Estructura de Computadores – Parcial 2 — 4 Junio - 2019

Nombre: **Grupo:**

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

- **Cache de Instrucciones**: 8 KB, correspondencia directa, tamaño de bloque de **32 Bytes**.
- **Cache de Datos**: 1 KB, correspondencia asociativa por conjuntos de 2 vías, tamaño de bloque de **64 bytes**. Emplea ubicación en escritura (*write-allocate*), actualización posterior (*write-back*) y LRU para reemplazos.

Nº de conjuntos = Nº de líneas / nº de vías = 16 / 2 = 8

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

Cache de Instrucciones	
Etiqueta	32 - 8 - 5 = 19 19
Línea	Nº de líneas = T(MC) / T(Bl.) = 8 KB / 32 B = 256 -> Línea = log(256) = 8 8
Desplazamiento	Desp = log(T(Bl.)) = log(32) = 5 5

Cache de Datos	
Etiqueta	32 - 3 - 6 = 23 23
Conjunto	Conjunto = log(Nº de conjuntos) = log(8) = 3 3
Desplazamiento	Desp = log(T(Bl.)) = log(64) = 6 6

b) *(0.5 puntos)* Calcule el tamaño de la memoria de control requerido para la cache de datos

	Cache de Datos
Número de líneas en la memoria de control	Nº de líneas = T(MC) / T(Bl.) = 1 KB / 64 B = 16 16
Número de bits de cada entrada (indique el nombre de los campos)	1 (Valid) + 23 (etiqueta) + 1 (Modified - dirty, solo si es write-back) + 1 (LRU -> si no es correspondencia directa) = 26 bits
Tamaño total de la memoria de control (en bits)	16 × 26 = 416 bits

c) El siguiente programa realiza la multiplicación de la matriz M por el vector V, dando como resultado el vector Z. Todos los elementos corresponden a enteros de 4 bytes. La matriz M está constituida por 256 elementos, organizados en 16 filas y 16 columnas. Los elemenos de la matriz se hallan almacenados en memoria de forma consecutiva, empezando por los de la fila 1 y terminando con los de la fila 16.

$$\begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & \dots & m_{1,16} \\ m_{2,1} & m_{2,2} & m_{2,3} & \dots & m_{2,16} \\ m_{3,1} & m_{3,2} & m_{3,3} & \dots & m_{3,16} \\ & & \dots\dots & & \\ m_{16,1} & m_{16,2} & m_{16,3} & \dots & m_{16,16} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{16} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{16} \end{bmatrix}$$

