Estructura de Computadores

Examen parcial 2 – Recuperación. 15 de junio de 2022

Apellidos, Nombre:	**Grupo:**

1 ***(3 puntos)*** Un computador con CPU MIPS R2000 tiene una cache L1 dual configurada como sigue:

- **Cache de Instrucciones**: capacidad 128 B, correspondencia directa y tamaño de bloque 8 Bytes.
- **Cache de Datos**: 256 B, correspondencia asociativa por conjuntos de 2 vías, tamaño de bloque de 16 bytes, remplazo LRU y políticas de escritura posterior y con ubicación (*write-back, write-allocate*).

a) *(0.5 puntos)* Obtenga el número de bits de cada campo de la dirección de memoria para ambas caches.

Cache de **INSTRUCCIONES**		
Etiqueta	Línea	Desplazamiento
32 – (3+4) = **25**	$\log_2$ (128/8) = **4**	$\log_2$ (8) = **3**

Cache de **DATOS**		
Etiqueta	Conjunto	Desplazamiento
32 – (4+3) = **25**	$\log_2$ (256/16/2) = **3**	$\log_2$ (16) = **4**

En la memoria del computador está almacenado este programa, que se analizará más adelante:

```
        .data 0x10000000
X:      .byte 0x07,0x93,... 0xF2  # 64 componentes
        .data 0x10000100
Y:      .space 256                # 256 componentes
        .text 0x00400000
        lui $t0,0x1000
        ori $t0,$t0,0x100   # Puntero a Y(0)
        ori $t1,$zero,240   # Num. de componentes de Y que se inicializan
        ori $t2,$zero,0x01  # Valor con que se inicializan
L1:     sb  $t2,0($t0)      # Inicializa componente del vector Y
        addi $t0,$t0,1      # Incrementar puntero
        addi $t1,$t1,-1     # Decrementar contador
        bnez $t1,L1         # Iterar mientras contador <> 0

        lui $t0,0x1000      # Puntero a X(0)
        ori $t1,$t0,0x100   # Puntero a Y(0)
        addi $t2,$zero,64   # Num. de componentes de X a copiar sobre Y
L2:     lb $t3,0($t0)       # Lectura de X(i)
        srl $t3,$t3,1       # X(i) / 2
        sb $t3,0($t1)       # Almacena en Y(i)
        addi $t0,$t0,1      # Incrementar puntero sobre X
        addi $t1,$t1,1      # Incrementar puntero sobre Y
        addi $t2,$t2,-1     # Decrementar contador
        bnez $t2,L2         # Iterar mientras contador <> 0
```

b) *(0.25 puntos)* ¿A qué instrucción o instrucciones del programa corresponde ubicarse en la línea 2 de la cache de instrucciones? ¿Con qué etiqueta se almacenaría su bloque en la cache?

Instrucción o instrucciones	*Etiqueta (hex)*
`sb $t2,0($t0)` y `addi $t0,$t0,1`	0x8000

c) *(0.25 puntos)* Si el tamaño de bloque de la cache de instrucciones fuera de 4 bytes en vez de 8, ¿a qué instrucción/es correspondería entonces ubicarse en la línea 2? ¿Con qué etiqueta lo harían?

Instrucción o instrucciones	*Etiqueta (hex)*
`ori $t1,$zero,240`	0x8000

El programa anterior accede a dos vectores de bytes, X e Y, de 64 y 256 componentes, respectivamente. En el primer bucle (L1), el programa escribe **los primeros 240** componentes de Y a un valor inicial 0x01. En el segundo bucle, el programa lee cada uno de los 64 componentes de X, divide su valor por dos y lo escribe en los primeros 64 componentes de Y. El programa no utiliza pseudoinstrucciones.

d) *(0.4 puntos)* Para la **cache de instrucciones**, obtenga:

Número de bloques que ocupa el código	9	Núm. de Instrucciones ejecutadas	4 + 240×4 + 3 + 64×7 = 1415
Total de FALLOS y tipo (inicio, conflicto, capacidad)	9 fallos de inicio		
Tasa de ACIERTOS (Con cuatro dígitos decimales. Indique el cálculo.)	H= $\frac{1416-9}{1416} = 0{,}9936$ (99,36%)		

e) *(0.4 puntos)* Obtenga el número de bloques de datos **a los que accede el programa** (recuerde que no accede a todo el vector Y). Indique también los números del primer y último bloque al que se accede de cada uno, sus etiquetas y los conjuntos a los que corresponden en la **cache de datos**.

