Estructura de Computadores

Parcial 2 **11 Junio - 2018**

Nombre: **Grupo:**

1 ***(3 puntos)*** Un sistema basado en procesador MIPS R2000 posee una cache L1 dual configurada como sigue:

- **Cache de Instrucciones**: 2 KB, correspondencia directa, tamaño de bloque de **16 Bytes**.
- **Cache de Datos**: 2 KB, correspondencia asociativa por conjuntos de 2 vías, tamaño de bloque de **64 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	
Etiqueta	21
Línea	7
Desplazamiento	4

Cache de Datos	
Etiqueta	22
Conjunto	4
Desplazamiento	6

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	128	32
Número de bits de cada entrada (indique el nombre de los campos)	1 (Valid) + 21 (Tag) = 22 bits	1 (Valid) + 22 (Tag) + 1 (Dirty) + 1 (LRU) = 25 bits
Tamaño total de la memoria de control (en bits)	128 × 22 = 2816 bits	32 × 25 = 800 bits

c) El siguiente programa obtiene la tercera fila (vector Fila) de una matriz de convolución. El vector Fila se obtiene, tras haber sido inicializadas todas sus componentes a cero, almacenando sobre el mismo las componentes de un vector X desplazadas 2 posiciones a la derecha. A modo de ejemplo, suponiendo que $[x_0,x_1,x_2,x_3]$ fuesen los cuatro elementos del vector X y que [0,0,0,0,0,0,0,0] fuese el vector Fila, inicializado a cero, el vector Fila resultante de la matriz de convolución sería [0,0, x_0,x_1,x_2,x_3,0,0]

```
            .data 0x2C000000
       X:   .word 1,2,3,4, ... 256     # Vector de 256 valores enteros
    Fila:   .space 2048                 # Espacio para vector de 512 valores enteros

            .text 0x00400000
__start:    ori $s0,$zero,512           # Contador de elementos vector Fila
            lui $t0,0x2C00              # Puntero al vector Fila (parte superior)
            ori $t0,$t0,0x0400          # Puntero al vector Fila (parte inferior)
      L0:   sw $zero,0($t0)             # Inicializa a cero Fila[i]
            addi $t0,$t0,4              # Incrementa puntero
            addi $s0,$s0,-1             # Decrementa contador
            bne $s0, $zero,L0           # Sigue iterando mientras contador > 0

            ori $s0,$zero,256           # Contador de elementos vector X
            lui $t1,0x2C00              # Puntero al vector X
            lui $t0,0x2C00              # Puntero al vector Fila (parte superior)
            ori $t0,$t0,0x0400          # Puntero al vector Fila (parte inferior)
      L1:   lw $t2,0($t1)               # Lee X[i]
            sw $t2,8($t0)               # Almacena X[i] en Fila[i+2]
            addi $t0,$t0,4              # Incrementa puntero a vector Fila
            addi $t1,$t1,4              # Incrementa puntero a vector X
            addi $s0,$s0,-1             # Decrementa contador
            bne $s0, $zero,L1           # 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**
5	0x40000	0x40004	5
Total de ACCESOS a código (Indique el cálculo)	7(fuera de bucles) + 512×4 (bucle L0) + 256×6 (bucle L1) = 3591		
Tasa de aciertos (Con cuatro dígitos decimales. Indique el cálculo)	H = (3591 – 5) / 3591 = 0,9986		

