# Estructura de Computadores

## Recuperación Parcial 2 — 21 Junio - 2018

**Nombre:** **Grupo:**

**1** ***(2 puntos)*** Un sistema basado en procesador MIPS R2000 posee una cache L1 dual configurada como sigue:

- **Cache de Instrucciones**: 1 KB, correspondencia asociativa por conjuntos de 4 vías
- **Cache de Datos**: 4 KB, correspondencia directa, política de NO ubicación en escritura (***write-no allocate***) con actualización posterior (*write-back*)
  Ambas cache poseen un tamaño de bloque de **8 bytes** y usan LRU para los reemplazos

El procesador ejecuta el siguiente código en alto nivel y ensamblador, respectivamente:

```
byte A[60];
...
for (int i=0; i<60; i=i+22)
    A[i]=A[i] & 0xF0;
for (int i=0; i<60; i=i+15)
  -   A[i]=0;
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
        .data 0x20180000
A:      .space 60
        ...
        .text0x01400000
        la $t0,A          # $t0 apunta al principio del vector
        addi $t1,$t0,60   # $t1 apunta al final del vector
for1:   lb $t2,0($t0)
        andi $t2,$t2,0xF0
        sb $t2,0($t0)
        addi $t0,$t0,22
        blt $t0,$t1,for1
        la $t0,A          # $t0 apunta de nuevo al principio del vector
for2:   sb $zero,0($t0)
        addi $t0,$t0,15
        blt $t0,$t1,for2
```

Nótese que la pseudo-instrucción *blt $t0,$t1,eti* (aparece dos veces en el programa) se ensamblará **en dos instrucciones máquina**.
Supóngase que todas las líneas de ambas caches son inválidas inicialmente. Analice el comportamiento de las cache de instrucciones y de datos durante la ejecución del código anterior

**Cache de Instrucciones** *(0.5 puntos)*

| | |
|---|---|
| ¿Cuántos bloques ocupan las instrucciones del programa? | 7 (0.1 punt) |
| Calcule el número de accesos a instrucciones | bucle *for1* itera 3 vegades (i=0, 22 i 44); bucle *for2* itera 4 vegades (i=0, 15, 30 i 45); en total: 2 + 3x6 + 1 + 4x4 = 37 accessos (0.2 punts) |
| ¿Cuál es la tasa de aciertos de la cache de instrucciones al ejecutar este código? | Hi ha set fallades. Per tant,<br>30/37 (0.2 punts) |

**Cache de Datos** *(1.5 puntos)*

| | |
|---|---|
| ¿Cuántos bloques ocupa el vector A? | 8 blocs (0.1 punt) |
| ¿Con qué etiqueta se almacenarán estos bloques en la memoria cache? | 0x20180 |
| ¿Cuál es el número de accesos a la cache de datos? | bucle *for1*: 3x2 accessos<br>bucle *for2*: 4 accessos<br>Total 10 accessos (0.2 punts) |
| ¿Cuántos fallos se producirán en la cache de datos al ejecutar el bucle *for1* del código? | 3 fallades:<br>(0.2 punts) |
| ¿Qué componentes del vector A fallan? | components 0, 22, i 44 |

| | |
|---|---|
| ¿Cuántos fallos se producirán en la cache de datos al ejecutar el bucle *for2* del código? | 2 fallades:<br>(0.2 punts) |
| ¿A qué componentes del vector A fallan? | components 15 i 30.<br>components 0 i 45 encerta |
| ¿Cuántas escrituras se harán sobre la memoria cache de datos? | 3 escriptures durant *for1*<br>2 escriptura durant *for2*<br>(0.2 punts) |
| ¿A qué componentes del vector A afectan? | *for1*: 0, 22 i 44;  *for2*: 0, 45 |
| ¿Cuántas escrituras se harán sobre la memoria principal? | 2 escriptures durant for2<br>(0.2 punts) |
| ¿A qué componentes del vector A afectan? | for2: 15 i 30 |
| ¿Cuál es la tasa de fallos de la memoria cache de datos? | (3+2)/10= 0.5<br>(0.1 punt) |
| ¿Cuántos bloques quedarán como válidos en la cache al término de la ejecución del código? | Els tres blocs que van fallar en *for1*<br>(0.2 punts) |
| ¿Cuántos bloques quedarán modificados en la cache al término de la ejecución del código? | Tots tres blocs vàlids<br>(0.1 punt) |