	No. de bloques	Primer bloque			Último bloque		
		Nº bloque	Etiqueta	Conjunto	Nº bloque	Etiqueta	Conjunto
X	4	0x1000 000	0x200000	0	0x1000003	0x200000	3
Y	15	0x1000 010	0x200002	0	0x100001E	0x200003	6

f) *(0.6 puntos)* Obtenga los siguientes datos sobre el comportamiento de la **cache de datos** tras la ejecución del programa. Indique cómo obtiene cada resultado.

No. ACCESOS	= 240 (Y) + 64 (X) + 64 (Y) = **368**
No. FALLOS y su tipo	= 15 fallos **de inicio** (Y en bucle L1) + 4 fallos **de inicio** (X en L2) + 4 fallos **de conflicto** (primeros bloques de Y en L2, la cache no está llena) = **23** fallos
No. de reemplazos	**4** reemplazos, por el acceso en L2 a los 4 primeros bloques de Y
Tasa de aciertos	H = (368 accesos – 23 fallos) / 368 accesos = 0.9375 = **93.75%**

g) *(0.3 puntos)* Suponga ahora que, en el bucle L1, el programa inicializa todos los componentes de Y (256 en vez de 240). Indique cómo variará el numero de fallos en la cache de datos. En este supuesto, ¿se podría mejorar la tasa de aciertos variando el grado de asociatividad de la cache? Justifíquelo.

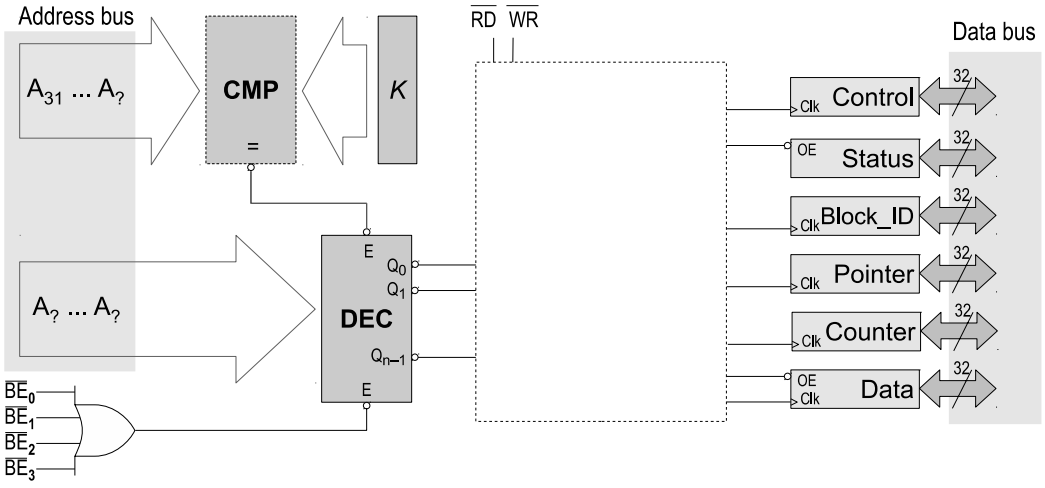
Solo habría un fallo más de inicio, causado por el último bloque del vector Y, al que no se accedía con solo 240 iteraciones pero sí con 256.

Aumentar la asociatividad no va a mejorar la situación, ya que en este caso no hay fallos de conflicto, sino de capacidad.

h) *(0.3 puntos)* Calcule el tamaño de la memoria de control requerido por cada una de las caches

	No. entradas en la memoria de control	Número de bits de cada entrada. (Indique qué bits de control se usan)	Tamaño total de la memoria de control (en bits)
Cache de instrucciones	16	1(V) + 25(Etiq) = 26 b	16×26 = 416 b
Cache de datos	16	1(V) + 1(M) + 1(LRU) + 25(Etiq) = 28 b	16×28 = 448 b

2 ***(2 puntos)*** La figura muestra (parcialmente) el esquema del adaptador de cierto dispositivo de bloques conectado a una CPU MIPS R2000. El dispositivo está ubicado en la **dirección base DB = 0xFFFF0100** y el adaptador ocupa el rango de direcciones mínimo necesario para incluir sus registros. Todos los registros son de 32 bits y se ubican en las direcciones que se indican en la tabla, relativas a la DB.