$$z_k = \sum_{j=1}^{j=16} m_{k,j} \times v_j$$

La componente k del vector Z corresponde al producto escalar de la fila k de la matriz M por el vector V

```
            .data 0x20000000
M:          .word 1,2,3,4, ... 256     # Matriz 256 enteros (16 filas x 16 columnas)
            .data 0x20002000
V:          .word 1,2,3,4, ... 16      # Vector columna de 16 valores enteros
            .data 0x20004000
Z:          .space 64                  # Vector resultado de 16 valores enteros

            .text 0x00400000
__start:    lui $t0,0x2000             # Puntero a matriz M en $t0
            ori $t2,$t0,0x4000         # Puntero al vector resultado Z en $t2
            li $t3,16                  # Inicializa contador de iteraciones en k
L0:         ori $t1,$t0,0x2000         # Puntero al vector V en $t1
            li $t6,16                  # Inicializa contador de iteraciones en j
            ori $s0,$zero,0            # Inicializa registro acumulador $s0
L1:         lw $t4,0($t0)              # Lee M[k,j] (k=0..15, j=0..15)
            lw $t5,0($t1)              # Lee V[j]
            mul $t4,$t5                # M[k,j] × V[j]
            mflo $t4                   # Acumula resultado multiplicación en $t4
            add $s0,$s0,$t4            # Acumula suma parcial en $s0
            addiu $t0,$t0,4            # Incrementa puntero a matriz M[k,j+1]
            addiu $t1,$t1,4            # Incrementa puntero a vector V[j+1]
            addi $t6,$t6,-1            # Decrementa contador de iteraciones j
            bne $t6, $zero,L1          # Sigue iterando mientras contador > 0
            sw $s0,0($t2)              # Almacena producto parcial en Z[k]
            addi $t3,$t3,-1            # Decrementa contador de iteraciones k
            bne $t3, $zero,L0          # 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**
3	0x20000	0x20002	3
Total de ACCESOS a código (Indique el cálculo)	3 + [3 + 9x16 (bucle L1) + 3]x16 (bucle L0) = 2403		
Tasa de aciertos (Con cuatro dígitos decimales. Indique el cálculo)	H = (2403 – 3) / 2403 = 0,9988		

c.2) *(0.5 puntos)* Indique los números de bloque del primer y último bloque correspondientes a la matriz y a los dos vectores, así como los conjuntos 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**
M	16	0x 800000	0x 80000F	0x0	0x7
V	1	0x 800080	0x 800080	0x0	0x0
Z	1	0x 800100	0x 800100	0x0	0x0

c.3) *(0.7 puntos)* Calcule, para la **cache de datos**:

Total de ACCESOS a datos	256 a M + 16x16 a V + 16 a Z = 528 accesos
Total de FALLOS de datos	**Fallos de inicio:** 16 fallos en M + 1 fallo en V + 1 fallo en Z = 18 fallos **Fallos de colisión/capacidad:** 1 fallo en V + 1 fallo en Z = 2 fallos Total Fallos= 20
Tasa de aciertos	H = (528 accesos - 20 fallos) / 528 accesos = 0.9621
Número de reemplazos de bloque	2 de M (filas 0 y 8) + 1 de V + 1 de Z = 4
Número de escrituras a memoria *(palabras de 64 bits)*	8 palabras de 64 bits (1 bloque) de Z

c.4) *(0.3 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 3 bloques correspondientes a fila 1 de M, al vector V y al vector Z, se mapearían sobre la misma línea de cache y colisionarían entre sí. Fila 1 de M y vector V fallarían continuamente durante el cálculo de su producto escalar (primera componente de Z). Una vez calculada la primera componente del vector Z, V y Z colisionarían al inicio (vector V) y al final (vector Z) del cómputo de las 15 componentes restantes, con lo que el total de fallos será:

Fallos de inicio: 16 fallos en M + 1 fallo en V + 1 fallo en Z = 18 fallos

