COGNOMS:

NOM: DNI:

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 mayusculas 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. (2,5 puntos)

Dado el siguiente código escrito en C y compilado para x86 linux 32 bits:

```
typedef struct {
   char c;
   short s[4];
   char c2;
   int i2;
} sx;
int examen(sx *p1, sx s1, short sh){
   int i;
   char ch;
   short aux;
   int *x;
   . . .
}
```

a) **Dibuja** como quedaría almacenada la estructura **sx,** indicando claramente los desplazamientos respecto del inicio y el tamaño de todos los campos

```
----------------------
|  s[0]   |  -- | c | c <- +0; s[0]<- +2
----------------------
|  s[2]   |  s[1]   | s[1]<- +4; s[2]<- +6;
----------------------
| --  c2  |  s[3]   | s[3]<- +8; c2 <-10
----------------------
|        i2         | i2<- +12 // tamaño 16 bytes
----------------------
```

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

```
---------------------
|         i         | i<- ebp-12;
---------------------
| aux     | --  ch  | ch <- ebp-8 ; aux<- ebp-6
---------------------
|       *x          | *x <- ebp-4
---------------------
|        ebp        | <- ebp
---------------------
|        RET        |
---------------------
|        *p1        | p1<- ebp+8
---------------------
|  s[0]   |  -- | c |  c<- ebp+12; s[0]<- ebp+14
---------------------
|  s[2]   |  s[1]   | s[1]<- ebp+16; s[2]<- ebp+18;
---------------------
| --  c2  |  s[3]   | s[3]<- ebp+20; c2<- ebp+22
---------------------
|        i2         | i2<- ebp+24
---------------------
| --   -- |   sh    | sh<- ebp +28
---------------------
```

c) **Traduce** a ensamblador x86 la sentencia **return(*x + s1.i2);** suponiendo que esta dentro la función examen:

```
movl -4(%ebp),%ecx
movl (%ecx), %eax
addl 24(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
```

d) **Traduce** a ensamblador x86 la sentencia **(*p1).s[2] = aux;** suponiendo que esta dentro la función examen:

```
movl 8(%ebp),%ecx
movw -6(%ebp),%ax
movw %ax,6(%ecx)
```

COGNOMS:

NOM: DNI:

Problema 2. (3,5 puntos)

Se dispone de un computador C con una memoria RAM de 64 MB (que corresponde al espacio físico direccionable). El procesador genera direcciones lógicas de 32 bits y está conectado a una memoria caché de datos de primer nivel L1D con política de escritura copy back-write allocate. La memoria cache L1D tiene un tamaño de 3 KB, es 3-asociativa y el tamaño de línea es de 64 bytes. El sistema dispone también de un TLB de 4 entradas, completamente asociativo y con política de reemplazo LRU. Se accede simultáneamente a TLB y cache. El tamaño de página del sistema operativo es de 4KB.

a) Un proceso P del sistema lanza un acceso de lectura que se mapea en la dirección física 0x2345678. Este acceso produce un acierto de lectura en L1D. **Indica** claramente cuántos bits tiene cada uno de los campos tag, conjunto y byte con que se accede a la cache, y **escribe EN HEXADECIMAL** el tag, el conjunto y el byte dentro de la línea al que se accederá en memoria cache L1D.

64 MB de RAM => 26 bits de @Física
L1D: 64 bytes/línea => 6 bits de línea
Tamaño 3KBytes, 3-asociativa => 3KB * 1 cjto/3 líneas * 1 línea/64 Bytes = 16 conjuntos => 4 bits conjunto
TAG: 26 bits @Física - 6 bits de línea - 4 bits conjunto = 16 bits

TAG	CONJUNTO	BYTE
16	4	6

tag=8d15 conjunto=9 byte=38

b) **Calcula** el tamaño en **MEGABYTES** de la tabla de páginas del proceso P, teniendo en cuenta que cada entrada de la tabla, además de los bits necesarios para codificar la página física, tiene un bit de presencia y un bit de página modificada.

@ Virtual 32 bits, @ Física 26 bits, tamaño página 4 KB (12 bits de desplazamiento)

@Virtual: 12 bits de desplazamiento, 32-12=20 bits de página virtual

@Física: 12 bits de desplazamiento, 26-12=14 bits de página física

Tamaño Tabla de páginas: 2^20 entradas de (14+2) bits = 2^20 * 16 = 2^20 * 2^4 = 2^24 =16 Mbits = 2 MBytes

El proceso P contiene el siguiente código escrito en ensamblador del x86:

```
      movl $0, %ebx
      movl $0, %esi
for:
      cmpl $256*1024, %esi
      jge end

  (a) movl 0x2000(%ebx, %esi, 4), %eax
  (b) movb %bh, 0x2000(%ebx, %esi, 4)
  (c) addl %eax, 0x3000(%ebx, %esi, 4)
  (d) movw %ax, 0x5000(%ebx, %esi, 4)

      addl $1, %esi
      jmp for
end:
```

c) **Calcula** la **tasa de fallos** de la cache para el fragmento de código anterior suponiendo que inicialmente está vacía.

