

Estructura de Computadores

Parcial 2

7 Junio - 2017

Nombre:

Grupo:

1

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

- **Cache de Instrucciones:** 1 KB, correspondencia directa, tamaño de bloque de 16 Bytes.
- **Cache de Datos:** 1 KB, correspondencia asociativa por conjuntos de 2 vías, tamaño de bloque de 8 bytes. Emplea ubicación en escritura (*write-allocate*) con actualización posterior (*write-back*) y usa LRU para los reemplazos.

a) (0.5 puntos) Indique el número de bits de los campos de la dirección de memoria para ambas caches

Cache de Instrucciones		Cache de Datos	
Etiqueta	22	Etiqueta	23
Línea	6	Conjunto	6
Desplazamiento	4	Desplazamiento	3

b) (0.5 puntos) Calcule el tamaño de la memoria de control requerido por cada una de las caches

	Cache de Instrucciones	Cache de Datos
Número de entradas en la memoria de control	64	128
Número de bits de cada entrada (indique el nombre de los campos)	1 (Valid) + 22 (Tag) = 23 bits	1 (Valid) + 23 (Tag) + 1 (Dirty) + 1 (LRU) = 26 bits
Tamaño total de la memoria de control (en bits)	$64 \times 23 = 1472 \text{ bits}$	$128 \times 26 = 3328 \text{ bits}$

c) El siguiente programa obtiene la luminancia de una imagen en color a partir de su representación RGB. En una imagen RGB, cada pixel está representado por 3 valores correspondientes a los niveles de rojo, verde y azul. El programa trabaja sobre una imagen de 256 pixels, cuyas componentes Rojo, Verde y Azul se almacenan en vectores consecutivos de enteros sin signo (tamaño byte). Tratando de imitar la percepción del ojo humano, el programa calcula la luminancia de cada pixel como la media ponderada de sus componentes RGB. En particular, para el pixel i :

$$Lum[i] = (3 \times Rojo[i] + 4 \times Verde[i] + Azul[i]) / 8$$

```

Rojo:      .data 0x10000000
Verde:     .byte 0x45, 0x68, ... 0x2D      # Vector de 256 valores para el Rojo
Azul:      .byte 0x13, 0x22, ... 0xA0      # Vector de 256 valores para el Verde
Lum:       .byte 0x6A, 0x6C, ... 0x09      # Vector de 256 valores para el Azul
          .space 256                       # Espacio para el vector Luminancia

__start:   .text 0x00400000
          ori $s0, $zero, 0x100           # Num. de pixels a procesar (256 en decimal)
          lui $t0, 0x1000                 # Puntero al vector Rojo
loop:      lbu $t1, 0($t0)                 # Carga $t1 con Rojo [i]
          lbu $t2, 0x100($t0)              # Carga $t2 con Verde [i]
          lbu $t3, 0x200($t0)              # Carga $t3 con Azul [i]
          sll $t4, $t2, 2                  # $t4 = 4*Verde
          add $t4, $t4, $t1                # $t4 = 4*Verde + 3*Rojo
          add $t4, $t4, $t1                # $t4 = 4*Verde + 3*Rojo + Azul
          add $t4, $t4, $t3                # $t4 = (4*Verde + 3*Rojo + Azul) / 8
          sra $t4, $t4, 3                  # Almacena Lum[i]
          sb $t4, 0x300($t0)               # Incrementa puntero
          addi $t0, $t0, 1                 # Decrementa contador
          bne $s0, $zero, loop             # Sigue iterando mientras contador > 0
          .end

```

c.1) (0.5 puntos) Obtenga, para la **cache de instrucciones**:

Número de bloques de código	Primer número de bloque	Último número de bloque	Total de FALLOS de código
4	0x40000	0x40003	4
Total de ACCESOS a código (Indique el cálculo)	2 (before loop) + 256×13 (loop) = 3330		
Tasa de aciertos (Con cuatro dígitos decimales. Indique el cálculo)	$H = 3330 - 4 / 3330 = 0,9988$		

c.2) (0.5 puntos) Indique los número de bloque del primer y último bloque correspondientes a los cuatro vectores, así como los conjuntos en los que se almacenan en la **cache de datos (en hex)**

