Apunts del Tema 8
Excepcions i Interrupcions

Joan Manuel Parcerisa

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Juny 2018

Licencia Creative Commons

Tema 8. Excepcions i interrupcions

Aquest document pretén servir de suport a l'assignatura EC de la titulació de Grau en Informàtica de la FIB. Encara que una gran part de la informació es pot trobar al llibre de text recomanat[1], aquesta es troba disseminada al llarg de múltiples capítols. Al marge esquerre hem referenciat els apartats d'aquest llibre relacionats. Les notes a peu de pàgina contenen informació que pot ser d'interès però no forma part del temari d'EC. La majoria de conceptes arquitectònics exposats són generals, no obstant els detalls referents al suport hardware per al mecanisme d'excepcions corresponen a l'arquitectura del processador MIPS.

4.9, B-7

1. Introducció

1.1 Concepte

Les excepcions són un mecanisme del processador que altera el flux de control normal d'un programa sense la intervenció de cap instrucció de salt. Inicialment, les excepcions van ser pensades per tractar events interns causats per condicions que alteren la interpretació normal d'una instrucció (p.ex. una fallada de pàgina, un accés a memòria no alineat, un codi d'operació il·legal), però aviat es va veure que el mateix mecanisme podia servir per tractar també events externs al processador, produïts per dispositius d'entrada/sortida que sol·liciten la seva atenció, i llavors els anomenem interrupcions[2].

Quan el processador detecta una condició d'excepció o una petició d'interrupció, interromp l'execució del programa, ja sigui finalitzant o cancel·lant la instrucció en curs, i comença a executar instruccions d'una rutina del sistema, la *Rutina de Servei d'Excepcions* (RSE), que farà el tractament adequat a la condició detectada. Més concretament, el processador escriu en el registre PC una adreça predeterminada per a aquests casos, l'adreça inicial de la RSE, de forma que la búsqueda d'instruccions continua a partir d'aquesta adreça com si s'hagués executat un salt. Un cop fet el tractament per part de la RSE, segons el tipus d'excepció es pot retornar el control al programa interromput o bé es pot avortar el programa. I si es retorna, segons el tipus d'excepció, es pot re-executar la instrucció causant de l'excepció o bé es pot continuar amb la següent.

1. D. Patterson, J. Hennessy, Computer Organization and Design, 4th edition. Morgan Kaufmann
2. En alguns textos les excepcions/interrupcions també es coneixen com interrupcions internes/externes o bé com excepcions síncrones/asíncrones.

1.2 Tipologia d'excepcions i interrupcions

a) Excepcions d'*overflow aritmètic*, *accés a memòria no alineat*, *divisió per zero de coma flotant*, *instrucció reservada*, etc: l'excepció és causada en intentar executar una instrucció que viola alguna restricció, ja sigui del seu format, dels seus operands o dels resultats. La instrucció en curs s'interromp sense escriure res en registres o en memòria. En alguns casos, el tractament de l'excepció per part de la RSE consisteix en *avortar* el programa, emetre un missatge d'error i donar pas a un altre programa en execució. En altres casos, la RSE pot resoldre el problema i escriu un resultat en el registre destinació de la instrucció causant de l'excepció. Un cop acabat el tractament, la RSE retorna al programa, *continuant* a partir de la instrucció següent.

b) Excepcions de *fallada de pàgina* i de *fallada de TLB* (veure apartat 5): es produeixen durant la traducció de l'adreça virtual a física d'un accés a memòria (ja sigui una búsqueda d'instrucció o un accés a dades). La instrucció en curs s'interromp sense escriure res en registres o en memòria, ni tampoc incrementar el PC. El tractament d'una fallada de TLB (la traducció no existeix en el TLB) consisteix a copiar la traducció de la taula de pàgines al TLB[3]. El tractament d'una fallada de pàgina (la pàgina no està present a la memòria física) consisteix a copiar la pàgina del disc a un marc de pàgina lliure en memòria i actualitzar la taula de pàgines i el TLB. Un cop fet el tractament, es retorna al programa *reexecutant la instrucció* que ha causat l'excepció.

c) Excepcions de *trap* i de *crida al sistema* (veure apartat 6): l'excepció és causada per la interpretació d'una instrucció especial del llenguatge màquina, la funció de la qual és precisament causar una excepció a fi que la RSE executi algun servei, com si es tractés d'una crida a subrutina (de fet, fins i tot pot rebre paràmetres i retornar resultats). MIPS disposa de la instrucció *syscall* i de diverses instruccions de trap condicional (*teq*, etc). Un cop acabada la RSE es retorna al programa, *continuant* a partir de la instrucció següent.

d) Interrupcions (veure apartat 7): Són causades per peticions d'elements externs al processador que requereixen la seva atenció, p.ex. dispositius d'E/S. Les peticions no es poden atendre immediatament si hi ha una instrucció en curs d'execució, sinó que queden registrades en un registre de peticions pendents. Les peticions pendents es monitoritzen i es poden atendre quan la instrucció en curs ha finalitzat i el PC s'ha incrementat. Per servir la interrupció, la RSE acostuma a transferir dades de/a algun dispositiu. Un cop servida, reprèn el programa interromput, *continuant* amb la següent instrucció.

