| COGNOMS: |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|----------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| NOM:     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**IMPORTANTE leer atentamente antes de empezar el examen:** Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

## Problema 1. (2,5 puntos)

Un programa P ejecuta  $5x10^9$  instrucciones dinámicas y realiza  $3x10^9$  operaciones de punto flotante. El programa P tarda en ejecutarse  $8x10^9$  ciclos en un procesador C que funciona a una frecuencia de 4 GHz.

a) Calcula el CPI y el tiempo de ejecución en segundos (Texec) del programa P.

```
CPI = 8x10<sup>9</sup> ciclos / 5x10<sup>9</sup> instrucciones = 1,6 ciclos / instrucción

Texe = Ciclos / F= 8x10<sup>9</sup> ciclos / 4GHz = 2 seg
```

El procesador C tiene una corriente de fuga de 10 A, se alimenta a un voltaje de 2 V y tiene una carga capacitiva equivalente de 5 nF. El consumo debido a cortocircuito es despreciable.

b) Calcula la energía consumida al ejecutar el programa P en el procesador C.

```
P = Pfuga + Pconmut = I*V + C*V<sup>2</sup>*F = 10 A * 2 V + 5 nF * (2 V)<sup>2</sup> * 4 GHz = 100 W
E = P*t = 100W * 2 s = 200 Joules
```

El programa P tiene un 80% del código que es perfectamente paralelizable.

c) Calcula el número (N) de procesadores idénticos a C que son necesarios para conseguir un speed-up de 4 en el programa P.

```
Ley de amdahl:

4 = 1/(0,2 + 0,8/N) ---> N = 16
```

d) Calcula los MFLOPS de P en dicho sistema paralelo (N procesadores C).

```
TexeP = Texe/4 = 0,5s

MFLOPS = OpsPF / TexeP*10^6 = 3x10^9 opsPF / 0,5*10^6 = 6000 MFLOPS
```

En un sistema paralelo con 25 procesadores idénticos a C, cada procesador tiene un MTTF de 10 millones de horas. El resto del sistema (sin contar las CPUs) tiene un MTTF de 100.000 horas.

e) Calcula el tiempo medio hasta fallo (MTTF) del sistema completo.

```
MTTF = 1/(25/10.000.000+1/100.000) = 80.000 horas
```

5 April 2022 10:34 am

| COGNOMS: |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|----------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| NOM:     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## Problema 2. (2.5 puntos)

Dado el siguiente código escrito en C que compilamos para un sistema linux de 32 bits:

```
typedef struct {
    short a[3];
    char b[3];
    short c[2];
    short c[2];
} s1;
typedef struct {
    short v;
    int v;
    char w[5];
    char *x;
} s2;
```

a) **Dibuja** como quedarían almacenadas en memoria las estructuras **s1** y **s2**, indicando claramente los desplazamientos respecto al inicio, el tamaño de todos los campos y el tamaño de los structs.

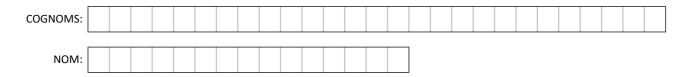
```
----- <---- 0
                                         ----- <---- 0
| a[1] a[0] |
                                         u[0] |
 -----4
                                         ----- <--- 14
| b[1]b[0] a[2] |
                                        | u[1] |
 ----- <---- 8
                                        ----- <---- 28
| c[0] - b[2] |
                                           V
 ----- <---- 12
                                        ----- <---- 32
     | c[1] |
                                       | w[3] ... w[0] |
    ----- <---- 14
                                        ----- <---- 36
                                        | - - - w[4] |
                                        ----- <---- 40
Tamaño de s1 = 14 bytes
                                       X |
                                        ----- <---- 44
                                      Tamaño de s2 = 44 bytes
```

b) **Escribe** UNA ÚNICA INSTRUCCIÓN que permita mover **z.u[1].c[0]** al registro **%ax**, siendo **z** una variable de tipo **s2** cuya dirección está almacenada en el registro **%ebx**. **Indica** claramente la expresión aritmética utilizada para el cálculo de la dirección.

```
\label{eq:condition} \mbox{La expresión aritmética para calcular la dirección del operando es @ini_z + 14*1 + 2*3 + 1*3 + 1 = @ini_z + 24, por lo tanto \%ebx + 24
```

