ESTRUCTURA DE COMPUTADORS
Curs 2016/2017

Pràctica 12: Sincronització per consulta d'estat

Objectius

Afermar els coneixements en sincronització amb perifèrics mitjançant consulta de l'estat. Per aconseguir aquest objectiu s'han integrat en el simulador dos perifèrics: el teclat i la consola.

Material

Heu de fer aquesta pràctica amb el simulador **PCSpim-ES** i els arxius de codi font *espera.asm* i *eco.asm*. Tot aquest material està disponible en la carpeta corresponent de PoliformaT.

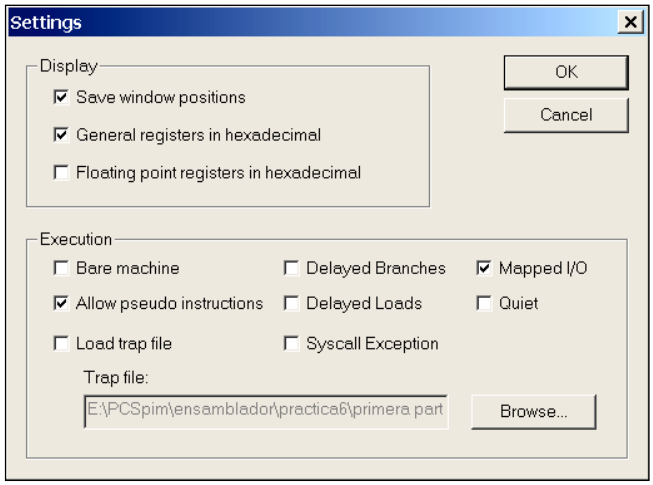


Figura 1. Configuració del simulador escaient per a la pràctica

PCSpim-ES és una versió del **simulador** *PCSpim* al què s'han afegit dos perifèrics. La fareu servir també també en la pràctica següent. Confirmeu que es tracta de la versió adaptada consultant el quadre *Help>AboutPCSpim* de l'aplicació una vegada oberta, que ha d'incluoure el text "PCSpim Version 1.0 - adaptación para las asignaturas ETC2 y EC, etc...". La figura 1 mostra la configuració escaient en el quadre de configuració del simulador (*simulator>settings*). Observeu que hi ha un botó nou, *Syscall Exception*, que heu de deixar sense marcar per ara.

L'E/S en PCSpim

Normalment, els programes d'usuari no accedeixen a les interfícies dels perifèrics mitjançant les instruccions d'accés a la memòria. És més, els sistemes operatius habituals prohibeixen als programes corrents aquest tipus d'accés por moltes raons. En el seu lloc, els programes han de cridar les funcions d'entrada/eixida del sistema que fan el treball de manera segura i eficient. Aquestes funcions sí accedeixen a les interfícies mitjançant les instruccions de memòria.

L'entorn de PCSpim és semblant a un sistema operatiu, però amb diferències importants:

- Les funcions d'E/S predefinides en PCSpim són simples i no estan pensades per al seu ús per part de processos concurrents, es poden vore al "Apèndix. Crides gestionades per PCSPIM".

- Els programes no estan obligats a usar les funcions predefinides i poden accedir (Figura 2) a la interfície dels perifèrics disponibles sense cap restricció

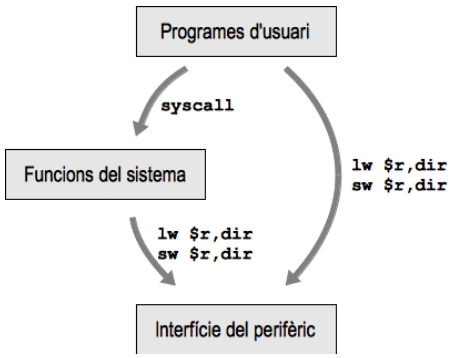


Figura 2. Accés als perifèrics sota PCSpim

Els perifèrics disponibles (Figura 3) són tres: teclat i consola. El teclat i la consola són els habituals del PCSpim que porten associades funcions conegudes, com `read_string` o `print_char`. Cadascun dels perifèrics té un adaptador simulat que crea una interfície amb la què heu de treballar en aquesta pràctica.

Figura 3. Esquema complet del computador simulat per PCSpim. A més a més del processador i la memòria están el monitor i el teclat.

1. La interfície del teclat

Quan PCSpim està actiu, el teclat és un perifèric d'entrada en el què l'operació elemental és la lectura d'un caràcter. Aquesta operació d'entrada es du a terme quan algú prem una tecla i la interfície subministra al programa el codi del caràcter teclejat. La interfície del teclat està formada pels dos registres que descriu la Figura 4. Ambdós registres estan accessibles en l'espai d'adreces del MIPS a partir de l'adreça base *DB* = `0xFFFF0000` i tenen 32 bits, encara que els bits útils es troben en la part menys significativa del contingut.

Observeu que per a llegir d'aquests registres podeu utilitzar les instruccions `lw`, `lhu`, `lh`, `lbu` o `lb` perquè els bits operatius es troben en el byte menys significatiu de la paraula. La utilització del bit E s'aplicarà en la pràctica següent; en aquesta pràctica heu de **mantenir el bit E = 0**.

