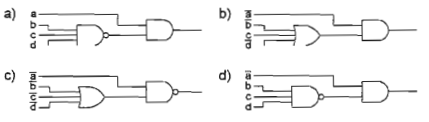
-3- Expresión en minterms de L3
a) L3=m9+m10+m11+m12+m13+ m14+m15
b) L3=m11+m12+m13+m14+m15
c) L3=m9+m10+m11+m12+m13
d) L3=m9+m11+m12+m13+m14+m15

-4- La función minimizada para L1
a) L1=$\bar{a}\bar{b}\bar{c}\bar{d}$
b) L1=$\bar{a}$ ($\bar{b}$+$\bar{c}$+$\bar{d}$)
c) L1=a (b+c+d)
d) L1=a$\bar{b}$+$\bar{a}$cd+b$\bar{c}$

-5- La función minimizada para L2
a) L2=$\bar{a}$ ($\bar{b}$+$\bar{c}$+$\bar{d}$)
b) L2=a$\bar{b}$+$\bar{a}$cd+b$\bar{c}$
c) L2=$\bar{a}$b+a$\bar{b}$+$\bar{a}$cd+b$\bar{c}\bar{d}$
d) L2=a (b+c+d)

-6- La función minimizada para L3
a) L3=$\bar{a}$b+a$\bar{b}$+$\bar{a}$cd+b$\bar{c}\bar{d}$
b) L3=$\bar{a}\bar{b}\bar{c}\bar{d}$
c) L3=a (b+c+d)
d) L3=b (a$\bar{b}$+$\bar{a}$cd)

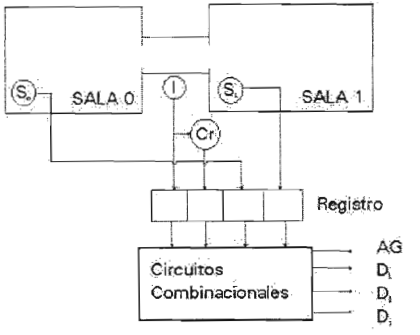
-7- Diga qué circuito podría servir para activar el diodo L1



PROBLEMA N°5

Se desea diseñar un circuito combinacional que ayude a vigilar un museo. El museo tiene dos salas (SALA 0 y SALA 1) unidas por un pasillo. Cada sala tiene un sensor (S_0 y S_1) que detecta la presencia de personas de forma infalible, de tal manera que cuando hay presencia pone su salida a 1, siendo 0 en caso contrario (S_i=0 no hay nadie; S_i=1 hay alguien).
La vigilancia se lleva a cabo con 2 guardias de seguridad, uno hace la ronda y otro observa un panel de control. El guardia que realiza la ronda tiene una llave con la que acciona el interruptor I, que se encuentra en el pasillo. Antes de entrar en la SALA 0 pone el interruptor en la posición 0 y antes de entrar en la SALA 1 pone el interruptor en la posición 1. Cada vez que se acciona este interruptor se pone en marcha un temporizador (señal Cr=0) que cronometra 10 minutos, al cabo de los cuales, de no haberse accionado el interruptor I pondrá la señal Cr=1.
El guardia que atiende el panel observa dos luces:
- La luz D_1 (diodo LED verde) informa que existe presencia del vigilante en la sala adecuada.
- La luz D_2 (diodo LED rojo) informa que el vigilante no acabó su ronda a tiempo.
Por último cuando se detecta una presencia extraña en cualquiera de las salas suena la alarma general AG.

I	Cr	S_0	S_1	D_1	D_2	AG	D
0	0	0	0	0	0	0	
0	0	0	1	0	0	1	
0	0	1	0	1	0	0	
0	0	1	1	0	0	1	
0	1	0	0	0	1	0	
0	1	0	1	0	1	1	
0	1	1	0	1	1	0	
0	1	1	1	0	1	1	
1	0	0	0	0	0	0	
1	0	0	1	1	0	0	
1	0	1	0	0	0	1	
1	0	1	1	0	0	1	
1	1	0	0	0	1	0	
1	1	0	1	1	1	0	
1	1	1	0	0	1	1	
1	1	1	1	0	1	1	



Se pide contestar a las siguientes preguntas:

Nota: el símbolo ‾ significa el complemento de la variable.

-1- Expresión en minterms de AG.
A) AG=$\prod M_i$ con i=1,3,5,7,10,11,14,15
B) AG=$\sum m_i$ con i=0,2,4,6,8,9
C) AG=$\sum m_i$ con i=1,3,5,7,10,11,14,15
D) AG=$\prod M_i$ con i=0,2,4,6,8,9

-2- Expresión en maxterms de D_2.
A) $D_2=\sum m_i$ con i=4,5,6,7,12,13,14,15
B) $D_2=\prod M_i$ con i=4,5,6,7,12,13,14,15
C) $D_2=\sum m_i$ con i=0,1,2,3,8,9,10,11
D) $D_2=\prod M_i$ con i=0,1,2,3,8,9,10,11

-3- Expresión mínima de la función $D_1 = f(I, Cr, S_0, S_1)$.
A) $D_1=\widehat{I}\,S_0\,\widehat{S_1} + I\,S_0\,S_1$
B) $D_1=\widehat{I}\,S_0\,\widehat{S_1} + I\,\widehat{S_0}\,S_1$
C) $D_1=I\,S_0\,S_1 + \widehat{I}\,\widehat{S_0}\,S_1$
D) $D_1=\widehat{I}\,S_0\,S_1 + I\,\widehat{S_0}\,\widehat{S_1}$

-4- Expresión mínima de la función $D_2 = f(I, Cr, S_0, S_1)$.
A) $D_2=Cr$
B) $D_2=\widehat{Cr}$
C) $D_2=\widehat{I}\,Cr + I\,\widehat{Cr}$
D) $D_2=I\,Cr + \widehat{I}\,\widehat{Cr}$

-5- Expresión mínima de la función $AG = f(I, Cr, S_0, S_1)$.
A) $AG=\widehat{I}\,S_1 + I\,\widehat{S_0}$
B) $AG=\widehat{I}\,S_1 + I\,S_0$
C) $AG=I\,S_1 + \widehat{I}\,S_0$
D) $AG=\widehat{I}\,S_1 + I\,S_0$

Se pretende incorporar una tercera luz D_3 que informe cuando el guardia abandona la sala en que se encuentra pero va llegando tarde (Cr=1). Confeccione su tabla de verdad y conteste a las siguientes preguntas:
-6- Expresión en minterms de D_3.
A) $D_3=\prod M_i$ con i=4,12
B) $D_3=\sum m_i$ con i=0,4,8,12
C) $D_3=\prod M_i$ con i=0,4,8,12
D) $D_3=\sum m_i$ con i=4,12

-7- Expresión mínima de la función $D_3 = f(I, Cr, S_0, S_1)$.
A) $D_3=\widehat{S_0}\,\widehat{S_1}$
B) $D_3=\widehat{Cr}\,S_0\,S_1$
C) $D_3=S_0\,S_1$
D) $D_3=Cr\,\widehat{S_0}\,\widehat{S_1}$

PROBLEMA N°6
Se tiene un sistema de computación basado en MC68000.
En un momento dado sus registros contienen lo siguiente:

D0: 0204 0608	A0: 0000 7F00
D1: F305 2BC9	A1:
D2: 4E4F 2000	A2: 0000 7F00
D3: 1000 30FF	A3:
D4: 8E55 2900	A4:
D5: 0000 0100	A5: 0000 7F00
D6: 1237 8915	A6:
D7: 1234 FEDC	A7:

Se pide analizar el efecto de las siguientes instrucciones examinándolas en el orden que se dan.

-1- Cómo afecta MOVE.W D3,D4 a los registros D3 y D4:
a) D3:1000 30FF ;D4:1000 30FF
b) D3:1000 30FF ;D4:8E55 30FF
c) D3:1000 30FF ;D4:1000 8E55
d) D3:1000 30FF ;D4:8E55 1000

-2- Cómo afecta MOVE.B (A0),D7 a los registros A0 y D7:
a) A0:0000 7F00 ;D7:1234 FE09
b) A0:0000 7F00 ;D7:0000 0009
c) A0:0000 7F01 ;D7:1234 3C09
d) A0:0000 7F02 ;D7:1234 0009

-3- Cómo afecta MOVE.W (A5)+,D2 a los registros A5 y D2:
a) A5:0000 7F00 ;D2:0000 09BA
b) A5:0000 7F01 ;D2:0000 09BA
c) A5:0000 7F02 ;D2:0000 09BA
d) A5:0000 7F02 ;D2:4E4F 09BA

-4- Cómo afecta MOVE.B -(A2),D1 a los registros A2 y D1:
a) A2:0000 7F02 ;D1:0000 003C
b) A2:0000 7EFF ;D1:0000 003C
c) A2:0000 7EFF ;D1:F305 2B3C
d) A2:0000 7F02 ;D1:F305 2B3C

-5- Cómo afecta MOVE.W $100(A0),D0 a los registros A0 y D0:
a) A0:0000 7F00 ;D0:0204 10BB
b) A0:0000 8000 ;D0:0204 10BB
c) A0:0000 7F00 ;D0:0000 10BB
d) A0:0000 7F00 ;D0:0000 00BB

Y en su zona de memoria hay los siguientes datos:

$DIR	$DATO
007EFF	3C
007F00	09
007F01	BA

008000	10
008001	BB
008002	2F
008003	90
008004	22
008005	04

-6- Cómo afecta MOVE.L 2(A0,D5.W),D4 a los registros A0, D5 y D4:
a) A0:0000 7F00 ;D5:0000 0100
D4:2F90 2204
b) A0:0000 7F02 ;D5:0000 0100
D4:2F90 2204
c) A0:0000 7F02 ;D5:0000 0102
D4:2F90 2204
d) A0:0000 7F04 ;D5:0000 0100
D4:2F90 2204

-7- Si cargamos D5 con el dato $12345678 y ejecutamos la secuencia de instrucciones:
MOVE.B #$3A,D5
MOVE.W #$9E00,D5
MOVE.L #10,D5
¿Cual es el contenido de D5?
a) D5:9E00 003A
b) D5:003A 9E00
c) D5:0000 000A
d) D5:0000 003A

-8- Cómo afecta MOVEQ #$8F,D3 al registro D3:
a) D3:0000 008F
b) D3:0000 FF8F
c) D3:008F FFFF
d) D3:FFFF FF8F

PROBLEMA N°7

Se tiene un sistema de computación basado en MC68000. En un momento dado sus registros y su memoria contienen lo siguiente:

D0: 1234 5678 A0: 0000 0060
D1: 5F02 C332 A1: 0000 0061
D2: 1012 A1B2 A2: 0000 0062
D3: 3F2E 5983
D4: 1A9B F082
D5: 5F02 C332
D6: 5F02 C302
D7: 3141 5926

ÁREA DE DATOS (memoria)

$DIR	$DATO
5E	EF
5F	9A
60	4D
61	12
62	DD
63	F0

Se pide analizar el efecto de las siguientes instrucciones **examinándolas en el orden que se dan**.
Esto quiere decir que se debe tener en cuenta el efecto sobre los registros de cada instrucción de las preguntas anteriores.

-1- Cómo afecta MOVE.W D0,D1 a los registros D0 y D1:
a) D0: 1234 5678 ;D1: 5F02 5678
b) D0: 1234 5679 ;D1: 5F02 5678
c) D0: 1234 5679 ;D1: 5F02 5679
d) D0: 1234 5678 ;D1: 5F02 C379

-2- Cómo afecta MOVE.B (A0),D2 a los registros A0 y D2:
a) A0: 0000 0060 ;D2: 1012 A14D
b) A0: 0000 0061 ;D2: 1012 A14D
c) A0: 0000 0060 ;D2: 1012 4DA1
d) A0: 0000 0061 ;D2: 1012 4DA1

-3- Cómo afecta MOVE.W (A1)+,D3 a los registros A1 y D3:
a) A1: 0000 0062 ;D3: 3F2E 12DD
b) A1: 0000 0063 ;D3: 3F2E 12DD
c) A1: 0000 0062 ;D3: 3F2E 9AA1
d) A1: 0000 0061 ;D3: 3F2E 9AA1

-4- Cómo afecta MOVE.B -(A2),D4 a los registros A2 y D4:
a) A2: 0000 0061 ;D4: 1A9B F012
b) A2: 0000 005E ;D4: F012 1A9B
c) A2: 0000 005F ;D4: F012 1AEF
d) A2: 0000 005F ;D4: 1AEF F012

-5-Cómo afecta ADD.B D0,D5 a los registros D0 y D5:
a) D0: 1234 5678 ;D5: 5F02 C3AA
b) D0: 1234 5679 ;D5: 5F02 C3AA
c) D0: 1234 5678 ;D5: 5F02 C37B
d) D0: 1234 5679 ;D5: 5F02 C37B

-6-Cómo afecta ANDI.B #$F0,D6 al registro D6:
a) D6: 5F02 C330
b) D6: 5F02 C300
c) D6: 5F00 C302
d) D6: 5F00 C332

-7-Cómo afecta LSR.L #4,D7 al registro D7:
a) D7: 0314 1592
b) D7: 1415 9260
c) D7: 0314 1590
d) D7: 1415 9000

-8-Cómo afecta BCHG #6,D7 al registro D7:
a) D7: 0314 15D2
b) D7: 0304 1592
c) D7: 0304 1590
d) D7: 0314 15D0

PROBLEMA N°8
Se tiene un sistema de computación basado en MC68000. En un momento dado sus registros y su memoria contienen lo siguiente:

D0: 0000 000F A0: 0000 0550
D1: 0F0F F0F0 A1: 0000 054F
D2: 1357 AB3B A2: 0000 0550
D3: 1357 B3BA
D4: 6007 6CAB
D5: 0000 0008
D6: 537A AAAA
D7: 0A7B C3A2

ÁREA DE DATOS (MEMORIA)

$DIR	$DATO
54C	09
54D	BA
54E	7F
54F	AB
550	3C
551	09
552	4E
553	4F

Se pide analizar el efecto de las siguientes instrucciones **examinándolas en el orden que se dan**.
Esto quiere decir que se debe tener en cuenta el efecto sobre los registros de cada instrucción de las preguntas anteriores.

-1- Cómo afecta MOVE.W D0,D1 a los registros D0 y D1:
a) D0: 0000 000F ;D1: 0F0F 0010
b) D0: 0000 0010 ;D1: 0F0F 0010
c) D0: 0000 000F ;D1: 0F0F 000F
d) D0: 0000 0010 ;D1: 0F0F 000F

-2- Cómo afecta MOVE.B (A0),D2 a los registros A0 y D2:
a) A0: 0000 0550 ;D2: 1357 AB3C
b) A0: 0000 0550 ;D2: 1357 3C09
c) A0: 0000 0551 ;D2: 1357 3BAB
d) A0: 0000 0551 ;D2: 1357 AB3C

-3- Cómo afecta MOVE.W (A1)+,D3 a los registros A1 y D3:
a) A1: 0000 0551 ;D3: 1357 3C09
b) A1: 0000 0551 ;D3: 1357 BA7F
c) A1: 0000 0551 ;D3: 1357 AB3C
d) A1: 0000 0551 ;D3: 1357 094E

-4- Cómo afecta MOVE.L -(A2),D4 a los registros A2 y D4:
a) A2: 0000 0550 ;D4: 09BA 09BA
b) A2: 0000 0552 ;D4: 09BA 09BA
c) A2: 0000 054E ;D4: 09BA 7FAB
d) A2: 0000 054C ;D4: 09BA 7FAB

-5- Cómo afecta MULU D0,D5 a los registros D0 y D5:
a) D0: 0000 0010 ;D5: 0000 0070
b) D0: 0000 000F ;D5: 0000 0070
c) D0: 0000 000F ;D5: 0000 0078
d) D0: 0000 0010 ;D5: 0000 0078

-6- Cómo afecta EOR.W D2,D6 al registro D6:
a) D6: 537A B323
b) D6: 537A 0196
c) D6: 537A 8888
d) D6: 537A AAAA

-7- Cómo afecta ROR.W D0,D7 al registro D7:
a) D7: 0A7B 2C3B
b) D7: 0A7B C3A2
c) D7: 0A7B 8745
d) D7: 0A7B A2C3

-8- Suponga que existe la rutina RUTI. Se ejecutan las siguientes instrucciones
CMP.W D2,D3
BNE RUTI
Señale el enunciado cierto:
a) Z=0 y no se produce el salto a RUTI.
b) Z=1 y no se produce el salto a RUTI.
c) Z=0 y se produce el salto a RUTI.
d) Z=1 y se produce el salto a RUTI.

PROBLEMA N°9

Un sistema de vigilancia por computador está instalado en un edificio de oficinas. Tiene ocho despachos, cada uno con una puerta, una ventana y un dispositivo detector de incendios. Cada puerta y cada ventana tienen un interruptor que da un valor lógico '0' cuando está abierta, y un valor lógico '1' cuando está cerrada. Asimismo, los detectores de incendios dan un '0' lógico cuando no hay incendio, y un '1' lógico si lo hay. Las puertas se agrupan en un registro de 8 bits, de manera que los despachos (numerados de 0 a 7) tienen una indicación del estado de su puerta en el bit correspondiente del registro. Análogamente se hace con las ventanas y los detectores. Todo esto se detalla en el siguiente mapa de registros:

$dir	contenido	comentario
007000	P7 P6 P5 P4 P3 P2 P1 P0	puertas
007001	V7 V6 V5 V4 V3 V2 V1 V0	ventanas
007002	D7 D6 D5 D4 D3 D2 D1 D0	detectores

cada registro ocupa una posición de memoria.

el sistema funciona como sigue:

- las puertas 0, 1, 2 y 3 deben estar cerradas siempre.
- las puertas 4, 5, 6 y 7 pueden estar abiertas o cerradas indistintamente.
- todas las ventanas deben estar cerradas.
- todos los detectores deben tener la condición "sin incendio".
- el sistema vigila de la siguiente manera:

    -1. registra (anota) el estado de las puertas 4, 5, 6 y 7 en cada piso.
    -2. comprueba que las puertas 0, 1, 2 y 3 y las ventanas permanecen cerradas en todos los pisos. De no ser así, alarma grave.
    -3. comprueba que las puertas 4, 5, 6 y 7 siguen en el estado inicial en todos los pisos. De no ser así, alarma leve.
    -4. comprueba que todos los detectores de incendio tengan la condición 0 lógico. De no ser así, alarma incendio.
    -5. Vuelve al punto 2.

Para implantar este sistema se necesita contestar a las siguientes preguntas:

-1- Cuando el computador consulte el registro $7000, en condiciones normales debería encontrar:
a) 0000 0000
b) 1111 1111
c) XXXX 1111
d) 1111 XXXX

-2- Cuando el computador consulte el registro $7001, en condiciones normales debería encontrar:
a) 0000 0000
b) 1111 1111
c) XXXX 1111
d) 1111 XXXX

-3- Cuando el computador consulte el registro $7002, en condiciones normales debería encontrar:
a) 0000 0000
b) 1111 1111
c) XXXX 1111
d) 1111 XXXX

NOTA: Con el símbolo X denotamos que puede contener cualquier valor (1 ó 0 indistintamente).

Lea atentamente el programa propuesto y suponga que existen las rutinas ALARMAG, ALARMAL y ALARMAI, que gestionan las alarmas grave, leve y de incendio respectivamente.

```
1           ORG $8000
2 regist    EQU $7000
3 estado    DS.B 1
4           ORG $8100
5           MOVEA.L #regist, A0;
6           MOVEA.L #estado, A1;
7           MOVE.B (A0),D1;
8           LSR.B #4, D1;
9           MOVE.B D1, (A1);
10 vigilar  MOVEA.L #regist, A0;
11          MOVE.B (A0)+, D1;
12          MOVE.B (A0)+, D2;
13          MOVE.B (A0), D3;
14          MOVE.B D1, D4;
15          MOVE.B #$0F, D5;
16          AND.B D5, D1;
17          EOR.B D5, D1;
18          BNE ALARMAG;
19          MOVE.B #$ FF, D5;
20          EOR.B D5, D2;
21          BNE ALARMAG;
22          MOVE.B (A1), D5;
23          LSR.B #4, D4;
24          EOR.B D5, D4;
25          BNE ALARMAL;
26          MOVE.B #$ 00, D5;
27          EOR.B D5, D3;
28          BNE ALARMAI;
29          BRA vigilar;
```

-4- Antes de empezar la rutina 'vigilancia' (ver las lineas 7,8 y 9) la posición de memoria etiquetada con el nombre 'estado', contendrá:

a) 0000 P3 P2 P1 P0

b) 0000 P7 P6 P5 P4

c) P7 P6 P5 P4 0000

d) P4 P5 P6 P7 0000

-5- ¿Qué hace el bloque de instrucciones de las lineas 10-14 (ambas inclusive)?

a) puertas->D1, ventanas->D2, detectores->D3

b) puertas-> D1, D2 y D3.

c) puertas->D1 y D4, ventanas->D2, detectores->D3

d) Borrar los registros $7000-$7002

-6- Cual es la misión de las instrucciones
MOVE.B #$0F, D5;
AND.B D5, D1;
EOR.B D5, D1;

a) eliminar los bits P7 P6 P5 P4 y comprobar que los bits P3 P2 P1 P0 estan a '1' lógico.

b) eliminar los bits P7 P6 P5 P4 y comprobar que los bits P3 P2 P1 P0 estan a '0' lógico.

c) eliminar los bits P3 P2 P1 P0 y comprobar que los bits P7 P6 P5 P4 estan a '1' lógico.

d) eliminar los bits P3 P2 P1 P0 y comprobar que los bits P7 P6 P5 P4 estan a '0' lógico.

-7- Otro modo de escribir las instrucciones
MOVE.B #$ FF, D5;
EOR.B D5, D2;
para que ejecuten una tarea equivalente es:

a) EOR.B D5, D2;
   MOVE.B #$ FF, D5;

b) EORI.B #FF, D2;

c) EORI.B #FF, D5;

d) ORI.B #FF, D2;

-8- La instrucción BNE ALARMAG trasfiere la ejecución del programa a la posición ALARMAG sólo si:
a) el bit Z del CCR es 0.
b) el bit Z del CCR es 1.
c) el bit V del CCR es 0.
d) el bit V del CCR es 1.

PROBLEMA N°10

Se tiene un sistema de computación basado en M68000, cuyos registros contienen:

D0: 0123 9876
D1: 1596 7536
D2: 250C 30F7
D3: 3F2E 5983
D4: 0000 0009

Se pide responder a las siguientes preguntas:

-1- Cómo afecta ROR.W D4, D3 al registro D3.

A) D3: 3F2E 06B3 B) D3: C1A3 F2E5 C) D3: 3F2E C1AC D) D3: 06B3 F2E5

-2- Cómo afecta ADDI.B #$10, D2 al registro D2.

A) D2: 250C 3107 B) D2: 251C 30F7 C) D2: 250C 3007 D) D2: 251C 310F

-3- Genere el código de máquina producido por la instrucción MOVE.B D3, (A5)+.

A) 1AC3 B) 161D C) 1743 D) 10DD

-4- Genere el código de máquina producido por la instrucción CLR.W D4.

A) 4424 B) 4244 C) 4260 D) 427C

Lea atentamente este fragmento de programa:

Consejos: 1°) Cree un mapa de memoria de las posiciones desde $7000 hasta $7017.
2°) Observe que A1 Y A2 son punteros.
3°) El bucle que empieza en INICIO sirve para detectar el carácter de control $00.

```
            ORG $7000
VECTOR1     DC.W 1000, $1000, 100, $100
VECTOR2     DS.L  4
            ORG $7100
            MOVEA.L #VECTOR1, A1
            MOVEA.L #VECTOR2, A2
INICIO      CLR.L D0
            CLR.L D1
            MOVE.B (A1)+, D0
            MOVE.L D0, D1
            SWAP D1
            MOVE.L D1, (A2)+
            CMPI.B #0, D0
            BNE INICIO
```

Ejecútelo completamente y conteste a las siguientes preguntas:

-5- ¿ Qué contiene la posición de memoria cuya dirección es $7001 ?

A) 01 B) 10 C) 00 D) E8

-6- ¿ Qué contiene el registro A1 ?

A) A1: 0000 7004 B) A1: 0000 7002 C) A1: 0000 7003 D) A1: 0000 7001

-7- ¿ Qué contiene la posición de memoria cuya dirección es $700D ?