Las 4 instrucciones (a), (b), (c) y (d) acceden al mismo conjunto de la cache.

Las instrucciones (a) y (b) acceden además a la misma línea dentro del conjunto,por lo que los accesos de (b) serán siempre aciertos. Las instruccions (c) y (d) acceden a líneas distintas del conjunto, pero como la cache es 3-asociativa no hay conflicto entre los accesos de (a)-(b), (c) y (d).

La instrucción (a) produce un fallo de cache la primera vez que se ejecuta, pero los 15 accesos siguientes son aciertos, porque se accede a la misma línea de cache (línea de 64 bytes, operandos de 4 bytes, 16 operandos por línea a los que se accede de forma consecutiva).

Los accesos de la instrucción (b) son siempre aciertos.

La instrucción (c) realiza dos accesos, el primero de lectura y el segundo de escritura. El segundo acceso será siempre un acierto. El primer acceso seguirá el mismo patrón que para la instrucción (a)

Los accesos de la instrucción (d) siguen el mismo patrón que para (a).

Cada 16 iteraciones se sigue el siguiente patrón de aciertos y fallos:

(a) 1F 15A

(b) 16A

(c) 1F, 15A+16A

(d) 1F, 15A

Cada 16 iteraciones se producen, por lo tanto, 80 accesos, de los cuales 3 son fallos.

m=3/80= 0,0375, o 3,75% de fallos

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

iteración	0	1*256	2*256	3*256	4*256	5*256	6*256	7*256	8*256	9*256
a	2	2	2	2	3	3	3	3	4	4
b	2	2	2	2	3	3	3	3	4	4
c	3	3	3	3	4	4	4	4	5	5
d	5	5	5	5	6	6	6	6	7	7

e) **Calcula** el **número de fallos** de TLB en el fragmento de código.

El TLB es completamente asociativo de 4 entradas. Para la secuencia del apartado d) se produce un fallo de TLB la primera vez que se acccede a una página, pero el resto de accesos son aciertos de TLB. Por ejemplo, en la iteración 0 se producen fallos en los accesos a las páginas 2 (la primera vez que se accede, la otra es una cierto), 3 y 5, pero el resto de iteraciones todos los accesos son aciertos hasta la iteración 4*256. En esa iteración se producen dos aciertos en la página 3 y dos fallos en las páginas 4 y 6. Al final de la iteración, el TLB contiene las páginas 5, 3, 4 y 6. Las siguientes iteraciones son todo aciertos hasta la iteración 9*256, en la que se producen tres aciertos al acceder a las páginas 4 y 5 y un fallo al acceder a la página 7. Al final de la iteración, el TLB contiene las páginas 6, 4, 5 y 7. Este patrón de comportamiento se repite hasta el final del bucle.

Como se produce un fallo de TLB sólo la primera vez que se accede a una página, calcular el número de fallos de TLB es equivalente a calcular el número de páginas a las que accede el código.

Número de páginas:

(256*1024 iteraciones * 4 bytes/iteración * 1 página / 4096 bytes) + 3 = 259 páginas => 259 fallos

Es preciso añadir 3 páginas extra, ya que hay 3 páginas de distancia entre los accesos (a) y (d)

COGNOMS:

NOM: DNI:

Problema 3. (4 puntos)

Tenemos un procesador (que llamaremos Pmp) sin memoria cache, todos los accesos se realizan directamente sobre memoria principal. En este procesador hemos ejecutado una aplicación A (que usaremos a lo largo de todo el problema) y hemos obtenido los siguientes datos: tiempo de ejecución 18 s, $2x10^9$ instrucciones ejecutadas, $3x10^9$ accesos a memoria (incluye tanto los accesos a datos como los accesos a instrucciones). Sabemos además que Pmp funciona a una frecuencia de 2 GHz. Para simplificar el problema supondremos que todos los accesos son lecturas.

a) **Calcula** el CPI del procesador Pmp al ejecutar la aplicación A (CPIpmp).

18 s * $2x10^9$ c/s = $36x10^9$ ciclos
CPIpmp = $36x10^9$ c / $2x10^9$ i = **18 c/i**

Con un simulador hemos simulado una versión ideal (que llamaremos Pideal) del procesador, en donde todos los accesos a memoria tardan un ciclo. En el Pideal hemos obtenido un CPI (CPIideal) de 3 ciclos/instrucción.

b) **Calcula** el tiempo de penalización de los accesos a memoria (en ciclos) del procesador Pmp respecto el Pideal.