Així doncs, hem vist que en alguns casos, després que la RSE acaba el tractament de l'excepció, el processador retorna el control al programa interromput perquè continuï executant-se. Per a poder continuar el programa en el futur, cal que la RSE salvi inicialment l'estat del programa i, un cop finalitzat el tractament corresponent, en restauri l'estat i executi a partir del punt d'interrupció. En molts d'aquests casos, el tractament de l'excepció pot requerir llargues esperes a algun dispositiu d'E/S, p.ex. llegir al disc, en un fallo de pàgina. Tenint en compte que el temps d'accés a un dispositiu d'E/S sol ser diversos ordres de magnitud major que el d'un accés a memòria, i per evitar esperes improductives, el sistema operatiu pot optar per fer altres tasques o programes mentre dura l'operació d'E/S.

3. A diferència del MIPS, en processadors com els d'Intel, la fallada de TLB no causa una excepció sinó que es resol per hardware, executant una seqüència de microcodi d'una ROM interna del processador.

B-7

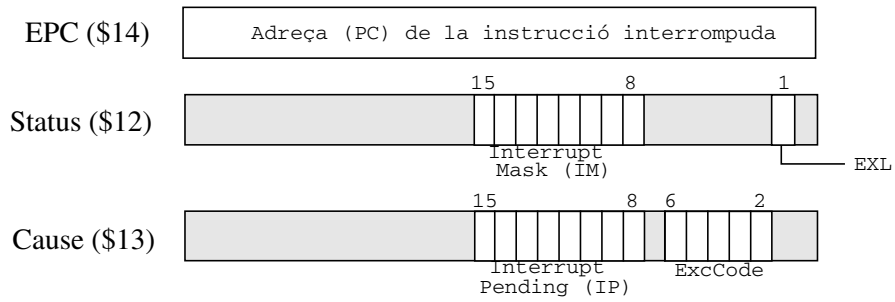
2. El Coprocessador del Sistema CP0

El tractament d'una excepció el realitza el Sistema Operatiu, però necessita suport de l'arquitectura del processador. En el MIPS, aquesta funció de suport la desenvolupa un circuit annex al processador que rep el nom de Coprocessador del Sistema (CP0). Aquest coprocessador controla les excepcions i la traducció ràpida d'adreces virtuals (TLB).

El CP0 disposa d'un banc de registres específics que denominem pel seu nom, encara que el codi màquina els designa pel seu número (de 5 bits). Aquests registres no poden ser usats com a operands de cap de les instruccions estudiades fins ara. Solament poden aparèixer com a operands explícits de les instruccions *mfc0* ("move from coprocessor 0") i *mtc0* ("move to coprocessor 0"):

```
mfc0   rt, c0_rd     # copia el registre c0_rd del CP0 en rt
mtc0   rt, c0_rd     # copia rt en el registre c0_rd del CP0
```

Els registres del CP0 també són usats, com a operands implícits, per instruccions especials, com *eret* i *tlbwr,* que s'explicaran més endavant. A continuació s'expliquen tres d'aquests registres, que intervenen en el tractament d'excepcions. Més endavant s'explicaran uns altres tres registres que intervenen en la gestió del TLB.



a) Registre EPC (*Exception Program Counter*)

Registre de lectura/escriptura. En detectar una condició d'excepció o una petició d'interrupció, abans de procedir a atendre-la, el processador escriu el valor actual del PC en el registre EPC. En cas d'excepció, com que la instrucció en curs s'interromp sense finalitzar, el PC que es guarda és el de la instrucció interrompuda. En canvi, si es tracta d'una interrupció, com que la instrucció en curs ja ha finalitzat, el PC que es guarda és el de la pròxima instrucció a executar.

b) Bit EXL (*Exception Level*) del registre *Status*

Bit de lectura/escriptura. Indica el *mode d'execució* i el *permís d'interrupcions*:

- Si EXL=0, el processador està en "mode usuari", és el valor per defecte en els nostres programes. Si EXL=1, el processador està en "mode sistema"[4] (té accés a adreces del sistema i pot executar instruccions privilegiades), és el valor que adquireix quan es produeix una excepció o una interrupció.
- Si EXL=0, les interrupcions estan permeses. Si EXL=1, totes les interrupcions resten inhibides (les peticions s'ignoren). Cal observar que no existeix cap bit similar per inhibir les excepcions.

4. S'explicarà en detall a l'apartat 6. Crides al Sistema, a la pàg. 13.

c) Camp *ExcCode* del registre *Cause*

Conté 5 bits de sols lectura. Quan es detecta una excepció o interrupció, el processador hi escriu un número que codifica la causa de l'excepció. La següent taula mostra alguns valors del *ExcCode*[5]:

Núm	Nom	Causa
0	Int	interrupció d'un dispositiu hardware d'E/S
1	Mod	fallada de TLB per pàgina modificada (primera escriptura a pàgina)
2	TLBL	fallada de TLB (o fallada de pàgina) per lectura
3	TLBS	fallada de TLB (o fallada de pàgina) per escriptura
4	AdEL	error d'adreça per lectura (accés mal alineat, o a espai de sistema no permès)
5	AdES	error d'adreça per escriptura (accés mal alineat, o a espai de sistema no permès)
6	IBE	error de bus (instruction fetch), p.ex. adreça física inexistent
7	DBE	error de bus (dades de load/store), p.ex. adreça física inexistent
8	Sys	crida al sistema (causada per la instrucció syscall)
9	Bp	breakpoint (causada per la instrucció break)
10	RI	instrucció reservada. p.ex. codis d'operació inexistents
11	CpU	coprocessador no implementat, p.ex. accés a CP0 en mode usuari
12	Ov	overflow aritmètic d'enters (instruccions add, sub, addi)
13	Tr	trap (causada per una instrucció de trap)
15	FPE	excepció de coma flotant: consultar detalls als registres del coprocessador CP1