A) 00 B) 10 C) E8 D) 01

-8- ¿ Qué contiene el registro A2 ?

A) A2: 0000 7004 B) A2: 0000 700D C) A2: 0000 7018 D) A2: 0000 700A

Soluciones a los problemas

problema n° 1	problema n° 2	problema n° 3	problema n° 4	problema n° 5
1-a	1-d	1-a	1-b	1-c
2-c	2-d	2-a	2-d	2-b
3-a	3-a	3-d	3-a	3-b
4-d	4-b	4-c	4-b	4-a
5-b	5-a	5-b	5-c	5-b
6-d	6-d	6-c	6-c	6-d
7-a	7-b	7-c	7-d	7-d

problema n° 6	problema n° 7	problema n° 8	problema n° 9	problema n° 10
1-b	1-a	1-c	1-c	1-c
2-a	2-a	2-a	2-b	2-c
3-d	3-b	3-c	3-a	3-a
4-c	4-a	4-d	4-b	4-b
5-a	5-a	5-c	5-c	5-d
6-a	6-b	6-b	6-a	6-a
7-c	7-a	7-c	7-b	7-c
8-d	8-a	8-b	8-a	8-c

FUNCIONES LÓGICAS.

CUESTIONES:

NOTA: La notación A' significa NOT(A), y (A+B)' significa NOT(A+B); además, el signo * representa la función AND.

Cuestión nº 1. Aplicando la Ley de De Morgan a la siguiente función [(A+B)'*C+A'+B'*D+C'*D']' se obtiene como resultado:

A)- [(A+B)+C']*A*(B+D')*(C+D);
B)- [(A+B)*C']*A*(B+D')+(C+D);
C)- [(A+B)+C']*(B+D')*(C+D);
D)- [(A+B)+C']*A*(B'+D')*(C+D);

Cuestión nº 2. La expresión de la función OR-exclusiva (XOR) se expresa como A*B'+A'B. Aplicando la Ley de De Morgan y las leyes que sean necesarias, la expresión simplificada de la función NOR-exclusiva (XNOR u OR-exclusiva negada) es:

A) A*B'+A'*B;
B) (A*B)'+A*B;
C) A'*B'+A*B;
D) (A+B')*(A'+B).

Cuestión nº 3. La función complementaria de f(A,B,C,D,E) = [A*B+C']*[(D+E)'+B*A'] es:

[f (A,B,C,D,E)]' = (A+B*C')*[(D+E)+B*A'];
[f (A,B,C,D,E)]' =(A'+B')*C+[(D+E)*(B'+A)];
[f (A,B,C,D,E)]' =(A*C)+[(D+E')*(B'+A)];
[f (A,B,C,D,E)]' =(A+B*C')+[(D+E)+B*A'].

Cuestión nº 4. Mediante la aplicación de las leyes fundamentales del Álgebra de Boole, la función (A+B)*(A+C)*(B+C) simplificada es:

A) (A+C)+(A*C)+(B+C);
B) (B+C)*(A+B*C);
C) (A+B*C)*B;
D) (A+B)*(B+C).

Cuestión nº 5. La función f(A,B,C,D,E) = [(A'*B')+D]*(B'+E)+C*D' para A = C = E = 0 y B=D=1 es:

A) 0;
B) 1;
C) 2;
D) Ninguna de las anteriores.

Cuestión nº 6. La función f(A,B,C) = A*B+B' es equivalente a:

A) f(A,B,C) = A;
B) f(A,B,C) = A+B';
C) f(A,B,C) = A*(C+C')*(B+B');
D) f(A,B,C) = A*B*C+A*B*C'+B'*A*C+B'*A'*C+B'*A*C'+B*A*C'.

Cuestión nº 7. La función f(A,B,C) = A'+C' es equivalente a:

A) f(A,B,C) = (A'+B'+C')*(A'+B+C);
B) f(A,B,C) = A'*B'*C'+A'*B'*C+A*B*C'+A'*B*C'+A*B'*C'+A*B*C+A'+C';
C) f(A,B,C) = (A'+C')*(A'+B+C)*(A'+B'+C');
D) f(A,B,C) = A'+A*B'*C'+A*B*C'.

RESPUESTAS

1) -A.
2) -C.
3) -B.
4) -B.
5) -A.
6) -B.
7) -D.

SIMPLIFICACIÓN DE FUNCIONES LÓGICAS

NOTA: El signo * representa la función AND, sin embargo y para simplificar, se utilizará la notación AB como equivalente a A*B.

Cuestión nº 1. La función f(A,B) = [A+(A*B')]' equivale a:

A) f(A,B) = m_0;
B) f(A,B) = m_0+m_1;
C) f(A,B) = m_1+m_2;
D) f(A,B) = m_3.

Cuestión nº 2. La función f(A,B) = [A+(A*B')]' equivale a:

A) f(A,B) = M_0*M_3;
B) f(A,B) = M_2*M_3;
C) f(A,B) = M_1*M_2;
D) f(A,B) = M_0*M_1;

Cuestión nº 3. La función f(A,B) = [A'*(B+A'*B')]' equivale a:

A) f(A,B) = $\Pi(0,2)$;
B) f(A,B) = $\Pi(2,3)$;
C) f(A,B) = $\Pi(1,3)$;
D) f(A,B) = $\Pi(0,1)$;

Cuestión nº 4. La función f(A,B) = [A'*(B+A'*B')]' equivale a:

A) f(A,B) = $\Sigma(0,1)$;
B) f(A,B) = $\Sigma(2,3)$;
C) f(A,B) = $\Sigma(0,3)$;
D) f(A,B) = $\Sigma(1,2)$;

Cuestión nº 5. La función f(A,B) = [A'+(A*B')]' equivale a:

A) f(A,B) = $\Sigma(0,1,2)$;
B) f(A,B) = $\Sigma(1,2)$;
C) f(A,B) = $\Sigma(3)$;
D) f(A,B) = $\Sigma(0)$;

Cuestión nº 6. La función f(A,B) = [A'+(A*B')]' equivale a:

A) f(A,B) = $\Pi(1,2,3)$;
B) f(A,B) = $\Pi(3)$;
C) f(A,B) = $\Pi(0,1,2)$;
D) f(A,B) = $\Pi(0)$;

Cuestión nº 7. Sean dos funciones lógicas f_1 y f_2 tales que: $f(A,B) = \Sigma(0,3)$ y $f_2(C,D) = \Pi(1,2)$. Represente en segunda forma canónica la función lógica $g(A,B,C,D) = f_1(A,B) \oplus f_2(C,D)$.

A) $g(A,B,C,D) = \Pi(0,3,5,6,9,10,12,15)$;
B) $g(A,B,C,D) = \Pi(0,1,5,6,9,10,12,13)$;
C) $g(A,B,C,D) = \Pi(1,2,4,7,8,11,13,14)$;
D) $g(A,B,C,D) = \Pi(0,1,2,3)$.

Cuestión nº 8. Se diseña un circuito combinacional que permite realizar el producto de dos números A y B de dos bits, $A = (a_1,a_0)$ y $B = (b_1,b_0)$. El número resultante, $P = (p_3,p_2,p_1,p_0)$, es de cuatro bits. Represente en 1ª forma canónica la función $p_3 = f(a_1,a_0,b_1,b_0)$.

A) $p_3 = f(a_1,a_0,b_1,b_0) = \Sigma(7,11,15)$;
B) $p_3 = f(a_1,a_0,b_1,b_0) = \Sigma(11,14)$;
C) $p_3 = f(a_1,a_0,b_1,b_0) = \Sigma(13,14,15)$;
D) $p_3 = f(a_1,a_0,b_1,b_0) = \Sigma(15)$.

Cuestión nº 9. Se diseña un circuito combinacional que permite realizar el producto de dos números A y B de dos bits, $A = (a_1,a_0)$ y $B = (b_1,b_0)$. El número resultante, $P = (p_3,p_2,p_1,p_0)$, es de cuatro bits. Represente en 1ª forma canónica la función $p_1 = f(a_1,a_0,b_1,b_0)$.

A) $p_1 = f(a_1,a_0,b_1,b_0) = \Sigma(6,7,9,11,13,14)$;
B) $p_1 = f(a_1,a_0,b_1,b_0) = \Sigma(10,11,14)$;
C) $p_1 = f(a_1,a_0,b_1,b_0) = \Sigma(0,3,5,7,8,9,10,11,12,13,14,15)$;
D) $p_1 = f(a_1,a_0,b_1,b_0) = \Sigma(3,5,7,10,11)$.

Cuestión nº 10. Se diseña un circuito combinacional que permite realizar el producto de dos números A y B de dos bits, $A = (a_1,a_0)$ y $B = (b_1,b_0)$. El número resultante, $P = (p_3,p_2,p_1,p_0)$, es de cuatro bits. Represente en 1ª forma canónica la función $p_0 = f(a_1,a_0,b_1,b_0)$.

A) $p_0 = f(a_1,a_0,b_1,b_0) = \Sigma(5,7,11,15)$;
B) $p_0 = f(a_1,a_0,b_1,b_0) = \Sigma(5,6,11,14)$;
C) $p_0 = f(a_1,a_0,b_1,b_0) = \Sigma(5,7,13,15)$;
D) $p_0 = f(a_1,a_0,b_1,b_0) = \Sigma(6,7,11,15)$.

Cuestión nº 11. Se diseña un circuito combinacional que permite realizar el producto de dos números A y B de dos bits, $A = (a_1,a_0)$ y $B = (b_1,b_0)$. El número resultante, $P = (p_3,p_2,p_1,p_0)$, es de cuatro bits. Represente en 1ª forma canónica la función $p_2 = f(a_1,a_0,b_1,b_0)$.

A) $p_2 = f(a_1,a_0,b_1,b_0) = \Sigma(10,11,15)$;
B) $p_2 = f(a_1,a_0,b_1,b_0) = \Sigma(11,13,14)$;
C) $p_2 = f(a_1,a_0,b_1,b_0) = \Sigma(11,13,15)$;
D) $p_2 = f(a_1,a_0,b_1,b_0) = \Sigma(10,11,14)$.

RESPUESTAS

1) B.
2) D.
3) B.
4) B.
5) C.
6) A.
7) C.
8) D.
9) A.
10) C.
11) D.

REPRESENTACIÓN DE LA INFORMACIÓN (I)

Cuestión nº 1. El sistema de representación binario denominado módulo-signo consiste en:

A) Utilizar un bit para el signo y el resto para el valor absoluto del número representado;
B) Representar sólo números negativos siendo imposible representar los positivos;
C) Representar números sólo por su valor modular sin tener en cuenta el signo;
D) Ninguna de las anteriores.

Cuestión nº 2. El conjunto de los números representables por un sistema de representación se conoce como:

A) Sistema o conjunto posicional;
B) Densidad de una representación;
C) Rango de una representación;
D) Conjunto denso.

Cuestión nº 3. Si todas las instrucciones de un microprocesador se codifican por campos, de tal forma que se destinan 5 bits para el código de operación y 11 para los operandos, ¿cuál de las siguientes afirmaciones es falsa?

A) Ese microprocesador puede tener 40 códigos de operación diferentes.
B) El dato para la instrucción puede ser un número de 8 bits.
C) Si el operando es un número representado en binario puro, entonces el decimal 2047 puede ser indicado como operando dentro de una de esas instrucciones.
D) Para esa instrucción puede estar en la memoria principal en un rango de 128 direcciones consecutivas.
E) Ese microprocesador puede disponer de un conjunto de 16 instrucciones.

Cuestión nº 4. Indicar cuál de las siguientes respuestas es falsa:

A) Al multiplicar dos números binarios enteros de 8 bits en complemento a 2 se puede obtener un número de 16 bits.
B) Para multiplicar un número positivo por números que son potencia de 2, se puede utilizar un desplazamiento a la izquierda de los bits del multiplicando en un número adecuado de posiciones.
C) El producto de un número binario positivo por uno negativo debe ser siempre distinto de 0.
D) El producto de un número binario positivo A por otro positivo B puede obtenerse repitiendo B veces la suma de A con el contenido de una variable que almacena el resultado de la operación, y que inicialmente es 0.

Cuestión nº 5. Tenemos un número X que es entero, siendo A el binario de su valor absoluto, B el complemento lógico de A y C el complemento lógico del contenido de A e incrementado en una unidad. Entonces, el complemento a 2 de X se obtiene:

A) Sumando 1 a A, cualquiera que sea su valor;
B) Sumando 1 a A si X es negativo;
C) Es B si X es positivo;
D) Es A si X es positivo.

Cuestión nº 6. Sean los números A=10011 y B=11101, representados en cierto sistema binario de representación. Si están representados utilizando palabras de 5 bits y si sabemos que la suma de A y B da como resultado S=10000, ¿qué sistema de representación hemos utilizado?

A) Representación signo-magnitud.
B) Representación en complemento a 1.
C) Representación en complemento a 2.
D) Representación en exceso a 16.

Cuestión nº 7. La suma de los números A= 11001 y B= 11101, representada en palabras de 5 bits y en complemento a uno, da lugar al siguiente resultado:

A) A+B= 10110.
B) A+B= 10111.
C) A+B= 11110.
D) A+B= 11101.

Cuestión nº 8. El número decimal N= 1 (positivo), se representa en complemento a 2 utilizando palabras de 4 bits como:

A) 0001.
B) 1110.
C) 1111.
D) 1001.

Cuestión nº 9. Se desea utilizar una representación que sigue el formato IEEE754 para palabras de 32 bits, pero limitando la representación de la mantisa a sólo 8 bits (signo incluido), o sea, utilizando 1 bit de signo, 8 bits de exponente representado en exceso a 127, y 7 bits de mantisa, la mantisa se representa en signo-magnitud. En esta representación se tiene el siguiente número: N=1000000011100000. ¿A qué número equivale en el sistema decimal?

A) 3'50.
B) $3'50*2^{-126}$.
C) $-1'75*2^{-126}$.
D) –3'50.

Cuestión nº 10. Representar en una palabra de 8 bits, B=(b7,b6,...,b1,b0) el número N=0'4, utilizando una representación en signo-magnitud, siendo b7 el bit de signo y representando los restantes la parte fraccionaria del número.

A) N=10110011.
B) N=01001100.
C) N=00110011.
D) N=00011001.

Cuestión nº 11. ¿Cuál es el error absoluto cometido en la representación en signo-magnitud del número 0'4, sobre 8 bits, en la cuestión anterior?

A) $0'0078125_{10}$.
B) $0'2046875_{10}$.
C) $0'00390625_{10}$.
D) $0'0015625_{10}$.

Cuestión nº 12. Se desea normalizar el número fraccionario N=1000011000111010, representado en signo-magnitud sobre una palabra de 16 bits. El byte más significativo contiene la parte entera con el signo, y el byte menos significativo contiene la parte fraccionaria. Al normalizarlo según la norma IEEE754, para 16 bits, el valor resultante del exponente (en exceso a 127) es:

A) 129.
B) 130.
C) 134.
D) 135.

Cuestión nº 13. El número binario 1000011000111010 de la cuestión anterior y que estaba representado en coma fija, se representaría en coma flotante según el IEEE754 para 16 bits por:

A) C047h.
B) C023h.
C) 4063h.
D) C0C7h.

La letra final –h indica que los números están representados en hexadecimal.

RESPUESTAS

1) A.
2) C.
3) A.
4) C.
5) D.
6) C.
7) B.
8) A.
9) C.
10) C.
11) D.
12) A.
13) D.

REPRESENTACIÓN DE LA INFORMACIÓN (II)

Cuestión nº 1. Cuál de los siguientes tipos de caracteres NO se encuentra en el código alfanumérico ASCII:

A) cifras del sistema decimal.
B) Letras mayúsculas y minúsculas.
C) Letras griegas.
D) Signos de puntuación.

Cuestión nº 2. Señale el enunciado que es cierto:

A) Para que exista la posibilidad de detectar un error el código debe ser redundante
B) La detección de error se realiza cuando la distancia del código es nula
C) Sólo los códigos redundantes pueden ser densos
D) La detección del error se puede realizar cuando la distancia del código es uno

Cuestión nº 3. Se dice que a un código denso se le ha implantado un control de paridad par transversal cuando:

A) A cada palabra se le añade un bit que hace que el número de unos en dicha palabra sea par.
B) Cada cierto número de palabras se añade un bit que hace que el número de unos del grupo sea par.
C) A cada palabra se le añade un bit que hace que el número de unos sea lo menor posible.
D) Cada bloque de ocho palabras tiene una palabra que hace que las columnas del bloque tengan un número impar de unos.

Cuestión nº 4. Entre los siguientes enunciados referidos a la compactación de la información hay uno falso. Señálelo:

A) La codificación diferencial es un método de compactar información.
B) La codificación por frecuencia de uso sirve para descompactar la información.
C) La compactación de información consigue almacenar más información en menos espacio.
D) La compactación suele complicar la codificación.

Cuestión nº 5. ¿Cuál de las siguientes afirmaciones es cierta?

A) Si se añade un bit de paridad a un código denso, la distancia del código obtenido es 1.

B) Un código corrector permite detectar un error.

C) Los códigos de paridad sólo utilizan un bit añadido, que es 1 cuando el número de unos en la palabra es 0 ó par, y que es 0 en caso contrario.

D) Los códigos de Hamming consisten en varios bits de paridad selectiva, que situados sólo en el fiñal de una cadena de bits permiten corregir al menos un error producido en un bit de dicha cadena.

Cuestión nº 6. La pregunta siguiente se compone de 4 enunciados de los cuales uno es falso. Señálelo.

A) El juego de caracteres de un código alfanumérico suele tener: las letras del alfabeto, las 10 cifras del sistema decimal, los signos de puntuación y algunos caracteres de control.

B) Puede considerarse que una instrucción de programa es una orden para que el computador realice una determinada operación de procesamiento de unos datos determinados.

C) La codificación según frecuencia de uso reduce el tamaño de la información a base de explotar las diferencias en el uso de las distintas unidades de información.

D) Las instrucciones nunca se deben codificar por campos pues no se podrían almacenar en la misma memoria que los datos, según Von Neumann.

RESPUESTAS

1) C.
2) A.
3) A.
4) B.
5) B.
6) D.

Estructura y Tecnología de Computadores I

Colección de problemas

Ejercicios y soluciones

Se puede encontrar el visor de archivos PDF en la sigui

http://www.adobe.com/products/acrobat/readstep.ht

FE DE ERRATAS

En la Colección de problemas de examen, los problemas 2 y 3 tienen su solución ma

La solución correcta es:

Problema 2: 1-d / 2-d / 3-a / 4-b / 5-a / 6-d / 7-b

Problema 3: 1-a / 2-a / 3-d / 4-c / 5-b / 6-c / 7-c

En los Ejercicios y soluciones, en la parte de simplificación de funciones lógicas, la cuestión 7 tiene como respuesta correcta la A.

Principal | Contenidos | Evaluación | Consultas | Material

UNIVERSIDAD NACIONAL
de EDUCACIÓN a DISTANCIA

Departamento de
Ingeniería Eléctrica, Electrónica y de Control

ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I (Sistemas)

Colección de PROBLEMAS RESUELTOS

Preparados por: **Carlos Cerrada**
Luis Rincón

Octubre 2001

13

UNIVERSIDAD NACIONAL
de EDUCACIÓN a DISTANCIA

Departamento de
Ingeniería Eléctrica, Electrónica y de Control

ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I (Sistemas)

Colección de PROBLEMAS RESUELTOS

Preparados por: **Carlos Cerrada**
Luis Rincón

Octubre 2001

La presente colección de problemas ha sido preparada para que sirva como material didáctico complementario para el estudio de la *asignatura Estructura y Tecnología de Computadores I (Sistemas)* de la Escuela de Informática de la UNED. Esta colección viene a suplir la carencia de ejercicios resueltos que tiene el texto base de esta asignatura, y ha sido elaborada por el Equipo Docente de la misma en la Escuela de Informática de la UNED.

La colección está organizada en tres Unidades Didácticas, siguiendo el mismo esquema que el texto base *"Fundamentos de Estructura y Tecnología de Computadores", Editorial Centro de Estudios Ramón Areces (2001).* Por cada Unidad Didáctica se presenta un conjunto de enunciados, cada uno seguido de su solución propuesta. Por tanto, la colección contiene diversos problemas de prácticamente de toda la materia que comprende la asignatura.

Conviene destacar que la colección presentada no sólo aporta un conjunto de soluciones a problemas concretos sino que pone de manifiesto una metodología de trabajo que permitirá al alumno abordar la solución de problemas dificultad similar.

Por último mencionar que, aunque es ésta una versión revisada de otras colecciones anteriores, no puede considerarse todavía una versión definitiva. En este sentido es probable que contenga alguna que otra errata todavía no detectada y que esperamos corregir en futuras versiones con la colaboración de todos.

El Equipo Docente

1.1. Simplificar las siguientes expresiones utilizando los teoremas del álgebra de Boole

a) $(A.\overline{A} + \overline{A}.B + B.B + A.B + \overline{B}) . (A + B)$

b) $A.B + C.\overline{D} + B.\overline{C}.D + \overline{A}.B$

c) $(A + \overline{B}) . (\overline{B} + C) . (\overline{C} + D)$

En la resolución de los tres apartados de este ejercicio se emplearán los teoremas fundamentales del álgebra de Boole, enunciados en la pregunta 2.3.1 del texto base (págs. 57-59). En el paso de una expresión a otra se notificará qué relaciones o propiedades se han utilizado, si bien no se avisará en ningún caso el empleo de las propiedades asociativa y conmutativa.

a) $E_{1a} = (A.\overline{A} + \overline{A}.B + B.B + A.B + \overline{B}) . (A + B)$

Aplicando

- $B.B = B$ (ley de idempotencia)
- $A.\overline{A} = 0$

la expresión del enunciado se convierte en

$$E_{1a} = (0 + \overline{A}.B + B + A.B + \overline{B}) . (A + B)$$

y como $B + \overline{B} = 1$

$$E_{1a} = (\overline{A}.B + A.B + 1) . (A + B).$$

La subexpresión encerrada entre paréntesis en primer lugar vale 1, quedando la expresión definitiva como

$$E_{1a} = A + B$$

b) $E_{1b} = A.B + C.\overline{D} + B.\overline{C}.D + \overline{A}.B$

Al aplicar la propiedad distributiva sobre los términos $A.B + \overline{A}.B$ queda

$$E_{1b} = (A + \overline{A}).B + C.\overline{D} + B.\overline{C}.D = B + C.\overline{D} + B.\overline{C}.D.$$

Por la ley de absorción, $B + B.\overline{C}.D = B$, y la expresión se reduce finalmente a

$$E_{1b} = B + C.\overline{D}$$

que ya no puede simplificarse más.

c) $E_{1c} = (A + \overline{B}) . (\overline{B} + C) . (\overline{C} + D)$

Aplicando la ley distributiva respecto del producto se tiene que

$$E_{1c} = A . (\overline{B} + C) . (\overline{C} + D) + \overline{B} . (\overline{B} + C) . (\overline{C} + D) =$$
$$= A . \overline{B} . (\overline{C} + D) + A . C . (\overline{C} + D) + \overline{B} . \overline{B} . (\overline{C} + D) + \overline{B} . C . (\overline{C} + D).$$

Puesto que $\overline{B}.\overline{B} = \overline{B}$, el tercer término de la expresión anterior queda reducido a $\overline{B} . (\overline{C} + D)$, y aplicando la ley de absorción resulta

$$A . \overline{B} . (\overline{C} + D) + \overline{B} . (\overline{C} + D) + \overline{B} . C . (\overline{C} + D) = \overline{B} . (\overline{C} + D)$$

con lo que nuestra expresión valdrá

$$E_{1c} = A . C . (\overline{C} + D) + \overline{B} . (\overline{C} + D) = A . C . \overline{C} + A . C . D + \overline{B} . (\overline{C} + D) =$$
$$= A . C . D + \overline{B} . (\overline{C} + D)$$

1.2. Simplificar las siguientes expresiones utilizando las leyes de De Morgan:

a) $\overline{(A.\overline{A} + \overline{A}.B + \overline{A.B + \overline{B}}).(\overline{A + B})}$

b) $\overline{A.B + C.\overline{D} + B.\overline{C}.D + \overline{A}.B}$

c) $\overline{(A + \overline{B}).\overline{(\overline{B} + C)}.(\overline{C} + D)}$

En los ejercicios de este apartado se emplearán, además de las leyes de De Morgan, las propiedades y relaciones utilizadas en el apartado anterior. Las leyes de De Morgan se enuncian en la página 59 del texto base, presentándose en su versión generalizada en la página 63.

a) $E_{2a} = \overline{(A.\overline{A} + \overline{A}.B + \overline{A.B + \overline{B}}).(\overline{A + B})}$

Por la primera ley de De Morgan, y eliminando el término $A.\overline{A} = 0$, la expresión pasa a ser

$$E_{2a} = \overline{(\overline{A}.B + \overline{A.B + \overline{B}})} + \overline{(\overline{A + B})}.$$

El segundo término entre paréntesis se halla doblemente invertido, con lo cual es igual a $A + B$. En cuanto a la subexpresión encerrada entre paréntesis en primer lugar, aplicándole las leyes de De Morgan, idempotencia, absorción y las relaciones fundamentales queda así

$$\overline{\overline{A}.B + \overline{A.B + \overline{B}}} = \overline{\overline{A}.B}.\overline{A.B + \overline{B}} = (A + \overline{B}).(A.B + \overline{B}) =$$

$$= A.A.B + A.\overline{B} + A.B.\overline{B} + \overline{B}.\overline{B} = A.B + A.\overline{B} + \overline{B} = A.B + \overline{B}$$

y entonces $E_{2a} = A.B + A + \overline{B} + B = 1$

b) $E_{2b} = \overline{A.B + C.\overline{D} + B.\overline{C}.D + \overline{A}.B}$

Fijándonos en los términos primero y último, y aplicando a los mismos la propiedad distributiva, observamos que

$$(\overline{A} + \overline{B}).(A + \overline{B}) = (A + \overline{A}).\overline{B} = \overline{B}$$

con lo que

$$E_{2b} = \overline{B}.(\overline{C} + D).(\overline{B} + C + \overline{D}).$$

Por la ley de absorción, $\overline{B}.(\overline{B} + C + \overline{D}) = \overline{B}$ y

$$E_{2b} = \overline{B}.(\overline{C} + D)$$

c) $E_{2c} = \overline{(A + \overline{B}).\overline{(\overline{B} + C)}.(\overline{C} + D)}$

Mediante el uso de la ley de De Morgan generalizada, se deduce que

$$E_{2c} = \overline{(A + \overline{B})} + \overline{\overline{(\overline{B} + C)}} + \overline{(\overline{C} + D)} = \overline{A}.B + \overline{B} + C + C.\overline{D}.$$

Aplicando la ley de absorción, $C + C.\overline{D} = C$ y

$$E_{2c} = \overline{A}.B + \overline{B} + C$$

Por la propiedad distributiva y el elemento neutro

$$\overline{A}.B + \overline{B} = (\overline{A} + \overline{B}).(B + \overline{B}) = (\overline{A} + \overline{B}).1 = \overline{A} + \overline{B}$$

Con lo que la expresión queda definitivamente como

$$E_{2c} = \overline{A} + \overline{B} + C$$

1.3. Escribir la tabla de verdad de las funciones booleanas:

a) $A.\overline{B} + \overline{A}.C + B$

b) $(A + \overline{B}) . (\overline{B} + C) . (\overline{C} + D)$

Las tablas de verdad se estudian en el apartado 3.1.2 del texto base (págs. 74-76) y apartados subsiguientes.