La instrucción es: movw 24(%ebx), %ax

5 April 2022 10:34 am 2/4



## Problema 3. (2.5 puntos)

Dado el siguiente código escrito en C:

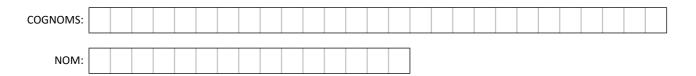
```
int examen(short int a, short int b, short int v[10], int M[10][10]) {
  int ii;
  short int aa, bb;
  short int vector[10];
  int *matriz;
  ...
  return M[ii][ii];
}
```

a) **Dibuja** el bloque de activación de la rutina examen, indicando claramente los desplazamientos respecto a **%ebp** y el tamaño de todos los campos.

```
----- i<- ebp-32
  ii
           ----- aa<- ebp-28
  | bb | aa | bb<- ebp-26
4
  -----<- <- ebp-24
20 | vector
  ----- <- ebp-4
  | *matriz |
             <- ebp
  | ebp old
  ______
      ret
  ----- <- ebp+8
  | -- | a
           ----- <- ebp+12
    -- |
         b
  ----- <- ebp+16
4
  g A
            ----- <- ebp+20
 M @ M
            ----- <- ebp+24
```

b) Traduce a ensamblador del x86 la instrucción return M[ii][ii] suponiendo que no hay que restaurar el valor de ningún registro.

5 April 2022 10:34 am 3/4



## Problema 4. (2.5 puntos)

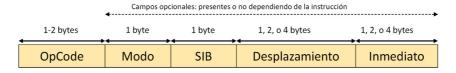
Cuando se va a ejecutar la siguiente instrucción escrita en ensamblador del x86:

```
movl $337, -40(%ebp, %esi, 4)
```

El contenido de los registros del procesador (en hexadecimal) es:

```
eax: 0x00000005 ebx: 0x00000004 ecx: 0x00000001 edx: 0x00000002 esp: 0x80800f30 ebp: 0x80800f48 esi: 0x00000002 edi: 0x00001001 eip: 0x808000fe eflags: 0x286
```

Sabemos que la codificación de instrucciones sigue el siguiente esquema tal y como se explica en clase:



Además sabemos que el procesador tiene una cache de datos de 32Kbytes (copy back+write allocate), y otra de instrucciones de 16Kbytes, ambas con lineas de 32 bytes y asociatividad 2. Sus páginas son de 4Kbytes y dispone de un TLB de 4 entradas. Asumimos que tanto el TLB como las caches están vacíos.

a) ¿Cuantos bytes ocuparía la instrucción anterior si sabemos que el OpCode de movl solo ocupa 1 byte?

```
1 opcode + 1 Modo + 1 SIB + 1 Desplazamiento + 2 Inmediato = 6 bytes
```

b) ¿Cuantos bloques de memoria se leen para ejecutar el movl? Escribe la dirección de dichos bloques.

```
2 instrucciones (0x808000e0 y 0x80800100) + 1 datos (0x80800f20)
```

c) ¿Cuantos bloques de memoria se escriben para ejecutar el movl? Escribe la dirección de dichos bloques.

```
Ninguno, es copy back + write allocate
```

d) ¿Cuantos fallos de TLB tendremos al ejecutar la instrucción anterior? Indica a que páginas es el fallo

```
1 fallo de TLB, instrucciones (0x80800) y datos están en la misma página
```

Suponiendo el siguiente código:

e) ¿Cuantos fallos de cache provoca el movl en función del valor de N? (suponemos de nuevo que las caches están vacías y los registros tienen valor indicado anteriormente antes de ejecutar el código).

```
1+N/8+1 si N no es múltiplo de 8: 1 para las instrucciones +N/8 para los datos pero aumenta en 1 fallo extra si N no es múltiplo de 8 por alineamiento.
```

5 April 2022 10:34 am 4/4