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 secuencial (PS) ejecuta $4x10^9$ instrucciones dinámicas que realizan $3x10^9$ operaciones de punto flotante. El programa PS tarda en ejecutarse $10x10^9$ ciclos en un procesador C que funciona a una frecuencia de 5 GHz.

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

CPI = $10x10^9$ ciclos / $4x10^9$ instrucciones = **2,5 ciclos / instrucción**

Texe = Ciclos / F= $10x10^9$ ciclos / 5GHz = **2 seg**

El procesador C tiene una corriente de fuga de 20 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 PS en el procesador C.

P = Pfuga + Pconmut = $I*V + C*V^2*F$ = 20 A * 2 V + 5 nF * $(2\ V)^2$ * 5 GHz = 140 W

E = P*t = 140W * 2 s = **280 Joules**

Hemos programado una variante paralela de PS (llamada PP) que tiene un 90% del código paralelo y un 10% secuencial.

c) **Calcula** el número (N) de procesadores idénticos a C que son necesarios para conseguir un speed-up de 5 en el programa PP respecto PS.

Ley de amdahl:
5 = 1/(0,1 + 0,9/N) ---> **N = 9**

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

TexeP = Texe/5 = 0,4s

MFLOPS = OpsPF / TexeP*10^6 = $3x10^9$ opsPF / 0,4*10^6 = **7500 MFLOPS**

Una CPU para móvil dispone de 1 procesador A (alto rendimiento) y 4 procesadores B (bajo consumo). Cada procesador B tiene un 75% del rendimiento de A, pero consume solo un 30% de la potencia. La CPU está limitada por la potencia disipada por lo que simultáneamente solo puede funcionar o un procesador A o los 4 procesadores B, pero nunca todos a la vez. PS se ejecuta exclusivamente sobre el procesador A. Mientras que para PP, la parte secuencial se ejecuta en el procesador A y la parte paralela en los 4 procesadores B.

e) **Calcula** la ganancia en energía de PP respecto PS en esta CPU.

Parte secuencial: GEs = 1
Parte paralela: Tb = 1/(0,75*4) Ta; Pb = 0,3*4 Pa -> Eb = (0,30*4)/(0,75*4) Ea = 0,4 Ea -> GEp = 2,5
G (energia) = 1/(0,1 + 0,9/2,5) = **2,17**

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 int v1[7];
 int g;
 char c;
 short int v2[4];
} sa;
```

```
typedef struct {
 sa tabla[50];
 short int *h;
} sb;
```

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

sa

```
----------------  <---- 0
|    v1[0]     |
----------------  <---- 2
|       ...    |
----------------
|    v1[6]     |
----------------  <---- 14 + 2 vacio = 16
|      g       |
----------------  <---- 20
|      c       |
----------------  <---- 21 + 1 vacio = 22
|    v2[0]     |
----------------  <---- 24
|       ...    |
----------------
|    v2[3]     |
----------------  <---- 30 + 2 vacio = 32
```

Tamaño de sa = 32 bytes

sb

```
----------------  <---- 0
|   tabla[0]   |
----------------  <---- 32
|       ...    |
----------------
|   tabla[49]  |
----------------  <---- 1600
|      *h      |
----------------  <---- 1604
```

Tamaño de sb = 1604 bytes

b) **Escribe** UNA ÚNICA INSTRUCCIÓN en ensamblador que traduzca la instrucción en C: **y.tabla[36].c = 'A'**, siendo **y** una variable de tipo **sb** cuya dirección está almacenada en el registro **%ebx**. **Indica** claramente la expresión aritmética utilizada para el cálculo de la dirección.

La expresión aritmética para calcular la dirección es @ini_y + 32*36 + 20 = @ini_y + 1172, por lo tanto la expresión es: %ebx + 1172

La instrucción es: **movb $'A', 1172(%ebx)**

COGNOMS:

NOM:

Problema 3. (2.5 puntos)

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