**2** ***(1 punto)*** Un procesador MIPS de 4GB de espacio de direccionamiento dispone de una memoria cache L1 dual de 64 + 64 KB. Ambas cache poseen un tamaño de bloque de **8 bytes** y usan LRU para los reemplazos

- **Cache de Instrucciones**: correspondencia asociativa por conjuntos de 8 vías
- **Cache de Datos**: correspondencia asociativa por conjuntos de 4 vías, con política actualización posterior (*write-back*)

Calcule el volumen del directorio para ambas caches (esto es, el número de bits de control de las mismas).

**Cache de Instrucciones** (64 KB, 8 bytes/bloque, 8 vías, LRU)

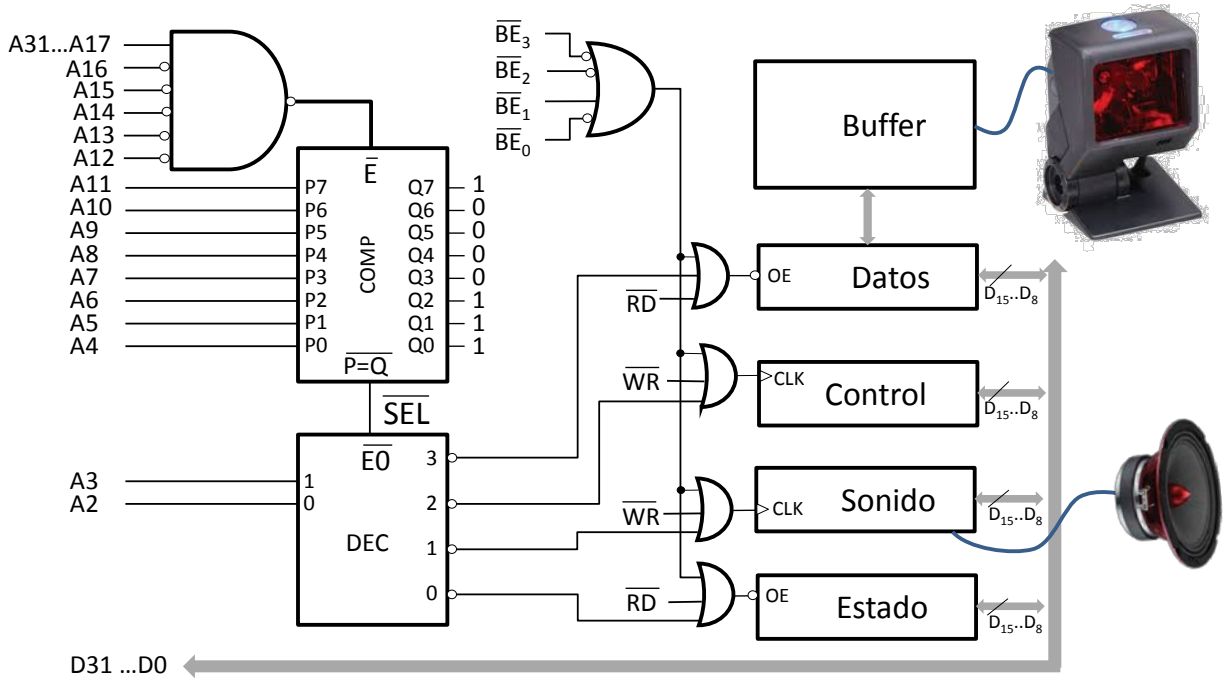
| | |
|---|---|
| Bits de etiqueta | 19 |
| Número de bits de control por línea | 23 (comptant V i LRU) |
| Número de líneas | 8192=2^13 |
| Número de conjuntos | 1024=2^10 |
| Volumen total del directorio (bits) | 188416 |