El bit de preparat del teclat

El bit R indica que el teclado està preparat. Cada vegada que algú prem una tecla, aquest bit pren el valor 1. Llegint aquest bit, els programes poden saber quan cal tractar el perifèric.

Interfície del teclat (DB = 0xFFFF0000)

Nom	Adreça	Accés	Estructura
Estat/ordres	DB	L/E	E R
Dades	DB+4	L	COD

Registre d'ordres i estat (Lectura/escriptura. Adreça = Base).

- R (bit 0, només lectura). Indicador de dispositiu preparat: R = 1 cada vegada que es prem una tecla. Per tornar R = 0 cal fer un accés de lectura al registre de dades.
- E: (bit 1, lectura/escriptura). Habilitació de la interrupció (mentres E = 1, el valor R = 1 activa la línia d'interrupció del dispositiu).

Registre de dades (Només lectura. Adreça = Base + 4).

- COD (bits 7...0). Codi ASCII de la tecla premuda. Llegir aquest registre provoca que R = 0.

Figura 4. Descripció de la interfície del teclat. En aquesta pràctica heu de mantenir el bit E = 0

Activitat número 1. Comprensió de la consulta del bucle d'espera.

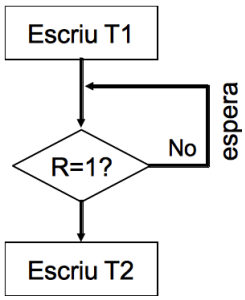
►Amb un editor de texts, obriu el programa *espera.asm* i observeu el seu codi. Identifiqueu les tres parts principals: l'escriptura d'una cadena T1 en la consola mitjançant una funció del sistema, el procés d'espera que se estudiarà tot seguit i l'escriptura de T2 en la consola.

La Figura 5 mostra l'estructura del programa, que conté un bucle que espera a que el bit de preparat valga 1. Observeu els detalls següents sobre l'espera:

- Carrega `$t0` amb l'adreça base de la interfície `0xFFFF0000`.
- Llig el registre d'estat de la interfície (desplaçament 0 respecte de l'adreça base).
- Aïlla el bit R mitjançant una operació lògica (and) i la màscara escaient.
- Només ix del bucle quan R=1.

► Carregueu el programa *espera.asm* amb el simulador i executeu-lo amb l'ordre *Simulator>Go* (F5). Notareu que inicialment apareix el text T1 en la consola, i que quan premeu una tecla apareix el text T2.

► Torneu a executar el programa sense tancar el simulador (per exemple, seleccionant *Simulator>Reload* en el menú de PCSpim). Voreu que ara no hi ha espera: la raó és que el bit de preparat de la interfície conserva el valor R = 1 des de l'execució anterior.




```
# Espera a que prema una tecla
        la $t0, 0xffff0000
espera: # Espera bit R == 1
        lw $t1,0($t0)
        andi $t1,$t1,1
        beqz $t1,espera
```

Figura 5. A l'esquerra, estructura del programa *espera.asm*. A la dreta, un detall del procés d'espera.

► ***Qüestió 1.***

Si canviara la interfície del teclat de manera que el bit *R* passara a ocupar la posició 5 de la paraula de control/estat, quina instrucció canviaríeu i com en ***espera.asm***?

Activitat número 2.
La cancel·lació del perifèric.

La cancel·lació és el mecanisme que torna a R el valor 0 i el deixa disposat per a una nova espera. En aquesta interfície, la cancel·lació es fa mitjançant la lectura del registre de dades.

►**Qüestió 2.** Modifiqueu *espera.asm* afegint una instrucció que llisca del registre de dades i deixe el contingut en *$t2*, com indica la Figura 6. Observeu que l'adreça d'aquest registre s'expressa en la Figura 4 com "DB+4".

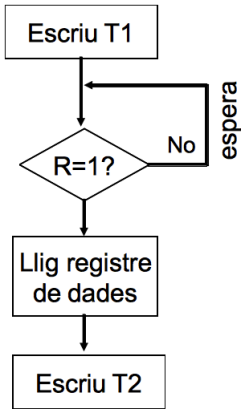


Figura 6. Estructura de *espera.asm* modificat.

► Proveu a executar vàries vegades seguides el nou programa modificat, sense tancar el simulador. Notareu que el programa sempre espera que algú prema una tecla, perquè el bit de preparat queda amb el valor R = 0 al final de l'execució.

2. Sincronització per consulta d'estat

Quan un programa espera que un perifèric estiga preparat llegint repetidament el seu registre d'estat, parlem de sincronització per consulta d'estat. En general, l'esquema de treball amb el perifèric és el que mostra la figura 7. El tractament depén del dispositiu i de l'operació que se faça amb ell, però en tot cas ha de cancel·lar el bit de preparat.