c.2) *(0.5 puntos)* Indique el número de bloques que ocupa cada vector, así como los números de bloque del primer y último bloque de cada uno y sus correspondientes números de conjunto en los que se almacenan en la **cache de datos (en hex)**

	Num. de bloques de datos	**Primer bloque**	**Último bloque**	**Primer conjunto**	**Último conjunto**
X	16	0x B00000	0x B0000F	0x0	0xF
Fila	32	0x B00010	0x B0002F	0x0	0xF

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

Total de ACCESOS a datos	512 a Fila + 256 a X + 256 a Fila = 1024 accesos
Total de FALLOS de datos	**Fallos de inicio:** 32 fallos en la inicialización del vector Fila + 16 fallos en el acceso al vector X **Fallos de colisión/capacidad:** 17 fallos en el acceso a Fila durante el bucle L1 Total Fallos= 65
Tasa de aciertos	H = (1024 accesses - 65 misses) / 1024 accesses = 0.9365
Número de reemplazos de bloque	32 bloques del vector Fila y 1 bloque del vector X
Número de palabras (32 bits) escritas a memoria	32 bloques reemplazados del vector Fila que se han modificado×16 palabras por bloque= 512 palabras escritas a memoria

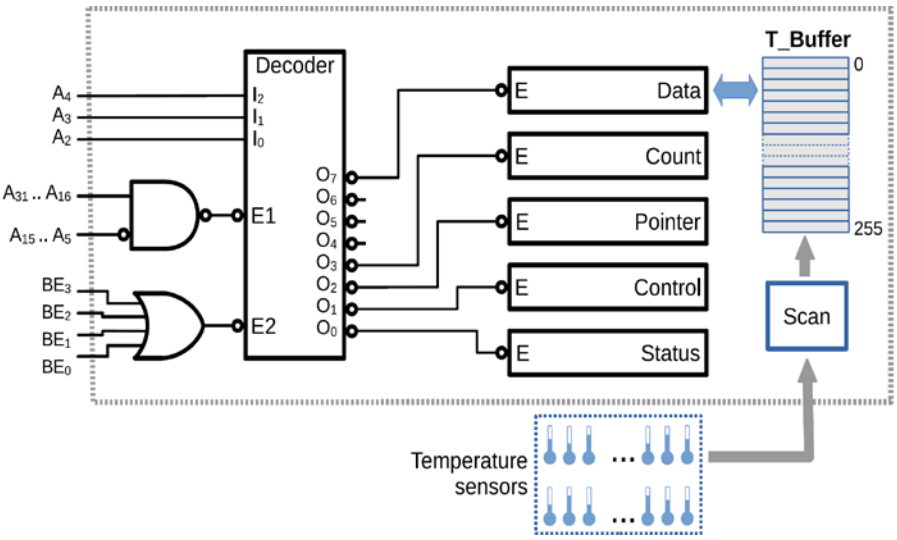
c.4) *(0.5 puntos)* ¿Cuál sería el número de fallos de la cache de datos si se empleara una correspondencia directa en lugar de asociaciativa de 2 vías? Justifique la respuesta.

Si la cache de datos fuese de correspondencia directa, los vectores X y Fila competirían entre sí por la misma línea de cache durante la mayor parte de los accesos. Sin embargo, dado que ambos vectores se acceden de forma desfasada (vector X está desplazado dos posiciones respecto a vector Fila), habrán accesos en que ambos vectores no colisionan y ello dará lugar a aciertos.

Vector X: de los 256 accesos al vector, además de los 16 fallos de inicio, tendremos 16x14=224 fallos de colisión, y sólo 16 aciertos (el último acceso a cada uno de los bloques).

Vector Fila: además de los 32 fallos de inicio (fase de inicialización), habrá 16x14= 224 fallos de colisión, dado que los dos primeros accesos a 16 de los 17 bloques referenciados serán siempre aciertos.

2 ***(2.5*** puntos***)*** La figura muestra los detalles del adaptador de un dispositivo de adquisición de temperaturas conectado a un procesador MIPS R2000. Dicho dispositivo puede ser configurado para registrar las temperaturas de hasta 256 sensores ubicados en las estancias de un edificio. Los valores de temperatura se hallan codificados como enteros con signo de 32 bits. Todos los registros del interfaz son de tamaño 32 bits.