**Cache de Datos** (64 KB, 8 bytes/bloque, 4 vías, write-back, LRU)

| | |
|---|---|
| Bits de etiqueta | 18 |
| Número de bits de control por línea | 22 (comptant V, M i LRU) |
| Número de líneas | 8192 |
| Número de conjuntos | 2048 |
| Volumen total del directorio (bits) | 180224 |

**3** ***(4* puntos)** El código de barras EAN13 (European Article Number) mostrado en la figura es el utilizado habitualmente en los productos vendidos en Europa. Mediante un conjunto de barras verticales (negras y blancas) se codifica un valor numérico compuesto por 13 dígitos, distribuidos en 4 partes: Código del país (3 dígitos) + código de la empresa (4 o 5 dígitos) + código de producto (resto hasta 12 dígitos) y un dígito de control que se utiliza para validar el código.


Un lector de código de barras es el periférico con el que se puede leer dicho código. La figura siguiente muestra la interfaz de tal periférico a un sistema basado en un MIPS R2000. Cuando el lector se activa, éste escanea lo que se pone delante del lector y descodifica el código, transmitiendo la información (13 bytes) al buffer interno de la interfaz. Cada byte representa el valor binario del dígito correspondiente, leído de izquierda a derecha.




La descripción de los registros es la siguiente.

Registro **CONTROL**: (8 bits; sólo escritura)

- ACT (bit 7). Un '1' activa el lector. '0' lo apaga.
- IE (bit 3): Habilita interrupción *Int_5*.
- CL (bit 0): Un 1 hace que R=0 en el registro de estado (cancelar).

Registro **ESTADO**: (8 bits; sólo lectura)

- ERROR (bits 0 a 2): Código de error que el lector transmite cuando descodifica el código de barras. **Código correcto** (0). **Código incorrecto** (6).
- R (bit 3) se activa a 1 cuando el lector ha leído y transmitido el código de barras COMPLETO al buffer interno de la interfaz por lo que ya se puede leer. Si el bit IE está a '1', se activa la interrupción *Int5*.

Registro **DATOS**: (8 bits; sólo lectura)

- Código binario de cada dígito del código de barras. Para leer el código completo hay que hacer 13 lecturas de este registro.

Registro **SONIDO**: (8 bits; sólo escritura) Sirve para emitir tonos sonoros que indican al usuario si la lectura ha sido correcta o no.

- Sonido (bits 0 a 2): Sin sonido (0); Tono de inicialización (5); **Tono de código correcto** (1); **Tono de código incorrecto** (7).

Se pide:

a) *(0.2 puntos)* Indique la dirección base (DB) de la interfaz.

DB = 0xFFFE0870

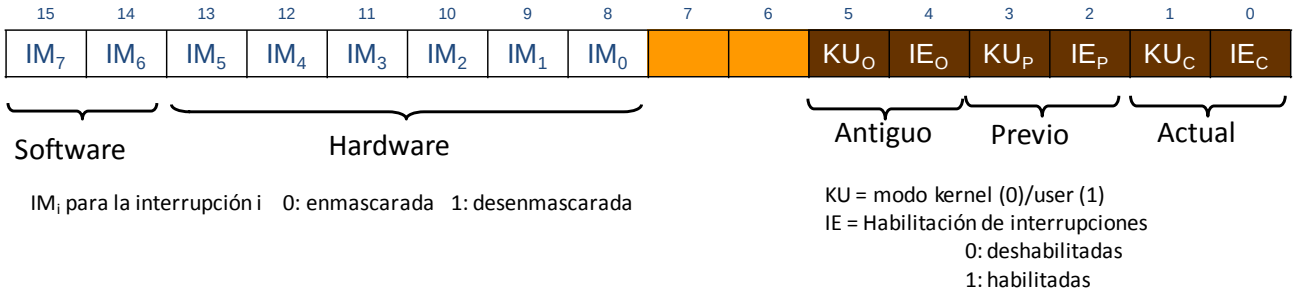
b) *(0.8 puntos)* Indique la dirección de cada uno de los registros en formato DB + 'n':