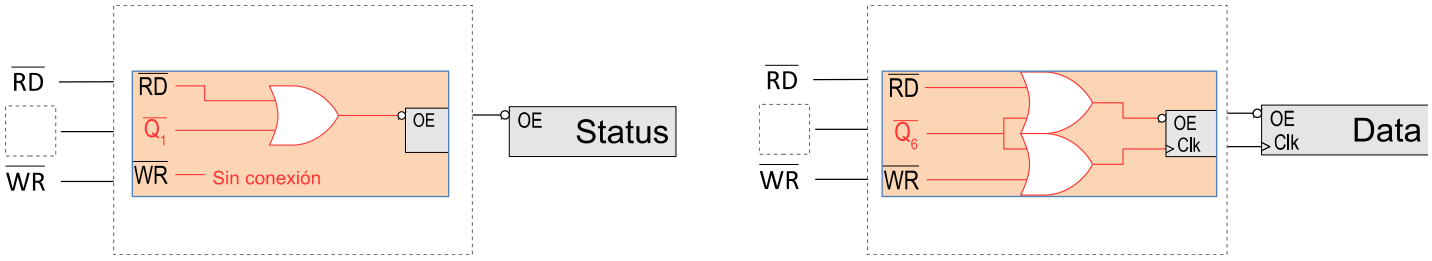


Registro	Dirección
Control	DB + 0
Status	DB + 4
Block_ID	DB + 12
Pointer	DB + 16
Counter	DB + 20
Data	DB + 24

a) *(0.5 puntos)* Complete la tabla indicando en ella cuáles son las líneas de dirección que se conectan en la entrada izquierda del comparador CMP; qué valor ha de tener *K*, la entrada derecha del comparador; y de cuántas entradas y salidas es el descodificador DEC.

Dirección → CMP ($A_{31} - A_?$)	Valor de K (en binario y con todos sus bits)	Tamaño de DEC (No. de entradas y No. de salidas)
A_{31} hasta A_5	1111 1111 1111 1111 0000 0001 000	3 entradas y 8 salidas

b) *(0.5 puntos)* Dibuje los detalles de la lógica de selección de los registros Status y Data. Indique qué salida de DEC (Q_i) hay que conectar en la entrada que no está rotulada.



El registro **Control** del adaptador tiene 5 bits útiles; el resto de los bits pueden tomar cualquier valor:

- **Bit 0**: *START*. Al ponerlo a uno, se inicia una operación de transferencia de un bloque.
- **Bit 1**: *IE* (*Interrupt Enable*). Cuando se pone a uno, habilita las peticiones de interrupción al final de cada transferencia.
- **Bit 2**: *DIR*. Dirección de la transferencia. Hay que escribir un uno para indicar una operación de lectura del dispositivo a la memoria o un cero para escritura desde memoria al dispositivo.
- **Bit 3**: *ADM*. Al ponerlo a uno, la transferencia se hace por ADM. Con cero, por PIO.
  - En modo ADM, el dispositivo indica el final de la transferencia (en el bit *Ready* del registro Status) cuando el bloque se ha transferido completamente.
  - En modo PIO, el bit *Ready* se activa cada vez que el dispositivo está preparado para transferir una nueva palabra de un bloque, a través del registro Data.
- **Bit 4**: *CLEAR*. Para cancelar el bit *Ready* hay que escribir este bit a uno y el resto de los bits a cero. Escribir cero en *CLEAR* no tiene ningún efecto.

En el registro **Status** solo hay un bit de interés; los demás están siempre a cero:

- **Bit 0**: *Ready*. Al ponerse a uno, indica el fin de la transferencia de un bloque por ADM o la disponibilidad para transferir una palabra mediante el registro de datos, en modo PIO. Si las interrupciones están habilitadas (ver bit *IE* de Control), la activación de este bit implica el envío de una petición de interrupción por la línea int3* del MIPS.

En el registro **Block_ID** se le indica al dispositivo el número de bloque a transferir. **Pointer** se usa en transferencias por ADM para darle la dirección del buffer en memoria desde el que leer o al que escribir. También en modo ADM, en **Counter** se programa el número de palabras (32 bits) que se han de transferir para completar un bloque. El registro **Data** solo se usa en modo PIO para transferir cada palabra del bloque.

