2º Grado Informática
Estructura de Computadores
3 de septiembre de 2015

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
c	b	b	d	d	a	d	a	a	c	b	b	d	d	c	d	b	c	d	c	a	c	d	b	b	a	a	a	c	b

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
b	a	c	d	d	c	a	c	b	a	b	a	b	d	a	c	d	b	d	c

Examen de Problemas (3.0 p)

1. **Acceso a arrays**. (0,4 puntos). Valores de H y J

constante	valor
H	5
J	7

El enunciado no pide explicación, se ofrece como complemento

```
movl  8(%ebp), %edx              ; edx=x (índice fila de array1)
movl  12(%ebp), %eax             ; eax=y
leal  0(,%eax,8), %ecx           ; ecx=8y
subl  %eax, %ecx                 ; ... ..-y=7y
addl  %edx, %ecx                 ; ... ... ...+x: ecx=7y+x
leal  (%edx,%edx,4), %edx        ; edx=5x
leal  (%edx,%eax), %eax          ; eax=5x+y
movl  array1(,%eax,4), %eax      ; array1[x][y]->[5x+y]-> 5cols=H
movl  %eax, array2(,%ecx,4)      ; array2[y][x]->[7y+x]-> 7cols=J
```

2. **Popcount**. (0,8 puntos). Valores retornados por la función (columna **`result`**) y valores que debieran haberse calculado (columna *popcount*).

`array`	**`result`**	*popcount*
`{0,1,2,3}`	7	4
`{1,2,4,8}`	10	4
`{1,16,256,1024}`	11	4
`{5,4,3,2}`	16	6

El enunciado no pide explicación, se ofrece como complemento

La columna popcount se calcula inmediatamente. El error consiste en que val no se reinicializa a 0 tras cargar cada nuevo elemento x=array[i]; y los bits acumulados en cada byte de val (byte3…byte0, usando x & 0x01010101) y *vueltos a acumular* en los LSB con los desplazamientos (val>>16, val>>8) siguen ahí al acumular los de un nuevo elemento.

Seguir la pista en general de los bits *vueltos a acumular* es entretenido para el byte 1 y difícil para el byte 0, pero afortunadamente estos tests se restringen al byte0 (salvo los elementos 256 y 1024 al final del test 3). Abajo se muestran los resultados calculados por el programa sin error y con el error (el **test 3** requiere más explicación)

```
0 1 2 3                1 2 4 8                1 16 256 1024          5 4 3 2
0                      1                      1                      2
0+1                    1+1                    1+1                    2+1
0+1+1                  1+1+1                  1+1  +1                2+1+2
0+1+1+2 = 4            1+1+1+1 = 4            1+1  +1  +1 = 4        2+1+2+1 = 6
```

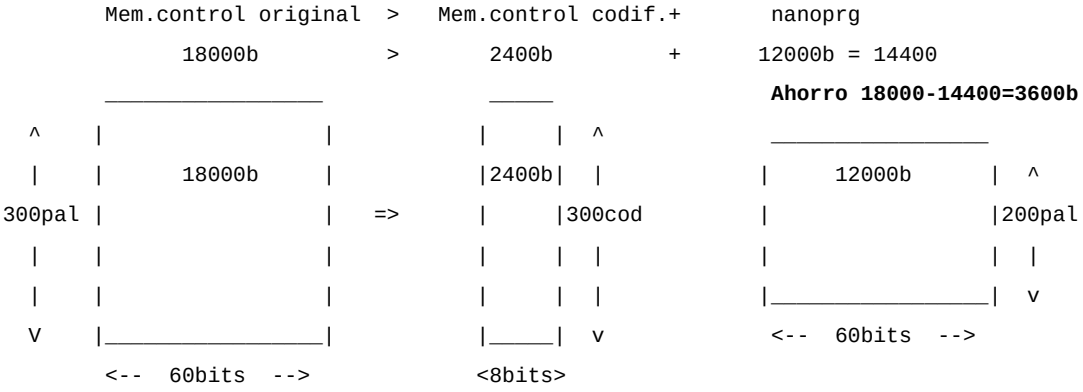
```
0+3+2+2 = 7          4+3+2+1 = 10      4+3  +2  +1 = 10       8+3+4+1 = 16
```

En el **test 3** el bit 2^8 se acumula en el byte1, se suma (correctamente) en su iteración (la 3ª), pero se vuelve a sumar (porque no se limpió val) en la siguiente iteración (4ª). Gráficamente se podría indicar así (columna izquierda). A la derecha se muestra una traza con los contenidos de x y val en cada iteración.

```
1 16 256 1024          elemento    x                 val         result
1                      1           |||   | 1|        |||  |1|     1
1+1                    16          |||   |1 |        |||  |2|     3
1+1  +1                256         |||  1|   |        |||1|3|     6
1+1  +1  +1 = 4        1024        |||1 |   |        |||2|5|     11
       \_+1                              byte  3 2 1  0    byte  32 1 0
4+3  +2  +2 = 11
```

3. **Unidad de control** (0,3 puntos). Ahorro en número de bits usando nanoprogramación (300pal 60bits / 200 dif.). Dibujos.