| Registro | Dirección | Registro | Dirección |
|---|---|---|---|
| ESTADO | DB + 1 | CONTROL | DB + 9 |
| SONIDO | DB + 5 | DATOS | DB + 13 |

Se dispone de varias funciones de sistema (*syscall*) para gestionar el lector, como se describen a continuación:

| Función | $v0 | Argumentos | Resultado | Resultados |
|---|---|---|---|---|
| *Inicializar_lector* | 400 | ------- | ------ | Apaga el sensor y emite el sonido de inicialización. Inicializa el sistema de interrupciones en la UCP. |
| *Leer_codigo_CE* | 410 | $a0: Puntero al buffer de memoria donde dejar el código | $v0: código error | Espera la lectura de un código de barras y lo copia en el buffer indicado. Sincronización por consulta de estado. |
| *Leer_codigo_INT* | 420 | $a0: Puntero al buffer de memoria donde dejar el código | -------- | Espera la lectura de un código de barras y lo copia en el buffer indicado. Sincronización por interrupción. |

El registro de estado de la UCP es el siguiente:




Se dispone también de dos variables del sistema, para ser usadas en su caso. La variable '*num_codigos_leidos*' debe contener el número de códigos de barras leídos correctamente hasta ese momento. La variable auxiliar '*dir_buffer*' contendrá la dirección del buffer de memoria del usuario donde dejar el código leído:

```
                        .kdata   0xA0000000
num_codigos_leidos: .word 0
dir_buffer:         .word 0
```

c) *(0.5 puntos)* Escriba el código de la función de sistema *'Inicializa_lector*. Las interrupciones en la UCP deben quedar habilitadas y **retornar a modo usuario**. En el periférico **NO** se debe habilitar la interrupción. También se deben inicializar a cero las variables del sistema..

```
Inicializa_lector:  la $t0, 0xFFFE0870
                    sb $zero, 9($t0)    # Apaga el lector IE=0
                    li $t1, 5
                    sb $t1, 5($t0)            # Tono de inicialización
                    sw $zero, num_codigos_leidos
                    sw $zero, dir_buffer
                    mfc0 $t1, $12
                    ori $t1,$t1,0x200C
                    mtc0 $t1, $12       # Desenmascara int_5, habilita interrupciones y retorna
                                          a modo usuario
                    b retexc
```

d) *(1.0 punto)* Escriba el código de la función de sistema *'Leer_codigo_CE'*. Activa el lector y espera por **consulta de estado** la lectura del código de barras. Una vez leído el código, éste se debe **transferir por PIO** al buffer cuyo puntero ha indicado el usuario en $a0. Se debe

retornar el código de error en $v0 y emitir el tono correspondiente *(obsérvese que éste coincide con el código de error + 1)*.

```
Leer_codigo_CE: la $t0, 0xFFFE0870
                li $t1, 0x80
                sb $t1, 9($t0)            # Activa lector IE=0
bucle_CE:       lb $t1, 1($t0)
                andi $t2, $t1, 0x08       # Bucle de consulta de estado
                beqz $t2, bucle_CE
                andi $v0, $t1, 0x07       # aíslo el código de error

                li $t2, 13
bucle_lect:     lb $t1, 13($t0)
                sb $t1, 0($a0)
                addi $t2, $t2, -1
                addi $a0, $a0, +1
                bnez $t2, bucle_lect      # Bucle de transferencia de datos

                li $t1, 0x01
                sb $t1, 9($t0)            # Apaga sensor IE = 0, CL=1 cancela

                bnez $v0, emitir_tono
                lw $t1, num_codigos_leidos # si código de error =o
                addi $t1, $t1, 1          # Incrementa variable num_codigos_leidos
                sw $t1, num_codigos_leidos

emitir_tono: addi $t1, $v0, 1             # El tono es igual al código de error + 1
                sb $t1, 5($t0)            # Emite el tono
                b retexc
```