```
int funcion(char x, short int y[3]) {
  int a;
  int *b;
  short int c[3];
  char d;
  . . .
  a = funcion(x, c) + *b;
  . . .
}
```

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

```
funcion
----------------------------- <- ebp-16
|              a              |
----------------------------- <- ebp-12
|             *b              |
----------------------------- <- ebp-8
|    c[1]     |    c[0]       |
-----------------------------
| --- |   d   |    c[2]       | <- d en ebp-2
----------------------------- <- ebp
|             EBP             |
-----------------------------
|             RET             |
----------------------------- <- ebp+8
| ----------------- |   x     |
----------------------------- <- ebp+12
|             *y              |
-----------------------------
```

b) **Traduce** a ensamblador x86 la sentencia `a = funcion(x,c) + *b;` que se encuentra en el interior de la subrutina, sabiendo que el valor de la expresión `*b` se encuentra en el registro **%edx** justo antes de la llamada. Se valorará usar el mínimo número de instrucciones y el mínimo número de accesos a memoria.

```
movl  %edx, %ebx          #  salvamos *b en un registro reguro
leal  -8(%ebp), %eax
pushl %eax                # c
pushl 8(%ebp)             # x
call  funcion
addl  $8, %esp
addl  %ebx, %eax          # valor retorno + *b
movl  %eax, -16(%ebp)     # a <- funcion2(x,c) + *b
```

Otras alternativas podrían ser guardar y restaurar %edx en la pila, o volver a leer *b.

COGNOMS:

NOM:

Problema 4. (2.5 puntos)

Disponemos de un sistema de cómputo que utiliza memoria virtual sin memoria cache ni TLB. El sistema funciona a 1GHz, tiene un CPI ideal (suponiendo que cada acceso a memoria principal se resuelve en 1 ciclo) de 8 ciclos y supondremos una tabla de páginas uninivel que es capaz de resolver todas las traducciones en un solo acceso. Supondremos también que no hay fallos de página.

Supongamos el siguiente código que se ejecuta en 0,424s. Las etiquetas "V" y "bucle" están alineadas a 1MB:

```
        movl $1000000, %ecx
        movl $0, %eax
bucle:  addl V(,%ecx,4), %eax
        decl %ecx
        jnz bucle
```

a) **Calcula** el número de accesos físicos a memoria principal producidos al ejecutar el bucle. ¿**Cuál** es el tiempo de acceso a la memoria principal?

Instrucciones ejecutadas = 3 * 1000000 = 3000000
Accesos a memoria = 4 (3 ins + 1 datos) * 2 (1 tabla páginas + 1 @física) * 1000000 = 8000000

Ciclos programa = 3000000 * 8 + 8000000 * (ta-1) = 424000000 = 24 000000 + 50*8 000000
-> ta -1 = 50 -> ta = 51 ciclos

Vista la ineficiencia del sistema decidimos incorporar al procesador un TLB y una memoria cache. La memoria cache que pensamos usar tiene líneas de 32 bytes. Tanto el TLB como la cache son unificados para instrucciones y datos.

b) Teniendo en cuenta que el TLB es completamente asociativo con reemplazo LRU, ¿**cuál** es el tamaño mínimo de TLB que nos permitiría tener el máximo número de aciertos en el bucle anterior?

2 entradas, 1 para datos y otra para instrucciones

Finalmente decidimos utilizar 2 caches (y 2 TLBs) separados de instrucciones y datos. Cada TLB tiene 8 entradas y cada cache 128 lineas. La cache de instrucciones es de acceso directo y la de datos 4 asociativa. Nuestro sistema con TLB y cache tiene un CPI ideal de 6 ciclos. Las páginas del sistema son de 4 KB.

c) ¿**Cuántos** fallos de cache dará en total el bucle anterior?

1 instrucciones y 1000000 / 8 de datos -> 125001

d) ¿**Cuántos** fallos de TLB dará en total el bucle anterior?

1 de instrucciones y 1000000 / 1024 -> 1 + 977 = 978