Codificar 200uinstr diferentes -> hacen falta 8bits (2^7=128 < 200 <= 256=2^8)
Memoria de control original: 300pal x 60bits = 18000bits 18000bits
M.C. codificada (nanoprogr): 300cod x 8bits = 2400bits ---
Microinstr.difer.(nanoprogr): 200pal x 60bits = 12000bits ---\---14400bits
Ahorro **3600bits**

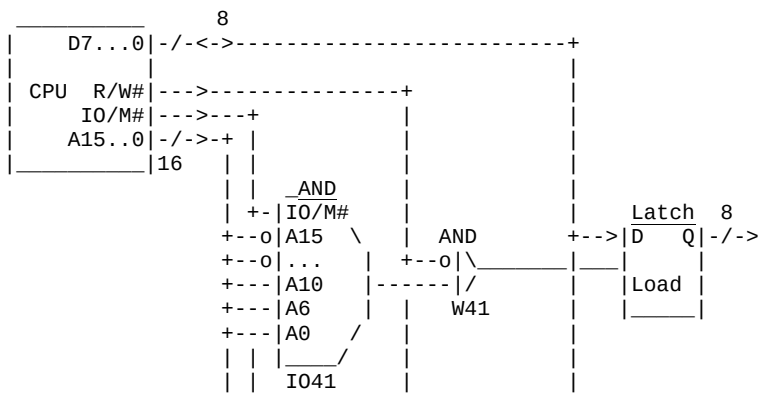


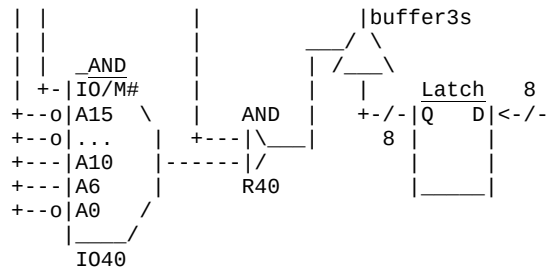
4. **Entrada/Salida** (0,5 puntos). Puerto 0x0440 de entrada y 0x0441 de salida.

Se podrían usar 2 (N)ANDs para decodificar:
Acceder a E/S 0x440 activa IO/M#, A10 y A6. Acceder a 0x441 activa además A0 (puertas marcadas IO40 / IO41)
Segundo nivel (N)AND según R/W para CLK-Load Salida o driver Entrada (puertas marcadas R40 / W41)
(driver Salida es el propio latch)

Según detalles temporización, R/W podría incorporarse a 1er nivel (N)AND y ahorrar 1 retardo propagación.
Se espera que temporización R/W <= IO/M-Addr-Data, para que CLK-Load pulse con datos estables
Driver/buffer triestado para entrada terminales Q a bus datos D7-0

```
Hex 0x0   4    4      1
Bin 0000 0100 0100 0001
          A10  A6     A0
```