e) *(0.5 puntos)* Escriba el código de la función de sistema *'Leer_codigo_INT'*. Activa el lector y espera **sincronización por interrupción**. Guarda la dirección del buffer en la variable `dir_buffer` del *kernel* para posterior uso en la rutina de interrupción. ***Nota**: Se asume un entorno multitarea en el que se hallan definidas las siguientes funciones: <suspende_este_proceso> y <activa_este_proceso>.*

```
Leer_codigo_INT: la $t0, 0xFFFE0870
                 li $t1, 0x88
                 sb $t1, 9($t0)           # Activa lector IE=1
                 sw $a0, dir_buffer
                 jal suspende_este_proceso
                 b retexc
```

f) *(1.0 punto)* Marque con una X, de entre la siguiente lista de tareas, aquellas que deberá realizar la rutina de servicio de interrupción ***Int5*** en respuesta al servicio previamente demandado por la función *'Leer_codigo_INT'*, dependiendo de si se detecta o no Error en la lectura del código. Dicha rutina deberá, en caso de error en la lectura del código, mantener el sistema preparado para volver a leer dicho código.

| Tarea | NO_Error | Error |
|---|---|---|
| Consultar si bit R=1 | | |
| Cancelar interrupción (CL=1) | X | X |
| Inhibir Interrupción (IE=0) | X | |
| Transferir código por PIO | X | X |
| Transferir código por ADM | | |
| Incrementar variable `num_códigos_leidos` | X | |
| Activar el proceso suspendido | X | |
| Retornar código de error en $v0 | | |

**4** ***(1.5 puntos)*** Los sensores de la serie PR-M/F de la marca Keyence son sensores fotoeléctricos miniatura que permiten detectar la presencia/ausencia de un objeto que atraviesa su haz láser. Se utilizan en las líneas de producción industrial para el conteo de piezas, como se indica en la figura.

Cada sensor se conecta a un sistema basado en MIPS R2000 mediante una interfaz compuesta por dos registros:

Registro **CONTROL**: DB + 0 (8 bits; sólo escritura)

- ACT (bit 7): Un '1' activa el sensor. '0' lo apaga.
- IE (bit 4): Habilita interrupción ***Int0***.
- CL (bit 0): Un 1 hace que R=0 (cancelar).

Registro **ESTADO**: DB + 4 (8 bits: sólo lectura)

- R (bit 7): se activa a 1 cuando el sensor detecta la presencia de una pieza. Si el bit IE está a '1' se activa la interrupción ***Int0***. Cuando se cancela (R=0) no se vuelve a activar hasta que pasa una nueva pieza.

En una línea de producción se han instalado estos sensores en dos cintas transportadoras para controlar el número de frascos que pasan por cada cinta. La dirección base (DB) de cada sensor se muestra en la tabla adjunta, pero **ambos sensores comparten la línea de interrupción *Int0*** del procesador.

| **Sensor** | **Dirección Base (DB)** |
|---|---|
| Cinta1 | 0xFFFF0010 |
| Cinta2 | 0xFFFF0020 |

Se dispone de 2 variables del sistema que contienen el número de frascos que han pasado por cada cinta.

```
                .kdata 0x00001000
Cinta1:         .word 0
Cinta2:         .word 0
```