Existen diversas formas de calcular la tabla de verdad de una función a partir de su expresión algebraica. La más directa es probar sobre la misma cada una de las posibles combinaciones de valores, mediante un procedimiento mecánico de comprobación.

Otra posibilidad es convertir la expresión a cualquiera de las formas canónicas, y de ésta obtener la tabla de verdad de la función. Este segundo procedimiento es el inverso del mostrado en el apartado 3.2.6 del texto base: cada minterm presente en la primera forma canónica da lugar a un 1 en la tabla de verdad, y cada maxterm de la expresión en segunda forma canónica aporta un 0 a la misma.

Se propone como ejercicio calcular las tablas de verdad mediante los dos métodos expuestos. Por tratarse de procedimientos rutinarios, esta tarea no ofrece mayor dificultad.

La solución presentada se basa en el análisis de la estructura de la función y los términos que contiene. Si la función se halla expresada como suma de productos, su valor será 1 en cuanto lo sea al menos uno de sus términos, y valdrá 0 en cualquier otro caso. En cambio, ante un producto de sumas, la función valdrá 1 sólo para combinaciones en las que todos los términos lo sean.

a) $f_a = A.\overline{B} + \overline{A}.C + B$

La función se encuentra expresada en forma de suma de productos. El examen de cada uno de sus términos indica lo siguiente:

- $t_1 = A.\overline{B}$: este término valdrá 1 si $A = 1$ y $B = 0$, para C cualquiera.
- $t_2 = \overline{A}.C$: será 1 si $A = 0$ y $C = 1$, cualquiera que sea el valor de B.
- $t_3 = B$: es 1 si $B = 1$.

La función f_a valdrá 1 en cuanto t_1, t_2 o t_3 lo valgan, y será 0 para combinaciones que hagan 0 a los tres términos simultáneamente. Con tales condiciones es fácil construir la tabla de verdad:

A	B	C	f_a
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

b) $f_b = (A + \overline{B}) . (\overline{B} + C) . (\overline{C} + D)$

En este caso, la expresión de la función viene dada en forma de producto de sumas. Por ello, el valor de la misma será 1 solamente para las combinaciones de valores de las variables que hagan 1 a todos y cada uno de los términos de la función.

A continuación se incluye el análisis pormenorizado de los términos de la expresión:

- $t_1 = A + \overline{B}$: vale 1 si $A = 1$ ó $B = 0$, y será 0 en caso contrario.
- $t_2 = \overline{B} + C$: será 1 cuando $B = 0$ ó $C = 1$.
- $t_3 = \overline{C} + D$: es 1 si $C = 0$ ó $D = 1$.

Las combinaciones que hacen simultáneamente 1 a los tres términos son:

- $B = 0$ (hace 1 a t_1 y t_2), $C = 0$ (a t_3), A y D cualesquiera.
- $B = 0$ (t_1 y t_2), $D = 1$ (t_3), A y C cualesquiera.
- $A = 1$ (t_1), $C = 1$ (t_2), $D = 1$ (t_3), B cualquiera.

A partir de estas restricciones se obtiene la tabla de verdad de la función, mostrada en la página siguiente.

A	B	C	D	f_b
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

1.4. Simplificar las siguientes expresiones booleanas mediante el método de Karnaugh:

a) $\overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.D + A.\overline{B}.\overline{C} + A.B.\overline{C} + \overline{A}.B.C.\overline{D}$

b) $(\overline{A}+\overline{C}+\overline{D}) . (\overline{A}+\overline{B}+D) . (A+\overline{B}+\overline{C}) . (A+B+\overline{C})$

La simplificación de funciones mediante el mapa de Karnaugh se explica en el apartado 3.3.3 del texto base (págs 86-94).

a) $f_a = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.D + A.\overline{B}.\overline{C} + A.B.\overline{C} + \overline{A}.B.C.\overline{D}$

La función se halla expresada en forma de suma de 5 productos, a partir de los cuales se obtienen los siguientes minterms:

- $t_1 = \overline{A}.\overline{C}.\overline{D} = \overline{A}.(B+\overline{B}).\overline{C}.\overline{D} = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.\overline{D} = m_4 + m_0$
- $t_2 = \overline{A}.\overline{B}.D = \overline{A}.\overline{B}.(C+\overline{C}).D = \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D = m_3 + m_1$
- $t_3 = A.\overline{B}.\overline{C} = A.\overline{B}.\overline{C}.(D+\overline{D}) = A.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} = m_9 + m_8$
- $t_4 = A.B.\overline{C} = A.B.\overline{C}.(D+\overline{D}) = A.B.\overline{C}.D + A.B.\overline{C}.\overline{D} = m_{13} + m_{12}$
- $t_5 = \overline{A}.B.C.\overline{D} = m_6$.

Y $f_a = m_0 + m_1 + m_3 + m_4 + m_6 + m_8 + m_9 + m_{12} + m_{13}$. La tabla de Karnaugh será:

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	1	1	0
$\overline{A}B$	1	0	0	1
AB	1	1	0	0
$A\overline{B}$	1	1	0	0

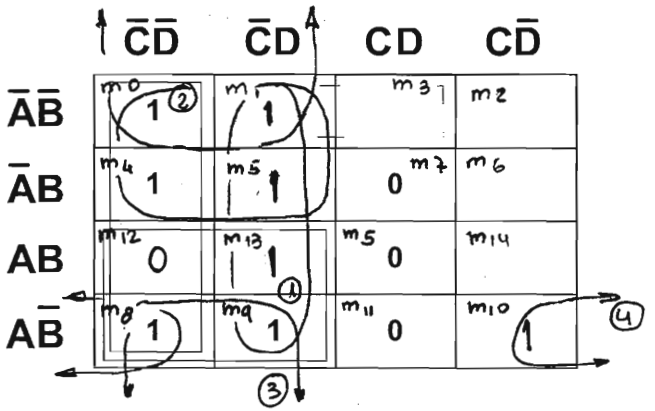
Así, la función queda $f_a = A.\overline{C} + \overline{C}.\overline{D} + \overline{A}.\overline{B}.D + \overline{A}.B.\overline{D}$

b) $f_b = (\overline{A}+\overline{C}+\overline{D}) \cdot (\overline{A}+\overline{B}+D) \cdot (A+\overline{B}+\overline{C}) \cdot (A+B+\overline{C})$

La función se halla expresada en forma de producto de sumas, con 4 términos. Para obtener los maxterms correspondientes se procederá como sigue:

- $t_1 = \overline{A}+\overline{C}+\overline{D} = \overline{A}+B.\overline{B}+\overline{C}+\overline{D} = (\overline{A}+B+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+\overline{D}) = M_4 . M_0$
- $t_2 = \overline{A}+\overline{B}+D = \overline{A}+\overline{B}+C.\overline{C}+D = (\overline{A}+\overline{B}+C+D).(\overline{A}+\overline{B}+\overline{C}+D) = M_3 . M_1$
- $t_3 = A+\overline{B}+\overline{C} = A+\overline{B}+\overline{C}+D.\overline{D} = (A+\overline{B}+\overline{C}+D).(A+\overline{B}+\overline{C}+\overline{D}) = M_9 . M_8$
- $t_4 = A+B+\overline{C} = A+B+\overline{C}+D.\overline{D} = (A+B+\overline{C}+D).(A+B+\overline{C}+\overline{D}) = M_{13} . M_{12}$

Entonces $f_b = M_0 . M_1 . M_3 . M_4 . M_8 . M_9 . M_{12} . M_{13}$. La tabla de Karnaugh expresada a partir de los maxterms será:



Y la expresión simplificada de la función es $f_b = (A+\overline{C}) . (\overline{C}+\overline{D}) . (\overline{A}+\overline{B}+D)$

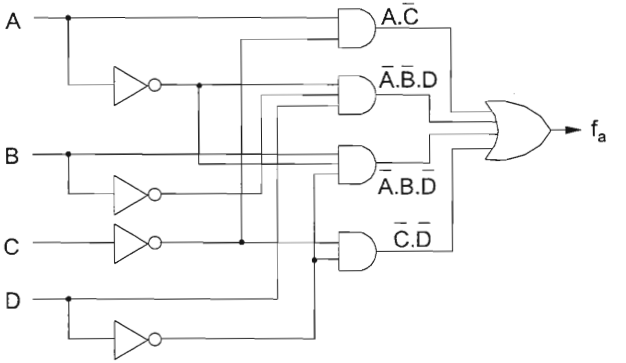
$\overline{f_b} = M_2 \cdot M_5 \cdot M_6 \cdot M_7 \cdot M_{10} \cdot M_{11} \cdot M_{14} \cdot M_{15}$

$f_b = \overline{\Pi(2,5,6,7,10,11,14,15)} = \Sigma(13,10,9,8,5,4,1,0)$

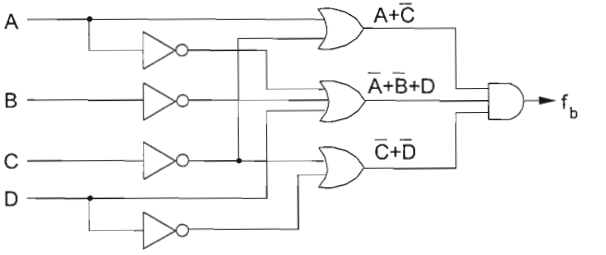
$f_b = \underbrace{\bar{c}\,d}_{1} + \underbrace{\bar{a}\,\bar{c}}_{2} + \underbrace{\bar{b}\,\bar{c}}_{3} + \underbrace{\bar{a}\,\bar{b}\,\bar{d}}_{4}$

1.5. Realizar las funciones simplificadas obtenidas en el ejercicio anterior mediante puertas OR, AND y NO.

a) La función $f_a = A.\overline{C} + \overline{C}.\overline{D} + \overline{A}.\overline{B}.D + \overline{A}.B.\overline{D}$ se representa mediante un circuito de orden 3 con 4 puertas NO, 4 AND y 1 OR.



b) El circuito que implementa la función $f_b = (A+\overline{C}) . (\overline{C}+\overline{D}) . (\overline{A}+\overline{B}+D)$ es de orden 3 y requiere 4 puertas NO, 3 OR y 1 AND.



1.6. En un registro de cuatro bits cuyas salidas están disponibles al exterior se almacena información en código BCD.

a) Determinar la tabla de verdad de un circuito que detecte que el número contenido en el registro es par.

b) Minimizar las expresiones canónicas algebraicas de este circuito por el método de Karnaugh

c) Realizar la expresión mínima con puertas NAND y NOR.

a) Tabla de verdad

El código BCD se explica en el apartado 4.3.2.5 del texto base. Su tabla es la siguiente:

Número decimal	R_3	R_2	R_1	R_0	f_{par}
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
-	1	0	1	0	X
-	1	0	1	1	X
-	1	1	0	0	X
-	1	1	0	1	X
-	1	1	1	0	X
-	1	1	1	1	X

A la derecha se ha incluido una columna que contiene un 1 si la cifra decimal representada en su fila es par, y 0 si es impar. Por tanto, dicha columna contiene los valores de la función del enunciado, y la tabla anterior constituye su tabla de verdad. Las seis últimas entradas representan redundancias, pues corresponden a combinaciones no

válidas en el código que, por tanto, nunca pueden darse. Por ello, el valor de f_{par} en estos casos es indiferente.

b) Minimización por el método de Karnaugh

La tabla de Karnaugh se construye a partir de la tabla de verdad de la función:

	$\bar{R}_1\bar{R}_0$	$\bar{R}_1 R_0$	$R_1 R_0$	$R_1\bar{R}_0$
$\bar{R}_3\bar{R}_2$	1	0	0	1
$\bar{R}_3 R_2$	1	0	0	1
$R_3 R_2$	X	X	X	X
$R_3\bar{R}_2$	1	0	X	X

En la simplificación se han tomado tres minterms correspondientes a redundancias para así obtener una expresión más reducida de la función. La expresión resultante es $f_{par} = \overline{R_0}$.

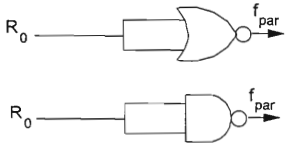
c) Circuito mínimo con puertas NAND y NOR

La representación de esta expresión en forma de circuito requiere emplear únicamente un inversor NOT. Sin embargo, en el enunciado se indica explícitamente que sólo pueden utilizarse puertas NAND y NOR. Es necesario pues adecuar la expresión de la función para que pueda representarse por tales tipos de puerta. Las funciones lógicas de estos dos modelos de puerta son:

- $f_{NAND} = \overline{A.B} = \bar{A}+\bar{B}$.
- $f_{NOR} = \overline{A+B} = \bar{A}.\bar{B}$.

Tanto una puerta NAND como una NOR son capaces de actuar como inversores, pues $\overline{A.A} = \bar{A}$; $\overline{A+A} = \bar{A}$.

Por tanto, el circuito resultante es uno cualquiera de los presentados en la figura.



Un ejemplo de mayor dificultad lo constituye la resolución del ejercicio utilizando el código BCD biquinario 5-4-2-1, cuyas tablas de verdad y de Karnaugh son:

Número decimal	R_3	R_2	R_1	R_0	f_{par}
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
-	0	1	0	1	X
-	0	1	1	0	X
-	0	1	1	1	X
5	1	0	0	0	0
6	1	0	0	1	1
7	1	0	1	0	0
8	1	0	1	1	1
9	1	1	0	0	0
-	1	1	0	1	X
-	1	1	1	0	X
-	1	1	1	1	X

	$\bar{R}_1\bar{R}_0$	$\bar{R}_1R_0$	R_1R_0	$R_1\bar{R}_0$
$\bar{R}_3\bar{R}_2$	1	0	0	1
$\bar{R}_3R_2$	1	X	X	X
R_3R_2	0	X	X	X
$R_3\bar{R}_2$	0	1	1	0

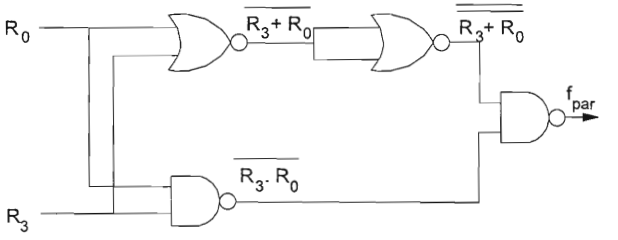
Teniendo en cuenta los tres minterms redundantes elegidos en la simplificación, la expresión resultante es $f_{par} = \overline{R_3}\cdot\overline{R_0} + R_3\cdot R_0$

La suma presente en la función f_{par} puede expresarse con una puerta NAND $f_{NAND} = \overline{A}+\overline{B}$ tal que

- $\overline{A} = \overline{R_3}\cdot\overline{R_0} = \overline{R_3+R_0} \rightarrow A = \overline{\overline{R_3+R_0}}$, que equivale a una NOR más un inversor.
- $\overline{B} = R_3\cdot R_0 \rightarrow B = \overline{R_3\cdot R_0}$, que corresponde con una NAND.

Con todo esto, la expresión de la función queda $f_{par} = \overline{\overline{\overline{R_3+R_0}}\cdot\overline{\overline{R_3\cdot R_0}}}$.

El circuito correspondiente contiene en total cuatro puertas, de las cuales 2 son NAND y las otras dos son NOR, una de ellas actuando como inversor.



1.7. Utilizando codificación binaria, complemento a 2 y coma flotante con formato de precisión ampliada (5 bits de exponente, 10 bits de mantisa y 1 de signo)

a) Hallar el equivalente decimal de 0110001000000000

b) Hallar el equivalente binario de –0'078125

Los formatos de representación en coma flotante se describen en los apartados 4.3.5.2 y 4.3.5.3 del texto base (págs. 140-148).

A la vista del enunciado, un posible formato de coma flotante que encaja con el del enunciado es el siguiente:

signo 1 bit	exponente 5 bits	mantisa 10 bits

Nos encontramos ante un formato de comparación rápida con mantisa normalizada y precisión ampliada con coma a la izquierda. Se parte del supuesto de que el exponente se representa en exceso a $2^{5-1} = 2^4$.

a) Equivalente decimal de 0110001000000000

Para calcular el equivalente decimal de 0110001000000000, desglosaremos el número en sus distintos campos:

- Signo: 0 → el número es positivo.
- Exponente: 11000 → $exp = 2^4+2^3-2^4 = 2^3 = 8$.
- Mantisa: (1)1000000000 → $m = 2^{-1}+2^{-2} = 0'5+0'25 = 0'75$.

Entonces el número representado es $X = m \cdot 2^{exp} = 0'75 \cdot 2^8 = 192$.

b) Equivalente binario de –0'078125

El primer paso consiste en hallar el exponente de la representación. Como el valor absoluto de la mantisa es siempre menor que la unidad, el exponente se calcula como el menor entero *exp* tal que $|x| \leq 2^{exp}$. En este caso, $exp = -3$.

El siguiente paso lo constituye el cálculo de la mantisa. Su primer bit significativo se corresponde con el de la potencia $i = exp-1$ debido a la condición empleada para calcular el exponente.

i	X(i)	2^i	$X(i) \geq 2^i$	x_i	$X(i-1) = X(i) - x_i \cdot 2^i$
-4	0'078125	0'0625	sí	1	0'015625
-5	0'015625	0'03125	no	0	0'015625
-6	0'015625	0'015625	sí	1	0

Puesto que la última diferencia calculada es 0, la representación es exacta.

De la tabla se deduce que la representación binaria normalizada de la mantisa es 0'101. Conocido el exponente y teniendo en cuenta que el número es negativo y que se emplea la técnica del bit implícito, el contenido de los distintos campos es:

- Signo: 1.
- Exponente: $-3+2^4 = 13 \rightarrow 01101$.
- Mantisa: 10100000000 → (0)1100000000 (complemento a 2).

Por tanto, la representación completa es 1 01101 1100000000.

1.8. Diseñar sistemas de codificación binaria de la fecha y la hora.

Se supondrá que la fecha se representa en el formato *dd-mm-aaaa*, mientras que la hora se expresa como *hh:mm:ss* con *hh* de 0 a 23.

Los formatos analizados se explican a lo largo de los capítulos 4 y 5 del texto base. En el análisis se indica el capítulo, el apartado y las páginas de las mismas donde se encuentra cada uno de ellos.

a) Códigos alfanuméricos

Estos códigos se emplean para representar caracteres, con lo que cada dígito de la fecha y la hora ocupará un carácter completo. Esto lleva a una codificación por campos, donde la fecha ocupa 8 caracteres y la hora requiere 6 (los símbolos '-' y ':' no se consideran).

En este apartado se puede citar el BCD alfanumérico (cap. 5, apdo. 5.1.2, págs 152-153), el ASCII (cap. 5, apdo. 5.1.3, págs. 154-155) y el EBCDIC (cap. 5, apdo. 5.1.4, págs. 154,157), que utilizan 6, 7 y 8 bits por carácter respectivamente. La siguiente tabla sintetiza el número de bits necesarios en cada código.

	BCD alfanumérico	ASCII	EBCDIC
Bits por carácter	6	7	8
Fecha (8 caract.)	8 x 6 = 48 bits	8 x 7 = 56 bits	8 x 8 = 64 bits
Hora (6 caract.)	6 x 6 = 36 bits	6 x 7 = 42 bits	6 x 8 = 48 bits
Total	84 bits	98 bits	112 bits

Con estos códigos se puede representar cualquier hora y fecha desde el 1 de enero del año 0 hasta el 31 de diciembre de 9999.

b) Código BCD numérico (cap. 4, apdo. 4.3.2.5, págs. 122-123)

En este código se emplean 4 bits por cada cifra decimal, con lo que:

- La fecha consumirá 8 x 4 = 32 bits.
- La hora ocupará 6 x 4 = 24 bits.
- Total ocupado: 56 bits.

c) Codificación optimizada por campos

Si se efectúa un análisis más detallado de los ítems que componen la fecha se deduce lo siguiente:

- El día del mes va de 1 a 31, con lo que puede codificarse con 5 bits.
- El mes va de 1 a 12, y bastan 4 bits para codificarlo.
- Para representar desde el año 1 hasta el 9999 se emplearán 14 bits, que realmente sirven para llegar hasta el año 16.384.
- Total ocupado por la fecha: 23 bits.

Con los ítems que componen la hora se puede hacer un análisis parecido:

- Hora: 5 bits.
- Minutos: 6 bits.
- Segundos: 6 bits.
- Total ocupado por la hora: 17 bits.

Por tanto, es posible codificar ambos conceptos con sólo 40 bits.

d) Codificación binaria directa

Aunque el código expuesto en el apartado anterior es más compacto que los de los apartados precedentes, existen en él combinaciones no válidas que hacen que se desaproveche una cierta cantidad de espacio. En efecto, con 5 bits es posible codificar 32 valores distintos, mientras que de ellos sólo se utilizarán 24 códigos para representar la hora. Un hecho análogo sucede con la codificación empleada para los minutos, los segundos, los días del mes y los meses del año.

