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.

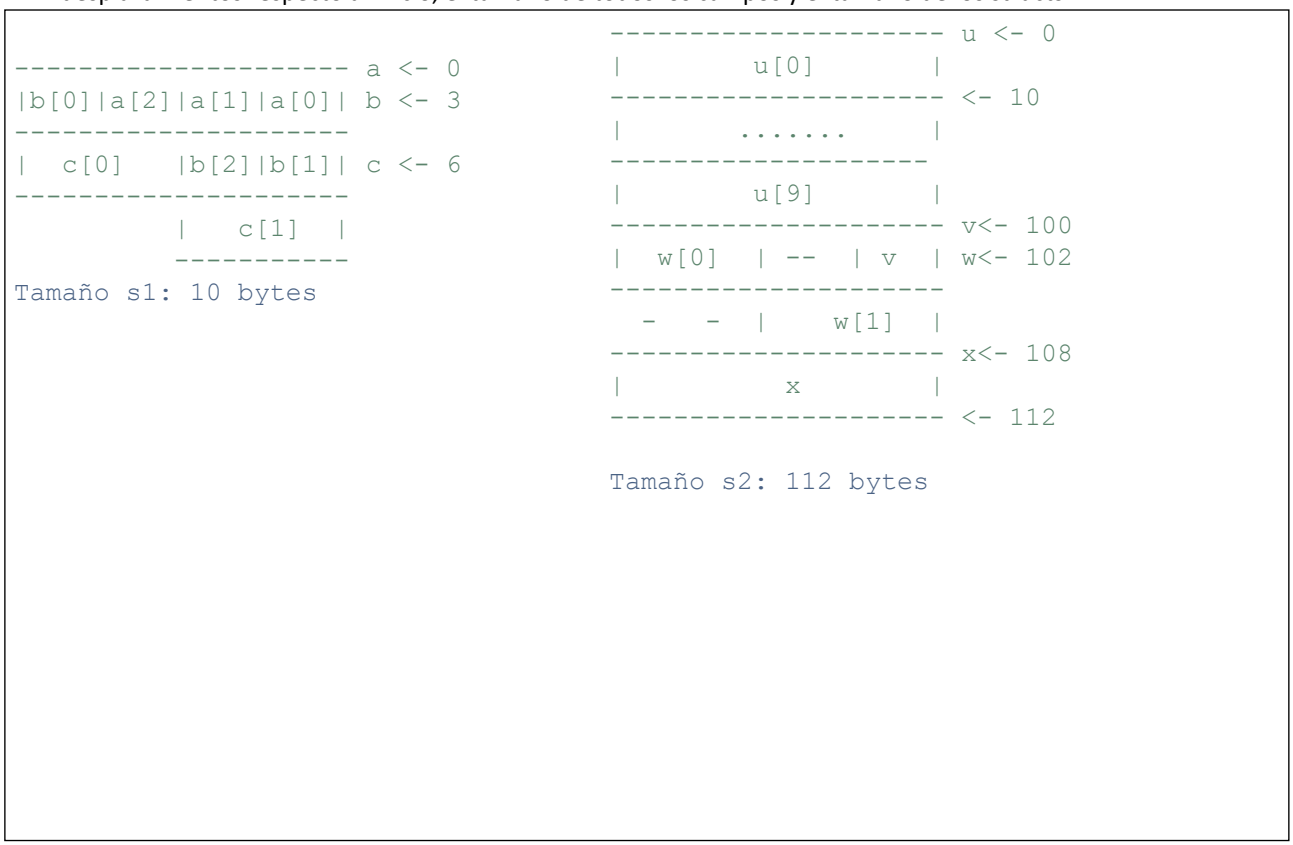
Problema 1. (5 puntos)

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

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

```
typedef struct {
  s1 u[10];
  char v;
  short w[2];
  int 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[5].b[1]** al registro **%dh**, 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*5+4, por lo tanto %ecx+54

La instrucción es: **movb 54(%ecx), %dh**

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

```
int examen(char b[2][3], char c, short d) {
  char y[2][3];
  short z;
  short w;
  int x;
  . . .
  x=examen(y,y[0][1],w);
  . . .
}
```

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
---------------------------- y<- ebp-16
|y[1,0] y[0,2] y[0,1] y[0,0]|          |
---------------------------- <- ebp-12
|      z      | y[1,2]y[1,1]  | z<- ebp-10
---------------------------- w<- ebp-8
|  --     --    |     w        |
---------------------------- x<- ebp-4
|           x                 |
-----------------------------
|          ebp                | <- ebp
-----------------------------
|          RET                |
-----------------------------
|         @b                  | @b <- ebp+8
-----------------------------
|  --   --   --         |  c   | c <- ebp+12
-----------------------------
|  --   --      |      d       | d <- ebp+16
-----------------------------
```

d) **Traduce** a ensamblador x86 la instrucción `x=examen(y,y[0][1],w);` que se encuentra en el interior de la subrutina, usando el mínimo número de instrucciones.

```
pushl -8(%ebp)
pushl -15(%ebp)
leal -16(%ebp), %eax
pushl %eax
call examen
addl $12, %esp
movl %eax, -4(%ebp)
```

COGNOMS:

NOM:

Problema 2. (5 puntos)

Un programa P tiene un 90% del código que es perfectamente paralelizable.

a) **Calcula** el numero mínimo de procesadores para conseguir un speed-up de 5 en el programa P.