La operación del dispositivo se inicia mediante una orden de *UPDATE*, escrita en el registro de Control. En respuesta a dicha orden, el sistema escaneará las medidas de temperatura de hasta 256 sensores y las almacenará en un buffer interno (*T_Buffer*), en posiciones consecutivas. Una vez *T-Buffer* se actualice con las medidas de temperatura, existen dos posibles modos de transferir su contenido al espacio de memoria de usuario: modos *PIO* y *ADM*. En modo *PIO*, el dispositivo se encontrará *Ready* tan pronto *T_Buffer* se actualice, de modo que la transferencia de datos desde *T_Buffer* a memoria de usuario debe hacerse por programa. En modo *ADM*, una vez *T_Buffer* se actualiza, los datos se transfieren por *ADM* a un buffer en memoria de usuario, de modo que el dispositivo se encontrará *Ready* al término de la transferencia por *ADM*. El adaptador contiene los siguientes registros:

STATUS (lectura y escritura)

- **RDY** (bit 0). Bit *Ready*. Es puesto a 1 por el hardware dependiendo del modo de transferencia
  - Modo *PIO*: un vez que *T_Buffer* ha sido actualizado con las medidas de temperatura
  - Modo *ADM*: cuando el contenido de *T_Buffer* ha sido totalmente transferido a la memoria de usuario por *ADM*.

  Para hacer RDY = 0, basta con escribir un 0 sobre dicho bit (obsérvese que no hay un bit específico de cancelación en este interfaz).
- **IEN** (bit 1). Bit *Interrupt Enable*. Si IEN = 1, la puesta a 1 del bit RDY causa la activación de INT3*.

CONTROL (solo escritura)

- **UPD** (bit 0). Bit *UPDATE*. Cuando se pone a 1 por software, el sistema inicia el registro de temperaturas de los distintos sensores.
- **MOD** (bit 1). Bit *MODO*. A 0 para *PIO* y a 1 para *ADM*. Se ignora cuando UPD = 0.

POINTER (solo escritura)

En modo *ADM*, este registro debe actualizarse con la dirección de inicio del buffer en memoria de usuario, donde el contenido de *T_Buffer* es transferido por *ADM*. No tiene ningún uso en modo *PIO*.

COUNT (solo escritura)

Debe actualizarse con el número de sensores de temperatura a registrar por el sistema con cada orden *UPDATE*.

DATA (lectura y escritura)

Este registro (contiene un valor de temperatura cada vez) se usa solo en modo *PIO*, durante la transferencia por programa del contenido de *T_Buffer*. El registro almacena temporalmente el siguiente valor de temperatura a ser transferido desde *T_Buffer*. Cada vez que se lee el registro, éste se actualiza automáticamente con el siguiente valor de temperatura almacenado en *T_Buffer*.

a) *(1 punto)* Calcule la dirección base del adaptador y los desplazamientos de cada uno de los registros (en hex).

Dirección Base (DB):	0xFFFF0000

Registro	**Desplazamiento**
Status	BA + 0x00
Control	BA + 0x04
Pointer	BA + 0x08

Registro	**Desplazamiento**
Count	BA + 0x0C
Data	BA + 0x1C

b) *(0.5 puntos)* De acuerdo a su descripción, se trata de un dispositivo de *bloques* o de *caracteres*? Justifique la respuesta.

This is a character device because there are no block identifiers.

c) (*0.6 puntos*) Escriba el código de la función del sistema encargada de dar una orden *UPDATE* para escanear *N* sensores y, una vez actualizado *T_Buffer*, almacenar su contenido en la dirección de memoria proporcionada por el usuario **empleando modo *PIO***. Asuma que *N* se pasa en el registro $a0 y el que el puntero al buffer de memoria de usuario lo hace en $a1. La transferencia se debe **sincronizar mediante *consulta de estado***.