	Primer bloque	Último bloque	Primer conjunto	Último conjunto
Rojo	0x 200 0000	0x 200 001F	0x0	0x1F
Verde	0x 200 0020	0x 200 003F	0x20	0x3F
Azul	0x 200 0040	0x 200 005F	0x0	0x1F
Lum	0x 200 0060	0x 200 007F	0x20	0x3F

c.3) (0.5 puntos) Calcule, para la **cache de datos**:

Total de ACCESOS a datos	256 iterations × 4 accesses/iteration = 1024 accesses
Total de FALLOS de datos	Since all vectors fit in the cache with no conflicts, only compulsory misses will occur, 1 per block accessed: 32 blocks × 4 vectors = 128 misses
Tasa de aciertos	$H = (1024 \text{ accesses} - 128 \text{ misses}) / 1024 \text{ accesses} = 0.875$

c.4) (0.5 puntos) ¿Cuál sería el número de fallos de la cache de datos si se adaptase el programa para trabajar con imágenes de 512 pixels, usando vectores R, V y A de 512 elementos? Justifique la respuesta.

With 512-byte vectors, all data accesses will be misses, hence we'd have $512 \times 4 = 2048$ accesses = 2048 misses. This is because the four vectors would map to exactly the same sets. With only two lines per set and accessing 4 vectors, the third access of each iteration (Blue) would replace the first (Red) and the fourth (Lum) would replace the second (Green).

2

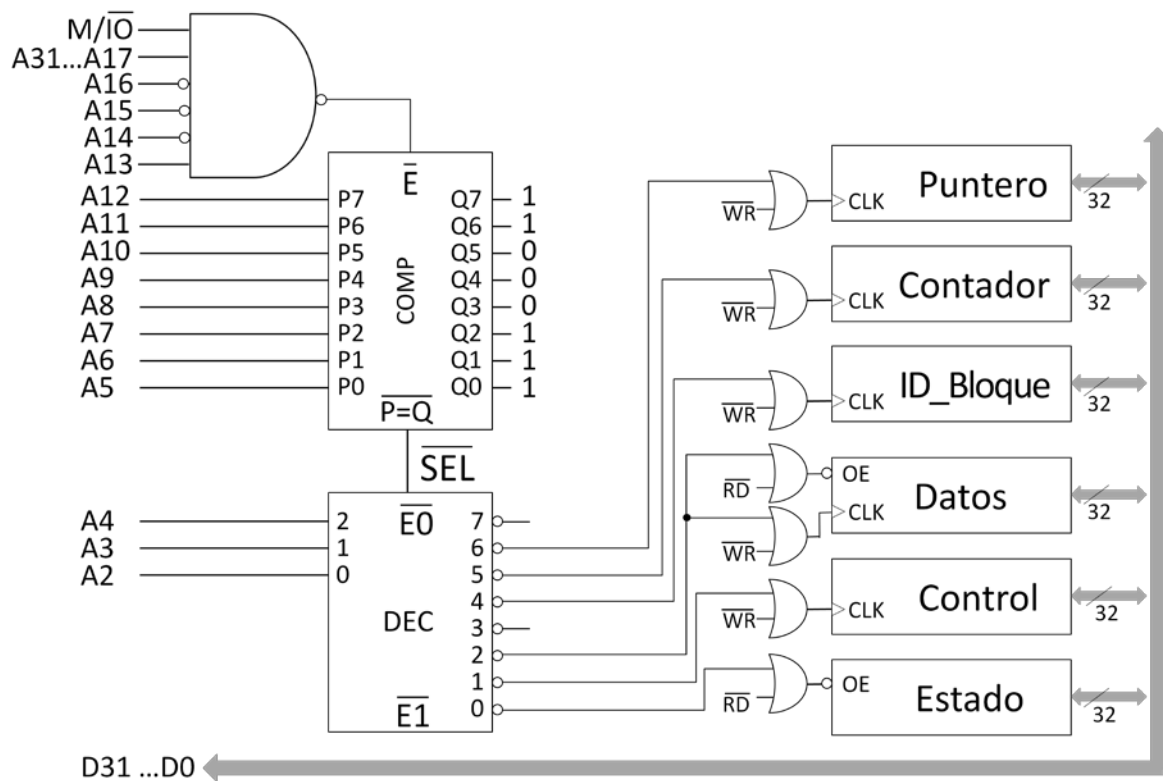
(2,5 puntos) La figura muestra la interfaz de un disco magnético. Esta interfaz se conecta a una CPU MIPS R2000 modificada, de manera que incluye espacios de direccionamiento separados para memoria y E/S. El acceso al espacio de E/S se realiza por medio de las instrucciones **InputW/InputH/InputB** y **OutputW/OutputH/OutputB**, con sintaxis similar a las **load/store**. El disco soporta únicamente transferencia por DMA. Los registros Estado y Control poseen los siguientes bits significativos:

Registro **CONTROL**:

- **A** (bit 7), a 1 ordena al interfaz el inicio de una operación de lectura/escritura sobre el disco magnético
- **IE** (bit 3) a 1 habilita la interrupción INT0*. Si IE=1, la interrupción se emitirá cada vez que R sea igual a 1
- **R/W** (bit 1), indica al interfaz si se trata operación de lectura (R/W= 0) o de escritura (R/W=1) sobre el disco magnético
- **CL** (bit 0), a 1 hace R=0

Registro **ESTADO**:

- **R** (bit 0) se activa a 1 cuando la transferencia del bloque a/desde memoria ha concluido



Nota: Todos los registros son de tamaño 32 bits

- a) (0,5 puntos) Calcule la dirección base (DB) del interfaz y el espacio de direccionamiento (Memoria o E/S) en el que se halla ubicado.

DB= 0xFFFE38E0

Espacio de direccionamiento: Memoria (selección con M/IO* = 1)

- b) (0,5 puntos) Calcule la dirección (DB+X) de cada uno de los registros del interfaz

Registro	Dirección
ESTADO	DB
CONTROL	DB+4
DATOS	DB+8

Registro	Dirección
ID_BLOCK	DB+16
CONTADOR	DB+20
PUNTERO	DB+24

- c) (1 punto) En el driver del disco magnético controlado a través del interfaz del esquema anterior se define la siguiente función:

Función	Índice	Argumentos
Write_Disk	\$v0= 400	\$a0: Puntero a buffer de memoria \$a1: Número de ciclos de transferencia \$a3: Identificador del bloque

La sincronización con el dispositivo se realiza por **INTERRUPCIÓN** al nivel de bloque. La función **Write_Disk** deberá configurar adecuadamente el **DMA** y habilitar la interrupción **int0*** en el interfaz. Suponiendo que el identificador del bloque que se desea escribir en el disco es **0xCC005555**, se pide:

Nota: Debe tenerse en cuenta que se está en un contexto en el que **múltiples procesos** pueden estar ejecutándose concurrentemente

c.1) (0,5 puntos) Genere el código que realiza la invocación de la función Write_Disk	c.2) (0,5 puntos) Genere el código que implementa la función Write_Disk
<pre> .data 0x20000000 Buffer: .space 512 # 512 bytes .text 0x00400000 la \$a0,Buffer li \$a1,128 li \$a3,0xCC005555 li \$v0,400 syscall </pre>	<pre> Write_Disk: la \$t0,0xFFFE38E0 sw \$a0,24(\$t0) sw \$a1,20(\$t0) sw \$a3,16(\$t0) li \$t1,0x8A sw \$t1,4(\$t0) jal suspende_proceso j retexc </pre>

- d) (0,5 puntos) Describa brevemente las acciones que debería realizar la rutina de servicio de **INT0*** al término de la transferencia por ADM.

Cancelar e inhibir la interrupción y activar proceso

3

(3 puntos) Se dispone de un modelo básico de máquina tragaperras que se halla controlada por un MIPS R2000. La máquina posee tres rodillos. Cada rodillo consiste en un generador de números aleatorios comprendidos en el rango [0..7].