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

Cada CPU funciona a un frecuencia de 3 GHz. Se ha ejecutado el programa P secuencial en un simulador con una única CPU ideal donde todos los accesos a memoria tardan un ciclo. Dicho programa ejecuta $6x10^9$ instrucciones, realiza $3x10^9$ operaciones de punto flotante y tarda $9x10^9$ ciclos.

b) **Calcula** el CPI ideal (CPI_{ID}) y el tiempo de ejecución en segundos (Texec) del programa P en este sistema ideal.

CPI_{IDEAL} = $9x10^9$ ciclos / $6x10^9$ instrucciones = **1,5 ciclos / instrucción**

Texe = Ciclos / F= $9x10^9$ ciclos / 3GHz = **3 seg**

c) **Calcula** los MIPS y MFLOPS en dicho sistema ideal.

MIPS = Instrucciones / Texe*10^6 = $6x10^9$ instrucciones / 3*10^6 = **2000 MIPS**

MFLOPS = OpsPF / Texe*10^6 = $3x10^9$ opsPF / 3*10^6 = **1000 MFLOPS**

La implementación (CPU_R) de dicha CPU dispone de una cache con una política de escritura *Copy Back* y *Write Allocate*. En caso de acierto en la cache el tiempo de acceso es de 1 ciclo. En caso de fallo, el tiempo de penalización es de 75 ciclos para reemplazar un bloque NO modificado y de 150 ciclos para reemplazar un bloque modificado.

El programa P realiza $9,6x10^9$ accesos a memoria, con una tasa de fallos (miss) del 10%. Sabemos que el 25% de los accesos son escrituras y que la probabilidad de que un bloque haya sido modificado en cache es del 20%.

d) **Calcula** el tiempo medio de acceso a memoria (Tma) para el programa P en CPU_R.

Tma = Tsa+m*(Pm*Tpfm + Pn*Tpfn) = 1 + 0,10*(0,20*150+0,8*75) = **10 ciclos/acceso**

e) **Calcula** el CPI del programa P en la CPU_R.

nr = $9,6x10^9$ accesos / $6x10^9$ instrucciones = 1,6 a/i
CPI = CPIid + nr*(Tma-1) = 1,5 c/i + 1,6 a/i * 9 c/a = 15,9 c/i

Alternativa: CPI = CPIid + CPImem = 1,5 + 1,6*0,10*(0,20*150+0,80*75) = 15,9 c/i

Cada acceso a memoria principal consume 100 nanoJoules (nJ).

f) **Calcula** el consumo total de energía de la memoria principal causada por los fallos de cache.

fallo que reemplaza bloque NO modificado genera 1 acceso a MP (lectura bloque)
fallo que reemplaza bloque modificado genera 2 accesos a MP (escritura bloque + lectura bloque)
AccesosMP = accesos * m * (Pn*1 + Pm *2) = $9{,}6x10^9$ * 0,1* (0,8*1 + 0,2*2) = $1{,}152x10^9$accesos a MP

E = $1{,}152x10^9$a * 100 nJ = **115,2 Joules**

Dicha CPU genera direcciones lógicas de 36 bits y direcciones físicas de 24 bits. La jerarquía completa de memoria está compuesta por un TLB (al que se accede ANTES de acceder a la cache), la memoria cache y la memoria principal. El TLB tiene 4 entradas y es completamente asociativo. La cache tiene un tamaño de 64 Kbytes, líneas de 64 bytes y es 4-asociativa. El tamaño de página del sistema es de 4 KBytes

g) **Calcula** el número de líneas, vías y conjuntos que tiene la cache. **Especifica claramente** cómo has realizado los cálculos.

64*1024 bytes / 64 bytes/linea = 1024 líneas
4-asociativa -> 4 vias
1024 lineas / 4 lineas/conjunto = 256 conjuntos.

La CPU lanza un acceso a la dirección lógica 0xEFABCD012 y sabemos que el contenido del TLB es:

VPN	PPN
0xFABCD0	0xA00
0xEFABCD	0xB01
0xABCD01	0xC02
0xBCD012	0xD03

h) **Indica** a qué dirección física se accede, en qué conjunto de la cache se encuentra el dato y cuál es la etiqueta guardada en memoria cache. **Justifica** la respuesta.

VPN = 0xEFABCD -> PPN = 0xB01 **@Fisica = 0xB01012**

VPN (24 bits)	desplaçament (12 bits)

Conjunto = 0x40 TAG = 0x2C0

TAG (10 bits)	cjt (8)	byte (6)

i) **Indica** el tamaño máximo que puede tener la cache para que sea posible acceder a la cache y al TLB en paralelo, suponiendo que se mantiene el tamaño de línea y el grado de asociatividad, y que se mantienen también el resto de parámetros de la jerarquía de memoria. **Justifica** la respuesta.

El tamaño máximo de la cache viene delimitado por los bits de byte + los bits de conjunto. Estos bits pueden direccionar como máximo una página del sistema (4 KBytes --> 12 bits).

Bits de byte = 6 -> número máximo de bits de conjunto = 6.

Tamaño máximo de cache = 64 conjuntos * 4 lineas/conjunto * 64 bytes/linea = 16 KBytes.