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.

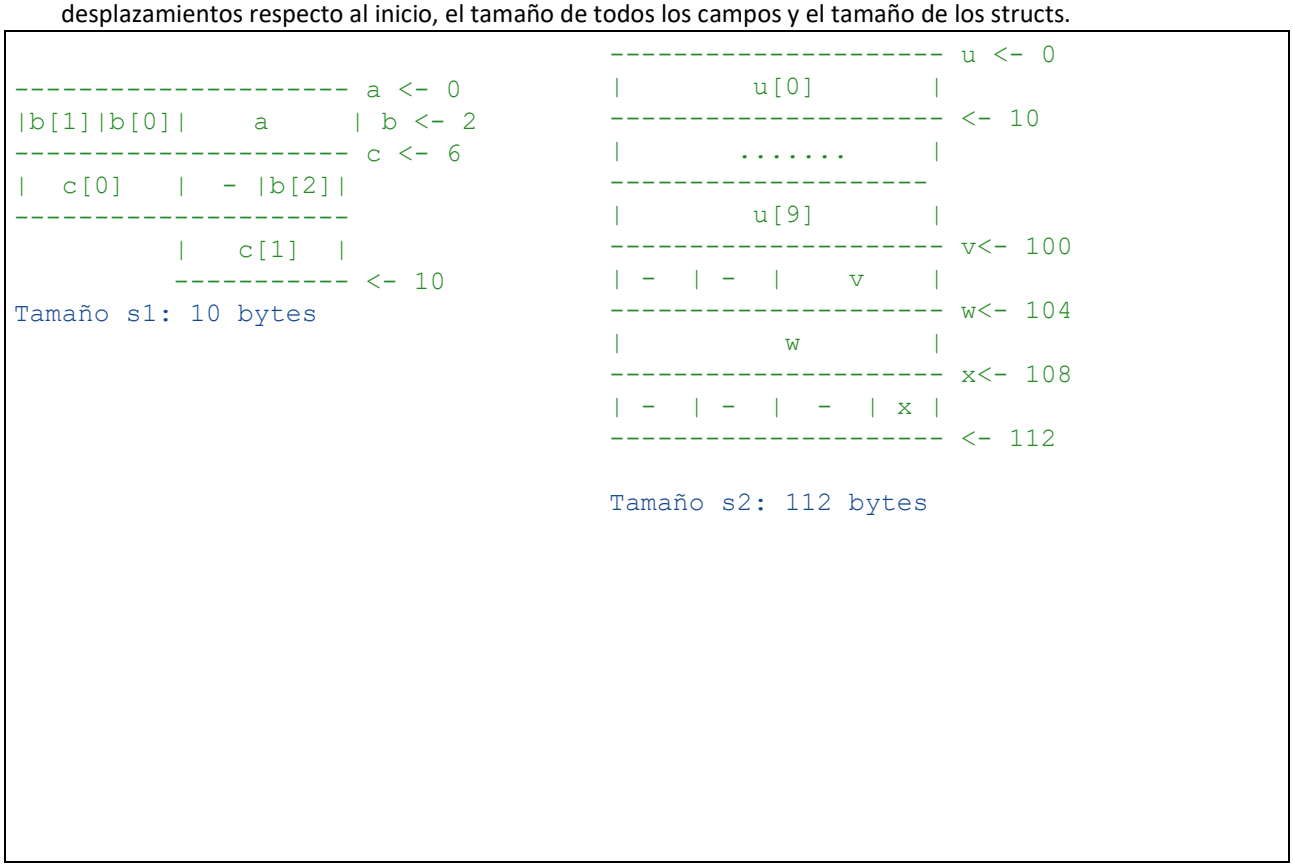
Problema 1. (5 puntos)

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

```
typedef struct {
  short a;
  char b[3];
  short c[2];
  } s1;
```

```
typedef struct {
  s1 u[10];
  short v;
  int w;
  char x;
  } 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.u[8].c[0]** a los 16 bits de menor peso del registro **%edx**, 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.

La expresión aritmética para calcular la dirección del operando es: @x+10*8+6, por lo tanto %ecx+86

La instrucción es: **movw 86(%ecx), %dx**

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

```
int examen(char a, char b[3][3], short c) {
  short x;
  char y[3][3];
  short z;
  int w;
  . . .
  w=examen(y[2][2],y,z);
  . . .
}
```

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

```
examen
----------------------------- x<- ebp-20
|y[0,1] y[0,0]|      x      | y<- ebp-18
----------------------------- <- ebp-16
|y[1,2] y[1,1] y[1,0] y[0,2]|          |
----------------------------- <- ebp-12
|  -    | y[2,2]y[2,1]y[2,0] |
----------------------------- z<- ebp-8
|  --     --    |    z        |
----------------------------- w<- ebp-4
|             w               |
-------------------------------
|          ebp                | <- ebp
-------------------------------
|          RET                |
-------------------------------
|  --   --   --        |  a    | a <- ebp+8
-------------------------------
|             @b              | @b <- ebp+12
-------------------------------
|  --   --      |     c       | c <- ebp+16
----------------------------- <- ebp+20
```

d) **Traduce** a ensamblador x86 la instrucción `w=examen(y[2][2],y,z);` que se encuentra en el interior de la subrutina, usando el mínimo número de instrucciones.

