COGNOMS: . . . . . NOM: . . . . . .

1er Control Arquitectura de Computadors **Curs 2014-2015 Q2**

- **Temps: 13:30 a 15:00**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

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 **%** 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.

d) **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.

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

La CPU (C1) tiene las caches de datos e instrucciones integradas en el mismo chip. Durante la ejecución del programa (P), en la cache de instrucciones se producen en media 5 millones de fallos por segundo mientras que en la de datos se producen en media 16 millones de fallos por segundo. Sabemos que la cache de datos tiene una política de escritura **copy back + write allocate** y que el 25% de los fallos reemplazan bloques con el dirty bit activado (D=1). Leer o escribir un bloque de memoria requiere un acceso a memoria principal.

f) **Calcula** el número de accesos por segundo a memoria principal cuando se ejecuta el programa (P).

Sabemos que la memoria principal tiene un consumo de 2W cuando esta inactiva y de 10W cuando está activa sirviendo un acceso y que está en estado activo durante 20 nanosegundos por cada acceso (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 graficas
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: . . . . . NOM: . . . . . .

1er Control Arquitectura de Computadors **Curs 2014-2015 Q2**

Problema 2. (5 puntos)

Dado el siguiente código escrito en C y ensamblador del x86:

```
typedef struct {
  char a;
  char b[2];
  int c;
} s1;

void Subr1 (double *par1, int par2[3],
            char par3, s1 par4) {
double *local1;
int local2[3];
char local3;
s1 local4;
...
  Subr1(local1,local2,local3,local4);
...
}

int Subr2 (s1 *par, int pos) {
  return (int) par->b[pos];
}
```

```
void Summat(int A[N][M], int B[N][M]);
Summat:
  ...
      xorl %ecx, %ecx
fori: cmpl $N, %ecx
      jge endi
      xorl %edx, %edx
forj: cmpl $M, %edx
      jge endj
      imull $M, %ecx, %eax
      addl %edx, %eax
      movl (%esi,%eax,4),%ebx
      addl %ebx,(%edi,%eax,4)
      incl %edx
      jmp forj
      incl %ecx
      jmp fori
  ...
```

a) **Dibuja** el bloque de activación de la rutina Subr1, indicando claramente los desplazamientos respecto a ebp y el tamaño de todos los campos.

b) **Traduce** a ensamblador del x86 la llamada recursiva de la rutina Subr1.

c) **Traduce** a ensamblador del x86 la rutina Subr2:

Dado el código en ensamblador de la rutina Summat y sabiendo que %esi y %edi contienen las direcciones iniciales de las matrices A y B respectivamente: traduce a C dicho código y explica tres formas de optimizarlo (una línea por optimización):

Código C:

1.

2.

3.