2º Grado Informática
Estructura de Computadores
8 Febrero 2016

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

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

Examen de Problemas (3.0p)

1. **Ensamblador** (0.6 puntos).

Otras soluciones son posibles, siempre que produzcan el resultado correcto y no añadan complejidad innecesaria.
El enunciado no requería comentar el código, el comentario se ofrece sencillamente como explicación.

Expresión C:
```
x = (x &      ~0x7C) | ( (x & 0x1C) << 2);
```
Alternativa:
```
x = (x & 0xFFFFFF83) | ( (x & 0x1C) << 2);
```
Braindamage:
```
x = (x&0xFFFFFF80) | (x&3) | ( (x<<2) & 0x70 );
```
Comentario: el primer & elimina los bits 6-2 (conserva el resto), el | con el segundo & recupera los bits 4-2 en posiciones 6-4
Versión braindamage es como el propio nombre indica http://onlineslangdictionary.com/meaning-definition-of/brain-damaged

Pequeño fragmento en ensamblador IA-32:
```
movb     x, %al     ; ó movl x, %eax
movb   %al, %dl     ; copia bits 7-0
andb $0x83, %dl     ; bits 7,1,0 en DL
andb $0x1C, %al     ; abc (4,3,2)en AL
shlb    $2, %al     ; pasarlos a 6,5,4
orb    %dl, %al     ; recup.resto bits
movb   %al,    x    ; ó movl %eax, x
```

Alternativa:
```
movl     x, %eax
movl  %eax, %edx
andl $0xFFFFFF83,%edx
andl $0x1C, %eax
shll    $2, %eax
orl   %edx, %eax
movl  %eax,    x
```

Alternativa GCC:
```
movl     x, %eax
leal (,%eax,4), %edx
andl $0xFFFFFF83,%eax
andl $0x70, %edx
orl   %edx, %eax
movl  %eax,    x
```

2. **Ensamblador** (0.2 puntos).

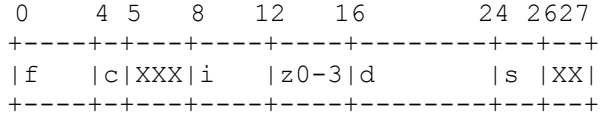
El enunciado no requería comentar el código, el comentario se ofrece sencillamente como explicación.
```
leal -48(%eax), %edx    ; edx = eax-48
cmpl  $9      , %edx;   ; > 9?
ja    .L2               ; above (sin signo)
```

EAX podría contener un código ASCII, en cuyo caso LEA lleva el char '0' a valor 0, ninguno de los caracteres desde '0' a '9' cumplen la condición ('9' se queda justo en el valor 9), y el resto de caracteres la cumplen

Respuesta: se salta a .L2 para valores de **EAX = [0..47] ∪ [58..0xffffffff]**
Alternativa: ***NO*** se salta para valores de EAX = [48..57]

3. **Disposición de estructuras** (0.6 puntos).

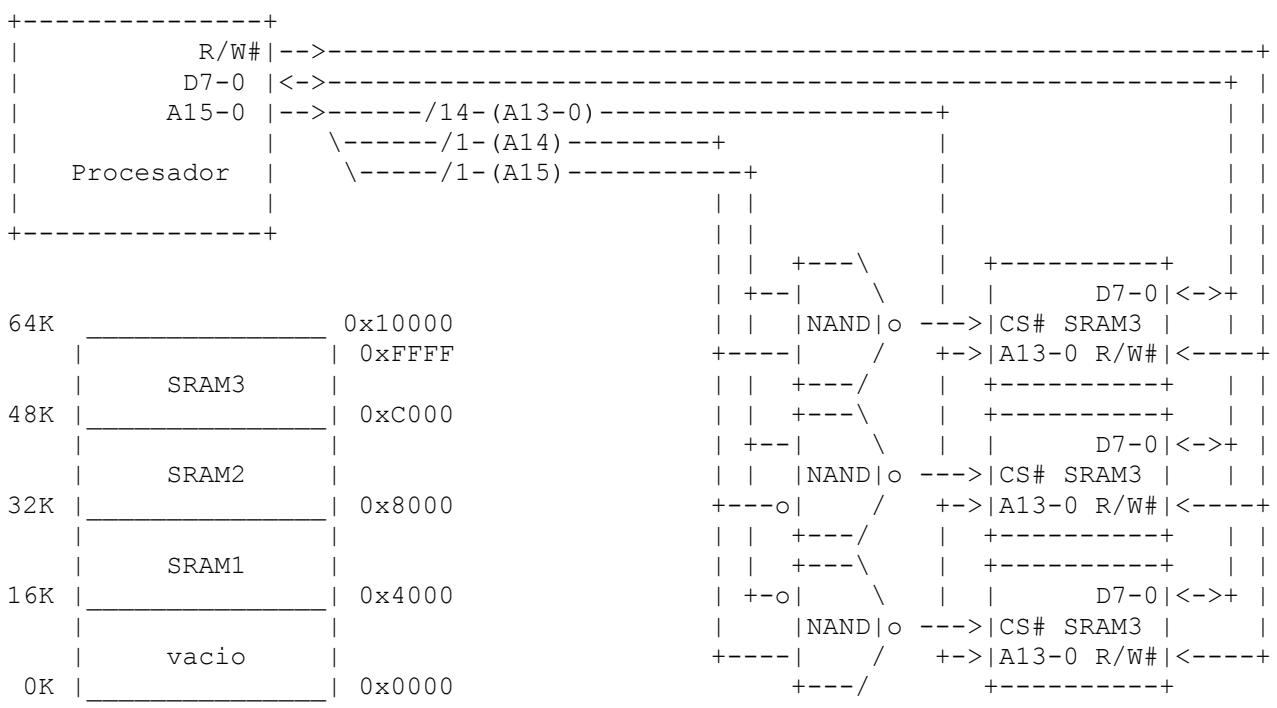
a) 28 bytes



b) 60 bytes

4. **Entrada/Salida y Memoria** (0.5 puntos).

a) Esquema de conexionado. El dibujo del mapa no se solicitaba, se añade como explicación.



b) **16K** direcciones disponibles para E/S (cada posición 1B). Etiquetado “vacío” en el mapa de memoria.