Ciclos mem = Ciclos Pmp - Ciclos Pideal = $36*10^9$ c - 3 c/i * $2x10^9$ i = $30*10^9$ ciclos
Penal accesos = $30x10^9$ ciclos / $3x10^9$ accesos = **10 ciclos/acceso**

-- alt --
CPImem = CPIpmp- CPIideal = 18 c/i - 3 c/i = 15 c/i
nr = accesos/instr = $3x10^9$ a/$2x10^9$ i = 1,5 a/i
Penalización/acceso = CPImem / nr = 15c/i / 1,5 a/i = **10 ciclos/acceso**

Para mejorar el rendimiento respecto el procesador Pmp, añadimos una cache 2-asociativa al procesador (que denominaremos procesador P2a). Para poder mantener la frecuencia de 2 GHz, los accesos a la cache 2-asociativa tardan 2 ciclos, es decir que en caso de acierto tenemos una penalización de 1 ciclo. La penalización media en caso de fallo es de 17 ciclos. La tasa de fallos de la cache 2-asociativa es del 5%.

c) **Calcula** el tiempo de ejecución de la aplicación A en el procesador P2a

Ciclos P2a = ciclos Pideal + ciclos penalizacion hits + ciclos penalización miss =
= 3 c/i * $2x10^9$ i + $3x10^9$ a * 0,95 h/a* 1 c/h + $3x10^9$ a * 0,05 m/a* 17 c/m = $11,4x10^9$ ciclos
Texe = ciclos / F = $11,4x10^9$ ciclos / $2x10^9$ c/s = **5,7 s**

-- alt ---
CPImem = nr * ((1-m) * 1 + m * 17) = 1,5*(0,95*1+ 0,05*17) = 2,7c/a
CPI2a = CPIideal + CPImem = 3 c/i + 2,7 c/i = 5,7 c/i
Texe = N*CPI*Tc = $2x10^9$ i * 5,7 c/i / $2x10^9$ c/s = **5,7 s**

Para reducir la penalización de accesos a la cache, uno de los ingenieros ha sugerido el uso de un predictor de vía. El predictor sugerido tendría una tasa de aciertos en la predicción de vía del 80% y tendría un impacto negligible tanto en área como en la frecuencia y consumo del procesador. Al procesador con cache 2-asociativa y predictor de vía lo denominaremos procesador Ppv. En caso de que el predictor acierte la vía no hay penalización respecto el procesador ideal. Si hay fallo de predictor pero acierto de cache se incurre en un ciclo de penalización (tal como ocurría con la cache 2-asociativa) ya que se necesita un ciclo adicional para acceder a la otra vía. Finalmente, si es fallo de predictor y también de cache, la penalización es de 17 ciclos (también la misma que la cache 2-asociativa).

d) **Calcula** el CPI del procesador Ppv (CPIppv).

80% acierto predictor -> 0 ciclos extra
15% fallo predictor acierto cache -> 1 ciclo extra
5% fallo predictor fallo cache -> 17 ciclos extra

Ciclos Ppv = Ciclos Pideal + ciclos fallo predictor acierto cache + ciclos fallo predictor fallo cache =
= 3 c/i * $2x10^9$ i + $3x10^9$ a * 0,15 mh/a* 1 c/mh + $3x10^9$ a * 0,05 mm/a* 17 c/mm = $9x10^9$ ciclos
CPIppv = ciclos / instr = $9x10^9$ ciclos / $2x10^9$ i = **4,5 c/i**

-- alt --
CPIp = CPI ideal + nr*(mphc*tpmp + m*tpf) = 3 c/I + 1,5 a/I * (0,15 mh/a * 1 c/mh + 0,05 mm/a * 17c/mm) = **4,5 c/i**

Queremos saber la potencia media debida a conmutación de la jerarquía de memoria del procesador P. Ignoraremos por tanto la potencia disipada por fugas así como la potencia de conmutación del procesador, y también la del predictor que ya hemos comentado que tiene un impacto despreciable. Sabemos que cada vez que se accede una vía de la cache (etiquetas + datos) se consumen 5 nJ (nanojoules) y cada vez que hay un fallo de cache se consumen 30 nJ adicionales.

e) **Calcula** la energía (de conmutación) consumida por la jerarquía de memoria del procesador Ppv al ejecutar la aplicación A

Si hay acierto de predictor solo se accede una via (5 nJ), si hay fallo de predictor se acceden las 2 vias (10 nJ) y si es fallo de cache 30 nJ adicionales (40 nJ en total)

Energia = $3x10^9$accesos * (0,8 * $5x10^{-9}$ J + 0,15 * $10x10^{-9}$ J + 0,05 * $40x10^{-9}$ J) = **22,5 Joules**

f) **Calcula** la potencia media (de conmutación) consumida por la jerarquía de memoria del procesador Ppv al ejecutar la aplicación A

P = 22,5 Joules / 4,5 s = **5 W**