

Test de Teoría (3.0p)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
b	d	b	b	d	a	b	a	d	d	c	d	b	b	a	a	c	a	c	c	d	a	c	b	d	d	b	a	a	a

Test de Prácticas (4.0p)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
c	a	b	c	b	b	b	d	d	c	d	c	c	b	c	b	b	c	d	b

Examen de Problemas (3.0p)

1. Ensamblador (0.7 puntos).

a) puntuía **0.1p**, b) puntuía **0.5p**, c) puntuía **0.1p** (total **0.70p**).

a) La función devuelve **29** cuando le pasamos 2020 como argumento.

b) Algunas alternativas posibles

Traducción directa:

```
unsigned ndof (unsigned y) {
    if (y % 4 != 0)
        return 28;
    if (y % 100 != 0)
        return 29;
    return 29 - (y % 400 != 0);
}
```

Función original:

```
unsigned ndof (unsigned y) {
    return y % 4 == 0 ?
        y % 100 == 0 ?
            y % 400 == 0 ?
                29 : 28 : 29 : 28 ;
}
```

c) La función "ndof" calcula el **número de días de febrero** (number days of february) **del año y**.

https://es.wikipedia.org/wiki/A%C3%B1o_bisiesto#Algoritmo_computacional

Material adicional:

```
ndof:
movl $28, %eax      # eax = 28
testb $3, %dil      # y % 4 ?
jne .L1             # y % 4 != 0 ==> return 28 (no bisiesto)

movl $100, %ecx     # ecx = 100
movl %edi, %eax     # eax = y
xorl %edx, %edx     # edx = 0
divl %ecx           # edx:eax / ecx ==> y / 100
movl $29, %eax     # eax = 29
testl %edx, %edx    # y % 100 ?
jne .L1             # y % 100 != 0 ==> return 29 (bisiesto)

xorl %edx, %edx     # edx = 0
movl $400, %ecx     # ecx = 400
movl %edi, %eax     # eax = y
divl %ecx           # edx:eax / ecx ==> y / 400
cmpl $1, %edx       # edx=0? => CF== 1 // edx>0? => CF == 0
sbb %eax, %eax      # edx=0? => eax== -1 // edx>0? => eax== 0
notl %eax           # edx=0? => eax== 0 // edx>0? => eax== -1
addl $29, %eax      # y % 400 == 0 ==> return 29 (bisiesto)
.L1:                # y % 400 != 0 ==> return 29+(-1) (no)
ret
```

2. Ensamblador(0.4 puntos).

Se puntúa **0.1p** por cada pregunta (total **0.1 x 4 = 0.40p**)

- a) (0.2p) num_t1 es **int**, por cualquiera de los movslq, corroborado por desplazamiento 4 en movsbl 4(...) num_t2 es **char**, por movsbl, corroborado por cálculo $\&r+i*4+j+4 = \&r+4+(i*4+j)*1$
- b) (0.2p) **N=4**, por el factor leaq i*4 + j
M=3, por el desplazamiento movslq 16(%rdi) para j (i 4B, a 12B, j 4B)

Material adicional:

	<pre>fun: movslq 16(%rdi), %rdx movslq (%rdi), %rax leaq (%rdi,%rax,4), %rax movsbl 4(%rdx,%rax), %eax ret</pre>	<pre># r[offset 16] j? 4B signo => rdx # r[offset 0] i 4B signo => rax # calcular &r + i*4 y seguir calculando # r[offset 4+j*i*4] a[i][j] 1B signo</pre>	
	<pre>typedef int num_t1; typedef char num_t2; #define M 3 #define N 4</pre>	<pre>r: +---+---+ 00 i(4B) +---+---+ --\ 04 a[0] \ +---+---+ 08 a[1] a(12B = 3*4 B) +---+---+ 12 a[2] / +---+---+ --/ 16 j(4B) +---+---+</pre>	

3. Unidad de control (0.6 puntos).

Cada solución (a,b) requiere **una ecuación, un dato x (únicas/repet.), un resultado r**, puntúa **0.1p** cada uno (total **0.1 x 6 = 0.6p**).