```
pushl -8(%ebp)
leal -18(%ebp), %eax
pushl %eax
pushl -10(%ebp)
call examen
addl $12, %esp
movl %eax, -4(%ebp)
```

COGNOMS:

NOM:

Problema 2. (5 puntos)

En un ordenador en el que tenemos instalado el entorno usado en el laboratorio de AC, hemos medido que un programa de 1000 instrucciones ensamblador se ha ejecutado en 2 segundos usando $6x10^9$ ciclos y ha ejecutado $4,8x10^9$ instrucciones y $2,4x10^8$ operaciones de coma flotante que debido a la falta de hardware específico ejecutan 10 instrucciones cada una.

a) **Calcula** el CPI del programa y la frecuencia de la CPU (usa el prefijo del sistema internacional más adecuado).

CPI = 6x10^9 ciclos / 4,8x10^9 instrucciones = 1,25 c/i

Frec = 6x10^9 ciclos / 2 segundos = 3x10^9 ciclos/segundo = 3 GHz

b) **Calcula** los MIPS y MFLOPS a los que se ejecuta el programa.

MIPS=4,8x10^9 instrucciones/ 2 segundos / 10^6 = 2400 MIPS

MFLOPS= 2,4x10^8 operaciones / 2 segundos / 10^6 = 120 MFLOPS

El tiempo de ejecución usado en el primer apartado se corresponde al tiempo de CPU (usuario + sistema). Usando el comando "time" de linux hemos obtenido que el tiempo de CPU representa solo el 20% del tiempo total del programa (wall time). El 80% restante es tiempo de entrada/salida (accesos al disco duro concretamente). Cada acceso al disco duro del sistema tarda 8 milisegundos, mientras que si los datos estuviesen en un disco SSD cada acceso tardaría 10 microsegundos.

c) **Calcula** la ganancia total en el programa que se obtendría con el cambio de tipo de disco.

Ganancia Disco = 8x10-3 s/a / 10x10-6 s/a = 800

Ganancia = 1/((1-fm)+fm/gm) = 1/((1-0,8)+0,8/800) = 4,975

A pleno rendimiento, la CPU funciona a una frecuencia de 3 GHz y está alimentada a 1,6 V. En modo bajo consumo la CPU funciona a una frecuencia de 0,8 GHz y está alimentada a 1 V. Hemos medido que el consumo de la CPU en alto rendimiento es de 120W y en modo bajo consumo es de 25 W. En estos datos solo se considera la potencia debida a conmutación y la debida a fugas. Tanto la corriente de fugas (I) como la carga capacitiva equivalente (C) son las mismas en ambos modos.

d) **Calcula** la corriente de fugas (I) y la carga capacitiva equivalente (C) de la CPU (usar prefijo más adecuado del SI) .

$P = Pc + Pf = V^2*F*C + I_f*V$

(1,6 V)^2 * 3x10^9 Hz * C + 1,6 V * I = 120 W (alto rendimiento)
(1 V)^2 * 0,8x10^9 Hz * C + 1 V * I = 25 W (bajo consumo)

resolvemos sistema de 2 ecuaciones lineales con 2 incognitas .....

C = 12,5 nF I = 15 A

Pare el resto del problema tendremos en cuenta solo la fase de calculo del programa, es decir solo tiempo de CPU (usuario + sistema).

e) **Calcula** la ganancia en energía que tendría el sistema si ejecutara el programa en el modo de bajo consumo en vez del modo de alto rendimiento suponiendo que el CPI medio no varía.

N y CPI no varian , el tiempo de Bc sera proporcional a la variación de frecuencia
Tiempo Bc = 2s * 3GHz/0,8Gz = 7,5 s

Ear = 120W * 2s = 240 J
Ebc = 25W * 7,5 s = 187,5

G = 240/187,5 = 1,28

Este procesador tiene direcciones físicas de 32 bits, una cache de datos de primer nivel (L1) 4-asociativa con tamaño de bloque 64 bytes y política de escritura *Copy Back + Write Allocate*. Las etiquetas (TAGS) de la cache son de 18 bits.

f) **Calcula** el numero de bloques (líneas) de la cache.

64 bytes/bloque -> 6 bits de offset (byte)
32 bits (@) - 18 bits (TAG) - 6 bits (offset) = 8 bits (conjunto)
256 conjuntos * 4 (asociatividad) = 1024 bloques

El procesador dispone ademas de un TLB que se accede en paralelo a L1.

g) **Calcula** el tamaño mínimo que pueden tener las páginas de memoria virtual para que sea posible el acceso paralelo a cache y TLB.

256 bloques / vía * 64 bytes / bloque -> 16Kbytes /vía
Tamaño Página >= Tamaño vía
Página >= 16K bytes

Para la fase de cálculo el tiempo medio de acceso a memoria (Tma) es de 1,3 ciclos. En caso de acierto en L1 el tiempo de acceso es de un ciclo. En caso de fallo hay una penalización (Tpf) de 10 ciclos adicionales si el bloque reemplazado tiene el *dirty bit* D=0 y de 20 ciclos si el bloque reemplazado tiene D=1. Sabemos que en media el 50% de los bloques tiene D=1. La influencia de los fallos de TLB y de los fallos de página es despreciable.

h) **Calcula** la tasa de fallos de la cache L1.

tpf = 0,5*10 + 0,5*20 = 15
Tma = Tsa + m*tpf
1,3 = 1 + m*15 - > m = 2%