```
PIO_Update:
      la $t0, 0xFFFF0000 # BA of Temperature Logger
      sw $a0, 0x0C($t0)  # Count
      sw $zero, 0($t0)   # Status: Disable interrupts
      li $t1, 1          # CLR=0, MOD=0 (PIO), UPD=1
      sw $t1, 0x04($t0)  # Control: send UPD command
Poll:
      lw $t1, 0($t0)     # Read Status
      andi $t1, $t1, 1   # Isolate RDY bit
      beqz $t1, Poll     # Retake polling loop
      sw $zero, 0($t0)   # Clear RDY bit
Transfer:
      lw $t1, 0x1C($t0)  # Read temperature from Data register
      sw $t1, 0($a1)     # Store in user buffer
      addi $a1, $a1, 4   # Update buffer address
      addi $a0, $a0, -1  # Decrease number of transfers
      bnez $a0, Transfer # Retake transfer loop for next temperature









b retexc
```

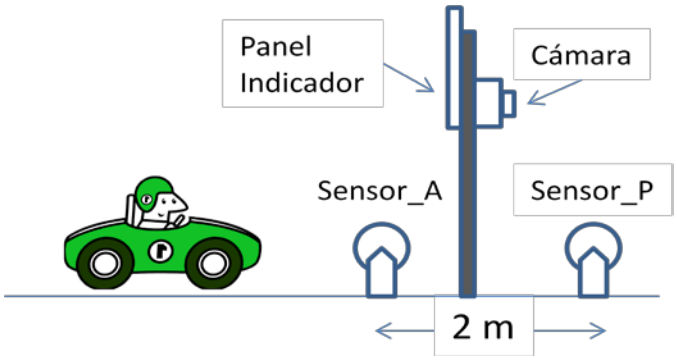
d) *(0.4 puntos)* Escriba el código de la función del sistema encargada de dar una orden *UPDATE* para escanear *N* sensores y, una vez actualizado *T_Buffer*, almacenar su contenido en la dirección de memoria proporcionada por el usuario **empleando modo *ADM***. Asuma que *N* se pasa en el registro $a0 y el que el puntero al buffer de memoria lo hace en $a1. La espera a que la tranferencia por *ADM* se complete, se hará asumiendo un **entorno multitarea**.

```
DMA_Update:
      la $t0, 0xFFFF0000 # BA of Temperature Logger
      sw $a0, 0x0C($t0)  # Count
      sw $a1, 0x08($t0)  # Puntoer
      li $t1, 2          # IEN=1
      sw $t1, 0x00($t0)  # Status: enable interrupts
      li $t1, 3          # CLR=0, MOD=1 (DMA), UPD=1
      sw $t1, 0x04($t0)  # Control: send UPD command
      jal suspende_este_proceso