Una posibilidad no considerada hasta ahora es codificar la fecha y la hora mediante un código binario denso que aproveche todas y cada una de las combinaciones permitidas de bits de una forma natural. Si se codifica la fecha y la hora completa a partir de los segundos transcurridos desde las 0 horas del día 1 de enero del año 0 (que correspondería con una codificación con todos los bits a 0), todas las combinaciones de bits disponibles en la codificación serán utilizables.

El rango disponible con esta codificación es mayor que el conseguido con las analizadas anteriormente. Con 40 bits existen 2^{40} = 1.099.511.627.776 combinaciones distintas, con lo que, tomando una media de 365 días al año, con este método se pueden representar fechas y horas de unos 34.865 años, frente a los 16.384 del caso mejor considerado en apartados previos. Esta mejora se consigue a costa de complicar el proceso de codificación y decodificación de la información, pues la tarea de extraer la fecha y la hora a partir del número codificado es más difícil que en los casos anteriores.

1.9. Obtener la representación binaria del número decimal –0'01 en formato normalizado IEEE 754 para coma flotante de 32 bits. Obtener el equivalente decimal de

0 10101100 10010001100000000000000.

El formato estándar IEEE 754 para representación de números en coma flotante se describe en el apartado 4.3.5.4 del texto base (págs. 148-150). Puesto que en este ejercicio se utiliza el formato de 32 bits, se cuenta con:

- 1 bit de signo.
- 8 bits de exponente, representado en exceso a $2^{8-1} - 1 = 127$.
- 23 bits de mantisa representada en módulo y signo, normalizada con bit implícito a la izquierda de la coma.

a) Representar en binario –0'01.

Como en este formato $1 \leq |m| < 1'9999$, el exponente se calcula como el mayor valor entero *exp* que cumple $|x| \geq 2^{exp}$. En este caso, *exp* = -7. La condición empleada para el cálculo del exponente conlleva que el primer dígito significativo de la mantisa corresponda a la potencia i = *exp*. Entonces, la tabla para calcular la mantisa es

i	X(i)	2^i	$X(i) \geq 2^i$	x_i	$X(i-1) = X(i) - x_i \cdot 2^i$
-7	0'01	0'0078125	sí	1	0'0021875
-8	0'0021875	0'00390625	no	0	0'0021875
-9	0'0021875	0'00195312	sí	1	0'000234375
-10	0'000234375	0'000976562	no	0	0'000234375
-11	0'000234375	0'000488281	no	0	0'000234375
-12	0'000234375	0'000244141	no	0	0'000234375
-13	0'000234375	0'00012207	sí	1	0'000112305
-14	0'000112305	$6'10352.10^{-5}$	sí	1	$5'12695.10^{-5}$
-15	$5'12695.10^{-5}$	$3'05176.10^{-5}$	sí	1	$2'0752.10^{-5}$
-16	$2'0752.10^{-5}$	$1'52588.10^{-5}$	sí	1	$5'49316.10^{-6}$
-17	$5'49316.10^{-6}$	$7'62939.10^{-6}$	no	0	$5'49316.10^{-6}$
-18	$5'49316.10^{-6}$	$3'8147.10^{-6}$	sí	1	$1'67847.10^{-6}$
-19	$1'67847.10^{-6}$	$1'90735.10^{-6}$	no	0	$1'67847.10^{-6}$

i	X(i)	2^i	$X(i) \geq 2^i$	x_i	$X(i-1) = X(i) - x_i \cdot 2^i$
-20	$1'67847.10^{-6}$	$9'53674.10^{-7}$	sí	1	$7'24792.10^{-7}$
-21	$7'24792.10^{-7}$	$4'76837.10^{-7}$	sí	1	$2'47955.10^{-7}$
-22	$2'47955.10^{-7}$	$2'38419.10^{-7}$	sí	1	$9'53674.10^{-9}$
-23	$9'53674.10^{-9}$	$1'19209.10^{-7}$	no	0	$9'53674.10^{-9}$
-24	$9'53674.10^{-9}$	$5'96046.10^{-8}$	no	0	$9'53674.10^{-9}$
-25	$9'53674.10^{-9}$	$2'98023.10^{-8}$	no	0	$9'53674.10^{-9}$
-26	$9'53674.10^{-9}$	$1'49012.10^{-8}$	no	0	$9'53674.10^{-9}$
-27	$9'53674.10^{-9}$	$7'45058.10^{-9}$	sí	1	$2'08616.10^{-9}$
-28	$2'08616.10^{-9}$	$3'72529.10^{-9}$	no	0	$2'08616.10^{-9}$
-29	$2'08616.10^{-9}$	$1'86265.10^{-9}$	sí	1	$2'23517.10^{-10}$
-30	$2'23517.10^{-10}$	$9'31323.10^{-10}$	no	0	$2'23517.10^{-10}$

El bit x_{-7} es el bit implícito de la mantisa y constituye su parte entera, no formando parte de la representación binaria del número. Entonces:

- Signo: 1.
- Exponente: $-7+127 = 120 \rightarrow 01111000$.
- Mantisa: 01000111101011100001010.

Por tanto, la representación binaria del número –0'01 según el estándar de coma flotante IEEE 754 de 32 bits es 1 01111000 01000111101011100001010.

Puesto que la última diferencia calculada en la tabla no es nula, la representación del número no es exacta. El error obtenido es $2'23517.10^{-10}$.

b) Obtener el equivalente decimal de
0 10101100 10010001100000000000000.

Separando la representación binaria en sus campos se observa que:

- Signo: $0 \rightarrow$ el número es positivo.
- Exponente: $10101100 \rightarrow exp = 2^7+2^5+2^3+2^2-(2^7-1) = 45$.
- Mantisa: $10010001100000000000000 \rightarrow m = 1+2^{-1}+2^{-4}+2^{-8}+2^{-9} = 1'56689$.

Entonces su representación decimal es $m.2^{exp} = 1'56689 \cdot 2^{45} = 5'51302 \cdot 10^{13}$.

1.10. Utilizando código de Hamming para la transmisión de datos de 6 bits se recibieron los siguientes datos:

a) 1010001110

b) 0100110110

Comprobar si hubo error en alguno de ellos y, en tal caso, corregirlo (se supone que los errores que pueden producirse son sólo de un bit).

Las características y utilización de los códigos de Hamming se describen en el apartado 5.4.2 del texto base (págs. 164-167).

El tratamiento de los dos apartados del ejercicio se realizará de forma conjunta. Los cálculos necesarios para comprobar los errores se condensan en la siguiente tabla:

Nº de orden	Bit	Bits asociados	Recepción a)	Detección a)	Recepción b)	Detección b)
0 0 0 1	P_1	- - - x	1	- - - 1	0	- - - 0
0 0 1 0	P_2	- - x -	0	- - 0 -	1	- - 1 -
0 0 1 1	D_1	- - x x	1	- - 1 1	0	- - 0 0
0 1 0 0	P_3	- x - -	**0**	- 0 - -	0	- 0 - -
0 1 0 1	D_2	- x - x	0	- 0 - 0	1	- 1 - 1
0 1 1 0	D_3	- x x -	0	- 0 0 -	1	- 1 1 -
0 1 1 1	D_4	- x x x	1	- 1 1 1	0	- 0 0 0
1 0 0 0	P_4	x - - -	1	1 - - -	1	1 - - -
1 0 0 1	D_5	x - - x	1	1 - - 1	1	1 - - 1
1 0 1 0	D_6	x - x -	0	0 - 0 -	0	0 - 0 -
			error	0 1 0 0	correcto	0 0 0 0

De la tabla se deduce que la recepción en el caso b) es correcta, no así en el caso a), donde se ha producido un error en el bit P_3, resaltado en negrita. Complementando dicho bit se obtiene la información correcta en a), que será 1011001110.

1.11. Para transmitir una información se utiliza paridad longitudinal y transversal (paridad par). Si al receptor le llega el siguiente bloque de información (dado en hexadecimal)

A3 2D FC 24 17 0D E4 B8

Decir si es correcto y, en caso contrario, corregirlo (se supone que sólo puede haber fallo en un bit).

Los códigos de paridad se revisan en el apartado 5.4.1 del texto base (págs. 163-164).

La comprobación de posibles errores en la información recibida se efectúa a través de la siguiente tabla:

A3	1	0	1	0	0	0	1	1	
2D	0	0	1	0	1	1	0	1	
FC	1	1	1	1	1	1	0	0	
24	0	0	1	0	0	1	0	0	
17	0	0	0	1	0	1	1	1	
0D	0	0	0	**0**	1	1	0	1	⇐
E4	1	1	1	0	0	1	0	0	
B8	1	0	1	1	1	0	0	0	
				⇑					

Tanto la fila como la columna recuadradas presentan un número impar de unos, en tanto en cuanto la paridad debe ser par. El bit en negrita, situado en la intersección de la fila y la columna marcadas, es el bit erróneo. La corrección consistirá simplemente en complementar dicho bit, convirtiéndose el sexto byte del bloque en 1D.

2.1. Suponer que el computador descrito en el capítulo 6 dispone de una memoria organizada en bytes, con un tamaño total de 64Kbytes.

a) Estudiar las dimensiones más adecuadas para los buses de datos y direcciones, el operador y los registros.

b) Analizar el formato de instrucción necesario para disponer de direccionamiento directo absoluto a todo el mapa de memoria. Estudiar si hay que introducir alguna modificación en la estructura del computador para disponer de este direccionamiento.

c) Calcular el número de períodos de reloj necesarios para la ejecución de una instrucción de suma como la descrita en el capítulo 6, en los supuestos de este ejemplo.

a) Dimensiones de los buses de datos y direcciones, el operador y los registros.

De los datos que ofrece el enunciado en cuanto a la memoria del computador se puede deducir que:

- El ancho del bus de datos es de 8 bits, ya que la memoria se organiza en bytes.
- El ancho del bus de direcciones es de 16 bits para poder direccionar 64Kbytes.

Los mismos 16 bits que tiene el bus de direcciones son los que tienen el registro de direcciones RD y el contador de programa CP.

El registro de memoria RM tendrá 8 bits, el ancho de palabra de la memoria. El operador y sus registros asociados (RO1, RO2 y el acumulador AC), cuyo tamaño viene dado por el bus de datos, también serán de 8 bits.

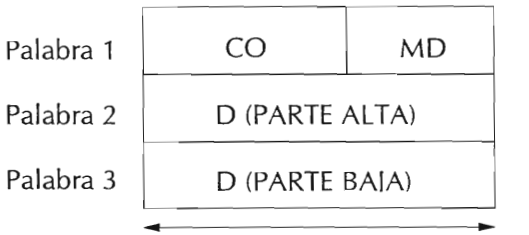
El tamaño del registro de instrucciones RI es independiente del ancho de los demás elementos. Sin embargo, se elegirá para él en este ejercicio un tamaño de 8 bits.

El tamaño de los registros de la batería de registros BR depende directamente de la utilidad que se les quiera dar. Si se van a emplear únicamente para manipular datos que deban atravesar el operador, lo más lógico es que tengan 8 bits cada uno. Sin embargo, los registros se usan a menudo para almacenar direcciones que serán más tarde empleadas en distintos tipos de direccionamientos. En estos casos es mejor disponer de registros con igual ancho que el bus de direcciones, que en nuestro caso es de 16 bits.

En principio, se asumirá que los registros del banco tienen 8 bits. En el apartado siguiente veremos cómo conjugar ambas situaciones adecuadamente.

b) Direccionamiento directo absoluto.

El formato de instrucción necesario para acceder a todo el mapa de memoria mediante direccionamiento directo absoluto requiere un campo de desplazamiento (D) de 16 bits. Suponiendo que el código de operación (CO) y el campo de modo de direccionamiento (MD) suman 8 bits, una instrucción de un operando con direccionamiento absoluto utilizará 3 palabras completas de 8 bits, donde la primera contiene los campos CO y MD, la segunda almacena la parte alta de la dirección absoluta y la tercera contiene la parte baja.



Para poder disponer de este modo de direccionamiento, es necesario trasvasar las dos partes de la dirección, obtenidas en dos lecturas sucesivas efectuadas sobre la memoria, al bus de direcciones. Esto hay que hacerlo, por supuesto, en dos ciclos de memoria diferentes. Para ello, es preciso disponer de un elemento de almacenamiento intermedio de 16 bits de ancho que admita cargas de datos de 8 bits procedentes del BD. Las cargas deben poder realizarse, alternativamente, en la parte alta o en la parte baja de dicho elemento, para lo que será preciso disponer de dos señales de carga, una para gobernar la entrada de información en la parte alta y otra para la entrada en la parte baja. La salida del elemento intermedio estará conectada directamente al bus de direcciones, lo que requerirá para éste una nueva señal de selección.

Muchos microprocesadores de 8 bits con mapa de memoria de 64 Kbytes disponen de una batería de registros organizada del siguiente modo: cada registro tiene 8 bits de ancho, pero a la vez existen algunos de ellos (o todos) que se encuentran agrupados de dos en dos, de manera que cada pareja de registros individuales conforma un único registro de 16 bits. El funcionamiento del banco de registros es dual:

- Cuando se trata de introducir información en el banco, o bien recuperar el contenido de algún registro para enviarlo al operador aritmético-lógico, se utiliza como un banco de registros de 8 bits.

- Cuando se requiere enviar información al bus de direcciones, se contempla como un banco de registros de 16 bits.

Serán las señales de control del banco de registros quienes decidirán en cada momento el modo de funcionamiento del mismo.

Es corriente que al menos una de las parejas de registros del banco sea transparente al usuario, es decir, que no sea posible acceder a ninguno de los dos registros explícitamente desde un programa, sino que los reserve la UCP para tareas internas. Esta pareja de registros transparentes será la que se emplee como elemento intermedio para almacenar y convertir dos datos de 8 bits en una dirección completa de 16 al utilizar direccionamiento absoluto directo a toda la memoria.

Como conclusión a todo ello, se deduce que en nuestro caso será preciso incorporar al diseño del computador las siguientes modificaciones:

- El banco de registros deberá tener al menos una pareja de ellos dotada de funcionamiento dual.

- Existirá una conexión directa unidireccional desde el banco de registros hasta el bus de direcciones, regulada por la señal de control SBABR.

Habrá que dotar al banco de registros de los siguientes grupos de señales de control:

- SBRS8: al activarlo, la salida obtenida será de 8 bits, y se encaminará hacia el registro RO2 a la entrada del operador.

- SBRS16: la activación de este grupo de señales producirá una salida de 16 bits encaminada hacia el bus de direcciones.

Estos dos grupos de señales de control serán incompatibles, es decir, no podrán ser activados simultáneamente en un mismo ciclo de reloj.

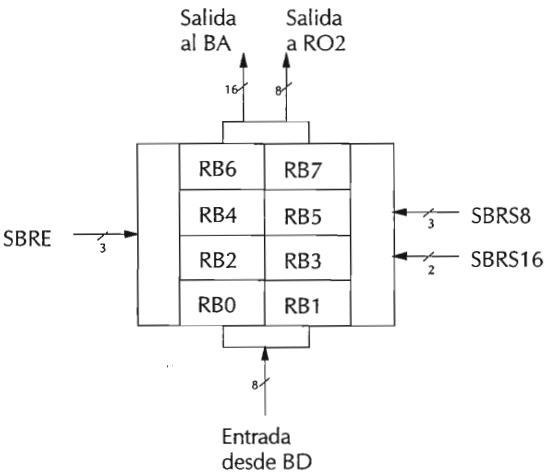
La carga de registros desde el bus de datos siempre se efectuará en modo 8 bits, para lo que será preciso activar el grupo de señales de control SBRE.

Si suponemos que el banco de registros cuenta con 8 registros de 8 bits, el grupo SBRS8 contendrá 3 señales de control, suficientes para referenciar a uno cualquiera de ellos. Si denotamos a los registros por RB0, RB1, ... , RB7, haciendo SBRS=011 obtendremos el contenido del registro RB3 en la salida de 8 bits.

En cuanto al funcionamiento en 16 bits, el banco se dividirá en cuatro parejas, a saber: RB0-RB1, RB2-RB3, RB4-RB5 y RB6-RB7. En cada una de las parejas habrá un registro que opere como parte alta (RB0, RB2, RB4 y RB6) y otro como parte baja (RB1, RB3, RB5 y RB7). El grupo SBRS16 se compone de dos bits que seleccionarán una de las parejas. Por ejemplo, con SBRS16 = 11, en la salida de 16 bits aparecerá concatenado el contenido de los registros RB6 y RB7, con los 8 bits de RB6 ocupando la parte alta y los 8 de RB7 la parte baja.

La entrada de datos al banco de registros se efectúa únicamente en modo 8 bits, y se regula mediante el grupo de señales SBRE, que funciona de manera análoga a como lo hace el grupo SBRS8.

El esquema resultante del banco de registros después de introducirle todas estas modificaciones se muestra en la siguiente figura.



c) Instrucción de suma

La secuencia de operaciones para ejecutar la instrucción *ADD n* es la siguiente:

a) Fase de búsqueda del código de instrucción:

1) Transferir el contenido del CP al registro de direcciones RD.
2) Lanzar un ciclo de memoria para leer la primera palabra de la instrucción y ponerla en RM.
3) Enviar la palabra leída al registro de instrucción RI.
4) Incrementar CP.

b) Fase de ejecución

5) Decodificar la instrucción.
6) Transferir el contenido de CP a RD.
7) Lanzar un ciclo de memoria para leer la parte alta de la dirección y ponerla en RM.
8) Enviar la parte alta de la dirección al registro transparente RB6.
9) Incrementar CP.
10) Enviar el contenido del CP a RD.
11) Lanzar un ciclo de memoria para leer la parte baja de la dirección y ponerla en RM.
12) Transferir la parte baja de la dirección al registro transparente RB7.
13) Incrementar CP.
14) Enviar el contenido del par de registros RB6-RB7 a RD.
15) Lanzar un ciclo de memoria para leer el dato y ponerlo en RM.
16) Enviar el dato de RM al registro de entrada al operador RO2, y el contenido del acumulador AC a RO1.
17) Efectuar la suma y cargar el resultado en el acumulador.

Suponiendo que el ciclo de reloj dura *TCR*, en este desglose hay que tener en cuenta lo siguiente:

- Los ciclos de lectura en memoria son, por definición, de duración variable, y consumen en general varios ciclos de reloj completos. Por comodidad, en este ejercicio se supondrá que los accesos a memoria son constantes y duran *n* ciclos, siendo *n* un número natural. En total, las cuatro lecturas de la instrucción ocuparán $4*n*TCR$ ciclos de reloj.

- El incremento del CP puede realizarse durante el ciclo de memoria; así, el paso 4 se puede ejecutar junto con el paso 2, el 9 junto con el 7 y el 13 dentro del 11. Por tanto, no se toman en cuenta, ahorrándonos entonces 3 ciclos de reloj.

- Si el paso 9 se ejecuta al mismo tiempo que el paso 7, los pasos 8 y 10 pueden ejecutarse simultáneamente, ahorrando un ciclo de reloj más.

El total de operaciones contabilizado será:

- 4 ciclos de memoria: $4*n*TCR$.
- 3 incrementos del CP: 0 ciclos.
- 10 operaciones normales, solapando 8 y 10: $9*TCR$.

Tomando, por ejemplo, $n=3$, tendremos que la instrucción completa necesitará para su ejecución un total de 21 ciclos de reloj.

2.2. Supóngase que el computador elemental desarrollado en el capítulo 6 posee una instrucción del tipo *JUMP n*, para realizar una bifurcación en la ejecución de un programa. La nueva dirección de ejecución será *n*, que se encuentra codificada, junto con el código de instrucción, en un formato de una sola palabra.

a) Analizar la fase de ejecución de esta instrucción, indicando las operaciones elementales para llevarla a cabo.

b) Dibujar y comentar el cronograma de esta instrucción.

a) Análisis de la fase de ejecución de la instrucción.

El desarrollo de la instrucción comprende dos fases:

- Fase de búsqueda de la instrucción, que es igual para todas las instrucciones. En el apartado 6.3.1 del capítulo 6 del texto base de la asignatura se explican detalladamente las operaciones elementales involucradas en esta fase.

- Fase de ejecución de la instrucción, que es la que se va a analizar seguidamente.

La instrucción *JUMP n* provoca una bifurcación incondicional en la secuencia de ejecución del programa. La siguiente instrucción ejecutada tras ella será la que se encuentre almacenada en la dirección de memoria *n*. Por tanto, la única acción que debe ser llevada a cabo por esta instrucción es cargar el CP con el valor *n*, que es una dirección absoluta contenida en el campo D del formato de instrucción.

Puesto que la instrucción completa se halla almacenada en una única palabra de memoria, en el instante en que se ejecuta la instrucción *JUMP* el valor *n* está en el campo correspondiente del registro de instrucciones del computador (RI). Entonces, las señales que deben activarse para transferir *n* desde RI a CP son:

- SBARI para enviar *n* desde RI al bus de direcciones: (RI) → BA.
- CCPBA para cargar el CP desde el bus de direcciones: (BA) → CP.

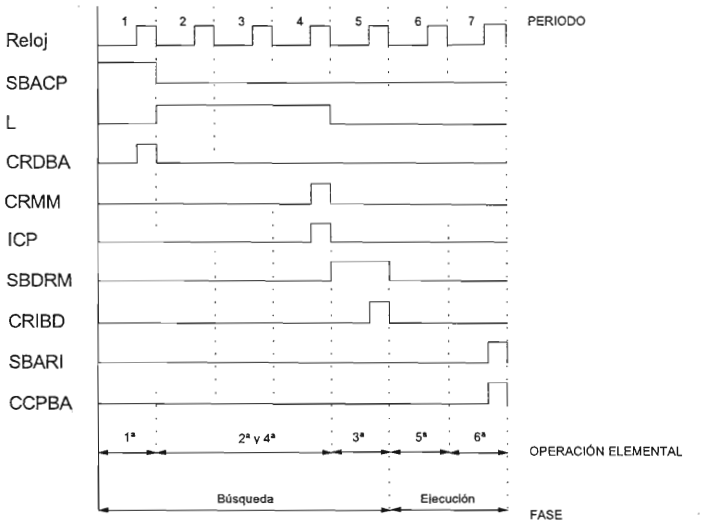
Estas dos señales deben ser activadas en el mismo período de reloj, por lo cual la fase de ejecución de la instrucción tan sólo comporta dos ciclo de reloj, asumiendo que la decodificación de la misma ocupa una única fase.

b) Dibujar y comentar el cronograma de esta instrucción.

El cronograma completo de la instrucción se presenta en la página siguiente. Paradójicamente, en él se observa que la fase de ejecución de la instrucción ocupa muchos menos ciclos de reloj que la fase de búsqueda de la misma en memoria. Esto pone de relieve que la simple acción de búsqueda en memoria de las instrucciones que componen un programa ocupa buena parte del tiempo que tarda en ejecutarse.

La explicación a este hecho es bien sencilla: los accesos a memoria son muy costosos en tiempo de ejecución, pues el procesador debe esperar normalmente varios ciclos de reloj desde que inicia la lectura de un dato hasta que la memoria responde con el dato solicitado. En este ejemplo se ha supuesto que el ciclo de lectura de memoria consume 3 periodos de reloj más la carga del registro de direcciones, lo que constituye el 57% de la duración total de la ejecución de la instrucción. Las instrucciones que utilicen operandos almacenados en memoria deben efectuar accesos suplementarios a la misma, con lo que serán más lentas que aquéllas que manejen operandos almacenados en los registros de la máquina, a los que se accede mucho más rápidamente. Por tanto, es recomendable limitar en lo posible los accesos a memoria para acelerar la ejecución de los programas.

En este ejercicio, el único operando de que consta la instrucción se encuentra almacenado en el propio registro de instrucción, con lo cual, y debido a las peculiares características de la operación realizada, la fase de ejecución es especialmente corta.



2.3. Se quiere dotar al computador del capítulo 6 con un tipo de instrucciones aritmético lógicas diádicas, que utilice dos operandos en memoria y deposite el resultado en el registro AC. Una instrucción de este tipo podría ser:

Instrucción	Descripción
SUB n1,n2	*(n1) - (n2) → AC*

donde los valores *n1* y *n2* son las direcciones en memoria de cada uno de los operandos, y forman parte del código de la instrucción. Supóngase que estas direcciones ocupan una palabra de memoria cada una. Por tanto, este tipo de instrucciones ocupará tres palabras de memoria consecutivas: la primera para el código de instrucción y las dos siguientes para las direcciones de los operandos.

a) Estudiar si este tipo de instrucciones implica algún cambio en la estructura del computador.
b) Analizar la ejecución de la instrucción *SUB* descrita.

a) Cambios en la estructura del computador.