Fallos de colisión: 15 fallos en M (fila 1) + 30 fallos en V (15 en colisión con fila1 de M y 15 en colisión con Z) + 15 fallos en Z= 60 fallos

2 **(3 puntos)** Se dispone de un cajero automático, el cual se halla controlado por un MIPS R2000. Actualmente, dicho cajero solo procesa tarjetas tipo MONEDERO y permite seleccionar únicamente tres importes para retirar: 50€, 100€ y 300€. El sistema está constituido, entre otros, por tres módulos de interfaz: el **TARJETERO,** el **TECLADO** y el **EXPENDEDOR DE BILLETES**, definidos más abajo. Asimismo, existe la variable de sistema `sesion`, cuya función se explica más adelante junto a su definición. La operación en el cajero se inicia introduciendo una tarjeta en el tarjetero, que verifica su clave y obtiene su tipo y saldo. Un proceso se encarga de gestionar de forma continuada el Tarjetero por consulta de estado.

El TECLADO dispone de teclas especiales para cada uno de los tres importes: <50>, <100> y <300>, además de la tecla <CANCEL>, todas ellas gestionadas por la interrupción INT1*. También dispone de las teclas numéricas <0-9> y <ENTER> para introducir el código clave, pero estas teclas no generan interrupción y se procesan al margen por el hardware. El EXPENDEDOR DE BILLETES incluye una impresora para imprimir el justificante del importe retirado en €. Dicho importe debe ser almacenado previamente en el correspondiente registro de su interfaz.

TARJETERO (Dir. Base 0XFF000000) Este dispositivo se controla mediante sincronización por consulta de estado

- Registro de **ESTADO** (solo lectura, 8 bits, DB+0)
  - Bit 0 – **R**: (bit Ready) La interfaz lo pone a 1 cada vez que se introduce una tarjeta y se valida su clave.
  - Bits 7,6: **TIPO-** Tipo de tarjeta:
    00: DEBITO
    01: CREDITO
    11: MONEDERO
- Registro de **CONTROL** (solo escritura, 8 bits, DB+2)
  - Bit 2 – **EX**: (bit de Expulsión) Poniendo EX=1 se ordena la expulsión de la tarjeta
  - Bit 6 – **CL**: (bit Cancel) Se pone a 1 para poner a cero (cancelar) el bit R
- Registro de **SALDO** (lectura y escritura, 32 bits, DB+4): Importe del saldo en € disponible en la tarjeta

TECLADO (Dir. Base 0XFF010000) Este dispositivo se controla mediante sincronización por interrupción

- Registro de **CANTIDAD** (solo lectura, 32 bits, DB+4) Almacena cantidad € según tecla 50/100/300 pulsada
- Registro de **ESTADO** (lectura y escritura según bits, 8 bits, DB+8)
  - Bit 0 – **R**: (bit Ready – solo lectura) La interfaz lo pone a 1 con cada vez que se pulsa alguna de las siguientes teclas clave: <CANCEL>, <50>, <100> y <300>. Además, si E=1, entonces se activa la interrupción **INT1*** del MIPS. R se pone a 0 automáticamente tras leer el registro de ESTADO.
  - Bit 3 – **E**: (bit de Enable – solo escritura) Se pone a 1 para habilitar la interrupción en la interfaz y a 0 para inhibirla.
  - Bits 7 – **TECLA** (bit de tecla especial pulsada **–** solo lectura):
    0: 50/100/300
    1: CANCEL

EXPENDEDOR DE BILLETES (Dir. Base 0XFF020000) Este dispositivo emplea E/S Directa

- Registro de **CONTROL** (solo escritura, 8 bits, DB+0)
  - Bit 2 – **P**: (bit de Print – solo escritura) Se pone a 1 para imprimir justificante del importe retirado.
  - Bit 3 – **EB**: (bit de Expulsión Billetes – solo escritura) Se pone a 1 para abrir trampilla de entrega de billetes.
- Registro de **ENTREGA** (solo escritura, 32 bits, DB+8): Importe a entregar en €

Se supone la existencia de la siguiente variable del sistema:

```
                .kdata
sesion:         .byte 0  ;sesión 'open'(=1) o 'closed'(=0)
```

Se pide:

a) *(0.75 puntos)* Programe la siguiente función del sistema:

Función	Índice	Argumentos	Resultado
inicializar	$v0 = 600	-----	Inicializa la variable `sesion` a cero. Habilita la interrupción en el periférico de TECLADO, desenmascara la interrupción INT1 y deja las interrupciones globalmente habilitadas al retornar.

```
inicializar:  sb $zero,sesion          # inicializa var sesion=0
              la $t0,0xFF010000        # DB del Teclado
              li $t1,0x08
              sb $t1,8($t0)            # E=1 en Teclado
              mfc0 $t1,$12             # lee reg.estado MIPS ($12)
              ori $t1,$t1,0x0204       # bits IM1=IEp=1
              mtc0 $t1,$12             # actualiza $12
              j retexc
```

Registro de Estado ($12) del MIPS

b) *(1 punto)* Programe el código del proceso del Kernel encargado de gestionar el TARJETERO. Dicho código debe estar a la espera de que se introduzca una tarjeta con clave correcta. Una vez introducida, deberá identificar su tipo. Si la tarjeta es de tipo MONEDERO, se iniciará sesión poniendo la variable `sesion` a 'Open' (=1), en caso contrario se ordenará su expulsión. Este código **debe estar en permanente ejecución**.

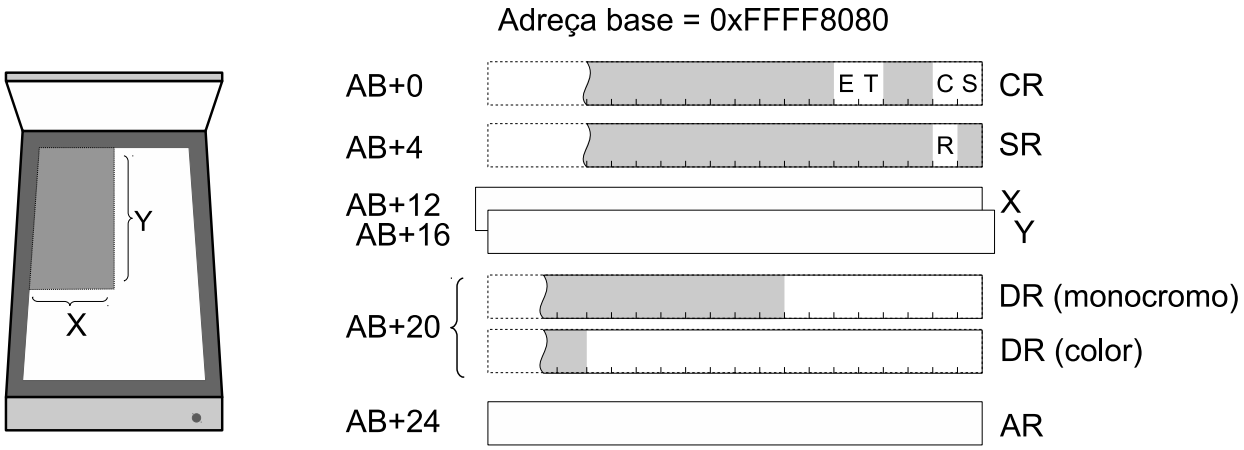
```
              la $t0,0xFF000000     # DB del Tarjetero
        L0:   lbu $t1,0($t0)        # lectura ESTADO
              andi $t2,$t1,1        # máscara consulta bit R
              beqz $t2, L0          # si R=0, seguir en bucle
              li $t2,0x40
              sb $t2,2($t0)         # cancela R
              andi $t1,$t1,0xC0     # máscara consulta tipo tarjeta
              li $t2,0xC0
              beq $t1,$t2,ini_sesion # si tarjeta tipo Monedero, inicia sesión
              li $t1,0x04           # mascara para EX=1
              sb $t1,2($t0)         # expulsa tarjeta
              j L0                  # regresar a bucle
ini_sesion:   li $t1,1
              sb $t1,sesion         # apertura sesión (sesion='open')
              j L0                  # regresa a bucle
```

c) *(1.25 puntos)* Programe el código de la rutina de servicio de interrupción INT1* del TECLADO, encargada de gestionar las teclas especiales <50>, <100>, <300> y <CANCEL>. Dicha rutina deberá comenzar comprobando si `sesion`='open' (=1), en cuyo caso procederá a averiguar cuál de las teclas especiales se ha pulsado y a actuar de acuerdo a la misma, tal y como muestra el pseudocódigo a continuación. Las teclas numéricas normales <0-9> sirven solo para introducir el código y se procesan independientemente por el hardware, siendo pues ignoradas.