Por su parte, el sistema cuenta con las funciones *suspende_este_proceso* y *activa_proc_en_espera* descritas en el aula. En el manejador de excepciones quedan disponibles los registros desde $t0 hasta $t3. Contando con ello, se debe implementar una función de sistema con el siguiente perfil:

Función	Índice ($v0)	Argumentos
Read_DMA_Block	100	$a0: Bloque a transferir $a1: Número de palabras del bloque $a2: Puntero a buffer de memoria

La función ha de ordenar la transferencia de lectura del bloque dado por ADM, que ha de guardarse en el buffer de memoria indicado. El proceso que llama ha de quedar suspendido hasta el fin de la transferencia del bloque. Para sincronizar con el final de la lectura, Read_DMA_Block ha de habilitar la interrupción. Esto se controla exclusivamente con el bit *IE* del adaptador, ya que el código de inicio se encarga de dejar la int3 desenmascarada y las interrupciones globalmente habilitadas.

c) *(0.8 puntos)* Implemente la función Read_DMA_Block y el manejador de la interrupción int3*. El manejador debe volver a dejar la interrupción del dispositivo inhibida.

```
Read_DMA_Block:

   la $t0, 0xFFFF0100
   sw $a0, 12($t0)     # Block_ID
   sw $a1, 20($t0)     # Counter
   sw $a2, 16($t0)     # Pointer
   li $t1, 0xF         # START=IE=DIR=ADM=1
   sw $t1, 0($t0)      # Control

   jal suspende_este_proceso


   b retexc
```

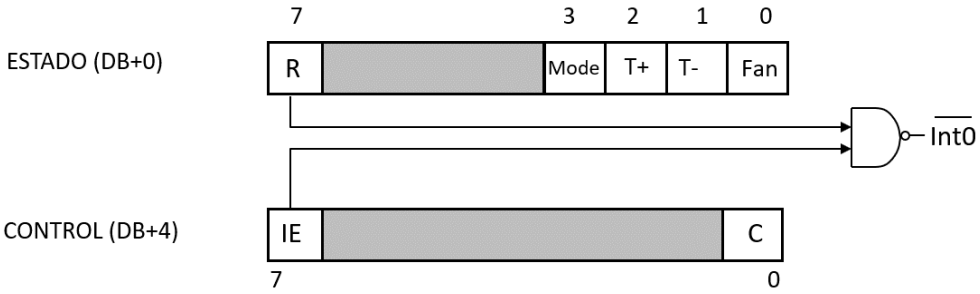
```
Int3:

   la $t0, 0xFFFF0100
   li $t1, 0x10        # CLEAR=1
   sw $t1, 0($t0)      # Control

   jal activa_proc_en_espera




   b retexc
```

d) (0.2 puntos) Un programa de usuario hace uso de Read_DMA_Block para leer el bloque número 42 del dispositivo, que tiene un tamaño de 512 bytes. En su sección *data* declara una variable, *buffer*, con espacio para un bloque. Escriba el código de usuario que prepara los argumentos y llama a la función.

```
   li $a0, 42          # Bloque a leer
   li $a1, 128         # No. de transferencias = 512 byte/bloque / 4 bytes/palabra
   la $a2, buffer      # Dirección inicio del buffer

   li $v0, 100         # Read_DMA_Block
   syscall
```

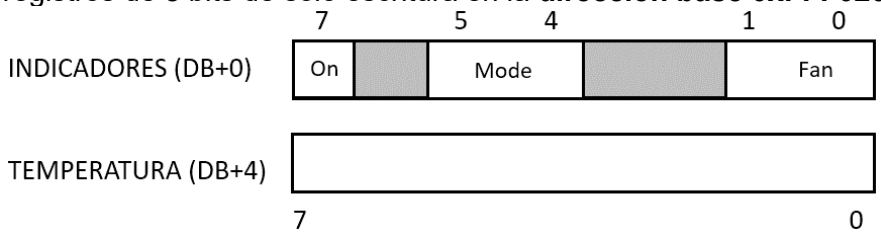
3 ***(3.75 puntos***) El mando a distancia de un equipo de aire acondicionado, está diseñado con un procesador MIPS R2000 y 3 dispositivos. Una botonera de 4 botones (3-Mode, 2-T+, 1-T-, 0-Fan), una pantalla LCD y un emisor de infrarrojos.

La **botonera** está en la **dirección base 0xFFFF0100,** está asociada a la interrupción 0 y tiene los siguientes registros de 8 bits:



El registro de estado (de sólo lectura) tiene un bit para cada uno de los 4 botones (en la posición que se indica) y el bit R que se activará cuando se pulse cualquiera de los 4 botones, generando una interrupción si el bit IE del registro de control (DB+4) está a 1. Para poner a cero el bit R hay que escribir un 1 en el bit C de dicho registro.