Solución: Los diseños originales ocupaban: Horizontal $79ui * 32b/ui = 2528b$, Vertical $160ui * 12b/ui = 1920b$
se reduce al $1920/2528 = 0.7595 = 75.95\%$, ahorro del 24.05%, aunque ese dato no se pide

a) (0.3p)

Para acercarse al 50% repetición, de 79 microinstrucciones se deberían repetir unas 40, y otras $x=40$ deberían ser únicas. Haciendo prueba y error alrededor de esa cantidad $x=40ui$ únicas, el tamaño de las memorias de control sería

		micro:	nano:		
x	[lg2 x]	79 * [lg2 x]	x * 32	Total bits	
64	6	474	2048	2522	~ 2528 por 6bits no merece la pena diferenciar de 79ui 64 únicas
40	6	474	1280	1754	< 1920 40ui únicas ahorra más que diseño vertical
32	5	395	1024	1419	<< 32ui únicas usarían micromemoria de $79*5$ en lugar de $79*6$

El límite estaría entre 40 y 64 microinstrucciones únicas, en concreto:

$$474+32x \leq 1920; x \leq (1920-474)/32 = 1446/32 = 45.19; x \leq 45 \text{ únicas}$$

Solamente por comprobar que el cálculo es correcto:

		micro:	nano:		
x	[lg2 x]	79 * [lg2 x]	x * 32	Total bits	
46	6	474	1472	1946	> 1920
45	6	474	1440	1914	< 1920

Empezamos buscando un 50% repetición porque el ejemplo de los apuntes es todavía más repetido ($640ui$, 280 únicas < 320), pero vale cualquier método (incluso fuerza bruta por prueba y error) con tal de localizar que $32 < x < 64$, para poder plantear la ecuación que puntúa ($474+32x \leq 1920$) evitando el redondeo hacia arriba [lg2x].

El porcentaje de repetición debería ser entonces mayor o igual que

$$79-45=34 \text{ microinstrucciones repetidas}; 34/79 = 43.04\%; \text{rep} \geq 43.04\%$$

b) (0.3p) mismo procedimiento

		micro:	nano:		
x	[lg2 x]	160 * [lg2 x]	x * 12	Total bits	
128	7	1120	1536	2656	> 1920
64	6	960	768	1728	< 1920

El límite estaría entre 64 y 128 microinstrucciones únicas, en concreto:

$$1120+12x \leq 1920; x \leq (1920-1120)/12 = 800/12 = 66.67; x \leq 66 \text{ únicas}$$

Solamente por comprobar que el cálculo es correcto:

x	[lg2 x]	micro: 160 * [lg2 x]	nano: x * 12	Total bits
67	7	1120	804	1924 > 1920
66	7	1120	792	1912 < 1920

El porcentaje de repetición debería ser entonces mayor o igual que
 $190-66=94$ microinstrucciones repetidas ; $94/160 = 58.75\%$; $rep \geq 58.75\%$

4. Entrada/Salida (0.4 puntos).

Nota completa **0.40p** si el programa funciona. Las soluciones ocupan unas 4-8 líneas, por eso en modo “rescate” se puntúa **0.05-0.1p** por cada línea “acertada” (total **0.05 x 8 = 0.1 x 4 = 0.4p**). Algunas alternativas posibles:

Versión de 4 líneas:

```
#define RED    6
#define GREEN  3
#define BLUE   5

void setup() {}

void loop() {
    /* Escriba aquí el código */

    // analogWrite(BLUE, 0); se puede obviar
    for (int i=0; i<=255; i++){
        analogWrite(RED,255-i);
        analogWrite(GREEN, i);
        delay(10000/256);
    }
    // 10000/256ms por iteración x 256 its = 10s
}
```

Versión de 7 líneas:

```
#define RED    6
#define GREEN  3
#define BLUE   5

void setup() {}

void loop() {
    int redValue  = 255;
    int greenValue = 0;

    analogWrite (BLUE, 0);
    for(int i = 0; i < 256; i++) {
        analogWrite (RED , redValue-- );
        analogWrite (GREEN, greenValue++);
        delay (39);      // 256*39ms = 9984ms ~ 10s
    }
}
```