```
si sesion == 1 entonces   ;consulta si sesion está 'open'
        si TECLA == CANCEL entonces   ;expulsar tarjeta, cerrar sesión y salir
        sino    ; TECLA == 50/100/300
                si SALDO > CANTIDAD  entonces
                        ;actualizar SALDO, anotar CANTIDAD en reg. ENTREGA del EXPENDEDOR DE BILLETES
                        ;entregar billetes, imprimir justificante, expulsar tarjeta, cerrar sesión y salir
                sino  ;ignorar y salir
sino  ;ignorar y salir
```

```
Int1:           la $t0,0xFF010000    # DB del Teclado
                la $t1,0xFF000000    # DB del Tarjetero
                lb $t2,8($t0)        #lee ESTADO y cancela (0→R)
                lb $t3,sesion        # lee variable sesion
                beqz $t3,fin
                andi $t2,$t2,0x80    # máscara para consultar bit TECLA
                bnez $t2,cancel      #si bit TIPO=1, tecla CANCEL
                                     # sino, teclas 50/100/300
50_100_300:     lw $t2,4($t0)        # lee cantidad euros solicitada
                lw $t3,4($t1)        # lee saldo tarjeta
                blt $t3,$t2,fin      # si saldo<cantidad, ignorar y salir
                sub $t3,$t3,$t2      # decrementa saldo
                sw $t3,4($t1)        # actualiza saldo tarjeta
                la $t0,0xFF02000     # DB del Expendedor de Billetes
                sw $t2,8($t0)        # almacena importe retirado en ENTREGA
                li $t2,0x0C          # P=1 y EB=1
                sb $t2,0($t0)        # entrega billetes, imprime justificante
     cancel:    li $t2,0x04          # máscara para hacer EX=1
                sb $t2,2($t1)        # expulsa tarjeta
cierra_sesion:  sb $zero,sesion      # cierra sesión. Var sesion='closed'
          fin:  j retexc
```

3 ***(2,5 puntos)*** Teniu un escàner connectat a un MIPS. Aquest perifèric digitalitza imatges rectangulars des del cantó superior amb una alçada *y* i amplada *x* donades en píxels, fent que l'escàner genere un total de *x*×*y* píxels en cada operació. Si l'escàner opera en monocromo, cada píxel es codifica en 8 bits; si ho fa en color, cada píxel es representa en 16 bits.
La interfície està ubicada en l'adreça 0xFFFF8080 i s'ha connectat a la línia d'interrupció int5*. Pot operar en mode PIO o en mode ADM. Conté els registres que es descriuen tot seguit:



- **CR:** Registre d'ordres (AB+0, només escriptura).
  - Bit 0: **S** (start): en escriure un 1 comença l'operació de l'escàner
  - Bit 1: **C** (color) a 1 indica color (16 bits per píxel) i a 0 monocromo (8 bits per píxel). Només útil si S=1.
  - Bit 4: **T** (mode de transferència): a 1 indica ADM, a 0 indica PIO
  - Bit 5: **E** (habilitació d'interrupció): mentre el bit R del registre d'estat val 1, activa la línia d'interrupció
- **SR:** Registre d'estat (AB+4, lectura/escriptura)
  - Bit 1: **R** (preparat) Depèn del mode de funcionament. En mode PIO, val 1 quan el registre de dades conté un nou píxel de la imatge. En mode ADM, val 1 quan l'escàner ha transferit tota la imatge. **Hi cal escriure un 0 per tal de cancel·lar-lo.**
- **X i Y:** Coordenades X (AB+12, escriptura) i Y (AB+16, escriptura):
  Especifiquen l'amplada i l'altura del rectangle que s'escaneja.
- **DR:** Registre de dades (AB+20, lectura):
  Només és útil en mode PIO. Conté les dades d'un píxel. Si opera en monocromo, són vàlids els bits 7 al 0; si ho fa en color, són vàlids els bits del 15 al 0.
- **AR:** Registre d'adreça (AB+24, escriptura):
  Només útil en mode ADM. Conté l'adreça inicial del buffer.

Noteu que la interfície no disposa de comptador per a operar en mode ADM. La lògica de l'adaptador dedueix internament el nombre de transferències de píxel multiplicant el contingut dels registres X i Y.

La intenció és oferir dues funcions del sistema:

MonoScan	1035	$a0 adreça del buffer $a1 amplada $a2 altura	—	Escaneja en monocromo
ColorScan	1036	$a0 adreça del buffer $a1 amplada $a2 altura	—	Escaneja en color

El codi inicial de la funció *MonoScan* és el següent:

```
1     fun1035:  la $t0,0xFFFF8080
```

```
2               sw $a1,12($t0)  # Paràmetre X
3               sw $a2,16($t0)  # Paràmetre Y
4               mult $a1,$a2
5               mflo $t2        # $t2 = nombre de píxels
6               li $t1,1
7               sw $t1,0($t0)   # Ordre d'escanear
8     sinc:     lw $t1,4($t0)   # Sinconització
9               andi $t1,$t1,2
10              beqz $t1, sinc
11              sw $zero,4($t0) # Cancel·lació
12              lbu $t1,20($t0) # Transferència
13              sb $t1,0($a0)
14              addi $a0,$a0,1  # Avança punter i comptador
15              addi $t2,$t2,-1
16              bgtz $t2,sinc
17              j retxec
```

a) *(0,2 punts)* És convenient aquesta implementació de la funció MonoScan si voleu que el computador treballe amb un sistema operatiu multiprogramat? Raoneu la resposta.

No és convenient. El tractament fa una consulta d'estat esperant que l'escàner complete l'operació. Els sistemes operatius multiprogramats aprofiten aquest temps d'espera per a assignar CPU a altres processos preparats

b) *(0,3 punts)* Escriviu el codi de la funció ColorScan fent servir les mateixes tècniques de sincronització i de transferència que aplica la versió inicial de MonoScan. No cal que escriviu tot el codi, només senyaleu les diferències amb el codi de MonoScan indicant la línia modificada, com mostra l'exemple.

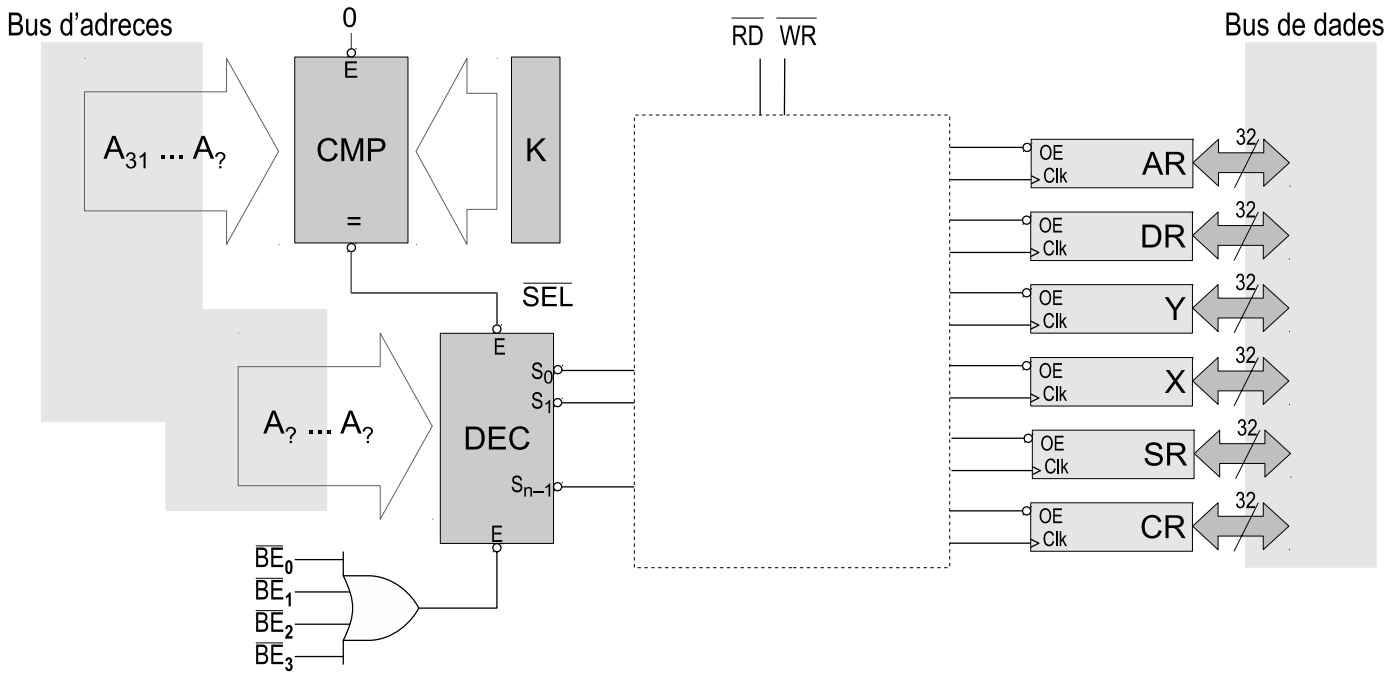
```
(1) fun1036:  la $t0,0xFFFF8080
(6) li $t1,3
(12) lhu $t1,20($t0)
(13) sh $t1,0($a0)
(14) addi $a0,$a0,2
```

c) *(1 punt)* Implementeu de nou la funció MonoScan amb sincronització per interrupcions i transferència ADM. Escriviu els tractaments de la nova funció MonoScan i de la interrupció int5. Feu servir les crides a suspen_aquest_procés i a activa_processos_en_espera on calga.