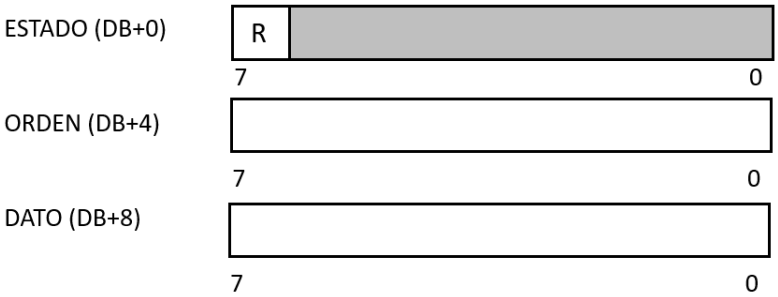
La **pantalla** tiene 3 secciones, la de la izquierda (MODE) permite mostrar un pequeño indicador en una de las 4 posibles posiciones, numeradas del 0 al 3, la de la derecha (FAN) es similar a la anterior e igualmente presenta otro indicador en 4 posibles posiciones. Por último, en la parte central se muestran dos dígitos con la temperatura de referencia. Este dispositivo es de salida directa y se gestiona con dos registros de 8 bits de solo escritura en la **dirección base 0xFFF0200**



El registro INDICADORES (DB+0) permite indicar el número de indicador a activar en las dos zonas: Mode en los bits del 5 al 4 y Fan en los bits del 1 al 0. En ambos casos se codificará un número del 0 al 3 en binario. Además, el bit 7 permite encender (1) o apagar (0) la pantalla.

El registro TEMPERATURA (DB+4) contiene el número con la temperatura de referencia que se representará en la parte central de la pantalla.

El **emisor de infrarrojos** está en la **dirección base 0xFFFF0300** y permite enviar órdenes con datos siempre y cuando el dispositivo esté listo. Ello se indica cuando el bit R=1 en su registro de ESTADO (DB+0) el cual es de sólo lectura. El bit se pone a cero de forma automática al escribir en el registro ORDEN (DB+4). El mensaje se transmite automáticamente al escribir en el registro DATOS (DB+8), por lo que el orden de pasos para enviar un mensaje debe ser: 1-verificar si bit R=1, 2-escribir orden, 3-escribir dato. Todos los registros son de 8 bits, los registros ORDEN y DATO son de solo escritura.



El comportamiento del sistema debe ser el siguiente:

Pulsando simultáneamente los botones 1 y 2 (bajar y subir la temperatura) el mando se enciende o se apaga.

Estando el mando apagado, la pantalla está apagada y solo reacciona a la combinación de encendido, cualquier otra pulsación es ignorada. Al detectar la combinación de encendido se enciende la pantalla y se transmiten por infrarrojos la orden de encendido.

Estando el mando encendido: Pulsando el botón 3 se cambia el modo del aire acondicionado (MODE), pasando al número siguiente de forma cíclica, es decir 0,1, 2, 3, 4 y del 4 se vuelve al cero. Lo mismo sucede con el botón 0 que cambia el modo del ventilador (FAN) igualmente de forma cíclica del 0 al 4. Pulsando los botones 2 y 1 se incrementa o decrementa la temperatura en un grado.

En todos los casos indicados, se actualiza la pantalla y se envía los mensajes por infrarrojos notificando los cambios.

El control se realiza con un programa de usuario, que se ejecuta sobre un sistema MiMoS similar al de las prácticas y que cuenta con las funciones del sistema siguientes:

Función	Índice	Argumentos	Resultado
Espera_boton	$v0 = 600	-----	Espera a que se pulse un botón. Una vez pulsado, devuelve en $v0 los 4 bits inferiores del registro de estado de la botonera.
Modif_pantalla	$v0 = 601	$a0 valor de indicadores $a1 valor de temperatura	Escribe en los registros de la pantalla los valores que se pasan como parámetros. El valor de los indicadores tiene el mismo formato que el registro INDICADORES
Enviar_infra	$v0 = 602	$a0 valor de la orden $a1 valor del dato	Envía por infrarrojos la orden y el dato que se especifican.