d) Camp IP (*Interrupt Pending*) del registre *Cause*

Conté 8 bits de sols lectura, un per a cada tipus d'interrupció. A cada senyal de petició d'interrupció d'un dispositiu d'entrada/sortida se li assigna un d'aquests bits. Quan un dispositiu activa el seu senyal de petició, el bit corresponent del camp IP es posa a 1, indicant al processador que un dispositiu d'aquest tipus ha sol·licitat una interrupció. El bit roman a 1 mentre el dispositiu mantingui el senyal de petició activat.

e) Camp IM (*Interrupt Mask*) del registre *Status*

Conté 8 bits de lectura/escriptura, un per cada tipus d'interrupció. Cada bit és un permís d'interrupció específic de cada un dels 8 tipus: si el bit val 1, les peticions d'aquest tipus s'atendran, altrament s'ignoraran. Naturalment, si EXL=1 s'ignoren totes les interrupcions, independentment dels bits d'IM.

4.9, B-7

3. Primer pas: Accions hardware en cas d'excepció o interrupció

Quan es produeix un event que causa una excepció o interrupció es desencadenen una sèrie d'accions en el processador. Les accions inicials tenen lloc en el hardware i s'expliquen aquí. El tractament de l'excepció o interrupció es completarà després per software en la rutina d'excepcions RSE, i ho veurem a l'apartat 4.

a) En el cas d'una excepció causada pel propi programa, la instrucció en curs s'avorta, és a dir que es dóna per finalitzada la seva execució sense escriure cap resultat en

5. MARS sols implementa les excepcions Int, AdEL, AdES, Bp, Ov, Tr i Sys, però les simula internament, com si es tractés d'una pseudoinstrucció, sense invocar la rutina d'excepcions. En canvi, per a la resta que no estan implementades sí que salta a la rutina d'excepcions.

memòria ni en registres, i sense incrementar el PC. En canvi, si es tracta d'una petició d'interrupció procedent d'un dispositiu d'entrada/sortida, aquesta no es monitoritza fins que la instrucció en curs ha finalitzat l'execució i el PC ja s'ha incrementat. Si llavors es comprova que el dispositiu *i* ha fet una petició (bit IP_i=1), per atendre-la caldrà comprovar també que estan activats el permisos d'interrupcions, tant el permís general (bit EXL=0) com l'específic del dispositiu *i* (bit IM_i=1).

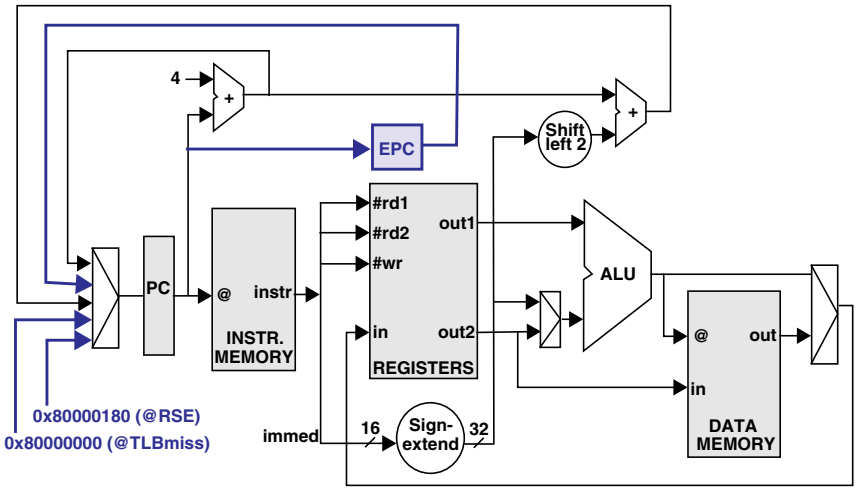
b) Es guarda el contingut del registre PC en el registre EPC (*Exception Program Counter*), per poder continuar més tard el programa interromput en cas necessari.

c) S'escriu la causa en el camp *ExcCode* de 5 bits del registre *Cause*. Si es produeixen dues causes o més simultàniament, el processador aplica unes regles de prioritat per determinar quina d'elles es codifica, així per exemple si es produeix una excepció qualsevol al mateix temps que una interrupció, es codifica sempre l'excepció.

d) Es posa a 1 el bit EXL del registre *Status*. A partir d'ara, i mentre EXL=1, les interrupcions dels dispositius restaran *inhibides* i el processador operarà en *mode sistema*, és a dir que tindrà dret a accedir a adreces de memòria del SO (on resideix la RSE) i tindrà dret a executar instruccions privilegiades (les que donen accés al coprocessador CP0).

e) El processador escriu en el PC l'adreça inicial predeterminada de la *Rutina de Servei d'Excepcions* (RSE) que val 0x80000180 (o bé de la rutina *TLBmiss*, que val 0x80000000, per a fallades de TLB)[6]. Aquestes adreces pertanyen a l'espai de memòria reservat al SO (adreces amb el bit 31=1).