b retexc
```

3 **(3 puntos)** Se dispone de un sistema de control de velocidad sobre una carretera de un solo carril, el cual se halla controlado por un MIPS R2000. Básicamente, el sistema se encarga de detectar el paso de vehículos (contabilizando su número) y de calcular su velocidad. En caso de que se supere el límite de velocidad, se hará una foto al vehículo. Para la detección de velocidad, el sistema dispone de un módulo **DETECTOR DE PASO** constituido por dos sensores (ver dibujo). El Sensor_A se halla ubicado 2 metros antes que el Sensor_P. La velocidad se calcula midiendo el tiempo que el vehículo tarda en recorrer los 2 m que separan ambos sensores. Cuanto menor sea dicho tiempo, mayor será la velocidad. Si la medida de velocidad supera la velocidad máxima establecida, el módulo **CÁMARA** realizará una foto del vehículo y registrará su velocidad (diferencia en tiempos de paso por los sensores).

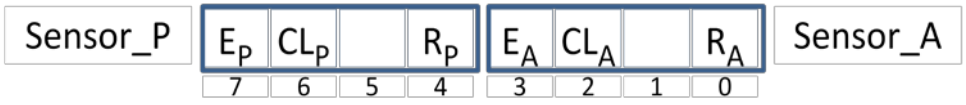


El sistema consta además de otros dos módulos: un **SENSOR DE LLUVIA** y un **PANEL INDICADOR**. El primero fija la velocidad máxima permitida (120 km/h ó 90 km/h) en función del estado del pavimento, la cual se almacena en una variable del sistema, al tiempo que se visualiza en el **PANEL INDICADOR**.

La estructura de cada uno de los referidos módulos de interfaz se indica a continuación:

DETECTOR DE PASO (Dir. Base 0xFFF01000)

- Registro de **SENSORES** (lectura y escritura, 8 bits, DB+0)
  - Bit 0 – **R_A**: (bit Ready – solo lectura) La interfaz lo pone a 1 con el paso de cada vehículo por el **Sensor_A**
  - Bit 2 – **CL_A**: (bit Cancel – solo escritura) Se pone a 1 para cancelar R_A
  - Bits 3 – **E_A**: (bit Enable – solo escritura) Se pone a 1 para habilitar la interrupción asociada al Sensor_A en la interfaz y a 0 para inhibirla. Si R_A=1, entonces se activa la interrupción **INT0*** del MIPS
  - Bits 4, 6 y 7: Idem. que bits 0, 2 y 3, pero para el **Sensor_P**, asociado a la interrupción **INT1*** del MIPS



CÁMARA (Dir. Base 0xFFF02000) Este dispositivo usa E/S Directa

- Registro de **CONTROL** (solo escritura, 8 bits, DB+0)
  - Bit 3 – **F**: (bit Foto) Se pone a 1 para ordenar la captura de una foto

- Registro de **VELOCIDAD** (sólo escritura, 32 bits, DB+4): Tiempo de paso (en *ms*) entre los dos Sensores

SENSOR DE LLUVIA (Dir. Base 0xFFF03000)

- Registro de **CONTROL** (lectura y escritura, 8 bits, DB+0)
  - Bit 3 – **E**: Se pone a 1 para habilitar la interrupción en la interfaz y a 0 para inhibirla.
- Registro de **ESTADO** (sólo lectura, 8 bits, DB + 1)
  - Bit 7 – **R**: (*Ready*) La interfaz lo pone a 1 cada vez que se detecta una variación en el estado del pavimento. R se pone a cero al leer ESTADO. Si E=1, entonces se activa la interrupción **INT5*** del MIPS.
  - Bits 1,0: **RES-** Resultado de la detección:
    01: Pavimento SECO
    10: Pavimento MOJADO

PANEL INDICADOR (Dir. Base 0xFFF04000) Este dispositivo usa E/S Directa

- Registro de **VELOCIDAD** (sólo escritura, 32 bits, DB+0): Valor de la velocidad máxima permitida en km/h.
- Registro de **INDICADOR** (solo escritura, 8 bits, DB+4)
  - Bits 1,0: **MSG-** Mensaje de texto a visualizar:
    01: NO muestra mensaje (si pavimento SECO)
    10: "CONDUZCA CON PRECAUCIÓN" (si pavimento MOJADO)

Se supone la existencia de las siguientes variables del sistema

```
                    .kdata
Reloj:              .word 0      # medida de tiempo real en milisegundos (ms)
tiempo_paso_A:      .word 0      # almacena el valor de tiempo de paso (ms) por el Sensor_A
contador_vehículos: .word 0      # contabiliza el número de vehículos que pasan
límite_velocidad:   .word 0      # velocidad máxima según estado del pavimento
```

Se pide:

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

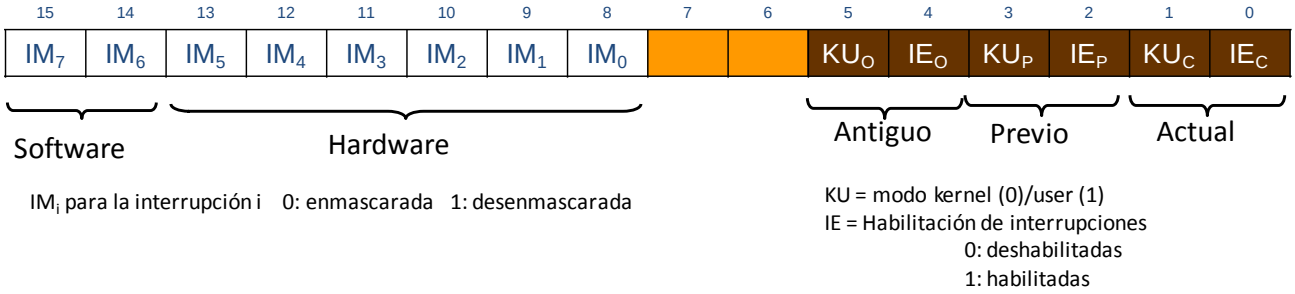
Función	Índice	Argumentos	Resultado
inicializar	$v0 = 800	-----	Habilita las dos interrupciones asociadas al DETECTOR DE PASO, así como la interrupción del SENSOR DE LLUVIA. Inicializa la variable `límite_velocidad` a 120 km/h y la visualiza en el PANEL INDICADOR. El resto de variables, salvo la de `Reloj`, las inicializa a cero. El PANEL INDICADOR no debe mostrar ningún mensaje adicional (MSG=01). Habilita globalmente las interrupciones y actualiza las máscaras de las interrupciones INT0, INT1 e INT5 en el procesador.
get_vehiculos	$v0=900		Retorna en $v0 el valor de la variable `contador_vehiculos`