Las órdenes posibles son:

Orden	Dato	Descripción
1	0	Encender el aire acondicionado
2	0	Apagar el aire acondicionado
3	MODE (0..3)	Actualizar el MODE
4	FAN (0..3)	Actualizar el FAN
5	TEMPERATURA	Actualizar la temperatura

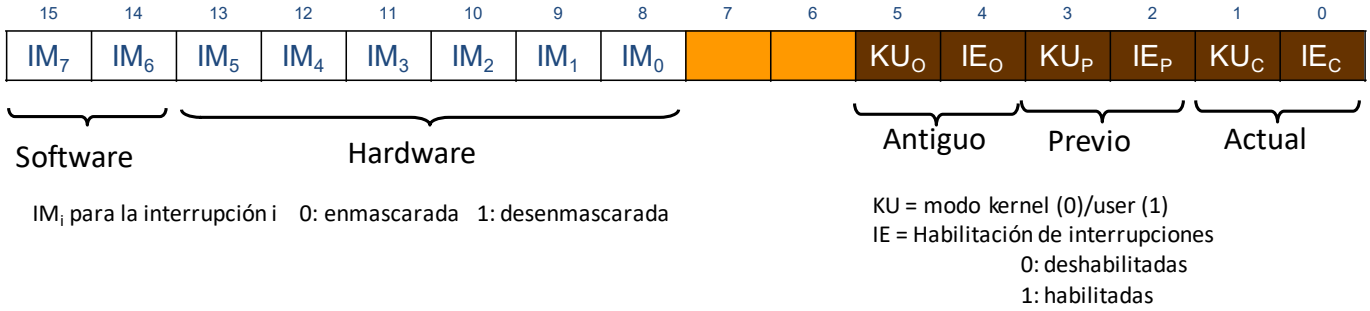
El programa de usuario tiene todas las variables para mantener el estado que son:

```
                .data
mode:           .byte  0          # Valor actual de MODE
fan:            .byte  0          # Valor actual de FAN
temp:           .byte  0          # Valor actual de temperatura
```

En el kernel se declara la variable estado, como en el MiMoS de prácticas, que permite decidir si ejecutar el proceso de usuario o el proceso ocioso del sistema:

```
                .kdata
                listo   = 0       # Posibles estados del proceso
                esperando_boton = 1
estado:         .word listo       # Estado del proceso
```

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



IM_i para la interrupción i 0: enmascarada 1: desenmascarada

KU = modo kernel (0)/user (1)
IE = Habilitación de interrupciones
0: deshabilitadas
1: habilitadas

a) *(1 punto)* Escriba el código de inicio del sistema, que inicializa la botonera para que no genere interrupciones, desenmascara la interrupción cero, apaga la pantalla, envía un mensaje de apagado por infrarrojos (debe esperar por consulta de estado a que el emisor esté libre), deja el procesador en modo usuario con las interrupciones habilitadas globalmente y después salta a la función main.

```
            .text    # Código de inicio del programa
__start:
            la $t0, 0xFFFF0100  # DB Botonera
            sb $zero, 4($t0)    # desactivamos interrupción en botonera
            la $t0, 0xFFFF0200  # DB pantalla
            sb $zero, 4($t0)    # apagamos pantalla
            la $t0, 0xFFFF0300  # DB infrarrojos
cde:        lb $t1,0($t0)       # reg. estado
            andi $t1,$t1,0x80   # bit R
            beqz $t1, cde       # esperamos hasta R=1
            li $t1,2            # orden de apagado
            sb $t1,4($t0)       # registro de orden
            sb $zero,8($t0)     # dato de apagado 0 en registro de dato (envío)
            li $t0, $t0, 0x0103 # Desenmascara Int0*, KUc=1 (USER)  IEc=1
            mtc0 $t0, $12









            la $k1, salvareg
            jal main
            li $v0, 10
            syscall                 # exit
```

b) *(0.5 puntos)* Programe la función del sistema Espera_boton:

Función	Índice	Argumentos	Resultado
Espera_boton	$v0 = 600	-----	Espera a que se pulse un botón. Entre tanto, se pasa el control al proceso ocioso. Debe habilitar la int0 y dormir (estado=esperando_boton) hasta que se pulse un botón. (La interrupción 0 ya se encargará de esto: devuelve en $v0 los 4 bits inferiores del registro de estado de la botonera)

```
Espera_boton:
        la $t0, 0xFFFF0100  # DB Botonera
        li $t1, 0x80
        sb $t1, 4($t0)    # activamos interrupción en botonera
        li $t1, esperando_boton
        sw $t1, estado




        b retexc
```