El següent esquema mostra un possible disseny del camí de dades d'un processador MIPS. Es poden veure, en traç més gruixut i en color blau, els elements afegits per al tractament d'excepcions: es pot observar el camí que permet salvar el contingut del PC de la instrucció en curs en el EPC (acció b)[7]; també hi ha tres entrades addicionals en el multiplexor d'entrada al PC: dues són les adreces de les rutines RSE i TLBmiss, que s'establei-



6. Els processadors tenen dues maneres de comunicar la causa d'una excepció a la rutina que en farà el tractament: escrivint un codi en un registre (MIPS usa aquest mètode per a la majoria d'excepcions, escrivint la causa en el camp ExcCode); o bé saltant a una adreça diferent (anomenada *vector*) per a cada tipus d'excepció (MIPS usa aquest mètode per a les fallades de TLB). Aquest últim mètode s'anomena *interrupcions vectoritzades.*

xen per fer el tractament de l'excepció (acció e). La tercera prové del registre EPC, i és on se salta al final de la RSE, executant la instrucció *eret* (veure següent apartat).

B-7

4. Segon pas: Accions a fer pel software (RSE)

Quan s'inicia l'execució de la RSE, el bit EXL val 1, i per tant les interrupcions estan inhibides. En EC suposarem que segueixen així fins al final de la RSE[8].

a) Primer de tot, la RSE ha de preservar l'estat del programa interromput, salvant tots els registres en memòria (normalment a la pila, on hi haurà reservat l'espai suficient). Aquí no regeixen les regles del ABI explicades al Tema 3, segons les quals alguns registres no és necessari salvar-los, ja que aquesta subrutina no ha estat invocada voluntàriament per una instrucció de salt del programa sinó com a conseqüència d'un event imprevisible. S'han de salvar tots. Les dues úniques excepcions són:

- Els registres $k0 i $k1, que són reservats per a l'ús exclusiu del sistema operatiu, i els programes no els haurien de fer servir
- Els registres de coma flotant, que es fan servir en molt poques excepcions i per tant se salvaran sols en aquelles subrutines que els facin servir i siguin invocades des de la RSE.

b) Després, la RSE ha d'identificar la causa de l'excepció, consultant el camp *ExcCode* del registre *Cause*, i saltar a una subrutina específica per a cada causa, que en farà el tractament i se sol conèixer com *exception handler*.

c) S'executa la rutina d'excepcions específica. El tractament que faci aquesta rutina pot consistir en: (i) avortar el programa; (ii) bloquejar-lo (donant pas a un altre programa, en sistemes multiprogramats[9]); o (iii) solucionar el problema i retornar després al programa interromput (com en el cas d'un fallo de pàgina).

d) Finalment, un cop que la rutina específica ha retornat, considerem si cal ajustar l'adreça de retorn guardada a l'EPC: en cas d'excepcions com les crides al sistema (*syscall*), si es retornés a l'adreça de l'EPC es reexecutaria la mateixa instrucció creant un bucle infinit. Per evitar-ho, la RSE incrementa en 4 el valor de l'EPC. En la resta de casos no cal cap ajust: en el cas d'interrupcions l'EPC ja conté l'adreça de la següent instrucció a executar; i en el cas d'excepcions com fallos de TLB o fallos de pàgina, conté l'adreça de la instrucció que l'ha causat i que volem que es reexecuti.

e) Restaurar tots els registres salvats al principi de la rutina RSE.

---

7. Aquest camí difereix del que apareix a la figura 4.66 del llibre de referència (veure nota al peu núm 1), però concorda amb la definició de MIPS, segons la qual EPC apunta a la instrucció causant de l'excepció (G.Kane and J.Heinrich, "MIPS Risc Architecture", Editorial Prentice Hall, 1992.)
8. En sistemes reals, no convé mantenir les interrupcions inhibides tot el temps. MIPS té dos bits addicionals en el Status Register que codifiquen de forma independent les funcionalitats de permís d'interrupcions (IE=1, permeses) i mode (KSU=0, sistema) quan el bit EXL val zero. D'aquesta manera, la RSE pot mantenir els permisos de sistema sense necessitat de inhibir les interrupcions tot el temps. La RSE programada d'aquesta manera ha d'escriure's seguint les regles de programació *reentrant*.
9. S'estudiarà amb més detall a les assignatures d'Interfícies de Computadors i de Sistemes Operatius

f) Retornar al programa interromput, executant la instrucció *eret* (exception return). Fa el següent:

- Posa EXL=0. Activa el permís d'interrupcions i desactiva el mode sistema
- Copia EPC al PC, és a dir que salta a l'adreça de retorn guardada en EPC.

