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. La siguiente tabla muestra la distribución de instrucciones para el programa (P) junto con el CPI medio de cada tipo de instrucción.

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

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

b) **Calcula** el rendimiento en MIPS 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 **%** de speedup (ganancia en tiempo) del programa (P) con la nueva CPU (C2) respecto a (C1).

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).

La CPU (C1) está conectada a una cache y esta a la memória principal (MP). Debido a los fallos de cache se realizan 15 millones de accesos a MP por segundo. La MP tiene un consumo de 2W cuando está inactiva y de 12W cuando está activa sirviendo un acceso. Por cada acceso, la MP está en estado activo durante 20 nanosegundos, el resto del tiempo está en estado inactivo (los accesos no se solapan).

g) **Calcula** la potencia media consumida por la memoria principal cuando se ejecuta el programa (P).

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.

h) **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. (5 puntos)

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

```
typedef struct {
  char c;
  short d;
  char e;
  char f[4];
  int h;
  short i;
 } s1;
```

```
typedef struct {
  int u;
  s1 v[10];
  char w;
  } 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].d** al registro **%ax**, siendo **x** una variable de tipo **s2** cuya dirección está almacenada en el registro **%ecx**.

Indica claramente la expresión aritmética utilizada para el cálculo de la dirección.

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

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

  return (x[1]+x[0])
}
```

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

d) **Traduce** a ensamblador x86 la instrucción `return(x[1]+x[0]);` sabiendo que la rutina ha usado los registros **%eax**, **%ebx**, **%ecx**, **%edx**, **%edi y %esi**