Puesto que cada dirección de operando ocupa una palabra completa de memoria, se puede suponer que el ancho del bus de direcciones y el del bus de datos coinciden, así como el de los registros del banco, el acumulador, el operador, el contador de programa, el registro de direcciones, el de memoria y el de instrucción.

Para efectuar direccionamiento directo, es preciso trasvasar datos desde el registro de memoria RM hasta el registro de direcciones RD. En nuestro computador, el único camino disponible para realizar la operación pasa por el operador y sus registros asociados (el registro de instrucciones no puede ser utilizado para tal fin, pues deberá conservar su contenido durante toda la ejecución de la instrucción).

Lo mejor es establecer un nuevo camino de comunicación que permita enviar información desde el bus de datos hacia el bus de direcciones. Puesto que los dos tienen el mismo ancho, lo más sencillo es colocar directamente una conexión unidireccional desde BD hacia BA, regulada por una nueva señal de selección que llamaremos SBABD.

b) Analizar la ejecución de la instrucción *SUB* descrita.

La instrucción que se va a ejecutar consta de 3 palabras de memoria consecutivas, de las cuales la primera contiene el código de operación y los modos de direccionamiento de los operandos, la segunda almacena la dirección en memoria del primer operando, y la tercera guarda la del segundo.

Por tanto, la secuencia de acciones que hay que llevar a cabo en la fase de ejecución de dicha instrucción es:

1) Leer la dirección del primer operando.
2) Leer el primer operando y ponerlo en uno de los registros transparentes del operador.
3) Leer la dirección del segundo operando.
4) Leer el segundo operando y ponerlo en uno de los registros transparentes del operador.
5) Hacer la resta y guardar el resultado en el acumulador.

Con las modificaciones introducidas en la arquitectura del computador, las operaciones elementales y las señales que hay que activar para ejecutar la instrucción propuesta son:

a) Fase de búsqueda del código de instrucción:

1) Transferir el contenido del CP al registro de direcciones RD.

(CP) → BA	SBACP
(BA) → RD	CRDBA

2) Lanzar un ciclo de memoria para leer la primera palabra de la instrucción y ponerla en RM.

Ciclo de lectura	L
Memoria(RD) → RM	CRMM

3) Enviar la palabra leída al registro de instrucción RI.

(RM) → BD	SBDRM
(BD) → RI	CRIBD

4) Incrementar CP.

(CP) + 1 → CP	ICP

b) Fase de ejecución

5) Decodificar la instrucción.
6) Transferir el contenido de CP a RD.

(CP) → BA	SBACP
(BA) → RD	CRDBA

7) Lanzar un ciclo de memoria para leer la dirección del primer operando y ponerla en RM.

Ciclo de lectura	L
Memoria(RD) → RM	CRMM

8) Enviar el dato leído (dirección del primer operando) a RD.

(RM) → BD	SBDRM
(BD) → BA	SBABD
(BA) → RD	CRDBA

9) Incrementar CP.

(CP) + 1 → CP	ICP

10) Lanzar un ciclo de memoria para leer el primer operando y ponerlo en RM.

Ciclo de lectura	L
Memoria(RD) → RM	CRMM

11) Enviar el operando al registro transparente RO1.

(RM) → BD	SBDRM
(BD) → RO1	CRO1BD

12) Transferir el contenido de CP a RD.

(CP) → BA	SBACP
(BA) → RD	CRDBA

13) Lanzar un ciclo de memoria para leer la dirección del segundo operando y ponerla en RM.

Ciclo de lectura	L
Memoria(RD) → RM	CRMM

14) Enviar el dato leído (dirección del segundo operando) a RD.

(RM) → BD	SBDRM
(BD) → BA	SBABD
(BA) → RD	CRDBA

15) Incrementar CP.

(CP) + 1 → CP	ICP

16) Lanzar un ciclo de memoria para leer el segundo operando y ponerlo en RM.

Ciclo de lectura	L
Memoria(RD) → RM	CRMM

17) Enviar el operando al registro transparente RO2.

(RM) → BD	SBDRM
(BD) → RO2	CRO2BD

18) Efectuar la resta y poner el resultado en el acumulador.

Operación(OPER) = SUB	OP
Sal(OPER) → AC	CAC

Asumiendo que cada ciclo de lectura en memoria consume 3 ciclos de reloj, la duración total de la ejecución de la instrucción es de 25 ciclos.

2.4. Razonar sobre cuál de los siguientes modos de direccionamiento es más rápido:

a) Mediante registro.
b) Indirecto.
c) Indirecto preindexado.
d) Relativo a base posindexado.

Para la resolución de este ejercicio, se analizará cada uno de los direccionamientos por separado, y se estimará el tiempo de acceso al objeto referenciado sumando las duraciones de cada una de las operaciones elementales necesarias para obtener el mismo. Se considerará que la duración del período es *TCR*, y que las operaciones de acceso a memoria consumen un tiempo representado por *TAM*. Hay que tener en cuenta que, en un computador convencional, $TAM = n*TCR$, donde $n \geq 1$.

Los tiempos de duración de las operaciones elementales se han calculado sobre el computador utilizado en el ejercicio 2.3, que es el mismo que se presentaba en el capítulo 6 del texto base, al que se ha añadido una conexión unidireccional desde el bus de datos hasta el bus de direcciones.

En todos los casos el objeto direccionado será depositado en uno de los registros transparentes del operador aritmético-lógico.

a) Direccionamiento mediante registro.

Este modo de direccionamiento se explica en el apartado 7.2.2.2 del texto base (págs. 228 y 229).

En este caso, el código de instrucción contiene el número identificativo de un registro de la batería de registros de la UCP, y el objeto buscado es precisamente el contenido del registro. Entonces, activando las señales de selección de la batería de registros el objeto estaría disponible automáticamente. De acuerdo con las indicaciones efectuadas al principio del ejercicio, este direccionamiento tiene un tiempo de acceso cuantificable como 1 *TCR*.

b) Direccionamiento indirecto.

El direccionamiento indirecto se describe en el apartado 7.2.5 del texto base (págs 234-235).

Suponiendo que el campo de dirección encuentre compactado junto con el código de instrucción en una única palabra de memoria, este direccionamiento requiere:

1) Cargar en RD la dirección contenida en el código de instrucción: 1*TCR*.
2) Lanzar una lectura en memoria para obtener la dirección del objeto: 1 *TAM*.
3) Cargar la dirección obtenida en RD: 1 *TCR*.
4) Efectuar una lectura en memoria para obtener el objeto: 1 *TAM*.
5) Llevar el objeto al registro transparente del operador: 1 *TCR*.

Total: 3 *TCR* + 2 *TAM* = $(2n+3)$ *TCR*.

c) Direccionamiento indirecto preindexado.

Este es un modo combinado de direccionamiento que se describe en el apartado 7.3.1.2 del texto base (pág. 237). Consta de una indirección en cuyo primer paso se suman el campo CD y el registro índice para obtener la dirección del puntero al objeto buscado.

Las acciones necesarias serán:

INDEXACIÓN Y 1er PASO DE LA INDIRECCIÓN
1) Cargar el desplazamiento (contenido en el registro de instrucción) y el valor del registro índice en los registros transparentes del operador de la UAL: 1 *TCR*.
2) Sumar el registro índice y el desplazamiento y cargar el resultado en el acumulador: 1 *TCR*.
3) Cargar el resultado en el registro de direcciones para acceso a memoria: 1 *TCR*.
4) Efectuar una lectura en memoria para obtener la dirección en memoria del objeto: 1 *TAM*.
2º PASO DE LA INDIRECCIÓN
5) Cargar la dirección obtenida en RD: 1 *TCR*.
6) Realizar una lectura en memoria para obtener el objeto: 1 *TAM*.
7) Llevar el objeto al registro transparente del operador: 1 *TCR*.

El tiempo total consumido será: 5 *TCR* + 2 *TAM* = (2*n*+5) *TCR*.

d) Direccionamiento relativo a base posindexado.

Este es un modo combinado de direccionamiento no descrito en el texto base. Ante ello, en la resolución de este apartado caben distintas interpretaciones. La que se propone en este ejercicio es la siguiente: el direccionamiento consta de una indirección en cuyo primer paso se suma el campo CD del código de instrucción al contenido de un registro base, con lo que se obtiene la dirección de memoria en la que se encuentra el puntero al objeto buscado. En el segundo paso de la indirección, al puntero obtenido anteriormente se le suma el contenido de un registro índice, de lo que resulta la dirección del objeto buscado.

Las acciones necesarias serán:

DIRECCIONAMIENTO RELATIVO A REGISTRO BASE Y 1er PASO DE LA INDIRECCIÓN
1) Cargar el registro base y el desplazamiento (contenido en el registro de instrucción) en los registros transparentes del operador de la UAL: 1 *TCR*.
2) Sumar ambos y cargar el resultado en el acumulador: 1 *TCR*.
3) Cargar resultado en el registro de direcciones para acceso a memoria: 1 *TCR*.
4) Efectuar una lectura en memoria para obtener el puntero al objeto: 1 *TAM*.
POSINDEXACIÓN Y 2º PASO DE LA INDIRECCIÓN
5) Llevar el puntero y el registro índice a los registros del operador: 1 *TCR*.
6) Sumar y cargar el resultado en el acumulador: 1 *TCR*.
7) Llevar el contenido del acumulador (dirección del objeto) al RD: 1 *TCR*.
8) Hacer una lectura en memoria para obtener el objeto: 1 *TAM*.
9) Llevar el objeto al registro del operador: 1 *TCR*.

El tiempo total consumido será 7 *TCR* + 2 *TAM* = (2*n*+7) *TCR*.

Por tanto, el más rápido es el direccionamiento mediante registro. Los valores obtenidos indican que en programas con requerimientos de velocidad altos será preferible usar intensivamente direccionamientos a registro frente a otros modos que requieren accesos a memoria.

2.5. Un computador posee 3 registros índices cuyos contenidos en un instante concreto, expresados en hexadecimal, son: *(R0) = 000A, (R1) = 000B* y *(R2) = 0001*. El computador tiene una memoria de 64Kpalabras de 16 bits. Supóngase que cada palabra contiene un valor igual a su dirección en memoria y que se utiliza una instrucción con un valor en el campo de direcciones *(CD) = (00C9)*. Calcular la dirección final en el caso de que se utilicen los siguientes direccionamientos:

a) Indirecto.
b) Preindexado al registro *R0*.
c) Posindexado al registro *R1*.

a) Direccionamiento indirecto

- *1er paso de la indirección*: la dirección inicial se obtiene leyendo el contenido de la posición de memoria dada por el CD de la instrucción: *(CD) = 00C9*.
- *2º paso de la indirección*: *(00C9) = 00C9*.

Por tanto, el objeto buscado se halla en la posición de memoria *00C9*.

b) Preindexado al registro *R0*.

- *Indexación y 1er paso de la indirección*: la dirección inicial se obtiene sumando el registro índice y el CD de la instrucción: *(R0) + (CD) = 000A+00C9 = 00D3*.
- *2º paso de la indirección*: *(00D3) = 00D3*.

Y el dato se encuentra en la posición de memoria *00D3*.

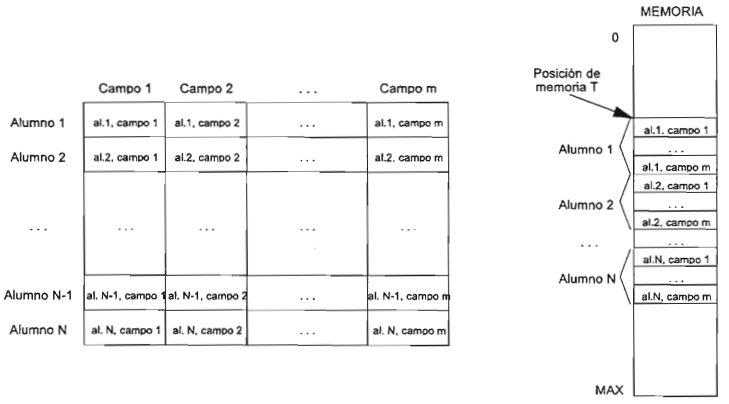
c) Posindexado al registro *R1*.

- *1er paso de la indirección*: *((CD)) = (00C9) = 00C9*.
- *Indexación y 2º paso de la indirección*: la dirección definitiva se calcula sumando el contenido del registro índice con la dirección obtenida en el paso anterior: *(R1) + 00C9 = 000B+00C9 = 00D4*.

La posición del dato con este direccionamiento será pues *00D4*.

2.6. Una tabla de datos correspondiente a los alumnos matriculados en una determinada asignatura se coloca en la memoria de un computador a partir de una posición conocida. Supóngase que cada alumno se identifica por su número de matrícula *n*, y que la tabla contiene una ficha por cada número de matrícula. Cada ficha consta de *m* campos de datos de idéntico tamaño. Estudiar el modo de direccionamiento más adecuado para realizar aplicaciones informáticas en las que la totalidad de las fichas se traten de igual forma.

La figura siguiente muestra la estructura de la tabla y cómo se almacena en memoria. Se supone que cada campo de la ficha ocupa *k* palabras de memoria completas consecutivas.



La tabla se encontrará almacenada a partir de la posición de memoria *T*, y contendrá una entrada por cada alumno, y en total tantas entradas como alumnos se hallen matriculados. Los datos de cada alumno se almacenan en la posición de la tabla correspondiente a su número de matrícula. Es decir, los datos del alumno con número de matrícula *n* se almacenarán en la entrada *n* de la tabla.

Comenzando a contar a partir de la ficha 1, y dado que cada entrada ocupará $m*k$ palabras, los datos del alumno *n* se ubicarán a partir de la posición de memoria $T+(n-1)*m*k$.

Numerando los campos de cada ficha desde el 1, el campo *i* del alumno *n* se encuentra a partir de la dirección de memoria $T+(n-1)*m*k+(i-1)*k$, o sea, desde la $T+((n-1)*m+i-1)*k$.

En cuanto a los direccionamientos idóneos para aplicaciones de tratamiento de la tabla, una posibilidad consiste en efectuar un direccionamiento indexado con posincremento. El registro índice apuntaría inicialmente al comienzo de la tabla (posición *T* de memoria), y se iría incrementando a medida que se accediese a los distintos campos de datos de la ficha. El incremento debería coincidir con el tamaño del campo en palabras (*k*). Este direccionamiento exige un tratamiento secuencial de las fichas y los campos dentro de ellas: se comenzaría por la ficha correspondiente al primer alumno, y se terminaría por la del último.

Otra posibilidad consiste en utilizar un direccionamiento indirecto posindexado, con el campo de dirección del código de instrucción conteniendo la dirección de una palabra con la dirección de comienzo de la tabla (*T*). El registro índice debería actualizarse en cada acceso a los datos, de cualquiera de estas dos formas:

- Con posincremento: acceso secuencial a todas las fichas y campos de la tabla.
- Con cálculo del desplazamiento: requiere efectuar en cada acceso el cálculo de la posición relativa del campo que se va a procesar ($((n-1)*m+i-1)*k$) y la posterior asignación del valor obtenido al registro índice, permitiendo acceso directo a cualquier campo *i* de cualquier ficha *n* de la tabla.

El microprocesador M68000 dispone de un modo de direccionamiento que resulta idóneo para acceder a tablas: el direccionamiento relativo a registro base con índice. Es similar al direccionamiento posindexado, pero en vez de realizar una indirección para calcular la dirección base a la cual se suma el contenido del registro índice, permite mantener la dirección base en otro registro, evitando los accesos a memoria propios de la indirección. Además, se puede añadir un desplazamiento constante, que podría utilizarse para especificar uno cualquiera de los campos de la ficha.

La instrucción contiene campos que identifican tanto al registro base como al registro índice, más un campo adicional para el desplazamiento. El esquema del direccionamiento es el siguiente:

- Contenido del registro base: dirección de comienzo de la tabla (*T*).
- Contenido del registro índice: desplazamiento al comienzo de la ficha: $(n-1)*m*k$.
- Contenido del campo de desplazamiento: $(i-1)*k$ (constante en cada campo de la ficha).

El acceso al campo *i* de la ficha *n* se efectúa mediante la suma de los contenidos de los tres campos: $T+((n-1)*m+i-1)*k$.

2.7. Sea un computador con ancho de palabra de 16 bits, y con dos bancos de 8 registros de 16 bits. Supóngase que este computador posee un total de 60 instrucciones diferentes, y que direcciona un total de 64Kbytes de memoria. Para ello utiliza los siguientes modos de direccionamiento: inmediato, absoluto directo, absoluto indirecto, relativo a registro base. Diseñar un posible conjunto de formatos para codificar el juego de instrucciones de este computador, suponiendo que utiliza instrucciones con dos campos de direcciones.

El único modo de direccionamiento útil para conferir un valor a un registro es el direccionamiento mediante registro. Al no estar incluido este modo entre los del enunciado, y como el computador del ejercicio posee dos bancos de registros, se hace imprescindible incluir el direccionamiento mediante registro en los modos disponibles. Se resolverá pues el ejercicio atendiendo a cinco modos de direccionamiento: los cuatro del enunciado más el direccionamiento mediante registro.

El formato de instrucción debe constar de los siguientes campos:

- Código de operación (CO).
- Campo de dirección del primer operando (CD1).
- Campo de dirección del segundo operando (CD2).

Puesto que hay 60 instrucciones diferentes, se tomará un campo de código de operación de longitud fija de 6 bits. Por tanto, de los 10 bits que restan en la palabra, 5 serán para direccionar el primer operando, y otros 5 para direccionar el segundo. Por tanto, el campo de dirección para cada operando constará de 5 bits.

Cada campo de dirección debe incluir dos subcampos:

- El identificativo del modo de direccionamiento (MD).
- El dato en sí, que puede ser de varios tipos:
  - Campo de registro (CR), que en este caso consta de dos subcampos: registro (REG) y banco de registros (BR).
  - D: es un campo que puede contener tanto datos inmediatos como direcciones absolutas o desplazamientos relativos.

Como el mapa de memoria del computador es de 64K, las direcciones absolutas deben ser de 16 bits para poder direccionar cualquier byte de la misma.

Los desplazamientos en el direccionamiento relativo no tienen por qué permitir el direccionamiento de la totalidad de la memoria, puesto que constituyen una porción de la dirección, que se suma o concatena al contenido de un registro para obtener la dirección completa. Por tanto, las necesidades del campo de desplazamiento nunca son mayores que las que comporta una dirección absoluta.

Por otra parte, al ser el ancho de palabra de 16 bits, se supondrá que los datos inmediatos tienen un tamaño de 16 bits. En caso de querer dotar al computador de la posibilidad de emplear operandos de otras longitudes, la información referente al tamaño del dato deberá incluirse en el propio código de operación, y el dato inmediato consumirá entonces 8, 16 ó 32 bits (tamaños más usuales).

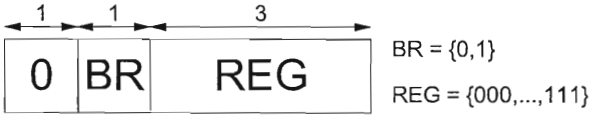
En resumen: el campo D tiene unas necesidades de almacenamiento que superan la capacidad de la palabra que contiene el código de operación. Por tanto, dicho campo se almacenará en palabras suplementarias (denominadas *palabras de ampliación*) ubicadas inmediatamente detrás de la palabra que contiene el código de operación.

El computador dispone de 5 modos de direccionamiento, lo que implica utilizar 3 bits para el subcampo MD de ser éste de tamaño fijo, con lo que en el campo de dirección quedarían disponibles tan sólo 2 bits para seleccionar el registro en los modos de direccionamiento mediante registro y en el relativo a registro base, cantidad del todo insuficiente para tal propósito. Por ello, es preferible contar con un subcampo MD de longitud variable, para de este modo optimizar al máximo el aprovechamiento de los bits disponibles en el campo de dirección.

Teniendo en cuenta las consideraciones recién expuestas, una posible solución para el ejercicio es la siguiente:

Direccionamiento mediante registro:

Este direccionamiento necesita 3 bits para referenciar al registro, más 1 bit para seleccionar el banco al que pertenece. En total hacen 4, y sólo sobraría 1 bit para el subcampo MD. Este bit irá en primer lugar dentro del campo de dirección.

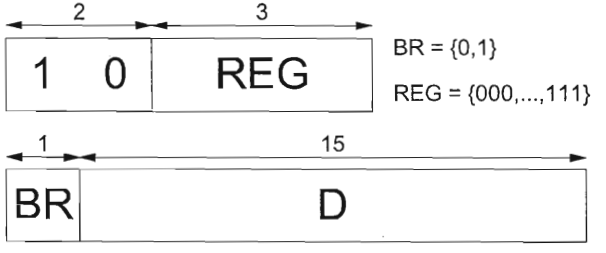


Según la figura, en operandos con direccionamiento mediante registro, MD = 0. El subcampo BR cuenta también con 1 bit, suficiente para seleccionar uno de los dos bancos de registros. Por su parte, el subcampo REG contiene tres bits, bastantes para seleccionar uno de los 8 registros dentro de un banco.

El subcampo MD para los restantes modos de direccionamiento deberá ser de 2 ó más bits, siendo el primero de ellos siempre un 1.

Direccionamiento relativo a registro base:

El desplazamiento irá incluido en una palabra de ampliación, mientras que son necesarios 4 bits para identificar al registro y al banco, con lo que sobraría un solo bit para indicar el modo de direccionamiento. Pero, según se ha indicado en el apartado dedicado al direccionamiento mediante registro, para tal efecto son necesarios al menos dos bits.



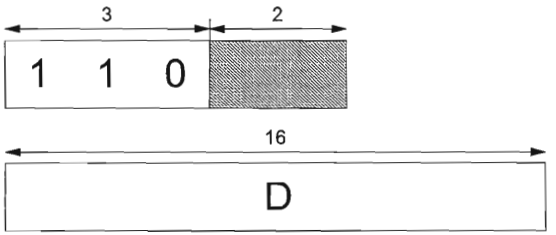
Entonces, para el direccionamiento relativo a registro base, MD=10. Los subcampos REG y BR siguen requiriendo 3 y 1 bits respectivamente, solo que ahora BR se encuentra en la palabra de ampliación.

Suponiendo que el desplazamiento se encuentre codificado en complemento a 2 para permitir desplazamientos positivos y negativos, con los 15 bits del campo D se pueden efectuar desplazamientos en el rango $[-2^{14},2^{14}-1]$ alrededor de la posición apuntada por el registro base.

Los restantes modos de direccionamiento deberán tener un subcampo MD de al menos 3 bits, y los dos primeros serán siempre 11.

Direccionamiento inmediato

El subcampo D, que contiene el dato inmediato, debe hallarse en una palabra de ampliación. Los 5 bits del registro de instrucción pueden utilizarse para designar el modo de direccionamiento, propósito para el que se emplearán tan sólo los 3 primeros. Los otros dos bits del campo de direcciones en el registro de instrucción quedan sin utilidad ninguna.

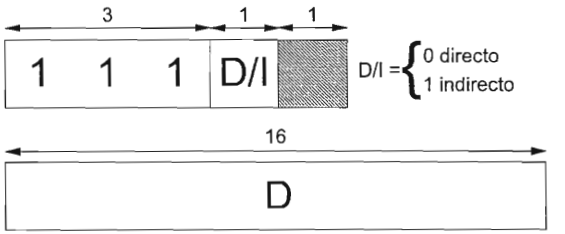


Así, MD=110 en operandos con direccionamiento inmediato, con un campo D que permite manejar datos de 16 bits.

Los restantes modos de direccionamiento deberán contener un subcampo MD de al menos 4 bits, donde los tres primeros siempre valdrán 111.

Direccionamiento absoluto directo o indirecto

La dirección en memoria ocupará una palabra de ampliación, permitiendo así direccionar la totalidad del mapa de memoria. Los 5 bits del registro de instrucción están disponibles para el subcampo MD, que ocupará los 3 primeros. El cuarto bit, que llamaremos D/I, servirá para distinguir entre direccionamiento directo o indirecto, y el quinto queda libre.