4.1 Un exemple senzill de RSE

El següent exemple simplificat reserva espai suficient a la pila[10], hi salva tots els registres (incloent-hi $hi i $lo, però no $k0, $k1 i $sp), i crida la funció *handler_dispatcher* passant-li els registres Cause i EPC com a paràmetres (així com tots els registres del programa, guardats al cim de la pila). Aquesta funció usarà l'ExcCode per seleccionar la rutina *exception_handler* que correspongui, i decidirà també si la RSE ha de retornar al punt del programa interromput (el que guardem en EPC), o a la instruccció següent (EPC+4) o a l'adreça d'algun altre procés (si és el cas que fa un canvi de context). La funció *handler_dispatcher* retornarà aquesta adreça com a resultat en $v0.

```
     .ktext 0x80000180          # secció "kernel text"
RSE:
     # Salvar TOTS els registres incloent $hi, $lo però no $k0, $k1, $sp
     addiu  $sp, $sp, -128
     sw     $1, 0($sp)
     sw     $2, 4($sp)
     sw     $3, 8($sp)
     sw     $4, 12($sp)
     . . .
     sw     $31, 124($sp)

     # Passar paràmetres Cause i EPC a una rutina
     mfc0   $a0, $13            # Passa Cause Register (amb ExcCode)
     mfc0   $a1, $14            # Passa EPC
     jal    handler_dispatcher  # invocarà el handler que correspongui
                                # i decidirà on ha de retornar la RSE
     mtc0   $v0, $14            # copiar resultat a EPC

     # Restaurar registres i pila
     addiu  $sp, $sp, -128
     lw     $1, 0($sp)
     lw     $2, 4($sp)
     lw     $3, 8($sp)
     lw     $4, 12($sp)
     . . .
     lw     $31, 124($sp)
     addiu  $sp, $sp, 128

     # Retornar al programa d'usuari, posant EVL=0 i PC=EPC
     eret
```

10. Un sistema real no faria servir la pila de l'usuari sinó que reservaria espai en una zona de memòria del sistema operatiu, i canviaria de pila per executar la RSE, ja que el registre $sp podria tenir un valor corromput (p.ex. zero), però obviarem aquest detall aquí.

5.4

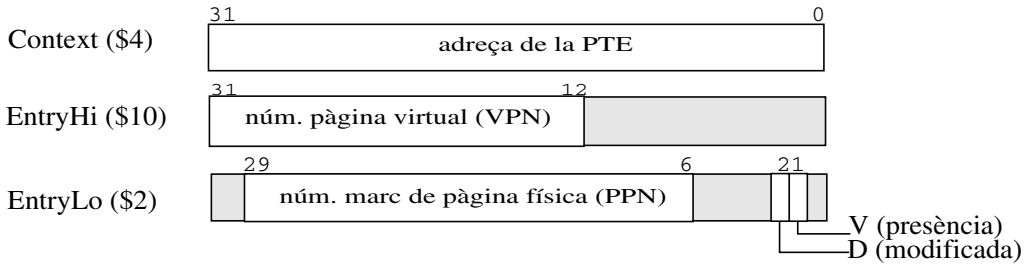
5. Cas especial d'excepció: la fallada de TLB

Una fallada de TLB succeeix quan el processador accedeix a una adreça virtual per a la qual el TLB no conté la corresponent traducció, com ja s'ha vist al tema anterior (recordem que el TLB és una caché de la taula de pàgines, i sols pot emmagatzemar una reduïda porció del seu contingut). Hi ha tres situacions on es pot produir: quan el processador va a buscar una *instrucció*, quan fa un load d'una *dada*, o quan fa un store. En el processador MIPS, en els dos primers casos, el *ExcCode* del registre *Cause* serà TLBL i en el tercer cas serà TLBS[11]. El tractament de la fallada consisteix a copiar la corresponent Entrada de la Taula de Pàgines (PTE) al TLB. Aquesta entrada conté la traducció (número de pàgina física PPN, de 24 bits) i diversos flags: el *bit de presència* (valid bit V) i el *bit de modificada* (dirty bit D). A més, cal escriure també al TLB l'etiqueta (el número de pàgina virtual VPN, de 20 bits).

En alguns processadors el tractament d'una fallada de TLB es fa per hardware, al igual que les fallades de cache, sense necessitat d'interrompre l'execució de la instrucció. En MIPS, a fi d'evitar complicar el hardware, el tractament es fa per software: quan una instruccció causa una fallada de TLB, aquesta s'interromp sense escriure cap resultat ni actualitzar el PC, s'executa una rutina de servei de l'excepció i finalment la instrucció ha de reexecutar-se. Les fallades de TLB succeeixen sovint i convé tractar-les d'una manera que penalitzi poc el rendiment. Com que la rutina d'excepcions genèrica RSE (adreça 0x80000180) resulta massa complexa i llarga, MIPS va decidir que les fallades de TLB invocarien la rutina TLBmiss (adreça 0x80000000). Aquesta rutina és molt ràpida (pot ser menys de 13 cicles) ja que ni tan sols salva registres, sols modifica $k1, que no cal salvar-lo perquè és d'ús exclusiu per al SO.

5.1 Suport hardware

En el processador MIPS, el TLB forma part del coprocessador CP0, i per gestionar-lo s'hi accedeix indirectament per mitjà dels registres *Context*, *EntryHi* i *EntryLo*:



En el moment que es detecta una fallada de TLB, el hardware, a més de les accions descrites a l'apartat 3, efectua les següents accions addicionals:

a) Escriu l'adreça de l'entrada de la taula de pàgines (PTE) al registre *Context*.

b) Escriu els 20 bits del número de pàgina virtual (VPN) al registre *EntryHi*.

11. MIPS identifica les fallades de TLB i les fallades de pàgina amb el mateix valor del ExcCode (TLBL/TLBS). Però no hi ha confusió, ja que MIPS invoca rutines diferents en cada cas.