► Escriviu un programa *ascii.asm* que llisca del teclat les tecles que se premen i escriga el seu codi ASCII en la consola. El programa ha d'acabar en prémer una tecla que trieu (*enter*, el punt ".", etc). El tractament serà:

- Llegir el registre de dades de la interfície del teclat
- Escriure en pantalla el valor llegit mitjançant la funció del sistema `print_int`

L'esquema complet del programa serà:

Repetir
  esperar a que el teclat estiga preparat (bit R = 1)
  Tractament:
  llegir el registre de dades del teclat
  escriure en pantalla el valor llegit
*Fins que (***`caràcter llegit = caràcter triat`***)*

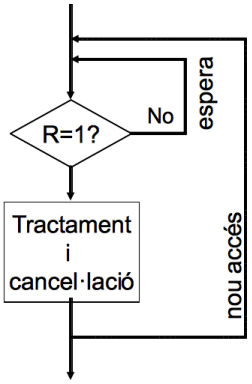


Figura 7. Flux habitual del tractament d'un perifèric mitjançant consulta de l'estat.

Per a escriure el programa podeu utilitzar moltes línies de codi d'*espera.asm*, per tant us recomanem que copieu *espera.asm* en un arxiu *ascii.asm* i anar introduint els canvis en aquest.

► ***Qüestió 3.*** Copieu açí les línies de codi que fan la sincronització i la lectura del registre de dades.

3. La interfície de la consola

La Figura 8 mostra la interfície de la consola de PCSpim. L'operació elemental d'eixida de dades per la consola consisteix en imprimir un caràcter. El caràcter se imprimeix quan el programa escriu el codi del caràcter en el registre de dades de la interfície. Respecte de la interfície del teclat, trobareu les diferències següents:

- L'adreça base ha canviat. Ara és `0xFFFF0008`.
- El registre de dades és d'escriptura.

Interfície de la consola (DB = 0xFFFF0008)

Nom	Adreça	Accés	Estructura
Estat/ordres	DB	L/E	E R
Dades	DB+4	E	COD

Registre d'ordres i estat (Lectura/escriptura. Adreça = Base).

- R (bit 0, només lectura). Indicador de dispositiu preparat: R = 1 quan la consola està disponible. R = 0 quan s'escriu en el registre de dades.
- E: (bit 1, lectura/escriptura). Habilitació de la interrupció (mentre E = 1, el valor R = 1 activa la línia d'interrupció del dispositiu).

Registre de dades (Només escriptura. Adreça = Base + 4).

- COD (bits 7...0). Codi ASCII del caràcter que voleu escriure en la consola. Escriure en aquest registre provoca que R = 0 mentres es processa l'operació.

Figura 8. Interfície de la consola de PCSpim. En aquesta pràctica heu de mantenir el bit E = 0.

El bit de preparat de la consola

La consola demana cert temps per a fer l'operació, al llarg del qual el bit R de preparat val 0 abans de tornar a valdre 1 quan l'operació termina. Per tant, abans d'escriure un caràcter cal esperar a que R = 1.

Activitat número 3. Funcions bàsiques amb el teclat i la consola.

Per al treball amb ambdós perifèrics se solen implementar dues funcions:

- `void putchar(char c);` escriu un caràcter en la consola
- `char getchar();` pren un caràcter del teclat

Ambdues funcions existeixen amb el mateix nom en la biblioteca d'entrada/eixida estàndard `<stdio.h>` dels llenguatges de programació C i C++, i tenen mètodes semblants en Java com `TextIO.put(char c)` i `TextIO.getAnyChar()`. L'entorn PCSpim les ofereix també com `read_char` i `print_char` (funcions 12 i 11).

► Obriu el programa *eco.asm* amb un editor i observeu-ne l'estructura. A partir de l'etiqueta __start trabarà el programa principal que crida a la funció *putchar* per a escriure en la consola els caràcters 'P12' i el final de línia. A continuació de l'etiqueta *bucle* hi ha una iteració que llig caracters del teclat cridant *getchar* i escriu l'*eco* en la consola cridant *putchar*. Adoneu-vos que el programa acaba en llegir la tecla d'escape (ASCII 27).

► Completeu el codi de les rutines `getchar` i `putchar` a continuació de les etiquetes corresponents. **Feu-lo seguint el conveni de programació habitual per a les funcions i sense utilitzar les funcions de PCSpim**. O siga:

- `getchar` s'ha de sincronitzar amb el teclat mitjançant consulta de l'estat, prendre el caràcter del seu registre de dades i desar-lo en *$v0*.
- `putchar` s'ha de sincronitzar amb la consola mitjançant consulta de l'estat i escriure en el seu registre de dades el contingut de *$a0*.

► ***Qüestió 4. Escriviu el codi de*** `getchar` ***i*** `putchar`**.**

► Executeu *eco.asm* (amb *Simulator>Run* o prement [F5]) i atureu-lo mitjançant *Simulator>Break* quan estiga esperant a llegir una tecla, després de mostrar en la consola el text "**`P12`**". Consulteu el valor del PC i identifiqueu la instrucció a la què apunta.

Instrucció:	Valor de PC (hex):

► ***Qüestió 5.*** Podeu explicar per què el programa s'ha aturat en aquest punt?

Apèndix. Crides gestionades per PCSPIM

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	