c) *(0.7 puntos)* Implemente el código de la interrupción 0 que completa las acciones de la llamada Espera_boton. Deberá copiar en $v0 los 4 bits inferiores del registro de estado de la botonera, cancelar la interrupción, deshabilitar la interrupción y poner la variable estado a listo:

```
Int0:
        la $t0, 0xFFFF0100  # DB Botonera
        li $t1, 0x01
        sb $t1, 4($t0)    # cancelamos y desactivamos interrupción en botonera
        lb $v0, 0($t0)
        andi $v0,$v0,0x0F
        li $t1, listo
        sw $t1, estado









    b retexc
```

d) *(0.75 puntos)* La función main que controla el mando en el espacio de usuario, debe ajustar se al siguiente pseudocódigo:

```
Repetir indefinidamente {
  Repetir
     Espera_boton
  Hasta combinación de encendido/apagado
  Enviar_infra (1,0) //encendido
  Modif_pantalla((mode*16)+fan+0x80,temperatura)

  jal Modo_encendido
}
```

La función Modo_encendido es una función de usuario que gestionará los botones cuando el mando esté encendido y que se implementará en el siguiente apartado.

Teniendo en cuenta que estamos en espacio de usuario y por tanto no podemos acceder a la entrada salida directamente, implemente usando las llamadas al sistema, el código en ensamblador de la versión mostrada del pseudocódigo.

```
                .data
fan:            .byte 0
mode:           .byte 0
temperatura:    .byte 17
                .text
main:
                li $v0, 600       # espera_boton
                syscall
                li $t1,0x06       # botones T+ y T- a la vez
                and $v0,$v0,$t1
                bne $v0,$t1,main
                li $v0, 602       # enviar_infra
                li $a0, 1
                li $a1, 0
                syscall
                li $v0, 601       # modif_pantalla
                lb $a0, mode
                sll $a0, $a0, 4
                lb $t1, fan
                add $a0,$a0,$t1
                addi $a0,$a0,0x80 #mode*16+fan+0x80
                lb $a1, temp
                syscall









                jal modo_encendido
                b main
```

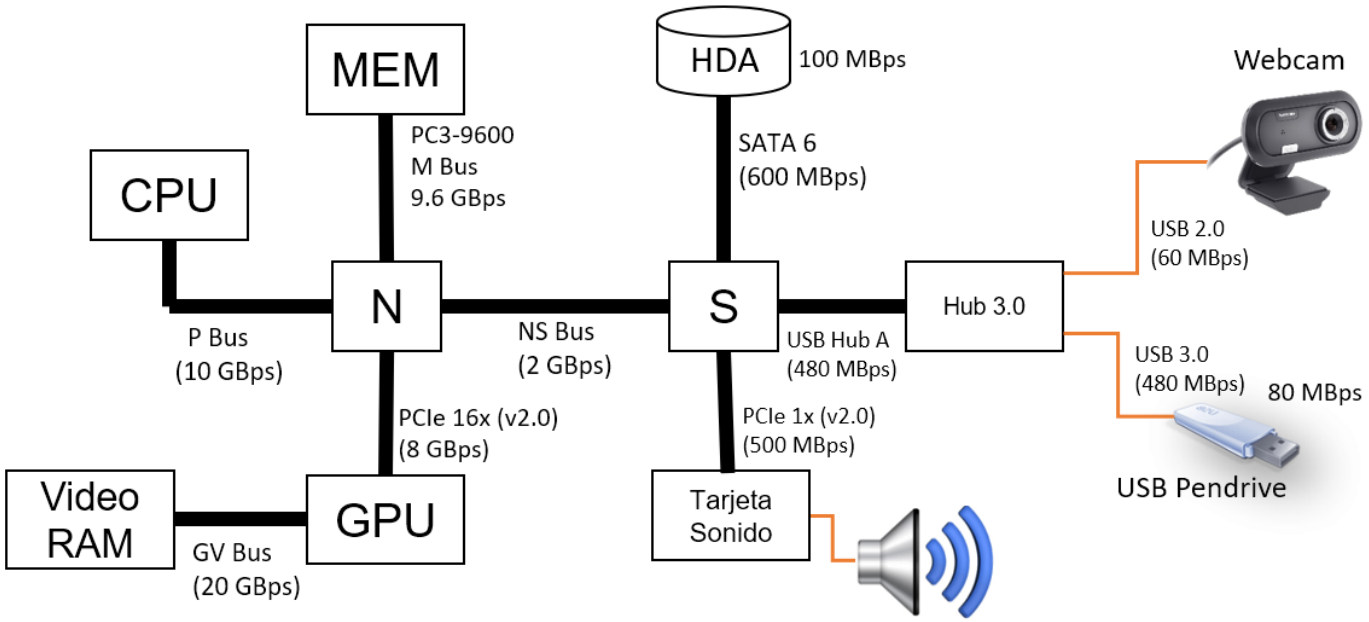
e) *(0.8 puntos)* A continuación se muestra una parte de la función modo_encendido en la que se gestiona únicamente el botón FAN y la combinación de encendido/apagado, que en este caso apagará.

```
modo_encendido{
  Repetir
     Espera_boton
     Si botón=0
       Fan = (fan + 1)&0x3
       Modif_pantall((mode*16)+fan+0x80,temperatura)
       Enviar_infra (4,fan)
  Hasta combinación de encendido/apagado
  Enviar_infra (2,0) //apagado
}
```

