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. (3 puntos)

Disponemos de un procesador de señal (DSP) que ejecuta el siguiente kernel (obsérvese que es un bucle infinito). **Las variables cambian de valor en cada iteración** ya que están mapeadas a diversos sensores/actuadores que las actualizan constantemente. Por ello **deben leerse y escribirse en cada iteración**.

```
while (true) {
    A=(F+B*C)-(D*E);
}
```

Las dos alternativas de las que disponemos son un procesador de tipo Acumulador y otro de tipo Memoria/Memoria. La descripción del ISA de ambos procesadores es la siguiente:

		Acumulador		**Memoria / Memoria**	
Tipo ins.	Opcodes	Ejemplo	Operación	Ejemplo	Operación
Aritmética	add, mul, sub, div	sub A	ACC=ACC-A	sub A, B, C	C=A-B
Memoria	load,store	load A	ACC=A	--	--
Salto	br	br Loop	PC=PC+despl	br Loop	PC=PC+despl

El siguiente código muestra la traducción a ensamblador del procesador Memoria/Memoria del kernel anterior. NO es posible optimizar el código haciendo loads/stores fuera del bucle dado que los datos son distintos en cada iteración.

```
Loop:                                   Instrucciones dinámicas/iteración: 5
  mul D,E,r1
  mul B,C,r2
  add F,r2,r2
  sub r2,r1,A
  br Loop
```

a) **Traduce** el bucle a ensamblador del procesador Acumulador usando el menor número de instrucciones posible. Podéis usar variables temporales tmp1, tmp2, ... si lo necesitáis. Escribid cuantas instrucciones dinámicas se ejecutan en cada iteración del bucle.

```
Loop:                                   Instrucciones dinámicas/iteración: 9
  load D
  mul E
  store tmp1
  load B
  mul C
  add F
  sub tmp1
  store A
  br Loop
```

Sabemos que la memoria de instrucciones es el cuello de botella del sistema y por tanto el rendimiento del procesador estará únicamente limitado por el ancho de banda con dicha memoria. La memoria de instrucciones es capaz de ofrecer, para el kernel anterior, un ancho de banda sostenido de 18 GB/s. Cada instrucción del procesador Acumulador ocupa 4 bytes y cada instrucción del procesador Memoria/Memoria ocupa 8 bytes.

b) **Justifica** cuantitativamente cuál es el procesador capaz de ejecutar el código más rápidamente y **calcula** cuál es su ganancia con respecto al más lento para el código dado (Pista: calculad cuántas iteraciones por segundo puede hacer cada procesador)**.**

Acumulador: es capaz de realizar 18E9bytes/s / 36 bytes/iter = 500E6 iter/s

Mem/Mem : 18E9bytes/s / 40 bytes/iter = 450E6 iter/s

El procesador acumulador será más rápido y su ganancia será de:
S= 500E6/450E6 = 1,11 con respecto al Mem/Mem.

Este DSP está conectado a una memoria principal formada por 1 DIMM de memoria DDR con 8 chips de 1 byte por DIMM y cada chip contiene un único banco. La latencia de fila es de 3 ciclos, la latencia de columna es de 2 ciclos y la latencia de precarga es de 1 ciclo.

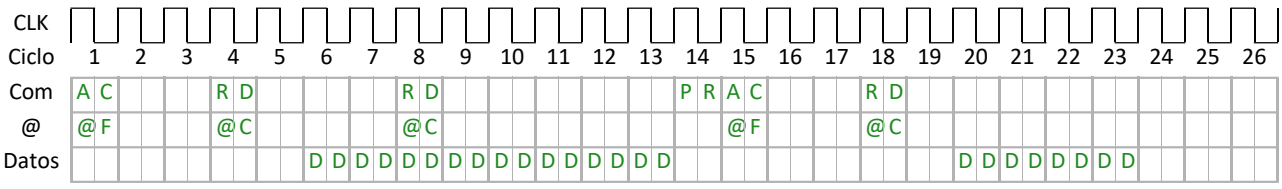
A la memoria DDR se realiza la siguiente secuencia de 3 accesos en los que se lee un bloque de 64 bytes en cada acceso: página X, página X, página Y

El sistema tiene un controlador de memoria que no cierra la página después de cada acceso. En caso de que un acceso se realice sobre una página abierta, no es necesario abrirla. Sin embargo si el acceso se realiza sobre una página distinta, tenemos que cerrar la página anterior y abrir la página que se desea acceder.