La máquina funciona únicamente en modo automático. **Cada vez que se baja la palanca y se ha hecho una apuesta, los 3 rodillos empiezan a girar**, mostrando números de forma aleatoria. Los rodillos se van deteniendo de forma automática al cabo de un cierto tiempo.

El sistema de premios es el siguiente: si el valor numérico mostrado por los tres rodillos coincide, el premio triplica la apuesta; si la coincidencia se limita a solo dos rodillos, el premio duplica la apuesta; si no hay ninguna coincidencia, no hay premio.

La máquina está constituida por tres módulos de interfaz: los **RODILLOS DIGITALES**, el **MONEDERO** y el **VISUALIZADOR**, definidos como sigue:

RODILLOS DIGITALES (Dir. Base 0xFF001000)

- Registro de **ESTADO / CONTROL** (lectura y escritura, 8 bits, DB+0)
 - Bit 0 – **P**: Se pone a 1 para ordenar el pago del premio, si lo hay.
 - Bit 3 – **R**: (Ready – solo lectura) La interfaz lo pone a 1 **una vez los tres rodillos dejan de girar** tras un cierto periodo de tiempo; R se pone a 0 al leer cualquiera de los registros de DÍGITO
- Registro de **DÍGITO 1** (Sólo lectura, 8 bits, DB+1): Valor numérico generado y visualizado por el Rodillo 1
- Registro de **DÍGITO 2** (Sólo lectura, 8 bits, DB+2): Valor numérico generado y visualizado por el Rodillo 2
- Registro de **DÍGITO 3** (Sólo lectura, 8 bits, DB+3): Valor numérico generado y visualizado por el Rodillo 3

MONEDERO (Dir. Base 0xFF002000)

- Registro de **CONTROL** (Lectura y escritura, 8 bits, DB+0)
 - Bit 0 - **A**: Se pone 1 para activar la detección de monedas. 0: Para detener la detección (las monedas son devueltas directamente).
 - Bit 1 – **D**: Se pone a 1 para devolver la moneda introducida (si moneda NO válida).
 - Bit 6 – **E**: Se pone a 1 para habilitar la interrupción en la interfaz y a 0 para inhibirla.
 - Bit 7 – **C**: Bit de cancelación: Si se escribe un 1 entonces la interfaz pone el bit R a 0.
- Registro de **ESTADO** (Sólo lectura, 8 bits, DB + 1)
 - Bit 7 – **R**: (Ready) La interfaz lo pone a 1 cada vez que una moneda es introducida por la ranura. Si E=1, entonces se activa la interrupción INT4* del MIPS.
 - Bits 1,0: **RES**- Resultado de la detección:
 - 00: Moneda NO válida (a devolver)
 - 01: Moneda de 50 céntimos
 - 10: Moneda de 1 €
 - 11: Moneda de 2 €

VISUALIZADOR (Dir. Base 0xFF003000) Este dispositivo usa E/S Directa

- Registro de **DATOS** (Sólo escritura, 32 bits, DB+0): Valor a visualizar, en céntimos de euro. Éste visualiza primero el importe de la apuesta y posteriormente el importe del premio obtenido.

El software del sistema está constituido, entre otros, por dos procesos: (1) Proceso Usuario y (2) Proceso Sistema, cuya estructura y variables que definen se muestran más abajo. Una interrupción de Reloj se encarga de la conmutación entre los mismos.

Proceso Usuario	Proceso Sistema
<pre>mi_premio: .data .word 0 .text <i>inicializar</i> #syscall <i>get_premio</i> #syscall sw \$v0, mi_premio</pre>	<pre>.kdata importe_apuesta: .word 0 importe_premio: .word 0 .ktext <i>inicio</i>: (<i>gestión de RODILLOS DIGITALES por consulta de estado</i>) <i>j inicio</i></pre>

Se pide:

a) (1 punto) Programe las siguientes funciones del sistema:

Función	Índice	Argumentos	Resultado
<i>inicializar</i>	\$v0 = 500	----	Activa MONEDERO y habilita su interrupción. Habilita INT4 en el procesador. Inicializa a cero el visualizador y las variables del sistema. Se asume que las interrupciones están globalmente habilitadas en el procesador (IE _c =1)
<i>get_premio</i>	\$v0=600		Retorna en \$v0 el valor de la variable <i>importe_premio</i>

```

inicializar:  la $a0,0xFF00200
              li $t1,0x41
              sb $t1,0($t0)
              la $t0,0xFF003000
              sw $zero,0($t0)
              sw $zero,importe_apuesta
              sw $zero,importe_premio
              mfc0 $t1,$12
              ori $t1,$t1,0x1000
              mtc0 $t1,$12
              j retexc

```

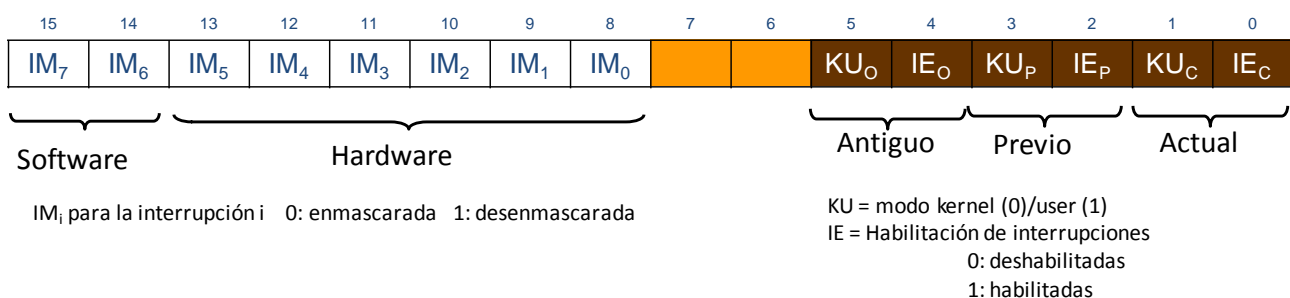
```

get_premio:  lw $v0,importe_premio

              j retexc

```

La figura adjunta muestra el contenido del Registro de Estado (\$12) del MIPS



- b) (1 punto) Programe el código constitutivo del proceso Auxiliar que gestiona, con sincronización por **CONSULTA DE ESTADO**, el módulo interfaz de RODILLOS DIGITALES. Una vez se detengan los rodillos, deberá comprobar la coincidencia de los dígitos – tres o dos iguales – y calcular el importe del premio a partir del valor de `importe_apuesta`; actualizar la variable `importe_premio`; mostrar el premio en VISUALIZADOR; ordenar el pago del premio (si lo hay). Este proceso se repite de forma ininterrumpida.

```

inicio:  la $t0,0xFF001000
bucle:  lb $t1,0($t0)           # bucle de consulta de estado
        andi $t1,$t1,0x08
        beqz $t1, bucle        # espera a que se detengan los rodillos

        lw $t4,importe_apuesta # carga importe_apuesta
        lb $t1,1($t0)          # carga el valor de los tres Dígitos
        lb $t2,2($t0)
        lb $t3,3($t0)
        li $t5,0               # inicializa $t5=0 como registro intermedio para calcular el premio

        beq $t1,$t2, dos_tres  # si Dígito 1 = Dígito 2 entonces pueden coincidir DOS ó TRES
        beq $t1,$t3, dos       # si Dígito 1 = Dígito 3 entonces coinciden solo DOS
        beq $t2,$t3, dos       # si Dígito 2 = Dígito 3 entonces coinciden solo DOS
        j premio

dos_tres: beq $t1,$t3, tres      # si Dígito 1 = Dígito 3 entonces coinciden TRES
dos:      add $t5,$t4,$t4        # actualiza el premio ($t5) con el doble de la apuesta
        j premio

tres:     add $t5,$t4,$t4
        add $t5,$t5,$t4        # actualiza el premio ($t5) con el triple de la apuesta

premio:   sw $t5, importe_premio # actualiza la variable importe_premio con el premio calculado (0, x2, x3)
        beqz $t5,visualizar
        li $t1,0x01
        sb $t1,0($t0)          # si $t5<>0 entonces ordena el pago del premio

visualizar: la $t1,0xFF003000
            sw $t5,0($t1)       # visualiza importe del premio
            j inicio