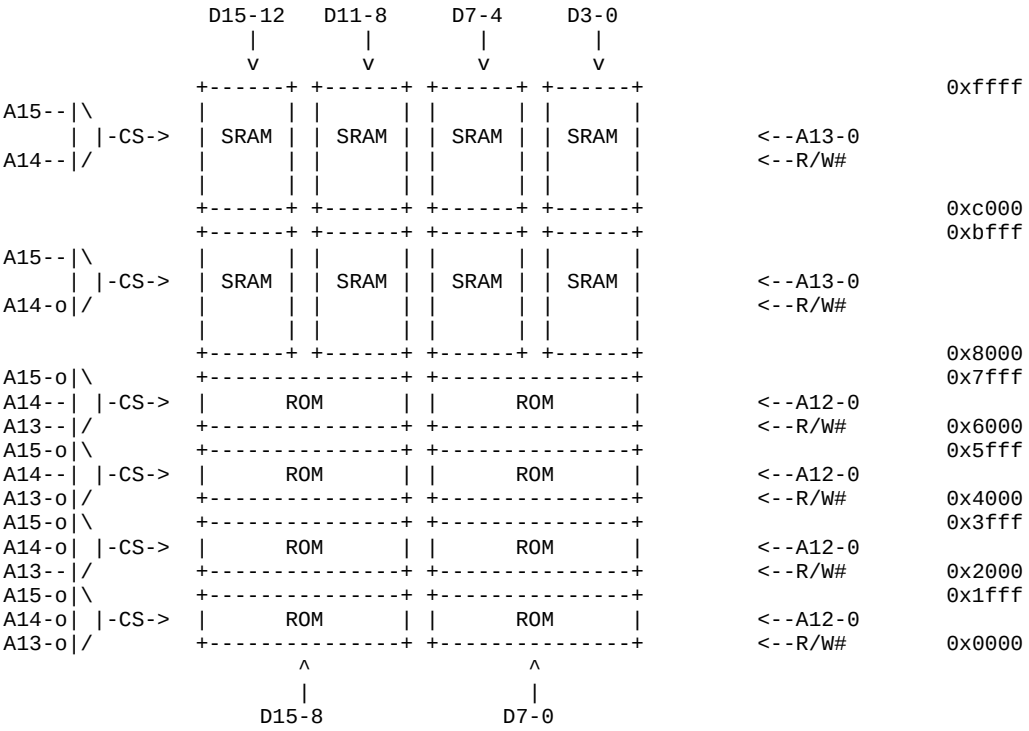




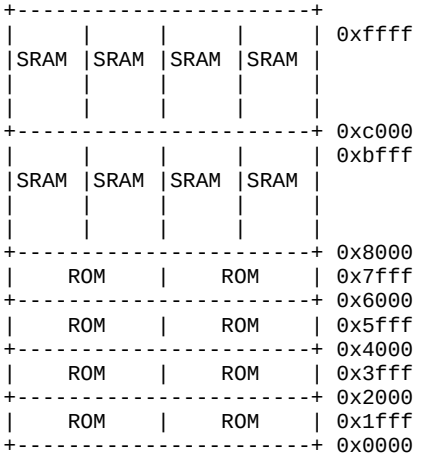
5. **Diseño del sistema de memoria** (0,5 puntos). SRAMs de 16Kx4 ocupando 0x8000 a 0xFFFF, ROMs de 8Kx8 ocupando 0x0000 a 0x7FFF. Mapa de memoria con direcciones.

ROM 0x0000-0x7fff = 2^{15}=32K pal = 32Kx16. En módulos de 8Kx8 x (4x2) -> 8 módulos
SRAM 0x8000-0xffff = 2^{15}=32K pal = 32Kx16. En módulos de 16Kx4 x (2x4) -> 8 módulos
Todos los módulos ROM conectados a A12…A0, decodificación con A15-A13 (000…011)
Todos los módulos SRAM conectados a A13…A0, decodificación con A15-A14 (10…11)

Dibujar: ROM: 4 hileras de 2 módulos, 0x0000-0x1fff, 0x2000-0x3fff, 0x4000-0x5fff, 0x6000-0x7fff
decodificación A15-A13=000,001,010,011 respectivamente
direcciones A12-0 a todos los módulos, datos D15-8, D7-0
SRAM: 2 hileras de 4 módulos, 0x8000-0xbfff, 0xc000-0xffff
decodificación A15-A14=10,11 respectivamente
direcciones A13-0 a todos los módulos, datos D15-12, D11-8, D7-4, D3-0
R/W# a todas las SRAM (R/W#) y ROM (OE) (R/W# invertida si OE# activa baja)



Mapa de memoria con direcciones



6. Memoria cache (0,5 puntos). 4GB MP/1MB cach/64B lin, para las tres correspondencias

Líneas de 64B = 2^6 B/lin → 6 bits para direccionar el byte dentro de cada línea
MP 4GB = 2^2 2^{30} = 2^{32} B → 32 bits para direccionar en memoria
→ 2^{32} B/MP / 2^6 B/lin = 2^{26} líneas en MP (64M líneas)
Cache 1MB = 2^{20} B/cache → 2^{20} B/cach / 2^6 B/lin = 2^{14} marcos en cache (16K marcos)
→ 16vías=2^4 lin/cjto → 2^{14} lin/cach / 2^4 lin/cjto = 2^{10} cjto/cach (1K conjuntos)

A. Asociativa: 32 = 26 + 6. etiqueta +offset.
B. Directa: 32 = 12+14+ 6. etiqueta+marco+offset. 2^{20} B/cach / 2^6 B/lin = 2^{14} lin/cache
C. Asociativa Conjuntos 16vías: 32 = 16+10+ 6. etiqueta+conjnt+offset. 2^{14} lin/cach / 2^4 lin/conj = 2^{10} conj/cache

Totalmente Asociativa: Cualquier bloque puede ir a cualquier marco, se identifica bloque por etiqueta

32 bits	
32-6 = 26 bits	6 bits
etiqueta	offset

Correspondencia directa: Bloques sucesivos van a marcos sucesivos. Al acabarse la cache, vuelta a empezar

32 bits		
32-20=12	20-6=14 bits	6 bits
etiqueta	marco (bloque,línea)	offset

Asociativa 16 vías: Bloques sucesivos van a conjuntos sucesivos, pero cada conj.tiene varios marcos (vías)

32 bits		
32-10-6= 16 bits	20-6-4=10bits	6 bits
etiqueta	conjunto	offset