```
fun1035:  la $t0,0xFFFF8080
          sw $a0,24($t0)
          sw $a1,12($t0)
          sw $a2,16($t0)
          li $t1,0x31
          sw $t1,0($t0)
          jal suspen_aquest_proces
          j retexc
```

```
Int5:     la $t0,0xFFFF8080
          sw $zero,4($t0)       # R=0
           sw $zero,0($t0)      # S=C=T=E=0 desconfigura ADM
          jal activa_proc_en_espera
          j retexc
```

d) *(1 punt)* El circuit de selecció de la interfície i dels seus registres té l'estructura següent:



d1) *(0,2 punts)* Quines línies del bus d'adreces caldrà connectar al comparador CMP?

A31...A5

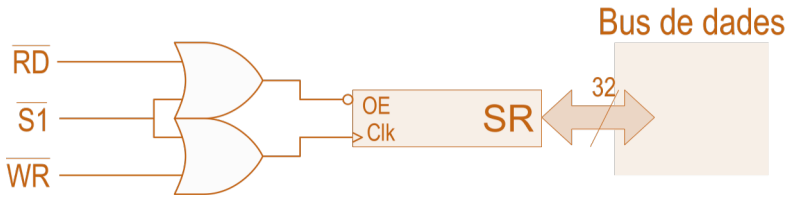
d2) *(0,2 punts)* Quina és la constant K? Escriviu-la en binari, abreujant amb punts suspensius si us convé.

1111 1111 1111 1111 1000 0000 100

d3) *(0,2 punts)* Quantes entrades i eixides ha de tindre el descodificador DEC?

3 entrades 8 eixides

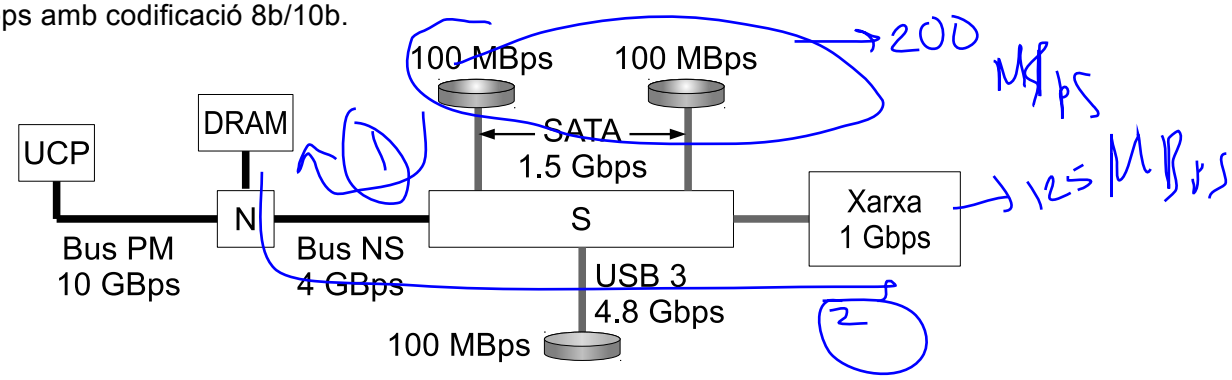
d4) *(0,4 punts)* Completeu la lògica de selecció del registre SR. Heu de dibuixar un circuit lògic amb tres entrades (RD*, WR* i el terminal del descodificador que indiqueu) i les dues eixides que es connectaran als terminals OE* i Clk del registre SR.



4 ***(1,5 punts)*** Un servidor ha de subministrar vídeo en temps real a través de la xarxa. Els clients s'hi connecten per la xarxa i demanen la transmissió d'un vídeo. Cada retransmissió implica la lectura d'un arxiu de vídeo i la transferència en temps real del seu contingut per la xarxa. Tots els vídeos estan codificats a una velocitat de dades de 800 kbps.

El servidor té els perifèrics i busos següents:

- Bus PM a 10 GBps i NS a 4 GBps.
- 2 discos interns, cadascun de 500 GB de capacitat, amb velocitat mitjana de transferència de 100 MB/s. Cada disc està connectat mitjançant un adaptador SATA que admet transferències de 1,5 Gbps amb codificació 8b/10b.