```
inicializar:  la $t0,0xFFF01000              # DB Detector de Paso
              li $t1,0x88
              sb $t1,0($t0)                  # E=1 en Sensor_A y Sensor_P
              la $t0,0xFFF03000              # DB Sensor de Lluvia
              li $t1,0x08
              sb $t1,0($t0)                  # E=1 en Sensor de Lluvia
              sw $zero,tiempo_paso_A         # var tiempo_paso_A=0
              sw $zero,contador_vehículos    # var contador_vehículos=0
              li $t1,120
              sw $t1,velocidad_máxima        # var límite_velocidad=120 km/h
              la $t0,0xFFF04000              # DB Panel Indicador
              sw $t1,0($t0)                  # visualiza límite_velocidad
              li $t1,0x01
              sb $t1,4($t0)                  # No muestra mensaje (MSG=01)
              mfc0 $t1,$12                   # lee reg.estado MIPS ($12)
              ori $t1,$t1,0x2304             # bits IM5=IM1=IM0=IEp=1
              mtc0 $t1,$12                   # actualiza $12
```

```
b retexc
```

```
get_vehiculos: lw $v0,contador_vehículos #almacena en $v0 var contador_vehículos
               b retexc
```

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



b) *(1 punto)* Escriba el código de las rutina de servicio de las interrupciones INT0* (asociada al Sensor_A) e INT1* (asociada al Sensor_B). El servicio de interrupción INT0* deberá únicamente almacenar el tiempo de paso (en *ms*) por el Sensor_A en la variable `tiempo_paso_A`. El servicio de interrupción de INT1* deberá leer el tiempo de paso por el Sensor_P y calcular la diferencia entre éste y el tiempo de paso por el Sensor_A (previamente almacenado por INT0 en la variable `tiempo_paso_A`). Si la diferencia calculada es inferior al valor de la diferencia en los tiempos de paso correspondiente al límite de velocidad establecido por la variable `límite_velocidad` (véase la tabla más abajo), se deberá ordenar a la CÁMARA la captura de una foto y proceder a registrar en la misma la diferencia de los tiempos de paso (en *ms*) calculada. En cualquiera de los casos, se deberá incrementar la variable `contador_vehículos`.

Tabla velocidad / diferencia_tiempos de paso

Límite de velocidad	*Diferencia tiempo de paso sensores*
120 km/h	60 ms
90 km/h	80 ms

La tabla proporciona las diferencias en los tiempos de paso equivalentes a cada límite de velocidad.
*Para cada uno de estos límites, si la diferencia de tiempos de paso calculada es **inferior** a la proporcionada por la tabla, significa que se ha excedido el límite de velocidad*

```
int0:           la $t0,0xFFF01000    # DB del Detector de paso
                lb $t1,0($t0)        # lee registro Sensores
                ori $t1,$t1,0x04     # aplica máscara con CLA=1
                sb $t1,0($t0)        # cancela interrupción (0→RA)
                lw $t1,Reloj         # lectura Reloj tiempo real
                sw $t1,tiempo_paso_A # almacena tiempo de paso por Sensor_A

b retexc
```

```
int1:           la $t0,0xFFF01000    # DB del Detector de paso
                lb $t1,0($t0)        # lee registro Sensores
                ori $t1,$t1,0x40     # aplica máscara con CLP=1
                sb $t1,0($t0)        # cancela interrupción (0→RP)
                lw $t1,contador_vehículos
                addi $t1,$t1,1             # incrementa contador_vehículos
                sw $t1,contador_vehículos
                lw $t1,Reloj         # lectura Reloj tiempo real
                lw $t2,tiempo_paso_A # lectura tiempo de paso por Sensor_A
                sub $t2,$t2,$t1      # calcula diferencia
# lee el límite de velocidad y establece su equivalencia en tiempo de paso
```