Para indicar la ocupación de los distintos recursos utilizaremos la siguiente nomenclatura:

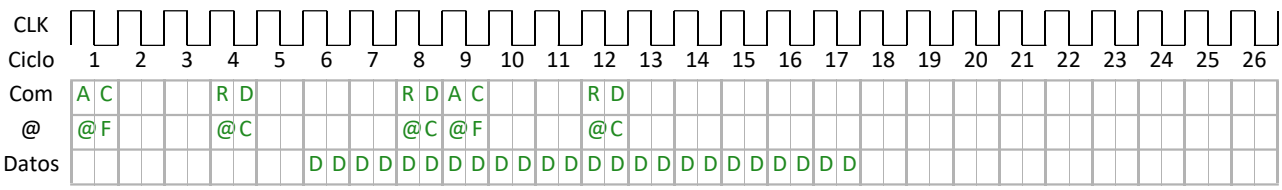
- AC: ciclo en que se envía el comando ACTIVE
- RD: ciclo en que se envía el comando READ
- PR: ciclo en que se envía el comando PRECHARGE
- @F: ciclo en que se envía la dirección de fila
- @C: ciclo en que se envía la dirección de columna
- D: transmisión de un paquete de datos

c) **Rellena** el siguiente cronograma, indicando la ocupación de los distintos recursos del sistema con **1 banco** por chip (inicialmente no hay ninguna página abierta), de forma que la secuencia se realice en el número mínimo de ciclos:



Una mejora adicional consiste en usar chips con múltiples bancos cada uno, de forma que solo se cierra página si accedemos a una página distinta a la abierta en el mismo banco. Sabemos que las páginas X e Y se encuentran en bancos distintos.

d) **Rellena** el siguiente cronograma, indicando la ocupación de los distintos recursos del sistema con **2 bancos** por chip (inicialmente no hay ninguna página abierta), de forma que la secuencia se realice en el número mínimo de ciclos:



COGNOMS:

NOM:

Problema 2. (3,5 puntos)

Tenemos un procesador conectado directamente a la memoria principal para los datos y las instrucciones, que llamaremos procesador **SIN**. En media, las instrucciones tienen un CPI de 181 ciclos/instr (incluyendo tanto el acceso a la memoria de instrucciones como la de datos). En este procesador, ejecutamos un código que tiene un 10% de las instrucciones que realizan 1 acceso a memoria de datos y un 20% de las instrucciones que realizan 2 accesos a la memoria de datos. Todas las instrucciones acceden a la memoria de instrucciones.

a) **Calcula** cuántos accesos a memoria por instrucción realiza el procesador **SIN** en media.

N = Número instrucciones

Núm accesos= Núm_accesos_inst + Núm_accesos_datos = 1*N + 0,1*1*N + 0,2*2*N

Accesos/Inst= 1,5*N/N= 1,5

El CPI del procesador si la memoria fuera ideal (1 ciclo de acceso siempre), es de 1 ciclo, a este procesador lo llamaremos **IDEAL**.

b) **Calcula** la penalización por acceso a memoria del procesador **SIN** respecto al **IDEAL**.