5.2 Rutina d'excepcions TLBmiss

El següent podria ser un exemple de la rutina TLBmiss:

```
     .ktext 0x80000000
TLBmiss:
     mfc0   $k1, $4       # Copia Context (adreça de la PTE) en $k1
     lw     $k1, 0($k1)   # Llegeix la PTE (traduccció) en $k1
     mtc0   $k1, $2       # Copia la traducció a EntryLo
     tlbwr                # Escriu EntryHi|EntryLo al TLB
     eret                 # Retorna al programa
```

A continuació analitzem pas a pas aquesta rutina TLBmiss:

`mfc0 $k1, $4`

Copia el registre Context al registre $k1. Aquest registre conté l'adreça exacta de memòria de l'entrada de la taula de pàgines que guarda la traducció de l'adreça virtual que ha causat la fallada de TLB.

`lw $k1, 0($k1)`

Usant el registre $k1 com a adreça base, accedeix a memòria i carrega el contingut sencer de l'entrada de la taula de pàgines sobre el propi registre $k1.

`mtc0 $k1, $2`

Copia $k1 al registre EntryLo. Podem observar al diagrama com queda el format del registre EntryLo: el número de pàgina física (PPN) de 24 bits, el bit de presència V i el bit de modificada D. Fa la còpia sense comprovar si V val 0 o 1.

`tlbwr`

Tots els operands d'aquesta instrucció són implícits. Copia el valor dels registres EntryHi (etiqueta) i EntryLo (traducció i flags) en una de les entrades del TLB, seleccionada de forma pseudoaleatòria. El TLB és completament associatiu, i l'algorisme de reemplaçament aleatori és molt més simple d'implementar que l'LRU i quasi igual d'eficient.

`eret`

Restaura el bit EXL=0 i retorna a l'adreça guardada en EPC. Per tant, la instrucció que ha causat el fallo de TLB es reexecutarà.

La rutina TLBmiss copia l'entrada de la taula de pàgines al TLB però sense comprovar si el bit de presència V val 1 o 0. Un cop resolta la fallada de TLB, com que no s'incrementa l'EPC, es reexecutará la mateixa instrucció per segon cop. En aquesta ocasió, torna a consultar el TLB i aquest cop es produirà un encert, i és en aquest moment quan es comprova el bit de presència V del TLB:

- Si el bit de *presència* val V=1 (la pàgina està present en memòria física). La instrucció s'executa amb normalitat.
- Si el bit de *presència* val V=0 (la pàgina no està present en memòria física). En aquest cas es produirà una nova excepció, aquesta vegada per "fallada de pàgina", la qual serà servida per la rutina genèrica RSE, que haurà d'iniciar una operació de lectura al disc.

5.3 Gestió i significat del bit V

Quan s'inicialitza un procés i cap pàgina està present en memòria física, totes les entrades del TLB tenen V=0, i es van posant a 1 a mesura que fan fallades de pàgina[12]. També es posa V=0 a l'entrada d'una pàgina, quan és reemplaçada en memòria física.

MIPS no especifica quin algorisme de reemplaçament usar per al TLB, però disposa d'una instrucció (*tlbwr*) que permet implementar fàcilment l'algorisme aleatori, com ja hem vist[13], el qual ignora el bit V. En canvi, en processadors que implementen LRU o altres algorismes, pot resultar útil per al rendiment ocupar una entrada buida abans que reemplaçar-ne una de plena, per a la nova traducció. En aquests casos no és necessari implementar un bit de control addicional que indiqui plena o buida, podem simplement considerar com a buida una entrada amb el bit de presència V=0, ja que reemplaçar-la no causa cap perjudici al rendiment, doncs el seu PPN no té cap significat útil.

5.4 Gestió i significat del bit D

A la taula de pàgines, el bit D forma part de la informació de *control* de la memòria virtual indicant que una pàgina en memòria física ha estat modificada respecte de la còpia en disc. Compleix la mateixa funció que en les caches d'escriptura retardada. En canvi, en el TLB el bit D forma part de les *dades* emmagatzemades (V, D, PPN), provinents de la taula de pàgines (la informació de *control* està formada per les etiquetes VPN). De les tres *dades*, l'única modificada pel programa és el bit D, que es posa a 1 quan s'executa un *store* per primer cop en una pàgina. En algun moment, aquesta escriptura del bit D s'haurà d'actualitzar també al següent nivell de la jerarquia (a la taula de pàgines) per mantenir-la coherent amb el TLB. Anàlogament a les caches, l'escriptura del bit D es pot gestionar amb política retardada o immediata. MIPS implementa *escriptura immediata* del bit D del TLB[14], però l'actualització sols és necessària per a la primera escriptura en cada pàgina, quan D passa de 0 a 1. Se segueixen les dues regles següents:

- Quan es produeix una fallada de pàgina, la RSE la copia del disc a memòria física i actualitza les entrades corresponents a la taula de pàgines i al TLB. Això inclou també inicialitzar el bit D als dos llocs: D=0 si la fallada l'ha causat una lectura, o bé D=1 si l'ha causat una escriptura.
- Quan una escriptura té un encert en una entrada vàlida del TLB, el hardware també examina el bit D d'aquesta entrada: si D=0 (és la primera escriptura en aquesta pàgina), es produeix una excepció (de "pàgina modificada"[15]), i el tractament per part de la RSE consistirà a posar el bit D=1 tant al TLB com a la taula de pàgines. En canvi, si el bit D ja val 1, l'accés prossegueix normalment.