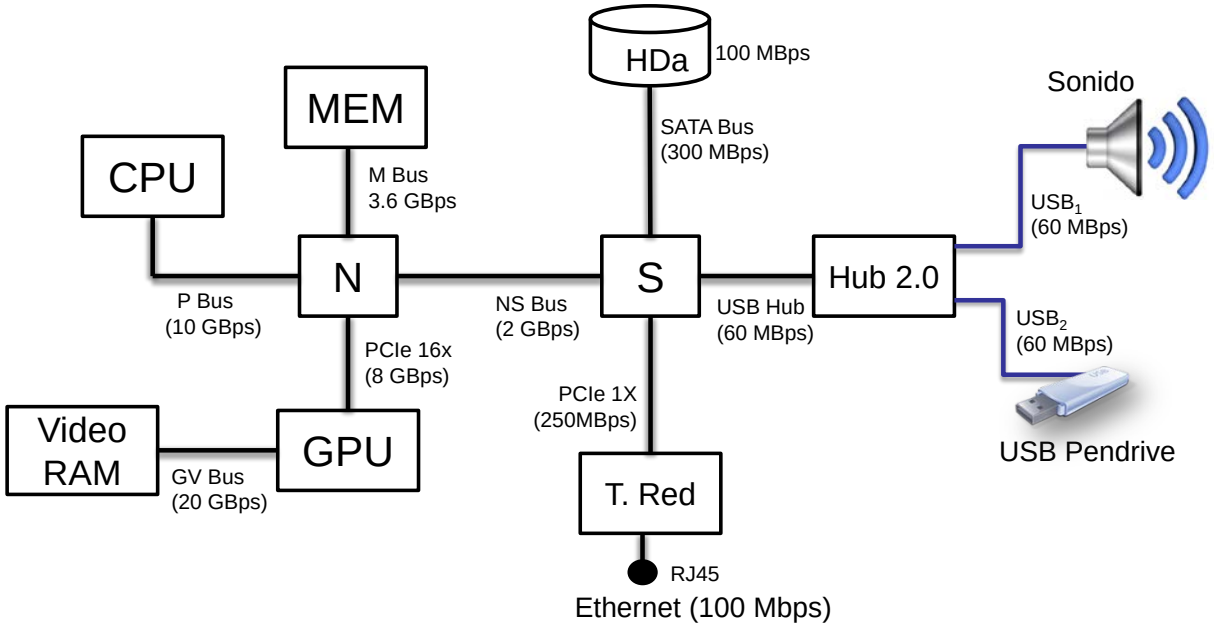
a) *(1.5 puntos)* Escriba el código de la rutina de servicio de la interrupción ***Int0***. Esta rutina debe actualizar la cuenta de frascos de cada cinta trasportadora.

```
Int_0:      la $t0, 0xFFFF0010        # Sensor 1
            lb $t1 0($t0)
            andi $t1, $t1, 0x80
            beqz $t1, Sensor2
            lw $t1, Cinta1
            addi $t1, $t1, 1
            sw $t1, Cinta1
            li $t1, 0x91              # cancelar, IE=1 Sensor activo
            sb $t1, 0($t0)

Sensor2:    la $t0, 0xFFFF0020        # Sensor 2
            lb $t1 0($t0)
            andi $t1, $t1, 0x80
            beqz $t1, retexc
            lw $t1, Cinta2
            addi $t1, $t1, 1
            sw $t1, Cinta2
```

```
                li $t1, 0x91              # cancelar, IE=1 Sensor activo
                sb $t1, 0($t0)
                b retexc
```

**5** ***(1 punto)*** Un alumno aplicado está estudiando EC en su computador reproduciendo un vídeo de la asignatura a través de la web. El audio/vídeo se transmite en formato MPEG 4 codificado a 32 Mbps. La tarjeta de red (10/100 Mbps) recibe la secuencia de datos y los transfiere por ADM a la memoria (MEM) del computador. Al mismo tiempo, la GPU lee la secuencia comprimida desde memoria, la descomprime y envía las imágenes a la RAM de Vídeo, y el audio al equipo de sonido, en ambos casos por ADM.