CPI = 181=1 + 1,5*Pmem

Pmem=120

Los ingenieros de computadores han decidido introducir una nueva versión de la memoria principal, una cache de instrucciones y una cache de datos. La latencia de acceso a las caches es de 1 ciclo. Si todos los accesos que realiza una instrucción son acierto en cache, esta se ejecuta con un CPI de 1 como en el procesador **IDEAL**, por lo que el CPI ideal (siempre acertamos en la cache) es de 1 ciclo. El tiempo de penalización en caso de fallo es de 60 ciclos para ambas caches. La tasa de fallos de la cache de instrucciones es de un 10%. A este procesador lo llamaremos procesador **CON**. Para simplificar el problema, asumir que en el procesador **CON** todos los accesos a datos son lecturas.

c) **Calcula** el CPI del procesador **CON** en función de la tasa de fallos de la cache de datos (t_f):

CPIcon = 1 + tf_inst*60+accesos_datos/inst*tf_datos*60

CPIcon= 1 + 0,1*60 + 0,5*tf*60= 7 + 30*tf

Como tenemos varios diseños de cache de datos con distintas tasas de fallos (t_f), debemos escoger el diseño que nos de el rendimiento deseado. En nuestro caso queremos que este código se ejecute 10 veces más rápido en el procesador **CON** que en el procesador **SIN**. Asumir que ambos funcionan a la misma frecuencia.

d) **Calcula** la mayor tasa de fallos posible de la cache datos para que el código se ejecute 10 veces más rápido en el procesador **CON** respecto al procesador **SIN**:

Tsin=N*CPIsin*Tc ; Tcon=N*CPIcon*Tc

Tsin/Tcon=10=CPIsin/CPIcon

10=181/(7+30*tf) => tf= 0,37

Después de un largo debate entre los ingenieros, la cache de datos implementada es write-through y write-no-allocate. Esta cache tiene una tasa de fallos de 0,2 tanto para lecturas como para escrituras. La tasa de fallos de la cache de instrucciones continua siendo de un 10%. Todas las instrucciones tienen un consumo base de 10nJ. Además, si acceden a cualquier cache tiene un consumo por acceso (escritura o lectura) de 30nJ. Un acceso (tanto de bloque como de palabra) a la memoria principal de datos consume 400nJ. En el código analizado, un 40% de los accesos a memoria de datos son escrituras. A este procesador lo llamaremos procesador **WT**.

e) **Calcula** la ganancia en energía por instrucción del procesador **WT** respecto al procesador **SIN**.

Esin = Ebase + accesos/inst*400 = 10 + 1,5*400=10+600=610nJ

Ewt = Ebase + Ecache-(inst+datos) + Einst-mem + Edatos-lecturas-mem + Edatos-escrituras_mem=

Ewt = Ebase + accesos/inst*Ecache + accesos_inst/inst*Emem + accesos_datos/inst*lecturas/accesos_datos*tfallo*Emem + accesos_datos/inst*escrituras/accesos_datos*Emem =

=10 + 1,5*30 + 0,1*400 + 0,5*0,6*0,2*400 + 0,5*0,4*400 = 10+ 45+40+24+80=199nJ

Esin/Ewt=610/199=3,06x

El procesador **WT** funciona a una frecuencia de 2GHz y tiene un CPI de 13 ciclos/instrucción.

f) **Calcula** la potencia en Watios del procesador **WT** si la frecuencia de operación es de 2GHz

N= Numero de instrucciones

Tc=1/f=0.5ns

P=E/T = (Ewt * N)/(N*CPIwt*Tc) = Ewt/(CPIcwt*Tc)

$P=199*10^{-9}/(13*0,5*10^{-9}) = 30,62W$

Debido al impacto en el rendimiento de los accesos a memoria principal, los diseñadores quieren introducir un buffer de escrituras entre la cache de datos y la memoria principal. Este buffer funciona como una cola y va guardando **todas** las escrituras que se realizan. Una vez se escribe en el buffer, el procesador sigue ejecutando el código (no se espera a que se escriba en memoria). Cuando la memoria está libre, el buffer escribe en la memoria principal. En el caso de una lectura de la memoria principal, primero se mira si los datos están en el buffer, si no lo están, se accede -después- a la memoria principal. Para simplificar el problema asumiremos que el buffer no se llena nunca y por tanto el procesador no se bloqueará por falta de espacio en el buffer. A este procesador lo llamaremos **BUFFER**.