```

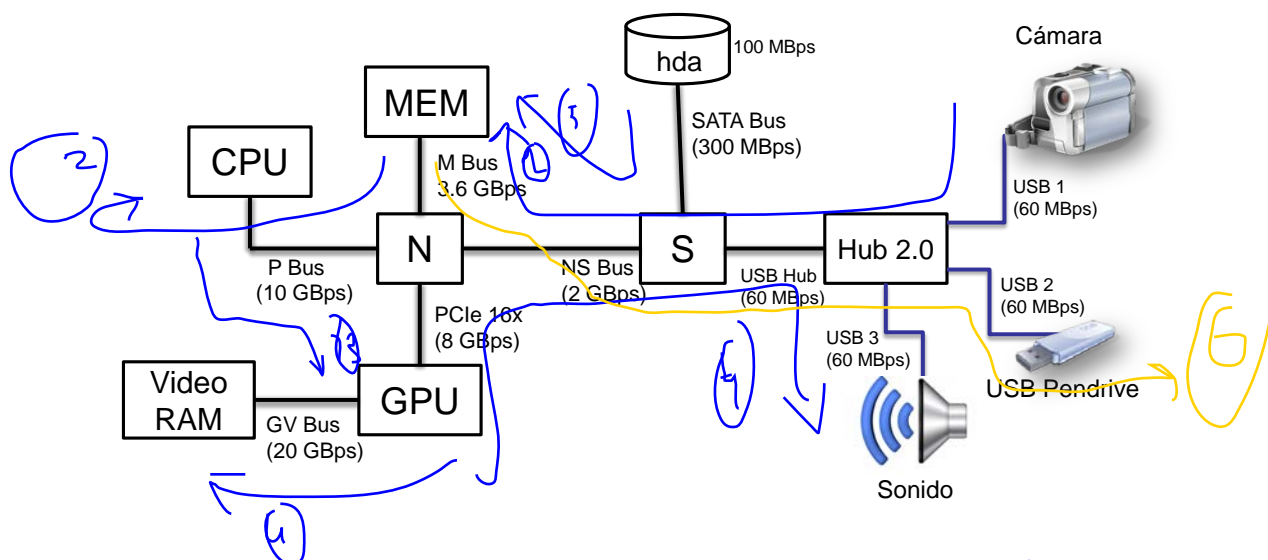
- c) (1 punto) Programe el código de la rutina de servicio de interrupción del MONEDERO (deberá detectar la moneda introducida, expulsarla si no es válida o, de lo contrario, añadirla a la variable `importe_apuesta`).

```

Int4:      la $t0,0xFF002000
           lb $t4,0($t0)                # carga $t4 con registro de CONTROL
           lw $t2,importe_apuesta       # inicializa $t2 con importe_apuesta (en céntimos)
           lb $t1,1($t0)                # lectura del registro de ESTADO
           andi $t1,$t1,0x03             # extrae bits RES mediante máscara
           beqz $t1,devolver             # si RES=0 entonces devolver moneda
           li $t3,0x02
           beq $t1,$t3,100_centimos      # si RES=2 entonces moneda de 1€ (100 céntimos)
           li $t3,0x03
           beq $t1,$t3,200_centimos      # si RES=3 entonces moneda de 2€ (200 céntimos)
50_centimos: addi $t2,$t2,50             # RES=1, moneda de 50 céntimos; incrementa apuesta
           j salir
100_centimos: addi $t2,$t2,100           # RES=2, moneda de 1€; incrementa apuesta
           j salir
200_centimos: addi $t2,$t2,200           # RES=3, moneda de 2€; incrementa apuesta
           j salir
devolver:  ori $t4,$t4,0x02             # ordena devolución moneda NO_Válida (D=1) mediante máscara
salir:     ori $t4,$t4,0x80             # cancela interrupción (C=1) mediante máscara
           sb $t4,$t4,0($0)             # actualiza registro de CONTROL
           sw $t2,importe_apuesta       # actualiza variable importe_apuesta
           j retexc