```
                lw $t3,límite_velocidad
                li $t1,120
                bne $t3,$t1,lim_90
     lim_120:   li $t1,60            # lim_vel=120 km/h, equivalente a 60 ms
                j comprobar_vel      #
     lim_90:    li $t1,90           # lim_vel=90 km/h, equivalente a 80 ms
# comprueba si se excede el límite de velocidad y, en su caso, hace la foto y
registra la velocidad (tiempo de paso equivalente) del vehículo
comprobar_vel:  bge $t2,$t1,fin     # si
     camara:    la $t0,0xFFF0200
                li $t1,0x08          # bit F=1
                sb $t1,0($t0)        # ordena hacer la foto
                sw $t2,4($t0)        # registra tiempo de paso en la Cámara
     fin:



b retexc
```

c) *(1 punto)* Programe el código de la rutina de servicio de interrupción INT5* del SENSOR DE LLUVIA, la cual deberá detectar el estado del pavimento, actualizar en consecuencia la variable `límite_velocidad` (Seco: 120 km/h; Mojado: 90 km/h) y visualizarla, junto al mensaje de texto correspondiente, en el PANEL INDICADOR.

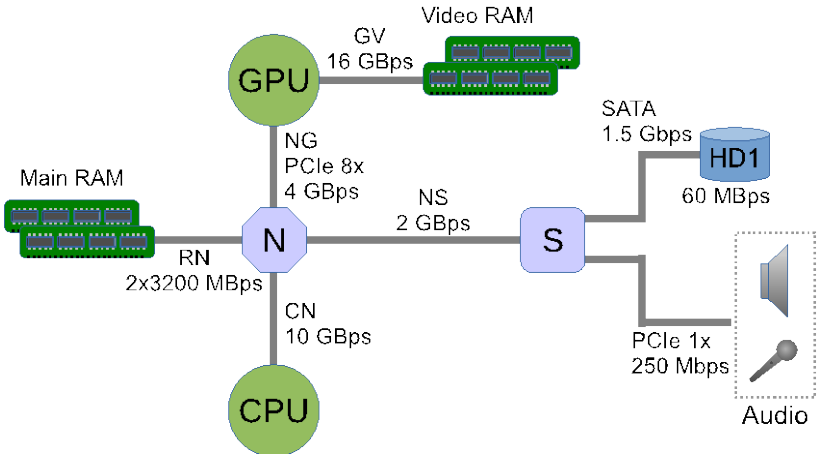
```
Int5:           la $t0,0xFFF03000    # DB del Sensor de Lluvia
                la $t4,0xFFF04000    #DB del panel Indicador
                lb $t1,1($t0)        #lee ESTADO y cancela interrupción (0→R)
                andi $t1,$t1,0x03    #máscara para consultar bits RES
                li $t2,0x01
                bne $t1,$t2,caso_10
     caso_01:   li $t3,120           # si RES=01, Seco (120 km/h)
                j actualizar
     caso_10:   li $t3,90            # si RES=10, Mojado (90 km/h)
     actualizar: sw $t1,4($t4)       # actualiza MSG según bits RES
                sw $t3, límite_velocidad   #actualiza limite_velocidad
                sw $t3,0($t4)        #actualiza velocidad en Panel Indicador