Por tanto, MD=111 identifica el direccionamiento absoluto, y D/I=0 significa direccionamiento directo, mientras que D/I=1 se emplea para direccionamiento indirecto. Las direcciones contenidas en el campo D, de 16 bits, permiten acceder a cualquier posición de la memoria.

3.1. Se tiene un número *N* en la memoria del M68000 que representa una codificación simplificada de otros dos números *C* y *P*. El número *N* es un entero comprendido entre *0* y *63*. El número *C* es un entero entre *0* y *3*. Por su parte, *P* es una máscara de 16 bits con sólo un bit puesto a *1* y el resto a *0*. La relación entre estos números es la siguiente:

Para $0 \leq N \leq 15$ $C = 0$ y *P* tiene a *1* el bit indicado por *N*.
Para $16 \leq N \leq 31$ $C = 1$ y *P* tiene a *1* el bit indicado por *N-16*.
Para $32 \leq N \leq 47$ $C = 2$ y *P* tiene a *1* el bit indicado por *N-32*.
Para $48 \leq N \leq 63$ $C = 3$ y *P* tiene a *1* el bit indicado por *N-48*.

Ejemplo: $N = 60 \Rightarrow C = 3$
$60 - 48 = 12 \Rightarrow$: bit *12* a *1* $\Rightarrow P = 0001000000000000$

Diseñar un programa en ensamblador del M68000 que calcule *C* y *P* a partir de *N* según la relación anterior. Seguir para ello el procedimiento indicado a continuación:

1) Especificar los datos de entrada y de salida mencionados en el enunciado, indicando los tamaños elegidos para los mismos.
2) Realizar una descripción textual del algoritmo propuesto.
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar el programa en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Datos de entrada y resultados del problema.

Datos de entrada

- El único dato de entrada es *N*, que es un número en memoria que se encuentra en el rango $0 \leq N \leq 63$. Por tanto, para almacenarlo será suficiente con un byte.

Resultados

- *C*: es un número en el rango $0 \leq C \leq 3$, con lo que ocupará 1 byte.
- *P*: es una máscara de 16 bits.

Tanto los datos de entrada como los resultados se almacenarán en la memoria principal del computador.

2) Descripción textual del algoritmo propuesto.

A la vista del enunciado, se puede deducir que:
- *C* es el cociente de la división entera *N*/16.
- *P* es una máscara con un único 1 representando el resto de la división *N*/16.

El procedimiento de cálculo de *C* y *P* a partir de *N* es sencillo. Basta con dividir el valor *N* por 16, obteniéndose el cociente y el resto de la división. El cociente se asignará a *C*, y el resto se utilizará para poner el correspondiente bit de *P* a 1.

Para ello se empleará la instrucción *DIVU* que, tomando un registro como dividendo y a la vez como destino, coloca el cociente en su parte baja y el resto en su parte alta. Ello obliga a copiar el valor del dividendo, inicialmente guardado en memoria, a un registro. Para extraer el resto se usará la instrucción *SWAP*, que intercambia las dos mitades del registro.

El resto servirá para poner a 1 un bit en *P* con la instrucción *BSET*. Esta instrucción tiene una limitación en el direccionamiento, consistente en que si el operando destino es una palabra de memoria, su tamaño es forzosamente 1 byte. Como *P* tiene 16 bits, para realizar la operación es preciso interponer un registro auxiliar.

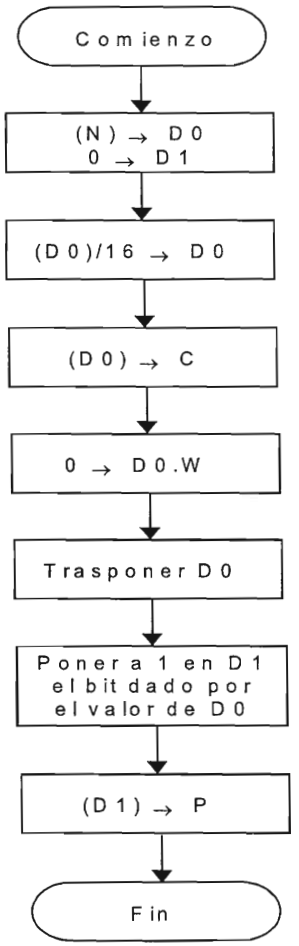
3) Descripción por pasos del algoritmo propuesto.

El programa debe constar de los siguientes pasos:

1. Inicializar datos: $(N) \rightarrow D0$, $0 \rightarrow D1$.
2. Dividir *D0* por 16: $(D0) / 16 \rightarrow D0$. Con esto queda el cociente en la palabra menos significativa de *D0*, y el resto en la palabra alta.
3. Almacenar el cociente en memoria: $(D0.B) \rightarrow C$ (sólo los 8 bits menos significativos).
4. Poner los 16 bits de la parte baja del registro *D0* a 0: $0 \rightarrow D0.W$.
5. Intercambiar las dos mitades del registro *D0*.
6. Colocar a 1 el bit correspondiente de *D1* según el contenido de *D0*.
7. Almacenar *D1* en memoria: $(D1) \rightarrow P$.
8. Fin del programa

En el algoritmo se han utilizado sendas variables intermedias en los registros *D0* y *D1*. Esto es debido a las limitaciones en los modos de direccionamiento permitidos en las instrucciones utilizadas.

4) Diagrama de flujo de la solución propuesta.



5) Codificación en ensamblador.

```
PROGRAMA  ORG      $1000     ; Origen de la zona de código en posición $1000
                             ; Paso 1: inicializar datos
          CLR.L    D0        ;   Poner D0 a 0
          MOVE.B   N,D0      ;   Cargar D0 con el valor de N
          MOVEQ    #0,D1     ;   Inicializar D1
          DIVU.W   #16,D0    ; Paso 2: dividir D0 entre 16
          MOVE.B   D0,C      ; Paso 3: almacenar el cociente en memoria
          MOVE.W   #0,D0     ; Paso 4: poner la parte baja de D0 a 0
          SWAP     D0        ; Paso 5: transponer registro
          BSET     D0,D1     ; Paso 6: poner el bit del resto a 1 en reg. auxiliar
          MOVE.W   D1,P      ; Paso 7: almacenar el reg. auxiliar en memoria
          STOP     #$2700    ; Paso 8: fin del programa

DATOS     ORG      $1200     ; Origen de la zona de datos en posición $1200
N         DC.B     55        ; Dato de entrada N (valor inicial ejemplo: 55)
C         DS.B     1         ; Dato de salida C
P         DS.W     1         ; Dato de salida P
          END
```

Comentarios adicionales:

- Al no especificar el enunciado nada al respecto, se ha colocado la zona de código desde la posición de memoria 1000 (hex.) y los datos en la posición 1200 (hex.).

- La reserva de datos en memoria se efectúa mediante pseudoinstrucciones *DS.S* y *DC.S*. Se empleará *DC.S* cuando se desee inicializar la variable con un valor concreto (por ejemplo, el caso de la variable *N*). La pseudoinstrucción *DS.S* sólo se ocupa de reservar una serie de posiciones de memoria, desentendiéndose de su contenido.

- En ninguna situación en el programa se ha incluido la pseudoinstrucción *EVEN*, que provoca que el siguiente dato o instrucción del programa comience en una posición de memoria con dirección par. Esto es porque que existen programas ensambladores en el mercado que alinean los códigos de instrucción y los datos en memoria de forma automática. Por ello, no se ha considerado necesario incluir la pseudoinstrucción *EVEN* en la solución de estos ejercicios, aunque en cualquier caso el alumno deberá comprobar la conveniencia de usarla en los manuales del programa ensamblador con el que trabaje.

- Se han incluido varias maneras de poner un registro o parte de él a 0: *CLR.L D0*, *MOVEQ #0,D0* y *MOVE.W #0,D0*. Las dos primeras son equivalentes en tanto en cuanto provocan los mismos efectos, que es poner el registro completo todo a 0. La instrucción *MOVE.W #0,D0* sólo pone a 0 la palabra baja de *D0* dejando la parte alta inalterada, con lo cual sería equivalente a *CLR.W D0*, si bien esta última opción resultaría más rápida y ocuparía menos espacio en memoria. Si el valor que se pretendiese almacenar en la palabra inferior de *D0* fuese distinto de 0, deberíamos recurrir a la instrucción *MOVE.W*.

- El programa finaliza con la instrucción de parada *STOP #$2700*, que activa el modo supervisor y hace *CPL=7*, es decir, inhabilita todo tipo de interrupciones excepto la interrupción no enmascarable l_7.

6) Modos de direccionamiento utilizados.

Puesto que este programa manipula datos que inicialmente se hallan almacenados o se deben grabar en memoria, lo más lógico sería que el modo de direccionamiento más importante en él fuese el direccionamiento absoluto. Sin embargo, las limitaciones del juego de instrucciones del M68000 en cuanto a los operandos permitidos en las distintas instrucciones aconsejan la utilización del direccionamiento a registro en varios puntos del programa. Por ello, el dato de entrada *N*, residente inicialmente en memoria, se copia en un registro de datos, y a partir de ahí la mayor parte de las operaciones se efectúan sobre registros auxiliares, que se copian en memoria para grabar los resultados finales.

Además de los mencionados, también se utiliza el modo de direccionamiento inmediato en las puestas a 0 de los operandos y en la división del dato inicial por 16.

3.2. Realizar un programa en ensamblador del M68000 que obtenga los *N* primeros términos de la serie de Fibonacci y los almacene en *N* posiciones de memoria consecutivas. La regla de formación es:

$$f(0) = 1$$
$$f(1) = 1$$
$$f(i) = f(i-1) + f(i-2)$$

Seguir para ello el procedimiento indicado a continuación:

1) Especificar los datos de entrada y de salida mencionados en el enunciado, indicando los tamaños elegidos para los mismos.
2) Realizar una descripción textual del algoritmo propuesto.
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar el programa en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Datos de entrada y resultados del problema.

Datos de entrada

- Sólo existe un único dato de entrada, que es *N*, el número de términos de la serie de Fibonacci que se pretende calcular. En el enunciado no se especifica el tamaño de *N*, pero podremos suponer que es de un byte (8 bits) y que se encuentra almacenado en la memoria del computador codificado en binario sin signo.

Resultados

- El resultado producido por el programa será una tabla almacenada en un conjunto de posiciones consecutivas de memoria en las que se irán grabando los términos pedidos. Como en el enunciado no se indica nada al respecto, podemos asumir, por ejemplo, que cada término ocupará una palabra de memoria (16 bits). La tabla se almacenará a partir de una posición de memoria etiquetada como *TABLA*.

2) Descripción textual del algoritmo propuesto.

El programa almacenará los dos primeros términos de la serie y entrará en un bucle que irá calculando los sucesivos términos, según la fórmula de recurrencia indicada en el enunciado, y guardándolos en la tabla, hasta llegar al término de índice *N*-1 (el primer término tiene índice 0). El bucle estará controlado por un contador que almacenaremos en un registro de datos de la máquina, por ejemplo, *D0*. Al inicio de cada pasada por el bucle, *D0* contendrá el número de términos calculados, y al final de cada pasada será incrementado. La comprobación de finalización del bucle se efectuará al final del mismo (igual que en una estructura REPEAT ... UNTIL), de modo que el bucle terminará cuando *D0* sea igual a *N*. Almacenaremos los términos calculados en la tabla mediante direccionamiento relativo a registro de direcciones con posincremento, y para acceder a los términos anteriores de la fórmula de recurrencia utilizaremos direccionamiento relativo a registro con desplazamiento.

3) Descripción por pasos del algoritmo propuesto.

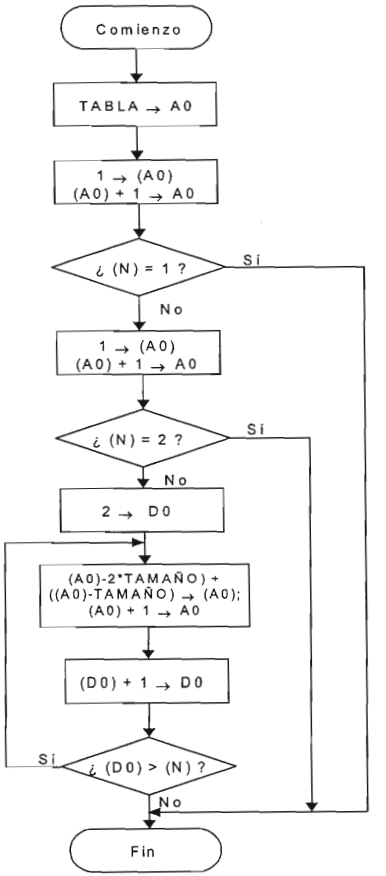
El programa debe constar de los siguientes pasos:

1. Inicializar datos: *TABLA* → *A0*.
2. Calcular el primer término de la tabla: 1 → *(A0)* ; *(A0)* + 1 → *A0*.
3. Si *(N)* = 1, saltar al paso 10.
4. Calcular el segundo término de la tabla: 1 → *(A0)* ; *(A0)* + 1 → *A0*.
5. Si *(N)* = 2, saltar al paso 10.
6. Inicializar el contador del bucle: 2 → *D0*.
7. Calcular el siguiente término de la serie mediante la fórmula de recurrencia: *((A0)* - 2**TAMANO)* + *((A0)* - *TAMANO)* → *(A0)*; *(A0)* + 1 → *A0*.
8. Incrementar el contador: *(D0)* + 1 → *D0*.
9. Si *(D0)* < *(N)*, ir al paso 7 porque no se han calculado todos los términos pedidos.
10. Ya se han calculado *N* términos: fin del programa.

En esta descripción se han incorporado dos nuevos elementos al programa:

- *A0*: es el registro de direcciones utilizado en la manipulación de la tabla.
- *TAMANO*: es una constante que indica el tamaño en bytes de cada elemento de la tabla, necesario para calcular los desplazamientos en los accesos a los elementos anteriores de la tabla citados en la fórmula de recurrencia.

4) Diagrama de flujo.



5) Codificación en ensamblador.

La codificación en lenguaje ensamblador será:

```
PROGRAMA  ORG       $1000          ; Comienzo de código en posición $1000
          MOVEA.L   #TABLA,A0      ; Paso 1: A0 apunta al inicio de la tabla
          MOVE.W    #1,(A0)+       ; Paso 2: calculo del termino 1 de la serie
          CMPI.B    #1,N           ; Paso 3: comprobar si N=1
          BEQ       FIN
          MOVE.W    #1,(A0)+       ; Paso 4: cálculo del termino 2 de la serie
          CMPI.B    #2,N           ; Paso 5: ver si N=2
          BEQ       FIN
          MOVE.B    #2,D0          ; Paso 6: carga del contador
BUCLE     MOVE.W    -2*TAMANO(A0),D1 ; Paso 7: calcular el termino como
          ADD.W     -1*TAMANO(A0),D1 ;   suma de los dos anteriores,
          MOVE.W    D1,(A0)+       ;   almacenarlo en (A0) e incrementar A0
          ADDQ.B    #1,D0          ; Paso 8: incremento del contador
          CMP.B     N,D0           ; Paso 9: comprobar si se han calculado
          BCS       BUCLE          ;   todos los términos solicitados
FIN       STOP      #$2700         ; Paso 10: parar
DATOS     ORG       $1200          ; Comienzo de datos en posición $1200
TAMANO    EQU       2              ; Tamaño en bytes de un termino
N         DS.B      1              ; Longitud de la serie (sin dar valor inicial)
TABLA     DS.W      255            ; Reserva para grabar la serie
                                   ; (máximo tamaño: 255 palabras)
          END
```

En la codificación ha sido necesario considerar una variable intermedia adicional en el registro *D1*, utilizada para realizar la suma indicada en la fórmula de recurrencia.

Comentarios adicionales:

- Se reserva espacio para almacenar los términos de la serie consecutivamente a partir de la dirección *TABLA* con una pseudoinstrucción *DS.S*. La variable *N* ocupa un byte, y se deja sin inicializar. El tamaño máximo reservado para la tabla corresponde con el número máximo de términos que se pueden solicitar mediante la variable *N*. Los datos de la tabla son palabras de 16 bits, por lo que cada elemento de la misma deberá obligatoriamente almacenarse a partir de una posición par de memoria. Como *N*, que se encuentra en la posición 1200 (hex.), ocupa un único byte, sería preciso incluir delante de la reserva de la tabla una pseudoinstrucción de alineamiento de tipo *EVEN* en los casos en que el ensamblador (traductor) empleado no realice dicha alineación por sí mismo.
- La carga de la dirección inicial de la tabla en *A0* se efectúa con una instrucción del tipo *MOVEA*, como corresponde a un registro de direcciones. Esto podría haberse hecho también mediante la instrucción *LEA TABLA,A0*.
- El almacenamiento de cada término de la serie se produce a través de una instrucción *MOVE*, cuyo operando destino viene referenciado mediante un direccionamiento relativo con posincremento sobre el registro *A0*, que actúa como puntero a la tabla. Así, los términos se almacenan consecutivamente en memoria en orden creciente. En este direccionamiento, el microprocesador M68000 realiza un escalado según el tamaño de los datos, de modo que el posincremento del registro es proporcional al tamaño del dato referenciado en la instrucción. Así, antes de almacenar el elemento 0 de la tabla, A0 apunta a la dirección *TABLA*; tras ejecutarse la instrucción *MOVE.W #1,(A0)+*, el registro *A0* apuntará a la dirección *TABLA+2*, al emplearse en la instrucción datos de tamaño palabra, y así sucederá en las demás pasadas por el bucle. Igual sucede con el direccionamiento relativo a registro con predecremento: éste depende del tamaño del dato direccionado.

- El acceso a los términos de la fórmula de recurrencia para el cálculo del nuevo término de la serie se realiza mediante direccionamiento relativo al registro de direcciones *A0* con desplazamiento negativo de 1 ó 2 elementos. También podría utilizarse direccionamiento relativo al registro *A0* con índice, empleando como índice el registro *D0* y dejando el contenido de *A0* inalterado a lo largo del programa a modo de registro base.

Al respecto de estos dos tipos de direccionamiento, es preciso hacer una aclaración: en el apartado 13.5 (pág. 397) del texto base de la asignatura, se indica que en el direccionamiento relativo a registro con desplazamiento se aplican desplazamientos independientes del tamaño de los datos manipulados: es decir, que en nuestro programa, la instrucción *MOVE.W -2(A0),D1* copiaría el contenido del elemento *i*-1 de la tabla en el registro de datos *D1* (suponiendo que *A0* apunta al elemento *i*). Dado que el desplazamiento se expresa con el número -2 y parecería más intuitivo que en realidad se apuntara al elemento *i*-2, pero para ello los desplazamientos aplicados deberían ser proporcionales al (y por tanto dependientes del) tamaño de los datos. Este modo de direccionamiento con escalado del desplazamiento está disponible en algunos microprocesadores de la familia 68000, pero no en el M68000. En él no se realiza escalado alguno, es decir, con la instrucción *MOVE.W -2(A0),D1* se accede a la posición marcada por *A0* menos dos, o sea, al elemento *i*-1 de nuestra tabla. Por tanto, es preciso multiplicar el número de elementos del desplazamiento por el tamaño en bytes de cada uno de ellos para así poder alcanzar el elemento deseado: *MOVE.W -2*TAMANO(A0),D1*, donde *TAMANO* es una constante. Esta multiplicación se realiza en tiempo de ensamblado, es decir, es una expresión calculada por el programa traductor, de modo que en el código objeto figura el desplazamiento real, y no se realiza producto alguno en tiempo de ejecución.

Lo mismo sucede con el direccionamiento relativo a registro con índice: ni el desplazamiento ni el índice se escalan en función del tamaño del dato accedido.

Concretando:

- El direccionamiento relativo a registro con desplazamiento sirve para acceder al dato que se encuentra en una posición calculada como la suma del contenido de los 32 bits de un registro de direcciones con un desplazamiento de 16 bits dado en complemento a 2. En tiempo de ejecución se efectúa una extensión de signo a 32 bits sobre el desplazamiento para hacer la suma. El contenido del registro de direcciones queda intacto. Ejemplos de este modo de direccionamiento son:

MOVE.B	*4(A1),D0*	*((A1)+4) → D0*
MOVE.W	*-9(A0),D1*	*((A0) - 9) → D1*
MOVE.L	*N(A0),D2*	*((A0)+N) → D2* (*N*: cte.)

- En el direccionamiento relativo a registro base con índice, la dirección del dato se calcula mediante la suma del contenido de un registro de direcciones que actúa como base, un registro de datos o de direcciones que actúa como índice, y un desplazamiento. Se puede usar el registro índice en modo palabra o en modo palabra larga. El desplazamiento es un número de 8 bits dado en complemento a 2. En tiempo de ejecución se efectúa una extensión de signo a 32 bits del desplazamiento, y también del registro índice si se toma en modo palabra. El contenido de los registros base e índice queda intacto. Ejemplos:

MOVE.B	*4(A1,A0),D0*	*((A1)+(A0.L)+4) → D0*
MOVE.W	*-9(A0,A2.W),D1*	*((A0)+(A2.W) - 9) → D1*
MOVE.L	*N(A0,D0.L),D2*	*((A0) + (D0.L)+N) → D2*(*N*: cte.)

En la figura 13.6 del texto base (pág. 403) se incluyen las palabras de ampliación asociadas a estos tipos de direccionamiento. La que corresponde con un direccionamiento relativo a registro con desplazamiento es la que en la figura aparece como palabra de ampliación de tipo 1, mientras que el

direccionamiento relativo a registro base con índice conlleva la utilización de una palabra de ampliación de tipo 2.

- La condición de finalización se determina mediante una comparación explícita del contador *D0* con la variable *N*, seguida de un salto condicional sobre el resultado obtenido. Hay que hacer notar que, cuando el computador ejecuta una operación de comparación, considera que los números tienen signo, y vienen dados en complemento a 2. Por tanto, si los números comparados son números sin signo en binario natural, la condición que hay que especificar en la instrucción *Bcc* subsiguiente será distinta que si los números son con signo. A continuación se adjunta una tabla en la que se indican las instrucciones que hay que utilizar en cada caso:

Comparación realizada	Números con signo	Números sin signo
≥	BGE, BPL	BCC
>	BGT	BHI
=	BEQ	BEQ
<>	BNE	BNE
≤	BLE	BLS
<	BLT,BMI	BCS

Como en este ejercicio se considera que *N* es un número sin signo, se utiliza la instrucción *BCS* para comprobar si se han calculado todos los términos pedidos.

6) Modos de direccionamiento utilizados.

La estrategia de la solución planteada en este ejercicio se basa en mantener apuntada en todo momento por el registro *A0* la posición de memoria del siguiente término de la serie que se quiere calcular. Esto se logra a través del direccionamiento relativo a registro con posincremento sobre *A0*. Con ello, el acceso a los términos que aparecen en la fórmula de recurrencia puede efectuarse mediante un direccionamiento relativo al registro *A0* con desplazamiento.

No se precisa mantener a lo largo del programa el registro *A0* apuntando a la dirección inicial de la tabla, ya que en todo momento es posible acceder a ella mediante direccionamiento absoluto a la etiqueta *TABLA*. En el caso de que fuese necesario conservar el registro *A0* apuntando al inicio de la tabla, habría sido preciso mantener una copia del mismo en otro registro, por ejemplo *A1*, y sobre éste se habría aplicado el direccionamiento relativo con posincremento. Otra alternativa habría sido utilizar direccionamiento relativo con índice.

Otros modos de direccionamiento empleados en el programa son:
- Absoluto: en las referencias a *N* y a *TABLA*.
- Mediante registro de datos: en las referencias a *D0* y *D1*.
- Mediante registro de direcciones: al cargar el valor inicial en *A0*.
- Inmediato.

3.3. Diseñar un programa en ensamblador del M68000 para calcular el término *T(n)* de la sucesión definida por:

$$T(0) = 1$$
$$T(i) = 2*T(i-1) \quad \text{si } i \text{ es impar}$$
$$T(i) = 2*T(i-1) - 1 \quad \text{si } i \text{ es par}$$

Seguir para ello el procedimiento indicado a continuación:

1) Especificar los datos de entrada y de salida mencionados en el enunciado, indicando los tamaños elegidos para los mismos.
2) Realizar una descripción textual del algoritmo propuesto.
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar el programa en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Datos de entrada y resultados del problema.

