COGNOMS (en majúscules): . . . . . .

NOM (en majúscules): . . . . . .

Duració: 1,5 hores

Problema 1. (5 puntos)

Un programa (P) se ejecuta en un computador cuya CPU (C1) funciona a una frecuencia de 2,4 GHz. El programa (P) contiene 50000 instrucciones estáticas, ejecuta $6x10^9$ instrucciones dinámicas y realiza $2x10^9$ operaciones de punto flotante. La siguiente tabla muestra la distribución de instrucciones dinámicas y estáticas para el programa (P) junto con el CPI medio de cada tipo de instrucción.

	punto flotante	**enteras**	**memoria**
% instrucciones estáticas	10%	40%	50%
% instrucciones dinámicas	40%	35%	25%
CPI	4,0	2,0	6,8

a) **Calcula** el CPI del programa (P).

b) **Calcula** el rendimiento en MFLOPS del programa (P).

En nuestro computador hemos cambiado la CPU (C1) por un nuevo modelo (C2) que soporta instrucciones SIMD y tiene una cache de datos más grande, aunque funciona a menor frecuencia (2,25 GHz). Una vez recompilado el programa (P), para hacer uso de las nuevas instrucciones SIMD, observamos que el número de instrucciones dinámicas de punto flotante se ha reducido a la mitad (el resto no ha cambiado). Además el CPI de las instrucciones de punto flotante ha aumentado en un 25% y el de las instrucciones de memoria es de 5,2 ciclos/instrucción (el CPI de las enteras es el mismo).

c) **Calcula** el **CPI** del programa (P) con la nueva CPU (C2).

d) **Calcula** el tiempo de ejecución del programa (P) con la nueva CPU (C2).

Se ha decidido que una rutina (R), que representa el 80% del tiempo de (P), sea programada en ensamblador.

e) **Calcula** la ganancia en tiempo de ejecución (speedup) de la rutina (R) que se debería obtener respecto al compilador para que el programa (P) se ejecute 3 veces más rápido.

La CPU (C1), tiene una capacidad efectiva equivalente de 8 nF (nanofaradios), y una corriente de fugas de 12 A y funciona a un voltaje de 1,25 V.

f) **Calcula** la potencia media debida a fugas, la debida a conmutación y la potencia total disipada por la CPU (C1).

Este computador está formado por los componentes mostrados en la tabla siguiente. La tabla también muestra el número de componentes de cada tipo y el tiempo medio hasta fallo (MTTF) de cada componente.

Componente	Fuente alimentación	CPU	Ventilador CPU	Placa base	DIMMs	Discos duros	Tarjetas gráficas
Nº	1	1	1	1	4	2	2
MTTF (horas)	100.000	1.000.000	100.000	200.000	1.000.000	125.000	500.000

El tiempo medio para reemplazar un componente que ha fallado (*mean time to repair*) es de 5 horas y la probabilidad de fallo sigue una distribución exponencial.

g) **Calcula** el tiempo medio hasta fallos del hardware (MTTF), el tiempo medio entre fallos (MTBF) y la disponibilidad del sistema.

COGNOMS (en majúscules): . . . . . .

NOM (en majúscules): . . . . . .

Duració: 1,5 hores

Problema 2. (2 puntos)

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

```
typedef struct {
  char a;
  short b;
  double c;
  short d;
  char e;
 } s1;
```

```
typedef struct {
  int u;
  s1 v[10];
} s2;
```

a) **Dibuja** cómo 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.

b) **Escribe** UNA ÚNICA INSTRUCCIÓN que permita mover **x.v[5].e** al registro **%ah**, siendo **x** una variable de tipo **s2** cuya dirección está almacenada en el registro **%ecx**.
Indica la expresión aritmética utilizada para el cálculo de la dirección de **x.v[5].e**, de forma que estén identificados claramente los desplazamientos (offsets) dentro de cada una de las estructuras s1 y s2.

COGNOMS (en majúscules): . . . . . .

NOM (en majúscules): . . . . . .

Duració: 1,5 hores

Problema 3. (3 puntos)

Dado el siguiente código escrito en C:

```
int examen(int a, int b[2][2], char *c) {
  char y, z;
  char *p;
  int x[4];
  . . .

  return (x[0]*x[3])
}
```

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

b) **Traduce** a ensamblador x86 la sentencia **return(x[0]+x[3])**; sabiendo que la rutina ha usado los registros **%eax, %ecx, %edx, %esi**

c) **Traduce** a ensamblador x86 la siguiente sentencia suponiendo que está dentro la función examen: ***p = y**.