5. Configuración de memoria (0.5 puntos).

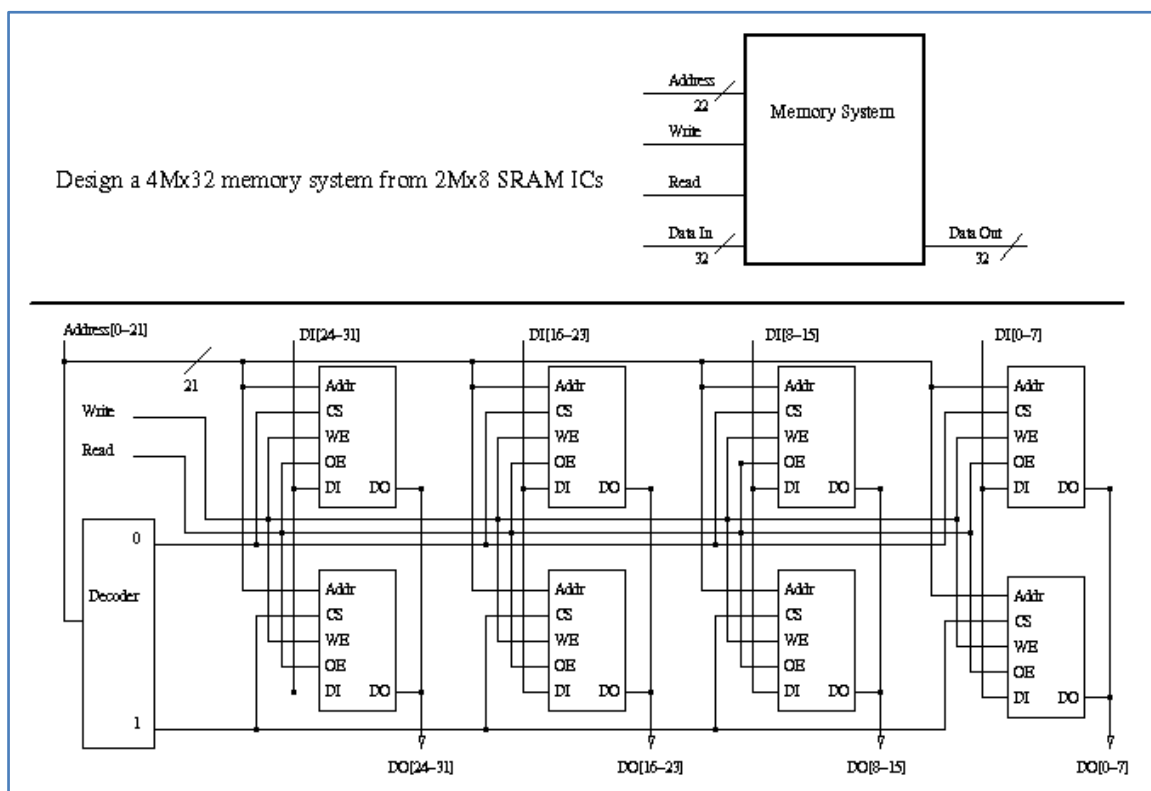
Se puntúa **0.1p** por cada detalle de la lista (total **0.1 x 5 = 0.50p**):

Apartado a): 2x4 chips, conexión R/W, decod.CS y conexión direcciones, conexión datos

Apartado b): los 4 números correctos

Solución:

a)



b)

Fila 0:
000000
...
1FFFFF

Fila 1:
200000
...
3FFFFF

6. Cache (0.4 puntos).

Se puntúa **0.1p** por cada número correcto (total **0.1 x 4 = 0.4p**).

Solución:

a) $2^{36} = 64 \text{ GB}$

b) $2^6 \text{ B/línea} = 64 \text{ B/línea}$

c) L3: $12 \text{ MB} = 3 \cdot 4 \cdot 2^{20} \text{ B} = 3 \cdot 2^{22} \text{ B}$

L3: $3 \cdot 2^{22} \text{ B} / 2^{14} \text{ conjuntos} = 3 \cdot 2^8 \text{ B/conjunto} = 768 \text{ B/conjunto}$

d) L3: $3 \cdot 2^8 \text{ B/conjunto} / 2^6 \text{ B/línea} = 3 \cdot 2^2 \text{ líneas/conjunto} = 12 \text{ líneas/conjunto} = 12 \text{ vías}$