---

12. L'arquitectura no defineix com s'ha d'inicialitzar el camp d'etiqueta VPN, però la traducció d'una adreça és indeterminada si coincideix amb múltiples entrades, i dependrà de la implementació. El més simple per resoldre la indeterminació és que el sistema operatiu garanteixi que cap etiqueta està repetida, inicialitzant les entrades amb VPNs diferents, ja siguin pàgines presents o no en memòria física.
13. També disposa d'instruccions (*tlbwi* i *tlbr*) que permetrien implementar fàcilment altres algorismes (e.g. FIFO), però no té suport per implementar LRU.
14. Això simplifica bastant la rutina TLBmiss, ja que aquesta pot reemplaçar una entrada del TLB sense necessitat d'actualitzar el bit D a la taula de pàgines, simplement reescrivint-la amb el nou contingut
15. El tractament d'aquesta excepció per part de la RSE no sols actualitza D=1 sinó que també comprova a la taula de pàgines el bit de permís d'escriptura. En cas de no tenir permís el programa s'avorta.

B-9

6. Cas especial d'excepció: les crides al sistema

El sistema operatiu és un programa resident al computador, el qual proporciona rutines útils per als programes, de forma que aquests puguin accedir als recursos compartits d'una forma simple, segura i eficient. Els recursos compartits són, entre altres, la memòria física, els dispositius d'E/S, i el propi ús del processador (execució concurrent).

6.1 Protecció: mode usuari/sistema

La gestió d'aquests recursos queda restringida al SO per raons d'eficiència i de seguretat, a fi i efecte que se segueixin els criteris establerts per l'administrador del computador. Per assegurar aquesta restricció, el codi del SO sols el pot executar el processador quan funciona en "mode sistema", un mode que li atorga una sèrie de privilegis. El mode sistema s'activa posant el bit EXL=1 en el registre *Status*. Els privilegis del mode sistema es necessiten bàsicament per a 2 coses:

- Per accedir a codi/dades del SO, els quals resideixen en una regió de "memòria protegida" ben definida: adreces virtuals amb el bit 31=1. En cas d'accés sense privilegis es genera una excepció amb ExcCode = AdEL (lectura) o AdES (escriptura).
- Per executar instruccions que operin sobre el CP0 (p.ex, per activar/desactivar el bit EXL que atorga els privilegis de sistema!)[16]. En cas d'accés al CP0 sense privilegis es genera una excepció amb ExcCode = CpU.

6.2 Entrada al SO canviant el mode

Per tal que els programes, que s'executen en mode usuari, puguin invocar les subrutines del SO cal que activin el "mode sistema", però concedir-los aquesta potestat sense limitacions anul·laria tota la protecció. L'única solució és que el processador passi a mode sistema al mateix temps que salta al codi del SO, i que aquest tingui un únic punt d'entrada. La solució és doncs causar una excepció, ja que aquesta activa EXL=1 automàticament, i s'executa la RSE, que pertany al SO. Així doncs, per invocar subrutines del sistema, els processadors implementen una instrucció especial de "crida al sistema" (*syscall*, en MIPS) que serveix precisament per causar aquesta excepció (ExcCode = Sys).

6.3 Les crides al sistema com a subrutines

El sistema disposa d'una subrutina per a cada servei, identificada per un *Codi*. Per indicar quina rutina es vol executar, el programa posa el seu codi en $v0 abans d'executar la instrucció *syscall*. Si la subrutina admet paràmetres, es passen en $a0-$a3 ($f12 si és de coma flotant). Si la subrutina retorna algun resultat, es passa en el registre $v0 ($f0 si és de coma flotant).

La següent taula mostra algunes de les crides emulades internament pel simulador MARS (vegeu l'opció de menú "Help" per a més informació). Cal tenir en compte que l'emulació d'aquestes crides en MARS es fa de forma atòmica (en 1 sol pas de simulació)

16. Nota: el simulador MARS ignora els privilegis de sistema, tant per a l'execució d'instruccions privilegiades com per a l'accés a memòria protegida.

i no segueix el mecanisme d'excepcions explicat aquí: ni es crida la RSE ni es modifiquen els registres del CP0. En canvi, les crides no implementades internament per MARS sí que se simulen com a excepcions, tal com hem descrit.

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

6.4 Exemples de crida al sistema

```
     .data
cadena:     .asciiz "Introdueix una frase\n"
     .text
     ...
     li     $v0, 4       # crida num 4: print_string(char *p)
     la     $a0, cadena
     syscall

     li     $v0, 10      # crida num 10: exit()
     syscall
```

6.6, B-7

7. Cas especial d'excepció: les interrupcions

Les interrupcions són un tipus especial d'events causats per l'activació de certs senyals de "petició d'interrupció" procedents dels dispositius d'entrada/sortida. Les interrupcions es gestionen amb el mateix mecanisme que les excepcions, tot i que tenen un origen molt diferent, ja que no estan causades per cap instrucció sinó per un event extern al processador.

7.1 Rol del Sistema Operatiu en l'entrada/sortida

Els dispositius d'entrada/sortida, al igual que altres recursos del sistema, són compartits entre múltiples programes i, com ja s'ha comentat a l'apartat anterior, la seva gestió està restringida al Sistema Operatiu. El SO proporciona als programes una interfície abstracta d'accés als dispositius, per mitjà de subrutines que gestionen les minucioses operacions de baix nivell, asegurant així el seu correcte control.

Qualsevol accés d'un programa a un dispositiu (lectura, escriptura, etc.) s'ha de fer de manera indirecta, sol·licitant-ho al SO per mitjà d'una crida al sistema. Però la comunicació també pot anar en direcció contrària: quan un dispositiu necessita comunicar informació al processador, activa un senyal de petició d'interrupció, i serà la rutina d'interrupcions, pertanyent també al SO, l'encarregada d'atendre aquesta comunicació. En els dos casos és el codi del SO qui ho gestiona.