b retexc
```

4 ***(1 punto)*** Se quiere evaluar la capacidad de un cierto computador (ver figura más abajo) para soportar una aplicación de *Karaoke*. El *vídeoclip* (audio/vídeo) de la canción reside en un archivo comprimido en el disco HD1. Durante la reproducción (*playback*) el archivo es transferido por ADM desde HD1 a memoria principal. De forma concurrente, la GPU va leyendo desde memoria el archivo y descomprimiendo el audio y el vídeo. Al mismo tiempo, la voz del cantante es capturada por el sistema de Audio y transferida

por ADM a memoria principal. Este flujo de audio es leído por la GPU desde memoria y combinado con el audio de la canción. La GPU envía a continuación el audio resultante al sistema de audio para su reproducción y el vídeo descomprimido a la RAM de vídeo.



Los anchos de banda dados en la figura son todos efectivos, excepto para el SATA y PCIe 1x, los cuales usan codificación 8b/10b. La velocidad del computador necesita ser suficiente para soportar las siguientes especificaciones:

- El *vídeoclip* con la canción está comprimido en un archivo MPEG codificado a 32 Mbps.
- El *vídeoclip* descomprimido está formado por video de 1920×1200×24 bits, para ser reproducido a 30 fps; y audio 5.1 a 24 bits por canal, muestreado a 96 KHz.
- El micrófono de audio está muestreado a 48 KHz, con16 bits por muestra.

a) *(0.4 puntos)* Calcular el ancho de banda requerido por cada una de las transferencias necesarias por la aplicación de *Karaoke*. Expresar todos los resultados en KBps, MBps o GBps, según el caso.

Vídeoclip comprimido de la canción:
32 Mbps = 4 MBps

Micrófono de audio:
48 KHz × 2 B = 96 KBps

Vídeo descomprimido:
30 fps × 1920 × 1200 × 3 B = 207.36 MBps

Audio combinado descomprimido(*):
6 channels × 96 KHz × 3 B = 1728 KBps

(*) Dado que el enunciado no especificaba claramente las características de reproducción del audio combinado descomprimido, en algún caso podría interpretarse que el resultado de la combinación era un sistema 6.1 (7 canales), por lo que se ha dado también como válida dicha respuesta.

b) *(0.6 puntos)* Justifica si los buses pueden soportar o no los requisitos de ancho de banda para las transferencias simultáneas durante la reproducción.

Bus SATA Supports compressed video clip only (4 MBps) U(SATA) = 4 MBps / 150 MBps = 2.67 % (□)	Bus PCIe 1x Supports
Bus NS Clip+Mic+comb. audio (4+0.096+1.728 = 5.824 MBps) U(NS) = 5.824 / 2000 = 0.3% (□)	Bus RN 2×Clip + 2×Mic (8+0.192 = 8.192 MBps) U(RN) = 8.192 / (2×3200) = 0.13% (□)
Bus NG Mic+Clip+Comb. audio (5.824 MBps) U(NG) = 5.824 / 4000 = 0.15% (□)	Bus GV Uncompressed video only (207.36 MBps) U(GV) = 207.36 / 16000 = 1.3% (□)

5 *(0.5 puntos)* Calcule el tiempo de transferencia de un archive de 10 MB ($K=10^3$) desde un disco magnético, cuyos parámetros se muestran más abajo. El archivo se halla almacenado de forma óptima en la Zona 1, esto es, en sectores consecutivos dentro de una misma pista y en cilindros consecutivos dentro de la zona.

Tamaño de sector: 512 bytes
Velocidad de rotación: 7500 RPM
Tiempo medio de posicionamiento: 6 ms
Trac-to-track seek time: 1 ms
Densidad lineal: 10000 tpi
Número de caras: 6
Radio interno: 0.5"
Radio externo: 1.5"

Formato: ZCAV, 2 zonas		
Zona	Límites	Sectores/pista
0	1.5" - 1.0"	800
1	1.0" – 0.5"	400

The file takes 10 MB / 512 B/sector = 19532 sectors.
One cylinder has a capacity of 400 sectors × 6 sides = 2400 sectors. Hence the file occupies (fully or partially) 19532 / 2400 = 9 cylinders; thus there will be 8 cylinder changes.

The time to transfer one sector from zone 1 is the time of one rotation / 400 sectors
At 7500 RPM, the time of one rotation is 60 seconds/minute / 7500 rotations/minute = 8 ms
Hence 8 ms / 400 sectors = 20 μs per sector

The transfer time for 19532 consecutive sectors is thus 19532 × 20 μs = 390.64 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 4 ms) plus the 8 cylinder changes times the track-to-track time plus the rotational latency. All together:

Time = 6 ms + 4 ms + 390.64 ms + 8 × 1 ms = 408.64 ms