Sólo un 10% de las lecturas encuentran el dato en el buffer. Cada acceso al buffer (lectura o escritura) consume 50nJ.

g) **Calcula** la energía media por instrucción del procesador **BUFFER**:

Ewt = Ebase + Ecache-(inst+datos) + Einst-mem + Edatos-lecturas-buffer +Edatos-lecturas-mem + Edatos-escrituras_buffer-i-mem=

Ebuffer= Ebase + accesos/inst*Ecache + accesos_inst/inst*Emem + accesos_datos/inst*lecturas/accesos_datos*tfallo*Ebuffer + accesos_datos/inst*lecturas/accesos_datos*fallo_lectura_buffer/escrutras*Emem + accesos_datos/inst*escrituras/accesos_datos*(Ebuffer+Emem)=

= 10 + 1,5*30 + 0,1*400 + 0,5*0,6*0,2*50+ 0,5*0,6*0,2*0,9*400 + 0,5*0,4*(50+400) =

= 209.6nJ

h) **Calcula** la latencia máxima en ciclos del buffer para que el procesador **BUFFER** sobrepase los 165 MIPS.

Lmem= 60 (latencia memoria)
Lbuffer = (latencia buffer)

CPIbuf = CPIbase + Pinst-mem + Pdatos-lecturas-buffer + Pdatos-lecturas-mem +Pdatos-escrituras-buffer=

CPIbuf = CPIbase + tfallo_inst*Lmem + accesos_datos/inst*lecturas/accesos_datos*tfallocache*Lbuffer + accesos_datos/inst*lecturas/accesos_datos*tfallocache*tfallobuffer*Lmem + accesos_datos/inst*escrituras/accesos_datos*Lbuffer = 1 + 0,1*60 + 0,5*0,6*0,2*lat + 0,5*0,6*0,2*0,9*60+0,5*0,4*lat= 10,24+0,26*lat

$MIPS = 1/(CPI*Tc)=165*10^{6}=1/(CPI*0,5*10^{-9})$ => CPI<=12,12 => lat= 7

COGNOMS:

NOM:

Problema 3. (3,5 puntos)

Queremos evaluar el rendimiento de un programa P en un sistema, que llamaremos PC1, con un solo procesador single-core y un solo disco duro D. Hemos tomado algunos valores de rendimiento ejecutando la aplicación en el PC1 y vemos que el programa se ejecuta en dos fases bien diferenciadas:

• Fase 1, o Fase de Lectura, en el que lee un bloque de datos de 300 GB del disco D almacenado en posiciones consecutivas. El disco D tiene un ancho de banda de 200 MB/s.

• Fase 2, o Fase de Cálculo, en el que se realizan $45 \cdot 10^{12}$ operaciones de coma flotante. El PC1 ejecuta esta fase a una velocidad de cómputo de 3 GFLOPS.

En todo el problema, no vamos a considerar los tiempos de búsqueda ni latencia de los discos.

a) **Calcula** el tiempo de ejecución de las Fases de Lectura y Cálculo, y el tiempo total de la aplicación en el PC1.

$T_{lectura}$ = 300 GB / 200 MB/s = 1500 s
$T_{cálculo}$ = $45 \cdot 10^{12}$ flops / $3 \cdot 10^{9}$ flops/s = 15000 s
$T_{PC1} = T_{lectura} + T_{cálculo}$ = 16500 s

Con el objeto de reducir el tiempo del programa, se baraja la opción de sustituir el procesador del PC1 por un sistema multiprocesador, que llamaremos PC2. Sin embargo, calculamos que solo el 80% de la Fase de Cálculo es paralelizable y que debido a la paralelización del código estaremos introduciendo una sobrecarga por la sincronización necesaria entre procesos. Estimamos que la sobrecarga introducida será de 15 segundos por cada procesador que se añada.

b) **Calcula** el número ideal de procesadores que debería tener el PC2 para obtener una ganancia máxima. **Calcula** la ganancia obtenida respecto al PC1. (Pista: recuerda que una manera de obtener el máximo de una función es encontrar la primera derivada, igualar a 0 y resolver)