Per assegurar que cap programa accedeixi directament als dispositius, l'accés es dissenya de manera que requereixi privilegis de sistema: ja sigui amb instruccions especials (p.ex. les instruccions *in* i *out* en els processadors x86), o bé amb instruccions normals de load/store però accedint a adreces físiques reservades al sistema (com en el MIPS).

7.2 Utilitat de les interrupcions per a la sincronització

La comunicació amb els dispositius d'entrada/sortida està dominada per la necessitat de tolerar les llarguísimes latències de les operacions. Típicament, el processador inicia una operació en un dispositiu (p.ex. lectura de disc) i l'operació tarda milions de cicles en acabar. Si el programa espera el resultat sense fer res útil, simplement consultant repetidament l'estat de l'operació, diem que se *sincronitza per enquesta*[17]:

```
do {
       estat = consultar_estat_dispositiu();
while (estat != READY);
```

No obstant, pot resultar molt més productiu que el processador segueixi executant altres tasques útils mentre el dispositiu realitza l'operació: ja sigui parts del mateix programa o parts d'altres programes, el SO pot decidir quines. Quan l'operació acaba, el dispositiu avisa el processador per mitjà d'un senyal que anomenem petició d'interrupció. Quan el processador rep aquest senyal es desencadena una excepció que anomenem interrupció. Aquest procediment rep el nom de *sincronització per interrupcions*[18].

17. A l'assignatura de Introducció als Computadors s'estudien exemples de sincronització per enquesta

18. Més detalls sobre l'entrada/sortida i la sincronització s'estudiaran a l'assignatura Interfícies de Computació. En EC estudiarem solament el suport hardware per tal de gestionar les interrupcions.

7.3 El camp *Interrupt Pending* del registre *Cause*

Quan un dispositiu vol notificar que ha finalitzat una operació o que necessita atenció, envia al processador un senyal de petició d'interrupció. En el MIPS, cada dispositiu té assignat un bit en el camp IP *(Interrupt Pending*) del registre *Cause* del CP0, el qual s'activa en el moment de rebre la petició, i es manté actiu mentre ho estigui el senyal de petició. Les condicions perquè un dispositiu desactivi el senyal de petició depenen de la seva implementació, però normalment succeeix quan la petició ha estat atesa pel processador (p.ex. quan s'ha llegit el codi de tecla polsada en un teclat), o bé quan ha expirat un determinat temps d'espera. Per tant, podem dir que aquest camp guarda memòria de quines peticions d'interrupció han arribat i encara no han estat ateses pel procesador (p.ex. perquè el processador ha tingut les interrupcions inhibides durant un temps). Per altra banda, el processador mai no interromp l'execució de la instrucció en curs a causa d'una interrupció. És al final de l'execució de cada instrucció quan la Unitat de Control comprova si hi ha algun bit actiu al camp IP, i si és així es desencadena una excepció amb *ExcCode* = Int al registre *Cause*.

7.4 El bit EXL i el camp *Interrupt Mask* del registre *Status*

L'activació del bit en el camp IP no significa que automàticament es generi una interrupció. El programa en curs s'interrompirà solament si les interrupcions estan permeses, és a dir si el bit EXL=0. Si aquest bit val 1, totes les interrucpions estan inhibides. Però el SO podria necessitar permetre o inhibir les interrupcions de determinats tipus de dispositius de forma selectiva. Amb aquesta finalitat, MIPS assigna a cada tipus de dispositiu un bit en el camp IM (*Interrupt Mask*) del registre *Status*. Cada un d'aquests bits actua com un permís d'interrupcions específic per a un sol tipus de dispositiu. Així, si un determinat dispositiu del tipus *i* ha fet una petició activant el bit IP_i, solament es produirà una interrupció si EXL=0 i a més a més $IM_i = 1$.

7.5 Identificació del dispositiu per software

Quan el processador detecta una petició d'interrupció i les interrupcions estan habilitades (EXL=0), el hardware executa totes les accions explicades a l'apartat 3 i a continuació salta a la rutina d'excepcions. Un cop que la RSE ha comprovat que ExcCode=Int (la causa és una interrupció), haurà d'examinar el camp *Interrupt Pending* per determinar quin és el dispositiu que ha fet la petició i el camp *Interrupt Mask* per decidir si té permís per interrompre, i si és així cridar a l'*exception_handler* corresponent. De fet, pot resultar que hi hagi més d'un tipus de dispositiu amb els bits IM i IP a 1 (p.ex. si múltiples peticions han arribat mentre les interrupcions estaven inhibides). En aquest cas la rutina d'excepcions haurà d'establir un sistema de prioritats entre ells. Però també pot resultar que no hi hagi cap dels bits IP activat, si el dispositiu no ha mantingut el senyal de petició actiu el temps suficient[19]. Això pot succeir si passen molts cicles de rellotge des de l'instant que es produeix la petició fins al moment que és acceptada i servida, ja que el processador podria haver passat molt de temps amb les interrupcions inhibides.

19. Això és el que passa en el simulador MARS, que cap dels dispositius d'E/S manté la petició activada si en el moment d'activar-la les interrupcions no estan permeses, de forma que aquestes peticions es perden irremisiblement. No és un comportament realista.