Datos de entrada

- Nuevamente nos encontramos con un único dato de entrada: *N*, el índice del término que se desea calcular. Nuevamente supondremos que ocupa un byte (8 bits) en memoria y que está codificado en binario sin signo.

Resultados

- Al igual que en el ejercicio 3.2, nos encontramos ante una sucesión definida mediante una fórmula de recurrencia. Sin embargo, en este caso no debemos guardar en memoria el valor de la sucesión completa, sino el de un solo termino. Supondremos que se guardará en una posición de memoria etiquetada como *TERMINO*, y que ocupará una palabra (16 bits).

2) Descripción textual del algoritmo propuesto.

El programa constará de un bucle de cálculo de los sucesivos elementos de la serie, y almacenará el último elemento calculado en un registro, por ejemplo *D1*. El bucle tendrá estructura de bucle FOR desde 1 hasta el contenido de la variable *N* y estará controlado por un contador almacenado en un registro de datos, que por ejemplo puede ser el *D0*. Este contador se inicializa antes de entrar en el bucle, y se incrementa al final de cada pasada. El bucle terminará, pues, cuando el contenido de *D0* sea mayor que el de *N*.

Como la fórmula de cálculo de los sucesivos elementos de la serie es diferente dependiendo de si el índice del elemento es par o impar, dentro del bucle será preciso establecer un tratamiento diferencial sobre los distintos términos de acuerdo con dicha condición.

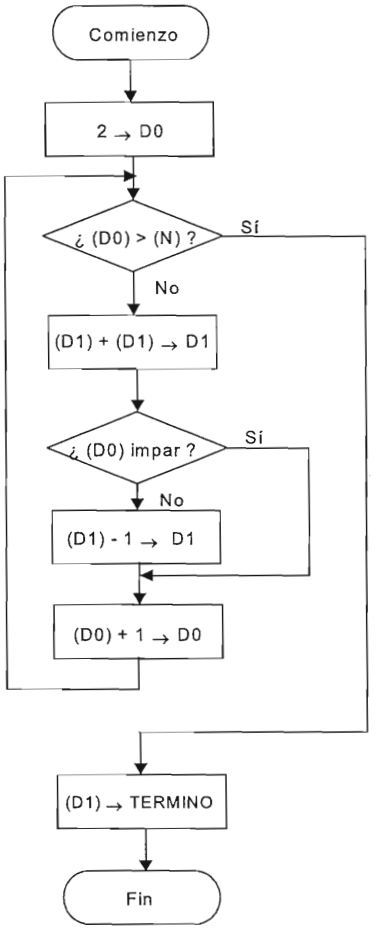
Para saber si un número codificado en binario natural sin signo es positivo o negativo, es suficiente con inspeccionar el bit de menor peso: si es 1, el número es impar, y si es 0, el número será par. El resultado de tal comprobación sobre el índice servirá para aplicar el tratamiento adecuado al término en curso.

3) Descripción por pasos del algoritmo propuesto.

El programa constará de los siguientes pasos:

1. Calcular el término con índice 0: $1 \rightarrow D1$.
2. Inicializar el contador del bucle: $1 \rightarrow D0$.
3. Si $(D0) > (N)$, saltar al paso 9.
4. Multiplicar el término anterior por 2: $(D1) + (D1) \rightarrow D1$.
5. Si *D0* es impar, saltar al paso 7.
6. *D0* es par: restar 1 al registro *D1*.
7. Incrementar el contador del bucle: $(D0) + 1 \rightarrow D0$.
8. Saltar al paso 3.
9. Grabar el término pedido en memoria: $(D1) \rightarrow TERMINO$.
10. Fin del programa.

4) Diagrama de flujo.



5) Codificación en ensamblador.

```
PROGRAMA    ORG       $1000       ; Comienzo de código en posición $1000
            MOVE.W    #1,D1       ; Paso 1: cálculo del primer término
            MOVE.B    #1,D0       ; Paso 2: inicializar D0 (contador)
BUCLE       CMP.B     N,D0        ; Paso 3: comprobar D0
            BHI       FINBUCLE    ;   Salir del bucle si (D0) > (N)
            ADD.W     D1,D1       ; Paso 4: multiplicar el término por 2
            BTST      #0,D0       ; Paso 5: si el último bit de D0 es 1 (índice impar)
            BNE       INCD0       ;   saltar hasta el paso de incremento
            SUBQ.W    #1,D1       ; Paso 6: restar 1 al término (si índice par)
INCD0       ADDQ.B    #1,D0       ; Paso 7: incrementar D0
            BRA       BUCLE       ; Paso 8: volver al principio del bucle
FINBUCLE    MOVE.W    D1,TERMINO  ; Paso 9: almacenar el término en memoria
            STOP      #$2700      ; Paso 10: parar la ejecución del programa

DATOS       ORG       $1200       ; Comienzo de datos en posición $1200
TERMINO     DS.W      1           ; Posición de memoria del término calculado
N           DS.B      1           ; Indice del término que se desea calcular
            END
```

Comentarios adicionales:

- El bucle tiene la estructura de un bucle FOR: se inicia el contador al principio, se compara si está dentro de los límites (entre 1 y *N*), se ejecutan las sentencias del bucle y finalmente se incrementa el contador y se vuelve al principio para comparar.

- El cálculo del término siguiente se efectúa a través de una suma y no de un producto porque la instrucción de multiplicación, además de ser más lenta, produce resultados de 32 bits.

- Para comprobar si el índice asociado al término en proceso es par o impar se utiliza la instrucción *BTST #0,D0*, que pone el flag *Z* a 1 si el bit de menor peso de *D0* es 0, y a 0 si el bit vale 1. El salto condicional *BNE INCD0* tiene lugar si el flag *Z* es 0, o sea, si el bit examinado es 1.

6) Modos de direccionamiento utilizados.

En este ejercicio sucede lo mismo que en el 3.1: el término calculado ha de almacenarse en memoria, con lo cual parecería lógico que la solución propuesta tuviese mayoría de accesos a memoria mediante direccionamiento absoluto. Sin embargo, por la misma razón que se esgrime en el ejercicio 3.1, se ha decidido emplear un registro de datos para calcular los sucesivos términos de la serie, y limitar los accesos a memoria a las comparaciones con *N* y a la grabación del resultado final. Además existen algunos direccionamientos inmediatos.

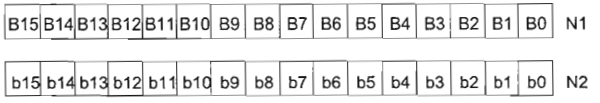
3.4. Razonar un método para obtener a partir de un número binario de 16 bits, otro que tenga los mismos bits pero en orden inverso (valor binario invertido). Por ejemplo, los dos números siguientes tienen sus bits invertidos:

0011011001110010 y *0100111001101100*

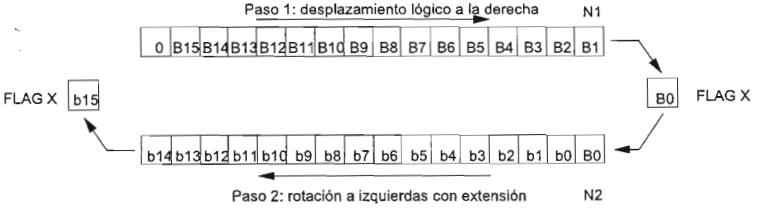
Diseñar un programa en ensamblador del M68000 que obtenga el valor binario invertido de un número dado según el método propuesto. Seguir para ello el procedimiento indicado a continuación:

1) Especificar los datos de entrada y de salida mencionados en el enunciado, indicando los tamaños elegidos para los mismos.
2) Realizar una descripción textual del algoritmo propuesto.
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar el programa en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

Supongamos que el número binario del que se parte como dato es *N1*, y el que se desea obtener como resultado es *N2*.



El proceso de inversión de *N1* ha de realizarse bit a bit. Una forma es efectuar un desplazamiento lógico elemental a la derecha sobre *N1* sacando el bit del extremo al flag *X*, e introducir este bit en *N2* mediante una rotación elemental a la izquierda.



Tras repetir este proceso 16 veces, *N2* contendrá exactamente los mismos bits que *N1* contenía inicialmente, pero en orden inverso.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	N1

B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	N2

Hay que hacer notar que esta operación destruye el operando *N1*, siendo conveniente utilizar entonces una variable intermedia, inicializada previamente con el contenido de *N1*.

1) Datos de entrada y resultados del problema.

Datos de entrada

- Existe un único dato de entrada, que es el número de partida que hemos llamado *N1*. Se encontrará almacenado en memoria en una palabra de 16 bits.

Resultados

- El resultado se grabará en el número *N2*, almacenado en una palabra de 16 bits en memoria.

2) Descripción textual del algoritmo propuesto.

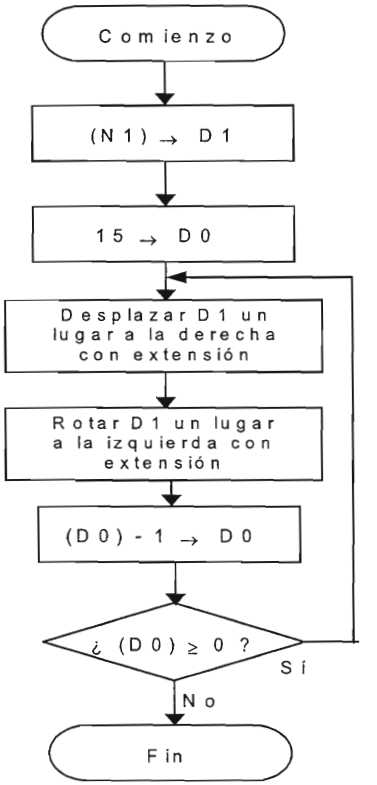
El algoritmo utilizado se basa en el método de inversión expuesto anteriormente: un bucle en el que se efectúan un desplazamiento elemental a derechas del número inicial seguido de una rotación elemental a la izquierda de *N2* con extensión al flag *X*. El número inicial se almacenará en una variable intermedia, por ejemplo el registro *D1*. El bucle se ejecutará 16 veces, y tendrá la estructura de un REPEAT ... UNTIL regido por un contador decreciente que ira desde 15 hasta 0, almacenado en el registro *D0*.

3) Descripción por pasos del algoritmo propuesto.

Esquemáticamente, el programa deberá actuar de acuerdo con el siguiente guión:

1. Copiar *N1* en una variable intermedia: $(N1) \rightarrow D1$.
2. Inicializar el contador del bucle: $15 \rightarrow D0$.
3. Hacer sobre *D1* un desplazamiento lógico a derechas afectando al flag *X*.
4. Hacer sobre *N2* una rotación a la izquierda con extensión, introduciendo por la derecha el flag *X*.
5. Decrementar el contador: $(D0) - 1 \rightarrow D0$.
6. Si $(D0) \geq 0$ saltar a 3.
7. Fin del programa.

4) Diagrama de flujo de la solución propuesta.



5) Codificación en ensamblador.

```
PROGRAMA    ORG       $1000       ; Comienzo de código en posición $1000
            MOVE.W    N1,D1       ; Paso 1: copiar el dato en un registro auxiliar
            MOVE.W    #15,D0      ; Paso 2: inicializar el contador
BUCLE       LSR.W     #1,D1       ; Paso 3: desplazar D1 y sacar un bit al flag X
            ROXL      N2          ; Paso 4: introducir el flag X en N2
                                  ; por el lado contrario
            DBF       D0,BUCLE    ; Pasos 5 y 6: decrementar el contador y
                                  ; saltar a bucle si es mayor que -1
            STOP      #$2700      ; Parar la ejecución del programa

DATOS       ORG       $1200       ; Comienzo de datos en posición $1200
N1          DS.W      1           ; Dato que se desea invertir
N2          DS.W      1           ; Dato invertido
            END
```

Comentarios adicionales:

- *N1* se reserva sin valor inicial, al igual que *N2*.
- La instrucción *ROXL N2* cuenta sólo con un operando, pues en el M68000 los desplazamientos y rotaciones sobre variables en memoria son por fuerza de longitud 1 y longitud palabra.
- El bucle se controla con *DBcc Di,dir*, instrucción que consta de tres fases:

  1. Se analiza la condición indicada de acuerdo con el contenido del registro de estado. Si la condición se cumple, finaliza la ejecución de la instrucción y pasa a ejecutarse la siguiente en la secuencia del programa. Si la condición no se cumple
  2. se decrementa el registro *Di* en modo palabra. Si el contenido de *Di.W* es igual a -1, finaliza la ejecución de la instrucción y pasa a ejecutarse la siguiente en la secuencia del programa. Si el contenido de *Di* es distinto de -1
  3. se produce un salto a la instrucción indicada por la dirección *dir*.

  Esta instrucción forma un bucle que se repite mientras la ejecución de *DBcc* llegue al punto 3), o sea, mientras no se cumpla la condición *cc* y el registro *Di* contenga un valor distinto de -1. Como en nuestro caso la condición especificada es *F* (*false*: falso), nunca se cumple, y el bucle únicamente acaba cuando *D0* vale -1. Los ensambladores admiten también el nemotécnico *DBRA*, equivalente a *DBF*. En el extremo contrario se encontraría la instrucción *DBT*, en la que la condición es *T* (*true*: verdad), y no forma bucle, pues la condición se cumple siempre.

6) Modos de direccionamiento utilizados.

En este programa se emplean principalmente direccionamiento absoluto para referirse a las variables *N1* y *N2*, direccionamiento mediante registro de datos para acceder a la variable intermedia y al contador, y direccionamiento inmediato.

3.5. Diseñar una subrutina para el M68000 que calcule, a partir de un número de 8 bits pasado en el registro *D0*, otro valor de 8 bits que se devolverá en el mismo registro *D0*. El nuevo valor se obtiene de la siguiente forma:

a) Realizar la función XOR de los bits *0*, *1*, *6* y *7* del número inicial.
b) Complementar el bit obtenido en a)
c) Desplazar a la derecha el número inicial, introduciendo por la izquierda el bit obtenido en b). El valor obtenido es el resultado.

Seguir el procedimiento indicado a continuación:

1) Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Datos de entrada y resultados de la subrutina.

La subrutina sólo dispone de un parámetro, el registro *D0*, que es tanto de entrada como de salida. De él sólo se emplearán los ocho bits menos significativos.

2) Descripción textual del algoritmo propuesto.

La subrutina constituye realmente un generador de secuencias de números pseudoaleatorios que funciona del siguiente modo: en primer lugar, se efectúa una llamada a la subrutina con un argumento cualquiera (contenido en *D0*), y posteriormente se realizan sucesivas llamadas a la misma introduciendo en cada caso el parámetro devuelto en la llamada anterior. La secuencia de números obtenidos en *D0* tras cada invocación a la subrutina constituye la secuencia de números pseudoaleatorios.

La subrutina cuenta pues con un único argumento, almacenado en el registro *D0*, que es a la vez la información externa recibida por la subrutina y el resultado que devuelve al programa llamante.

Por conveniencia, en este ejercicio se utilizarán las propiedades asociativa y conmutativa del Álgebra de Boole, que se cumplen para la operación lógica XOR: en vez de hacer $b_{0,1,6,7} = b_0 \oplus b_1 \oplus b_6 \oplus b_7$, se efectuarán las siguientes operaciones:

$$\left.\begin{aligned} b_{0,6} &= b_0 \oplus b_6 \\ b_{1,7} &= b_1 \oplus b_7 \end{aligned}\right\} b_{0,1,6,7} = b_{0,6} \oplus b_{1,7}$$

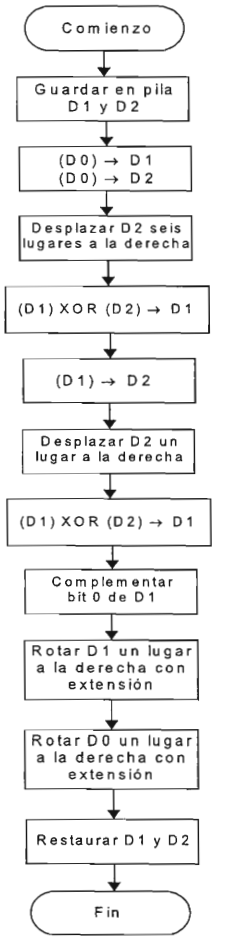
La realización de estas operaciones conlleva la utilización de variables temporales internas a la subrutina. Estas variables se almacenarán en registros de propósito general, cuyo contenido previo debe ser salvaguardado en memoria de pila y restaurado al final de la subrutina.

3) Descripción por pasos del algoritmo propuesto.

Las acciones efectuadas por la subrutina son las siguientes:

1. Guardar en la pila los registros *D1* y *D2*, elegidos para variables temporales: (*SP*) - 8 → (*SP*); (*D1*)/(*D2*) → (*SP*).
2. Inicializar las variables temporales: (*D0*) → *D1*; (*D0*) → *D2*.
3. Hacer un desplazamiento lógico de *D2* 6 lugares a la derecha para alinear los bits b_0 y b_1 de *D1* con los bits b_6 y b_7 de *D2*.
4. Realizar la operación de OR exclusivo de *D1* con *D2*, y almacenar el resultado en *D1*: (*D1*) *XOR* (*D2*) → *D1*. De este modo, los bits 0 y 1 de *D1* son respectivamente $b_{0,6}$ y $b_{1,7}$.
5. Copiar *D1* en *D2*: (*D1*) → *D2*.
6. Desplazar *D2* un lugar a la derecha para alinear $b_{0,6}$ de *D1* con $b_{1,7}$ de *D2*.
7. Hacer el OR exclusivo de *D1* y *D2* y guardar el resultado en *D1*: (*D1*) XOR (*D2*) → *D1*. De este modo, el bit 0 de *D1* es ahora $b_{0,1,6,7}$.
8. Negar el bit 0 de *D1*.
9. Poner dicho bit en el flag *X* del registro de códigos de condición mediante una rotación de *D1* a la derecha con extensión.
10. Introducir el bit en *D0* mediante una rotación a la derecha con extensión.
11. Restaurar los registros almacenados en pila: ((*SP*)) → *D0*/*D1*; (*SP*) + 8 → *SP*.
12. Retornar de la subrutina.

4) Diagrama de flujo de la solución propuesta.



5) Codificación en ensamblador.

```
SUBRUT   ORG       $1000          ; Origen de código en posición $1000
         MOVEM.L   D1/D2,-(SP)    ; Paso 1: salvaguarda de registros en la pila
         MOVE.B    D0,D1          ; Paso 2: Carga de los registros auxiliares
         MOVE.B    D0,D2          ;   con el valor del argumento
         LSR.B     #6,D2          ; Paso 3: alineación de los bits 0 y 1 con 6 y 7
         EOR.B     D2,D1          ; Paso 4: OR exclusivo de los bits 0 con 6 y 1 con 7
         MOVE.B    D1,D2          ; Paso 5: carga del registro auxiliar
         LSR.B     #1,D2          ; Paso 6: alineación del bit b(0,6) con el bit b(1,7)
         EOR.B     D2,D1          ; Paso 7: OR exclusivo de los bits b(0,6) y b(1,7)
         BCHG      #0,D1          ; Paso 8: inversión del bit b(0,1,6,7)
         ROXR.B    #1,D1          ; Paso 9: Carga del bit en el flag X
         ROXR.B    #1,D0          ; Paso 10: rotación extendida del argumento
         MOVEM.L   (SP)+,D1/D2    ; Paso 11: restauración de los registros auxiliares
         RTS                      ; Paso 12: retorno de la subrutina
```

6) Modos de direccionamiento utilizados.

Puesto que tanto el parámetro como las variables intermedias empleadas se encuentran en registros de datos, el modo de direccionamiento predominante es precisamente mediante registro de datos. Además, existen algunas referencias a operandos inmediatos, y el manejo de la pila se materializa a través de direccionamientos relativos al puntero de pila con posincremento y predecremento.

3.6. Realizar una subrutina para sumar dos vectores de 10 números enteros de 32 bits. Los elementos de cada vector se almacenan en posiciones de memoria consecutivas. Los parámetros que se le pasan a la subrutina son:

- La dirección de comienzo del primer vector en el registro *A0*
- La dirección de comienzo del segundo vector en el registro *A1*.

El vector suma se guardará en el primer vector, destruyendo su valor previo.

Seguir el procedimiento indicado a continuación:

1) Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Datos de entrada y resultados de la subrutina.

Datos de entrada:

La subrutina recibe como argumentos de entrada dos registros de direcciones, *A0* y *A1*, cuyos contenidos son punteros que señalan a la dirección inicial de sendos vectores de datos almacenados en memoria. Cada uno de estos vectores está constituido por una secuencia de 10 palabras largas ubicadas consecutivamente en memoria. Cada registro contiene la dirección del primer elemento de uno de los vectores.

Datos de salida:

La subrutina deberá almacenar en el vector apuntado por *A0* la suma componente a componente de cada elemento de *A0* con su homólogo en *A1*. La subrutina devolverá los registros *A0* y *A1* con sus contenidos originales.

2) Descripción del algoritmo propuesto.

Este ejercicio se resolverá como un ejemplo de utilización del direccionamiento relativo a registro con índice.

Se cuenta con *A0* y *A1* como registros de dirección base a los dos vectores, a los que denominaremos *V0* y *V1*. Para recorrer todos los elementos de los mismos, se utilizará un registro índice almacenado en un registro de datos (*D0*).

El algoritmo consistirá en un bucle que en cada pasada sumará el elemento i-ésimo de *V0* con el elemento i-ésimo de *V1*, y dejará el resultado en el elemento i-ésimo de *V0*. El bucle terminará cuando se

hayan sumado todos los elementos de los vectores, lo que se detectará en cuanto el índice *D0*, empleado además como contador del bucle e incrementado en cada pasada por el mismo, supere un valor límite. Cada elemento ocupa 4 bytes, con lo que los incrementos del índice serán de 4 en 4, y el bucle finalizará cuando el contenido de *D0* llegue al valor 10*4=40. Con este esquema, los registros *A0* y *A1* quedarán intactos durante la subrutina.

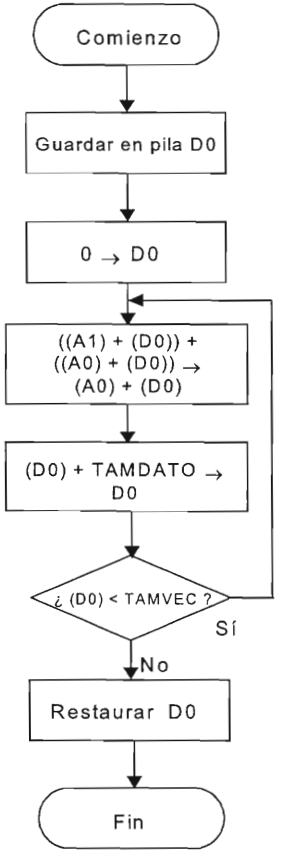
3) Descripción por pasos del algoritmo propuesto.

Los pasos que ha de seguir la subrutina se enlistan a continuación:

1. Guardar en la pila el contenido previo de los registros utilizados: $(SP) - 4 \rightarrow SP$; $(D0) \rightarrow (SP)$.
2. Inicializar el índice: $0 \rightarrow D0$.
3. Sumar la pareja de elementos de los vectores cuya posición viene marcada por el índice *D0*: $((A1)+(D0)) + ((A0)+(D0)) \rightarrow (A0)+(D0)$.
4. Incrementar el índice: $(D0)+TAMDATO \rightarrow D0$.
5. Si $(D0) < TAMVEC$, volver al paso 3.
6. Restaurar los registros salvaguardados en pila: $((SP)) \rightarrow D0$; $(SP) + 4 \rightarrow SP$.
7. Retornar de la subrutina.

En esta descripción se ha incluido el valor *TAMVEC*, que será una constante que expresará el tamaño del vector en bytes.

4) Diagrama de flujo de la solución propuesta.