$T_{secuencial} = T_{lectura} + T_{cálculo} \cdot 0,2$ = 4500 s
$T_{paralelo} = T_{cálculo} \cdot 0,8$ = 12000 s
T_{PC2} = 4500 + 12000/N + 15·(N-1)
Derivando e igualando a 0 --> N ≈ 28

T_{PC2} = 5333,57 s
G = 16500 / 5333,57 = 3,09

Después de acudir al mercado, decidimos optar por una propuesta más económica y adquirir un multiprocesador que, junto al disco D, montaremos en un nuevo sistema que llamaremos PC3. En este sistema, calculamos que la aplicación completa se ejecutará 3 veces más rápido que en el PC1.

Sabemos que la Fase de Cálculo del programa P tiene $25 \cdot 10^{3}$ instrucciones estáticas y ejecuta $60 \cdot 10^{12}$ instrucciones dinámicas, de las cuales $50 \cdot 10^{12}$ son instrucciones de coma flotante.

c) **Calcula** los MIPS y GFLOPS de la Fase de Cálculo al ejecutar el programa P en el PC3 sabiendo que este ejecutará un 20% de instrucciones más (en la fase de cálculo) respecto al PC1 debido a la sincronización.

$T_{cálculo\_PC3}$ = 16500/3 - 1500 = 4000 s
MIPS = $(1,2 \cdot 60 \cdot 10^{12}$ instr. / $10^{6}) / 4000$ s = 18000 MIPS
GFLOPS = $(45 \cdot 10^{12}$ flops / $10^{9}) / 4000$ s = 11,25 GFLOPS

Otra opción que se ha barajado para mejorar el rendimiento del sistema PC1 es añadir un RAID de discos en lugar del disco duro D. Un RAID nos permite paralelizar la Fase de Lectura, ya que en esta fase hay suficientes accesos para saturar el ancho de banda de todos los discos, además de añadir un mecanismo de tolerancia a fallos en disco. Para ello, disponemos de 6 discos iguales con ancho de banda de 200 MB/s y un sistema RAID que puede configurarse como **RAID 10** o **RAID 5**.

d) **Describe** las principales características de cada uno de estos sistemas RAID, dibujando un esquema de cómo se distribuyen los datos y especificando el tipo de entrelazado, el porcentaje de información redundante, el número de discos que han de fallar para que el sistema deje de ser operativo, el ancho de banda **máximo** de las lecturas y el ancho de banda **máximo** de las escrituras.
NOTA: Considerad el mejor de los casos entre accesos secuenciales y aleatorios.

RAID 10:
Entrelazado a nivel de tira. El 50% de la información es redundante. Es fiable ante el fallo de un disco cualquiera, pero podrían fallar hasta tres discos si están en mirrors distintos. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6 x 200 MB/s = 1,2 GB/s), mientras que el de las escrituras es sólo la mitad porque se ha de escribir en cada disco y en su mirror (600 MB/s).

RAID 5:
Entrelazado a nivel de tira. Tiene la paridad distribuida entre todos los discos, de forma que en total 1/6 de la información es redundante. Es fiable ante el fallo de un disco cualquiera. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6 x 200 MB/s = 1,2 GB/s). En el mejor caso, el ancho de banda de las escrituras sería el de escribir en 5 discos (en el sexto se escribiría la paridad de los otros 5), y por lo tanto 5 x 200 MB/s =1 GB/s.

Decidimos configurar el sistema de discos como **RAID 5** y montarlo en el PC3. A este sistema le llamamos PC4.

e) **Calcula** la ganancia al ejecutar el programa P en el PC4 respecto al PC1.

$T_{lectura\_PC4}$ = 300 GB / 1,2 GB/s = 250 s
$T_{PC4} = T_{lectura\_PC4} + T_{cálculo\_PC3}$ = 250 + 4000 = 4250 s
G = 16500 / 4250 = 3,88