COGNOMS:

NOM: DNI/NIE:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos, el nombre y el DNI/NIE 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, todo lo que haya fuera de ellos es ignorado. La identificación del alumno se realiza de forma automática, no seguir correctamente estas instrucciones puede comportar no tener nota.

Problema 1. (3 puntos)

Se dispone de la siguiente definición de estructura de un programa (izquierda) y del siguiente fragmento de código (derecha), escrito en C, que compilamos para un sistema linux de 32 bits:

```
typedef struct {
  char a[3];
  char b;
  char c[5];
  } s1;
```

```
int examen(short f, char e[7], s1 d) {
 s1 g;
 char h;
 short i;
 int j, k;
 int v[16];
 int M[16][32];
     . . .
  v[k]=M[k][j];
   . . .
}
```

a) **Dibuja** cómo quedaría almacenada en memoria la estructura **s1**, indicando claramente los desplazamientos respecto al inicio, el tamaño de todos los campos y el tamaño y alineamiento del struct.

b) Suponiendo que el vector V de la subrutina examen se encuentra almacenado a partir de la dirección simbólica @V, y que la matriz M se encuentra almacenada a partir de la dirección simbólica @M, **escribe** las expresiones aritméticas para calcular las direcciones del elemento V[i] del vector V y el elemento M[k][j] de la matriz M. .

@V[i] =

@M[k][j]=

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 `v[k]=M[k][j];` que se encuentra en el interior de la subrutina examen, usando el mínimo número de instrucciones, suponiendo que la variable k está almacenada en el registro %edx y la variable j está almacenada en el registro %ecx. Existe una solución con 4 instrucciones.

COGNOMS:

NOM: DNI/NIE:

Problema 2. (3,8 puntos)

Dado el siguiente código escrito en ensamblador del x86:

```
        movl $0, %ebx
        movl $0, %esi
for:    cmpl $256*1024, %esi
        jge end
(a)     movl (%ebx, %esi, 4), %eax
        shll $2, %eax
(b)     addl %eax, 4*1024(%ebx, %esi, 4)
(c)     addl 8*1024(%ebx, %esi, 4), %eax
        addl $1024, %esi
        jmp for
end:
```

a) **Indica** el número de iteraciones ejecutadas del bucle

Número iteraciones	

Sabemos que para los accesos a memoria de la instrucción (a), se accede a las siguientes páginas de la memoria virtual en la iteración indicada en la tabla siguiente.

Iteración	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
a	0	0	1	1	2	2	3	3	4

b) **Indica** el tamaño de la página del sistema

Tamaño de página	

Suponiendo que el sistema dispone de un TLB de 4 entradas completamente asociativo con reemplazo FIFO, responde a las siguientes preguntas:

c) Para cada uno de los accesos (etiquetas (a), (b) y (c)), **indica** a qué página de la memoria virtual se accede en cada una de las 16 primeras iteraciones del bucle

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
a	0	0	1	1	2	2	3	3	4							
b																
c																

d) Indica el contenido del TLB después de ejecutar la instrucción (c) en las 16 primeras iteraciones del bucle (si la entrada está vacía, indícalo con una X)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
TLB 0																
TLB 1																
TLB 2																
TLB 3																

e) **Calcula** la cantidad de fallos y aciertos de TLB en TODO el bucle

f) Si cambiamos el TLB, a uno de 2 entradas completamente asociativo con reemplazo FIFO, **indica** para cada una de las métricas listadas si su valor se incrementará, se reducirá o quedará igual respecto al TLB de 4 entradas utilizado hasta ahora. **Justifica** tu respuesta:

	Aumentará	Igual	Decrementará	Justificación
Número de accesos al TLB				
Número de fallos de TLB				
Tamaño de la tabla de páginas				

g) Si cambiamos el tamaño de página a 32KB y mantenemos el TLB de 4 entradas utilizado, **indica** para cada una de las métricas listadas si su valor se incrementará, se reducirá o quedará igual respecto al TLB de 4 entradas utilizado hasta ahora. **Justifica** tu respuesta:

	Aumentará	Igual	Decrementará	Justificación
Número de accesos al TLB				
Número de fallos de TLB				
El número de entradas de la tabla de páginas si mantenemos el tamaño de la memoria virtual y de la memoria física				
El número de entradas de la tabla de páginas si mantenemos el tamaño de la memoria virtual y aumentamos la memoria física				
El número de entradas de la tabla de páginas si mantenemos el tamaño de la memoria virtual y reducimos la memoria física				

COGNOMS:

NOM: DNI/NIE:

Problema 3. (3,2 puntos)

Dado el siguiente fragmento de código escrito en lenguaje C:

```
#define N 3
#define M 100
double a[N][M], b[M+1][N]; // recuerda: un double ocupa 8 bytes

main(){
   int i,j;
   . . .
   for (i = 0; i < N; i++){
      for (j = 0; j < M; j++){
         a[i][j] = b[j][0] * b[j+1][0];
      }
   }
   . . .
}
```

Disponemos de una cache de Datos de 8KB de mapeo directo con bloques de 16 bytes y políticas de escritura **copy-back + write-allocate**. Las matrices a y b no se encuentran en cache al comienzo de la ejecución del código. Ambas matrices se almacenan consecutivamente en memoria y su tamaño total es inferior al de la cache, por lo que no habrá fallos de conflicto.

a) **Indica** si los accesos a las matrices a y b están aprovechando o no el tipo de Localidad Espacial y porqué.

b) **Indica** cuáles son las 2 ocasiones en las que los accesos a la matriz b están aprovechando el tipo de Localidad Temporal y porqué.

c) **Rellena** la siguiente tabla indicando para cada uno de los accesos a las matrices a y b, si es acierto (AC) ó fallo (FA) en cada una de las 5 primeras iteraciones del for más interior (j) y para las iteraciones con los valores i=0 e i=2 del for más exterior.

	iteración i = 0					**iteración i = 2**				
iteración j	**0**	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**4**
a										
b_j										
b_{j+1}										

d) **Calcula** la cantidad de aciertos y de fallos de cache en todo el código.

Si todos los accesos a memoria fuesen acierto en cache, cada iteración del bucle interno tardaría 7 ciclos en ejecutarse (los ciclos consumidos en el externo son despreciables). Un fallo de añade una penalización de 100 ciclos.

e) **Calcula** cuántos ciclos tarda en ejecutarse el código anterior, teniendo en cuenta los fallos de cache.

Tras optimizar el código (insertando instrucciones de prefetch), hemos conseguido reducir en un 92.4% los fallos de cache. En este caso, si todos los accesos a memoria fuesen acierto en cache, el código optimizado se ejecutaría en 2500 ciclos.

f) **Calcula** cuántos ciclos en total tarda en ejecutarse el código optimizado, teniendo en cuenta los fallos de cache.

Sabemos que la frecuencia del procesador es de 667 MHz.

g) **Calcula** a cuántos MFLOPs se ejecutan el código original y el código optimizado.

h) **Calcula** cuál es la ganancia que obtenemos con esta optimización.