```

- 4 (1 punto) El computador del esquema reproduce una secuencia mientras la cámara la captura. Ésta genera audio/vídeo en formato MPEG codificado a 32 Mbps y lo transfiere por DMA desde la cámara a la memoria (MEM). Al mismo tiempo, la CPU envía la secuencia comprimida desde memoria a la GPU. A continuación, la GPU descomprime la secuencia y envía, por DMA, las imágenes a la RAM de Vídeo y el audio al equipo de sonido.



- a) (0.5 puntos) Suponiendo que el vídeo descomprimido fue adquirido a $1920 \times 1080 \times 24$ bits y 30 escenas/segundo, mientras que el audio es Surround 5.1 y ha sido muestreado a 48 KHz con 16 bit por muestra, calcule el ancho de banda (en MBps) requerido para:

Transferir el video comprimido desde la cámara a la memoria (MEM):

$$32 \text{ Mbps} / 8\text{bits} = 4 \text{ MBps}$$

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

$$1920 \times 1080 \times (24/8 \text{ Bytes}) \times 30 \text{ fps} = 186.624 \text{ MBps}$$

Enviar el audio desde la GPU al equipo de sonido:

$$6 \text{ channels} \times 48000 \text{ samples per second} \times 16/8 \text{ bytes per sample} = 0.576 \text{ MBps}$$

- b) (0.5 puntos) Mientras el computador reproduce la escena, se transfiere un archivo de 100 MB (10^8 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 DMA. Lecturas y escrituras son concurrentes. Calcule el tiempo que cuesta transferir el archivo y el porcentaje de utilización del bus NS.

Tiempo de transferencia del archivo: The limiting path is the USB Hub, which supports traffic of compressed video (4 Mbps) and audio (0.576 MBps). The available bandwidth is thus $60 - 4 - 0.576 = 55.424$ MBps. The transfer will therefore take: $10^2 \text{ MB} / 55.424 \text{ MBps} = 1.804 \text{ s}$

Porcentaje de utilización del bus NS:

$$4 + 0.576 + 2 \times 55.424 / 2000 = 114.424 / 2000 = 5.77\%$$

5

- (0.5 puntos) Calcule el tiempo de transferencia de un archivo de 512 KB ($K=10^3$) desde un disco duro con los parámetros mostrados más abajo. El archivo se halla almacenado de forma óptima en la Zona 0, esto es, usa sectores consecutivos en cada pista y pistas consecutivas dentro de un cilindro.

Tamaño del sector: 512 bytes
 Velocidad de giro: 4000 RPM
 Tiempo medio de posicionamiento: 6 ms
 Tiempo de salto pista-a-pista: 1 ms
 Densidad lineal: 6000 tpi
 Número de caras: 6
 Radio interno: 0.5"
 Radio externo: 1.75"

Formato: ZCAV, 2 zonas		
Zona	Límites	Sectores/pista
0	0.5" - 0.625"	500
1	0.625" - 1.75"	290

The file takes $512 \text{ KB} / 512 \text{ B/sector} = 1000$ sectors.

Hence the file takes just two tracks and there is no cylinder change because there are 6 sides.

The time to transfer one sector from zone 0 is the time of 1 rotation / 500 sectors

At 4000 RPM, the time of one rotation is $60 \text{ seconds/minute} / 4000 \text{ rotations/minute} = 15 \text{ ms}$

Hence $15 \text{ ms} / 500 \text{ sectors} = 30 \mu\text{s}$ per sector

The transfer time for 1000 consecutive sectors is thus $1000 \times 30 \mu\text{s} = 30 \text{ ms}$

The total time to transfer the file will also include the average seek time (6 ms) plus the rotational latency (time of half turn, or 7.5 ms). All together:

$$\text{Time} = 6 \text{ ms} + 7.5 \text{ ms} + 30 \text{ ms} = 43.5 \text{ ms}$$