- 1 targeta de xarxa que opera a 1 Gbps efectius
- 1 disc extern de 1 TB de capacitat, amb velocitat mitjana de 100 MB/s, connectat a un port USB 3.0 a 4,8 Gbps amb codificació 8b/10b.



Calculeu

a) *(0,5 punts)* Quantes transmissions simultànies pot mantenir el servidor si tots els vídeos implicats estan en el mateix disc? Suposeu que la resta del tràfic dins del computador és menyspreable. Quin serà el percentatge d'amplada de banda consumida en el bus NS en aquest cas?

Cada transmissió suposa llegir 800 kbps (100 kBps) del disc i transmetre 800 kbps per la xarxa. El límit l'imposa el disc a 100 MBps, així que podran llegir-se 100 MBps/100 kBps = 1000 vídeos.

Pel bus NS circularan 100 MBps llegits del disc + 100 MBps transmesos per la xarxa = 200 MBps; el consum serà del 5%

b) *(0,25 punts)* Quantes transmissions simultànies pot mantenir el servidor si els vídeos implicats estan repartits per igual entre els dos discos?

Ara el límit l'imposa la velocitat de la xarxa, 1 Gbps. En aquestes condicions, el màxim serà de 1 Gbps/800 kbps = 1250 videos

c) *(0,25 punts)* Veieu que ambdós discs interns estan ocupats al 80% de la seua capacitat. Quantes hores de vídeo contenen? Menyspreeu l'espai ocupat per la resta d'arxius.

80 % de 500+500 GB és 800 GB

(800·10^9) bytes x 8 bits/byte /
3600 segons/hora x 800·10^3 bits/segon
= 2222 hores

d) *(0,5 punts)* Si desconnecteu el servidor de la xarxa, quin seria el mínim temps per a fer una copia de seguretat del contingut dels discos interns (ocupats al 80%) en el disc extern?

Cal transferir 800 GB
El límit de velocitat el fixa el disc extern: 100 MBps.
Faran falta 8000 segons = 2,22 hores = 2h 13’