5) Codificación en ensamblador

```
SUBRUT    ORG          $1000          ; Comienzo de código en posición $1000
                                      ; Definición de constantes
TAMDATO   EQU          4              ; Tamaño de un elemento en bytes
TAMVEC    EQU          10*TAMDATO     ; Tamaño de los vectores en bytes
          MOVEM.L      D0/D1,-(SP)    ; Paso 1: salvaguarda de registros en pila
          MOVEQ        #0,D0          ; Paso 2: inicialización de D0 (contador)
BUCLE     MOVE.L(A1,D0),D1            ; Paso 3: suma de los elementos
          ADD.L        D1,(A0,D0)     ;   a través de un registro auxiliar
          ADDQ.L       #TAMDATO,D0    ; Paso 4: incremento del contador
          CMP.L        #TAMVEC,D0     ; Paso 5: comprobación de D0
          BCS          BUCLE          ;   Volver al principio del bucle si D0 < N
          MOVEM.L      (SP)+,D0/D1    ; Paso 6: restauración de registros
          RTS                         ; Paso 7: retorno de la subrutina
```

Comentarios adicionales:

- Para recorrer los vectores podría haberse empleado también un esquema de direccionamiento relativo a registro de direcciones con posincremento. Sin embargo, al hallarnos en una subrutina, los registros *A0* y *A1* han de quedar intactos al salir de la misma, lo que nos obliga a preservar su valor de entrada a la misma mediante la pila. Lo mismo sucedería si copiásemos *A0* y *A1* en otros registros de direcciones.

- Es necesario utilizar el registro *D1* para sumar los dos elementos de los vectores porque la instrucción *ADD.L (A1,D0),(A0,D0)* presenta una incompatibilidad en los direccionamientos de los operandos.

- La comparación empleada para comprobar el fin de la ejecución del bucle es *BCS*, pues se ha considerado que tanto *D0* como *TAMVEC* son números sin signo.

6) Modos de direccionamiento utilizados.

El modo de direccionamiento clave en la solución de este problema es el relativo a registro con índice. El recorrido a través de los vectores podría haberse realizado utilizando direccionamiento relativo a registro con posincremento, pero en cualquier caso habría sido necesario controlar la condición de fin de bucle, y la manera más sencilla de hacerlo es mediante un contador. Con el direccionamiento relativo a registro con índice, el propio índice se emplea como contador del bucle, y cumple así una doble función.

Otros modos empleados son el direccionamiento mediante registro de datos en las referencias al contador y a la variable auxiliar, relativo a registro con posincremento y predecremento en las operaciones con la pila, e inmediato.

3.7. Se tiene una estructura de datos en cadena simple en la memoria del M68000. Cada elemento de la estructura consta de 2 palabras largas en el siguiente orden: un puntero al siguiente elemento y el dato asociado. El último elemento de la cadena tiene un puntero igual a 0. Todos los datos de la cadena son positivos. El puntero al comienzo de la estructura se encuentra en la posición *020000* en hexadecimal.

Se quieren poner en una tabla que comience en la posición *030000* todos los datos de la cadena comprendidos entre dos valores máximo y mínimo dados, que se encuentran en las posiciones *020004* y *020008* de la memoria.

Escribir el programa que genere la tabla pedida. Para ello, seguir el procedimiento indicado a continuación:

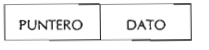
1) Especificar las estructuras de datos iniciales y los datos de entrada y de salida mencionados en el enunciado, indicando los tamaños elegidos para los mismos.
2) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) Describir por pasos el algoritmo propuesto, indicando las constantes, las variables y las estructuras de datos intermedias utilizadas.
4) Realizar un diagrama de flujo de la solución propuesta.
5) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.
6) Justificar la elección de los modos de direccionamiento empleados en la solución.

1) Estructuras de datos iniciales, datos de entrada y resultados del problema.

Estructuras de datos iniciales y datos de entrada:

Una cadena simple es una estructura de datos en la que cada elemento consta de dos componentes o campos:

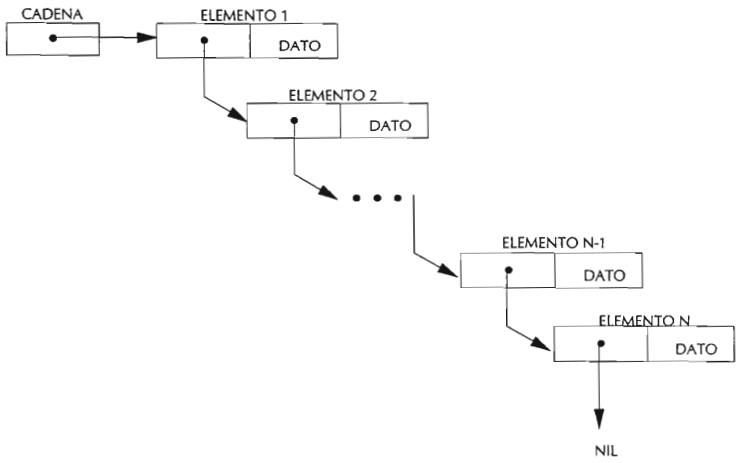
- El dato contenido en el elemento.
- La dirección (puntero) al siguiente elemento de la cadena.



Esta estructura de datos es de uso común en lenguajes de programación que permiten el manejo de memoria dinámica, tales como MODULA, PASCAL o C.

En nuestro caso, tanto el dato como la dirección del siguiente elemento de la cadena ocupan sendas palabras largas de 32 bits.

La siguiente figura representa una cadena simple, en la que el puntero se representa por una flecha que enlaza al elemento con el siguiente en la estructura.



En toda estructura de tipo cadena debe existir un puntero que señale hacia el primer elemento de la cadena. En nuestro caso, dicho puntero es un dato de entrada que se halla en la posición de memoria $020000 (*CADENA*). Suponemos que este dato ocupa una palabra larga de memoria (32 bits).

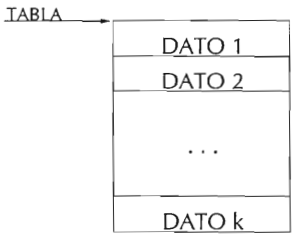
El último elemento de la cadena contiene un puntero nulo. Esto se representa de diferentes formas en distintos lenguajes de programación. Nosotros designaremos el puntero nulo mediante la constante *NIL*.

Los distintos elementos de la cadena pueden estar almacenados en cualquier ubicación en la memoria del computador, y el elemento siguiente a un elemento dado no tiene por qué hallarse en una posición de memoria más alta. La única restricción es que el puntero y el dato de un elemento dado deben permanecer en posiciones de memoria contiguas y, en nuestro caso, el puntero por delante del dato.

Como datos de entrada, el programa tiene otras dos variables, que llamaremos *MAXIMO* y *MINIMO*, situadas respectivamente en las posiciones $020004 y $020008 de memoria. Cada una de ellas ocupará una palabra larga.

Resultados:

El programa que se pide debe guardar secuencialmente en una tabla, almacenada a partir de la posición de memoria $030000 (*TABLA*), los campos de datos de la cadena cuyo valor esté dentro del rango definido por las variables *MAXIMO* y *MINIMO*. La estructura de la tabla se presenta en la siguiente figura.



2) Descripción textual del algoritmo propuesto.

El proceso consistirá en recorrer la estructura en cadena elemento por elemento, y en cada uno de ellos evaluar si el valor del dato se encuentra en el intervalo limitado por *MINIMO* y *MAXIMO*. Si es así, el dato se escribirá en la primera posición vacía de la tabla. Para recorrer la estructura en cadena, se empleará el registro de direcciones *A0*, mientras que el registro *A1* se utilizará para insertar en la tabla los valores que se encuentren en el intervalo. La cadena se recorrerá mediante un bucle de tipo WHILE, cuya condición de finalización será que el puntero del elemento alcanzado valga 0 (*NIL*).

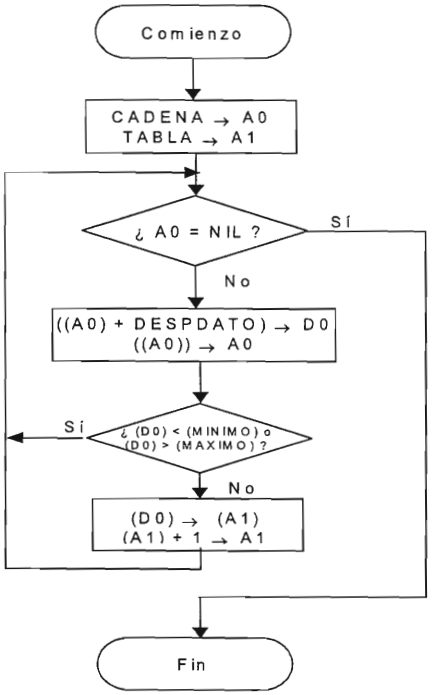
3) Descripción por pasos del algoritmo propuesto.

La secuencia de acciones realizada por el programa será:

1. Inicializar los punteros a las estructuras: (*CADENA*) → *A0*; (*TABLA*) → *A1*.
2. Si (*A0*) = *NIL*, hemos llegado al final de la cadena. Saltar al paso *7*.
3. Leer siguiente elemento de la cadena: (*DESPDATO* + (*A0*)) → *D0*; ((*A0*)) → *A0*.
4. Si (*D0*) < (*MINIMO*) o (*D0*) > (*MAXIMO*), saltar al paso 2.
5. Insertar (*D0*) en la tabla: (*D0*) → (*A1*); (*A1*) + 1 → *A1*.
6. Volver al paso 2.
7. Fin del programa.

La constante *DESPDATO* contiene el desplazamiento en bytes que hay que sumar al puntero que apunta al elemento para ser capaz de acceder al dato contenido.

4) Diagrama de flujo de la solución propuesta.



5) Codificación en ensamblador.

```
PROGRAMA  ORG       $1000           ; Origen de código en posición $1000
NIL       EQU       $00000000       ; Puntero nulo
DESPDATO  EQU       4
                                    ; Paso 1: inicialización de punteros
          MOVEA.L   CADENA,A0       ;   A0 apunta al principio de la cadena
          LEA       TABLA,A1        ;   A1 apunta al principio de la tabla
BUCLE     CMP.L     #NIL,A0         ; Paso 2: si A0 es NIL, hemos llegado
          BEQ       FINAL           ;   al final de la cadena
          MOVE.L    DESPDATO(A0),D0 ; Paso 3: lectura del campo de datos
          MOVEA.L   (A0),A0         ;         lectura del campo de puntero
          CMP.L     MINIMO,D0       ; Paso 4: si el dato es menor que MINIMO
          BLT       BUCLE           ;               volver al principio del bucle;
          CMP.L     MAXIMO,D0       ;         si el dato es mayor que MAXIMO
          BGT       BUCLE           ;               volver al principio del bucle;
          MOVE.L    D0,(A1)+        ;             ; Paso 5: meter el dato en la tabla
          BRA       BUCLE           ; Paso 6: volver al principio del bucle
FINAL     STOP      #$2700          ; Paso 7: fin del programa
DATOS     ORG       $020000         ; Zona de datos iniciales
CADENA    DS.L      1               ; Reserva del puntero al principio de la cadena
MAXIMO    DS.L      1               ; Reserva del extremo superior
                                    ; del rango de la tabla
MINIMO    DS.L      1               ; Reserva del extremo inferior
                                    ; del rango de la tabla
          ORG       $030000         ; Zona de datos de salida
TABLA     DS.L      100             ; Reserva para la tabla (tamaño arbitrario)

          END
```

Comentarios adicionales:

- Las comparaciones con *MINIMO* y *MAXIMO* se han realizado considerando que ambos son números con signo.
- Los datos del programa no se han inicializado en ningún caso, y se ha supuesto que los elementos de la cadena son almacenados en memoria por otro programa o fragmento de código no incluido en esta solución.

6) Modos de direccionamiento utilizados.

En este ejercicio se pueden distinguir dos estrategias distintas de direccionamiento, una en relación a la estructura de datos en cadena, y otra en cuanto a la inserción de elementos en la tabla. El recorrido de la estructura en cadena se realiza mediante direccionamiento relativo a registro (lectura del puntero al siguiente elemento través del contenido del registro *A0*) y direccionamiento a registro de direcciones (escritura del mismo en *A0*), mientras que el acceso al campo de datos de cada elemento necesita de un direccionamiento relativo a registro con desplazamiento. Por otra parte, la inserción en la tabla se lleva a cabo utilizando un direccionamiento relativo al registro *A1* con posincremento.

Otros modos empleados en la solución son el direccionamiento absoluto en las referencias a variables en memoria, el direccionamiento relativo a registro de datos y el inmediato.

Bienvenido a mis Tutorías © en la UNED

Ejemplos de utilización IEEE 754

Ejemplo 1:

César Moreno Fernández

El problema es el del examen del 3 de Febrero de 94.
Se pide la representación del Nº 53'2874 en formato normalizado IEEE 754 para coma flotante de 16 bit.

16 bit quiere decir, a no ser que especifiquen lo contrario, que utilizan 1 bit para el signo, 8 para el exponente y 7 para la man El mismo resultado se obtiene con la precisión normal de 32 bits, eliminando los últimos cuatro bytes de la derecha, que correspond a los 16 bits extras de la mantisa.

Pasos

1º) Calcular el signo:

En nuestro caso es + luego el bit de signo es 0

2º) Normalización de la mantisa:

53'2874 = 110101'010010

(se haya dividiendo por dos el entero y quedándote con el resto y multiplicando por 2 la parte fraccionaria y quedándote con los resultados enteros)

Como es el formato IEEE 754 hay un bit implícito. Esto quiere decir que vamos a correr la coma hasta el primer uno pero en luga dejarla a su izquierda como se hace en otros formatos lo dejamos derecha (específico para IEEE 754). Dicho bit implícito no será representado, con lo que ganamos un bit más en la precisión de la mantisa.

Esto es:

```
corremos coma   1'10101'010010
                 ^____ /
```

la hemos corrido +5 posiciones. Lo tendremos en cuenta para despu

el bit implícito no lo representamos en la mantisa, luego cogemos los 7 bits siguientes a la coma

Mantisa: 1010101

3º) El exponente:

Se calcula en base al exceso 2^(n-1) -1 (específico de IEEE 754)
Entonces tenemos : +5 (de la coma corrida) + 2^(n-1)-1=132= 10000
Luego la representación del número será:

```
0100001001010101
_ \_____/\_____/
 Sig  Expon   Mantisa
```

Ojo en algunos exámenes los posibles resultados los dan en Hex. Así que lo pasas de binario a Hex. Luego la solución es 4255 (hex

Ejemplo 2:

Demetrio Quirós

Este problema es del examen del 26 de Enero de 1999 de ETC-1 (Modelo B - Pregunta 15):

Obtener la representacin del número decimal (-0.00015) en el form normalizado IEEE754 para coma flotante de 16 bits (igual que el d pero con una mantisa de 7 bits)

A) 377B B) B77D C) B91D D) 377D

Empezaremos por calcular el equivalente de -0.00015 en formato m* procedimiento que yo sigo es dividir el número por 2 elevado a 'l hasta que el resultado esté en la forma 1.xxxx, en este caso: -0. 2^-13 = 1.2288, con lo que podemos decir que -0.00015 = 1.2288 * lo que ya tenemos la mantisa y el exponente.

Signo: negativo = 1
Exponente: 127 - 13 = 114 = 01110010
Mantisa: [1.]2288 = [1.]0011101

Con lo que el resultado es

```
1011 1001 0001 1101
 B    9    1    D
```

Ejemplo 3:

Ramón Quiñones Lozano

¿Cuál es el error, en valor absoluto, que se comete al representa número decimal 291.072 con el número en formato IEEE754 (16 bits)

Primero pasamos el número hexadecimal a binario:

4391 hex = 0100 0011 1001 0001

colocándolos segun el formato IEEE754

signo	exponente	mantisa
0	10000111	0010001

y convirtiéndolo queda:

signo (0) -> positivo
exp. (10000111) = 135 (quitando el exceso a 127) 135-127 =
mantisa 1.0010001 (ojo al bit implícito)

1.0010001 x 2^8 = 100100010 = 290 en decimal

Así que el error = 291.072 - 290 = 1.072

Ejemplo 4:

Dethais

Convertir el número -2.5675e15 al formato IEEE 754 de 32 bits:

1º Se coge el nº sin el menos, es decir positivo y escribimos lo

2.5675*10^15 = 2^exponente

Nos interesa despejar el "exponente". ¿Cómo se hace esto? Con log
La expresión simplificada quedaría:

[log(2.5675)+15*log(10)]/log(2) = exponente

Esto se hace con la calculadora y listo. Se puede hacer con logar
logaritmos neperianos, lo que se prefiera. El exponente sale:

exp = 51.189

(aproximamos al inmediatamente inferior, o sea al 51. Esto se hac
Si fuera negativo, por ej. el -81.6, cogeríamos el -82)

Entonces, ¿qué se ha conseguido con esto? Pues una aproximación a
nº que nos dan. Para tener el nº exacto entonces tendremos que ha

2.5675*10^15 = x*2^51

Calculamos 2^51 con la calculadora y nos sale:

2^51 = 2251799813685248

La x es el nº que multiplica a nuestra "aprox. " para que dé el n
(el 2.5675*10^15). Despejamos x y nos queda que:

x = 2567500000000000/2^51=1.14019904629003576701

2º Ahora tenemos que pasar a binario el nº 1.14. Lo bueno de este
método es que sólo tienes que hallar la parte decimal del nº
(0.140199...) porque la parte entera es 1 y va a ser el bit implí
el de ahorro para el IEEE754.

0.14019904629003576701(dec) = 0.00100011111001000001011(bin)

Con esto se obtiene la mantisa, ahora calculamos el exponente.
En la notación IEEE754 el exponente se pone en exceso, por lo que

2^(n-1)-1 + exponente = 2^7 - 1 + 51 = 127 + 51 = 178

3º Ahora se pasa todo a la notación IEEE754:

El primer bit es el de signo, como es negativo se pone un 1. Los
8 bits son los del exponente, por lo que ponemos el 178 en binari
bits y el resto (23 bits) son la mantisa que recuerda que se pone
de bit, eso quiere decir que el primer bit significativo de la ma
omitimos. Quedaría así:

1 10110010 x00100011111001000001011

(en la x estaría un 1, pero como es el bit implícito lo quitamos)

Ahora agrupándolos de 4 bits en 4 bits tenemos el nº en hexadecim

1101 1001 0001 0001 1111 0010 0000 1011

D 9 1 1 F 2 0 B

Es decir:

-2.5675e15 (dec) = D911F20B (ieee754)

Ejemplo 5:

Antonio Bello

Convertir el número -0.01 al formato IEEE 754 de 32 bits:

1º Se coge el nº sin el menos, es decir positivo y escribimos lo

0.01 = 2^exponente

Nos interesa despejar el "exponente". ¿Cómo se hace esto? Con log
La expresión simplificada quedaría:

log(0.01)/log(2) = exponente

Esto se hace con la calculadora y listo. Se puede hacer con logar
logaritmos neperianos, lo que se prefiera. El exponente sale:

exp = -6.643856...

(aproximamos al inmediatamente inferior, o sea al -7. Esto se hac
Si fuera positivo, por ej. el 81.6, cogeríamos el 81)

Entonces, ¿qué se ha conseguido con esto? Pues una aproximación a
nº que nos dan. Para tener el nº exacto entonces tendremos que ha

0.01 = x*2^-7

Calculamos 2^-7 con la calculadora y nos sale:

2^-7 = 0.0078125

La x es el nº que multiplica a nuestra "aprox. " para que dé el n
(el 0.01). Despejamos x y nos queda que:

x = 0.01/0.0078125=1.28

2º Ahora tenemos que pasar a binario el nº 1.28. Lo bueno de este
método es que sólo tienes que hallar la parte decimal del nº
(0.28) porque la parte entera es 1 y va a ser el bit implícito o
el de ahorro para el IEEE754.

0.28(dec) = 0.01000111101011100001010000(bin)

Con esto se obtiene la mantisa, ahora calculamos el exponente.
En la notación IEEE754 el exponente se pone en exceso, por lo que

2^(n-1)-1 + exponente = 2^7 - 1 - 7 = 127 - 7 = 120

3º Ahora se pasa todo a la notación IEEE754:

El primer bit es el de signo, como es negativo se pone un 1. Los
8 bits son los del exponente, por lo que ponemos el 178 en binari
bits y el resto (23 bits) son la mantisa que recuerda que se pone
de bit, eso quiere decir que el primer bit significativo de la ma
omitimos. Quedaría así:

1 01111000 x01000111101011100001010

(en la x estaría un 1, pero como es el bit implícito lo quitamos)

Ahora agrupándolos de 4 bits en 4 bits tenemos el nº en hexadecim

```
1011 1100 0010 0011 1101 0111 0000 1010

 B    C    2    3    D    7    0    A
```

Es decir:

```
-0.01 (dec) = BC23D70A (ieee754)
```

Ejemplo 6:

Enrique Buitrón

Representar el número -480 a coma flotante IEEE754

Las opciones son: A) C3E0 B) C3A0 C) C3F0 D) C3B0

1°/ Pasas 480 a binario: 480 = 111100000.

2°/ Como la repr. IEEE754 tiene bit implícito, colocas la coma de
del primer 1, es decir:

1.11100000 Como hemos corrido la coma 8 posiciones hacia la

izquierda, tenemos un exponente igual a 8.

3°/ El exponente está en exceso, por lo que sumamos 2^7-1 + 8 = 1

4°/ Lo representamos: El primer bit es el de signo, los 8 siguien
y los 7 restantes la mantisa.

1 10000111 1.1110000 ---> quitamos el bit implicito y ya nos q

la representación:

1 10000111 1110000 = C 3 F 0

Ejemplo 7:

Francisco Javier Alonso Álvarez

Convertir el número C19E0000 en formato IEEE754 en su equivalente

Primero paso hexadecimal a binario

C19E0000 = 1100 0001 1001 1110 0000 0000 0000 0000

s (1) exponente (10000011) mantisa (001 1110 0000 0000 0000 0000)

la formula es

(-1)^s * 1,mantisa * 2^(e-127)

s (1) significa, por tanto, signo negativo

exponente 10000011 es 131, como se representa en exceso a 2^(n-1)
restar 127 para volver a tener el exponente real, es por tanto 4.

mantisa (hacia la derecha se van multiplicando los bits por 2^-1,
2^-4, ...)

```
0*0,5
```



```
0*0,25
1*0,125
1*0,0625
1*0,03125
1*0,015625
----------
0,234375 en decimal
```

como está normalizada (se supone un 1 a la izquierda de la coma)
realidad 1,234375. Todo junto:

```
(-1)^1* 1,234375 * 2^4 = -1,234375*16 = -19,75
```

Ejemplo 8:

Antonio Bello

**Se desea normalizar el no fraccionario N=1000011000111010 represe
signo-magnitud sobre una palabra de 16 bits. El byte mas signific
contiene la parte entera con signo,y el byte menos significativo
fraccionaria. Normalizarlo segun la IEEE754, para 16 bits.**

Es decir:

```
Parte Entera: 10000110 (bin) = -6
Parte Fracc.: 00111010 (bin) = 0,2265625
```

Si utilizamos el conversor de la página

http://www.etsimo.uniovi.es/~antonio/uned/ieee754/IEEE-754.htm

sabemos que la solución tendría que ser: C0C7 (IEEE754)

1°) Calcular el signo:

Vamos ahora paso a paso. Lo primero es quitar el signo que ya sab
es negativo, entonces el número a normalizar sería:

```
110,00111010
```

2°) Normalización de la mantisa:

Como es el formato IEEE 754 hay un bit implícito. Esto quiere dec
vamos a correr la coma hasta el primer uno pero en lugar de dejar
izquierda como se hace en otros formatos lo dejamos a la derecha
para IEEE 754). Dicho bit implícito no será representado, con lo
un bit más en la precisión de la mantisa.

Esto es:

```
corremos coma   1,10,00111010
                  ^__|
```

la hemos corrido +2 posiciones. Lo tendremos en cuenta para despu

el bit implícito no lo representamos en la mantisa, luego
cogemos los 7 bits siguientes a la coma

```
Mantisa: 1000111
```

3°) El exponente:

```
se calcula en base al exceso 2^(n-1) -1 (especifico de IEEE 754)
Entonces tenemos : +2 (de la coma corrida) + 2^(n-1)-1=129= 10000
Luego la representación del número será:

   1 10000001 1000111
  _ \_______/ \_____/
  Sig  Expon   Mantisa

Ojo en algunos exámenes los posibles resultados los dan en Hex.
Así que lo pasas de binario a Hex. Luego la solución es C0C7 (hex
```

No dudes en escribir si tienes alguna pregunta sobre estos ejemplos

antonio@scig.uniovi.es