a) *(0.5 puntos)* Suponiendo que el vídeo descomprimido tiene una resolución de 1080x720x24 bits y 30 escenas/segundo y que el sonido es estéreo (audio 2.0), con muestreo a 48 KHz y 16 bits/muestra, calcule el ancho de banda (en MBps) requerido para:

Transferir el video comprimido desde la tarjeta de red a la memoria (MEM):

32 Mbps / 8bits = 4 MBps

Escribir las imágenes de vídeo desde la GPU a la RAM de vídeo:

1080×720× (24/8Bytes) × 30fps = 69.984 MBps

Enviar el audio desde la GPU al equipo de sonido:
2 canales × 48000 muestras por segundo × 16/8 bytes por muestra = 0.192 MBps

b) *(0.5 puntos)* Mientras se reproduce el vídeo, el alumno transfiere un archivo de 1GB ($10^9$ bytes) desde el disco duro al *pendrive* USB. La reproducción de audio/vídeo tiene prioridad sobre dicha transferencia. El archivo se lee del disco duro a la memoria (MEM) y desde ésta al *pendrive* mediante ADM. Lecturas y escrituras son concurrentes. Calcule el tiempo que cuesta transferir el archivo y el porcentaje de utilización de los buses indicados.
***Nota****: Todos los anchos de banda mostrados en el esquema son efectivos*

Tiempo de transferencia del archivo:

La velocidad que limita la transferencia es la del bus USB Hub, que soporta un tráfico de audio (0.192 MBps). El ancho de banda disponible será de 60 – 0.192 = 59.808 MBps. La transferencia del archivo requerirá de 1000 MB / 59.808 MBps =16.72 s

Porcentaje de utilización del bus NS:

NS = 4 + 0.192 + 2x59.808 / 2000 = 6.19%

Porcentaje de utilización del bus M:

M = 4 + 4 + 2x59.808 / 3600 = 3.544%

Porcentaje de utilización del bus SATA:

SATA = 59.808 / 300 = 19.936%

**6** ***(0.5 puntos)*** Sea un disco duro magnético formado por seis platos con formato ZCAV. El área útil de los platos es una corona circular de 2.4" de diámetro exterior y 0.4" de diámetro interior. El área útil se ha distribuido en 4 zonas de igual anchura con la siguiente distribución de sectores:

| | Zona 0 | Zona 1 | Zona 2 | Zona 3 |
|---|---|---|---|---|
| Sectores/pista | 1000 | 900 | 800 | 700 |

La densidad de pistas es de 200.000 tpi. El disco gira a 6000 rpm, el tiempo medio de posicionamiento de 6 ms, el *track-to-track time* de 0.5 ms y dispone de una conexión SATA de 3 Gbps con codificación 8/10.

a) *(0.25 puntos)* Calcule los siguientes parámetros del disco:

Anchura de cada zona = (Dext -.Dint)/2 /4 = 0.25":

Número de cilindros en cada zona= 200.000 x 0.25 = 50.000 cilindros en cada zona

Capacidad del disco en número de sectores: 12 caras x 50.000 ciclindros x (1000 + 900 + 800 + 700 sectores) = 2.040.000.000 sectores

Capacidad del disco en GB ($10^9$ B): 2.040.000.000 sectores x 512 bytes = 1.044,48 GB

b) *(0.25 puntos)* ¿Cuál es el tiempo medio para leer un archivo de 4 MB ($4x10^6$ B), ubicado en la Zona 0, y transferirlo al buffer interno de la interfaz? Se supone que el archivo está ubicado en el disco de forma óptima.

El archivo ocupará 4·10^6/512 = 7812.5~ 7813 **sectores**
Una vuelta de disco dura 10 ms; media vuelta 5 ms
En zona 0: puede alojarse en un cilindro (capacidad 12x1000 = 12000 sectores)
tiempo = 6 + 5 + 10 x 7813 sectores /1000 = 89.13 ms