implemente usando las llamadas al sistema, el código en ensamblador de la versión mostrada del pseudocódigo.

```
modo_encendido:
              li $v0, 600            # esperando_boton
              syscall
              andi $t3,$v0,$0F       # nos quedamos con los 4 bits
              li $t1,0x01            # botón 0
              bne $t3,$t1,sigue      # no es FAN
              lb $t2,fan
              addi $t2,$t2,1
              andi $t2,$t2,0x03
              sb $t2,fan             # actualizamos fan
              li $v0, 601            # modif_pantalla
              lb $a0, mode
              sll $a0, $a0, 4
              add $a0,$a0,$t2
              addi $a0,$a0,0x80
              lb $a1, temp
              syscall
              li $v0, 602            # enviar_infra
              li $a0, 4
              or $a1, $zero, $t2     # fan
              syscall
sigue:        li $t1,0x06       # botones T+ y T- a la vez
              bne $t3,$t1,modo_encendido
              li $v0, 602            # enviar_infra
              li $a0, 1
              li $a1, 0
              syscall









              jr $ra
```

4 ***(1.25 puntos)*** El computador del esquema reproduce una secuencia de vídeo en directo en formato MP4 codificado a 50Mbps desde la Webcam. Para hacer esto, se transfiere por ADM de la Webcam a memoria y a la vez de memoria a la GPU (USB2 → M, M → GPU). La GPU descomprime el vídeo que guarda en su memoria gráfica y el audio se envía al sistema de sonido por ADM (GPU → M, M → PCie 1x). Además, la GPU transcodifica la secuencia de vídeo en un formato H265 de 20Mbps y el resultado lo almacena en el Pendrive. Todas las transferencias se hacen de forma sincronizada y simultánea.



a) *(0.3 puntos)* Suponiendo que el vídeo descomprimido tiene una resolución de 3840x2160x24 bits y 50 escenas por segundo y que el audio es 5.1, con una frecuencia de muestreo a 44 KHz y 16 bits/muestra, calcule el ancho de banda (en MBps) requerido para:

Transferir el video comprimido MP4 desde la Webcam a la memoria (MEM):
50 Mbps / 8bits/byte = 6,25MBps

Transferir el video comprimido H265 desde la memoria (MEM) al Pendrive:
20 Mbps / 8bits/byte = 2,5MBps

Enviar las imágenes de vídeo desde la GPU a la RAM de vídeo:
3840×2160×24×50fps/8 = 1244160 MBps

Enviar el audio desde la GPU al equipo de sonido:
6 canales × 44000 muestras por segundo × 16/8 bytes por muestra = 0,528 MBps

b) *(0.7 puntos)* Indique la ocupación (%) de los buses siguientes, suponiendo que las transferencias se realizan sin problemas:

Bus USB 2.0: 6,25 / 60 = 10,41%
Bus USB Hub A: (6,25+2,5) / 480 = 1,82%
Bus GV: 1244,160 / 20000 = 6,22%
Bus PCIe x1: 0,528 / 500 = 0,11%
Bus NS: (6,25+2,5+0,528) / 2000= 0,46%
Bus PCIe x16: (6,25+2,5+0,528) / 8000 = 0,12%
Bus M: (6,25+2,5+0,528)*2/9600 = 0,19%

c) *(0.25 puntos)* Si mientras se están haciendo las transmisiones anteriores se quiere copiar un archivo de $1x10^9$ Bytes del HDA al Pendrive (vía HDA – M y M – HDA), indique cuánto tardaría en copiarse.

La velocidad más lenta es la del Pendrive (80 MB - 2,5MB) por tanto el tiempo en segundos será:
1000MB/77,5MB=12,90s