5. **Unidad de Control** (0.5 puntos).

CALL: Z = SP – 4
SP = Z ; MAR = Z
MBR = PC ; Y = PC
M[MAR] = MBR ; Z = IR + Y
PC = Z ; goto FETCH

RET: MAR = SP ; Z = SP + 4
MBR = M[MAR] ; SP = Z
PC = MBR ; goto FETCH

6. **Memoria cache** (0.6 puntos).

a) Correspondencia **directa** (i), asociativa por **conjuntos de 4** vías (ii), totalmente **asociativa** (iii)
b) i) **etiqueta, marco, palabra**
ii) **etiqueta, conjunto, palabra**
iii) **etiqueta, palabra**
alternativas: etiqueta = (marca, tag), marco = (línea, bloque), palabra = (desplazamiento, offset)
c) línea = 2^6 palabras = **64 pal**
d) cache = 2^4 líneas x 2^6 palabras/línea = 2^10 pal = 1024 pal = **1K pal**
e) 2^(6+4+6) = 2^(8+2+6) = 2^(10+6) = 2^16 = **64K pal**

---

2. **Ensamblador** (0.2 puntos). Comentarios adicionales

El enunciado está basado en un caso real de una necesidad que se dio en una base de datos (SWAD), cuyo lenguaje de consultas permite expresiones similares a las de lenguaje C.

La versión “pobremente diseñada” se ha calificado como tal porque utiliza 2 AND (&) para realizar el trabajo que se puede hacer con uno solo (0xFFFFFF83), e invierte el orden “lógico humano” máscara(0x1C)-desplazamiento resultando en desplazamiento-máscara(0x70), haciendo la máscara más difícil de entender para un programador humano.

Curiosamente, GCC con bitfields produce esa misma máscara **0x70**: en un esfuerzo por ahorrar operaciones usa LEAL para conseguir el efecto combinado de las instrucciones 2 y 5 (MOV y SHL), con lo cual es obligatorio retrasar la máscara.

En clase no se han explicado bitfields de lenguaje C. El lenguaje C se diseñó inicialmente como lenguaje de programación de sistemas, en donde se anticipa que las operaciones a nivel de bits serán muy frecuentes. Un programa C completo para realizar la operación deseada sobre un número cualquiera podría ser:

```
#include <stdio.h>

int main (void)
  {
   unsigned int x;
   union
     {
      struct {unsigned int :2, central:3;} original;
      struct {unsigned int :2, cero:2, central:3;} nueva;
      unsigned int x;
     } dato;

   printf("x? ");
if(scanf ("%x",&x) !=1) return 1;
   printf("x original = %#08X\n",x);

   dato.x             = x;
   dato.nueva.central = dato.original.central;
   dato.nueva.cero    = 0;
   x                  = dato.x;

   printf("x nuevo    = %#08X\n",x);

   return 0;
  }
```

Al compilar con distintos niveles de optimización se obtiene el siguiente código ensamblador:

-O0

```
movzbl  24(%esp), %eax        ; EAX= 8 bits menos significativos x
shrb    $2, %al               ; eliminar bits 1-0
andl    $7, %eax              ; dejar sólo bits 4-2 en pos.2-0
andl    $7, %eax              ; gcc -O0 es así
movl    %eax, %edx            ; EDX=bits "abc"
sall    $4, %edx              ;     ...en posiciones 6-4
movzbl  24(%esp), %eax        ; EAX= 8 bits menos significativos x
andl    $-113, %eax           ; ...& FFFFFF8F ->hueco en pos.6-4
orl     %edx, %eax            ; ...|EDX -> pegarle "abc" pos.6-4
movb    %al, 24(%esp)         ; gcc -O0 es así, machaca x
movzbl  24(%esp), %eax        ; pero se lo vuelve a traer porque
andl    $-13, %eax            ; ...& FFFFFFF3 ->hueco en pos.3-2
movb    %al, 24(%esp)         ; ahora sí, guardar x
```

-O1

```
movl    28(%esp), %edx        ; EDX=x
leal    0(,%edx,4), %eax      ; EAX=x<<2
andl    $112, %eax            ; EAX=(x<<2) & 0x70
andl    $-125, %edx           ; EDX=x & FFFFFF83
orl     %edx, %eax
movl    %eax, 28(%esp)        ; x= EAX | EDX
```

-O2

```
movl    28(%esp), %eax        ; EAX=x
leal    0(,%eax,4), %edx      ; EDX=x<<2
andl    $-125, %eax           ; EAX=x & FFFFFF83
andl    $112, %edx            ; EDX=(x<<2) & 0x70
orl     %edx, %eax
movl    %eax, 28(%esp)        ; x= EAX | EDX
```

Obsérvese que GCC deduce de las posiciones de bits especificadas en los bitfields las máscaras necesarias. Con –O0 usa $7 para extraer los bits 2-0 (“abc” ya desplazados a la derecha), $-113 (0xFFFFFF8F) para hacer hueco en bits 6-4, y muy aparatosamente $-13 (0xFFFFFFF3) para hacer hueco en posiciones 3-2. Con –O1 usa $112 (**el mencionado 0x70**) para extraer los bits 6-4 y $-125 (0xFFFFFF83) para hacer hueco en bits 6-2. Con –O2 usa las mismas máscaras.