Eccezioni

Che cos'è un eccezione? Un "salto" o una variazione non prevista. Va fatta un'immedaita distinzione tra eccezioni e interrupt.

- Le eccezioni sono eventi **sincroni**, possono essere volontari (break, syscall) o involontari (overflow).
- Gli interrupt sono considerati come un tipo di eccezioni, ma **asincroni.** Essi sono causati dalle periferiche o comunque provengono dall’esterno del processore.

Quando si verifica un'eccezioni è l'hardware che se ne accorge, ovvero il datapath. A quel punto viene cambiato il datapath con il coprocessore-0, ovvero il processore salta all’indirizzo del gestore delle eccezioni. In questo esatto momento la gestione dell’eccezione passa al software che si occupa del resto.

Ricorda solamente che saltare alla routine dell’OS per gestire l’eccezione è uguale a dire quello che è scritto qua sopra.

Ora vanno elencati i 2 metodi principali con cui si eseguono le routine dell’OS:

1. Indirizzo Fisso: quello solito che abbiamo visto a lezione, che utilizza 1 handler. Quello che utilizza MIPS.
2. Interruzioni Vettorizzate: ci sono più software/handler che gestiscono cose. Sarà compito dell’hardware scegliere a quale handler far saltare il PC. Esiste un vettore di indirizzi che l’hardware può consultare, questi indirizzi portano ognuno ad un handler diverso. Ogni eccezione normalmente al suo handler specifico.

Facciamo un esempio un overflow aritmetico. A quel punto cosa succede:

- L’Unità di Controllo di MIPS ha rilevato un’eccezione. In questo caso l’eccezione arriva dall’ALU in quanto è un overflow. Potrebbe però arrivare durante la decode dall’opcode dell’istruzione se magari stiamo cercando di eseguire un istruzione non valida.

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

- MIPS (hardware) si salva EPC, l'istruzione che causa l'eccezione. che sarebbe PC - 4
- Vengono salvati altri registri utili che potrebbero servire al coprocessore-0, ma questo dipende dal tipo di eccezione:

Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

- Registro Cause ($13 nel co-processore) = Registro da 32 bit che risponde alla domanda “Cosa è successo?”. Contiene:
  - Exception Code [2-6] = Un numero rappresenta il tipo di eccezione (0 per gli interrupt oppure 12 per l’overflow)

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

    - Pending Interrupts [8-15] = Tiene traccia degli interrupt non ancora gesitti dal gestore delle eccezioni.
    - Branch Delay [31] = Controlla se è avvenuta una branch non richiesta.
- Registro di Stato (Status Register → $12) = registro di controllo che salva lo stato della macchina (non è fondamentale negli esercizi).
- Registro BadVAddr = un registro che viene usato quando vogliamo accedere ad un dato non valido. Indica la posizione del dato a cui volevamo erroneamente accedere.
- $k0 e $k1 vengono riservati al co-processore per farci i calcoli quando si gestirà l'eccezione.
- Poi cambia PC a 0x80000180 che è nel kernel
- Ora è compito del coprocessore che deve salvare tutti i restanti registri macchina UTILI.
- Gestione dell'eccezione
- Attraverso "eret" (exception return) si torna al punto del programma dove si era interrotto. Ripristino del PC e dello stato precedente in 2 possibili modi:

- EPC → se è "interrupt". Così il programma può essere eseguito tranquillamente.
- EPC + 4 → se è exception (ovviamente non deve avvenire un errore irreparabile come una break).

Attenzione!

> 💡 All other exceptions are caused by the execution of the instruction at EPC, except when the istruzione incriminata is in the delay slot of a branch or jump. In that case, EPC points to the branch or jump instruction and the BD bit is set in the Cause register. When that bit is set, the exception handler must look at EPC + 4 for the istruzione incriminata. However, in either case, an exception handler properly resumes the program by returning to the instruction at EPC.

> 💡 The Count register is a timer that increments at a fixed rate (by default, every 10 milliseconds) while SPIM is running. When the value in the Count register equals the value in the Compare register, a hardware interrupt at priority level 5 occurs.

Cosa avviene però nel gestore delle eccezioni?

- Exception occurred at PC=0x00400030
- Cause = 0x00000030
- Status = 3000ff12

Salvataggio dello stato corrente dei registri.

```
.ktext 0x80000180        #prima di tutto, viene fatta la copia dei registri che (ol
.set noat       #direttiva necessaria per impedire l'uso del registro $at
move  $k1, $at       #copia dell'attuale contenuto del registro $at nel registro lo
.set at        #direttiva necessaria per consentire l'uso del registro $at, ad esen
```

```
sw $v0, save0      #copia dell'attuale contenuto di $v0 in memoria
sw $a0, save1   #copia dell'attuale contenuto di $a0 in memoria
```

Ora bisogna controllare se l’exception code all’interno del registro cause indica che si tratta di un interrupt (0) oppure di un’eccezione (>0).

```
mfc0  $k0, $13              #copia con mfc0 del contenuto del registro Cause de
andi  $a0, $k0, 0x007c      #confronto il contenuto del registro Cause (ora in $
                                    #ovvero salvo solo i bit da 2 a 6, in modo da
bgtz  $a0, Excp_ret     #se non e' un interrupt generato da una periferica (altri
```

Ovviamente l’overflow non è un interrupt e dunque si salta a Excp_ret. In questa porzione di codice non succede nulla di particolare oltre al fatto che vengono stampati i dovuti messaggi di errore.

```
Excp_ret:
  li $v0, 4             #nel caso di eccezione, stampo la stringa di output metten
  la $a0, __m1_           #e mettendo __m1_ in $a0
  syscall

  li $v0, 1           #nel caso di eccezione, stampo anche il codice e nome dell'e
  srl $a0, $k0, 2      #e mettendo in $a0 prima il contenuto di $k0 (copia del co
  andi $a0, $a0, 0x1f    #e poi mettendo in $a0 il risultato dell'AND con la costa
  syscall

  li $v0, 4             #per stampare il nome dell'eccezione corrispondente, mett
  lw $a0, __excp($k0)       #e metto in $a0 la word che si trova in memoria all'in
  syscall
```

Come funziona la cosa ora. Nel gestore ci sono un sacco di procedure una dietro l’altra. Ognuna di queste gestisce un certo tipo di istruzione. Però ognuna di queste procedure inizia con un instruzione particolare: bne $a0, codice dell’istruzione di cui si occupa la procedura, fine della procedura. Avviene un controllo per ogni procedura, quando si trova quella giusta verrà eseguita e così via…

In caso di overflow non si deve fare nulla.

```
Excp_Aritm:
   bne $a0, 0xc, End_Excp_Aritm          #se il codice l'eccezione NON e' 12=0x
   li $v0, 4                        #se invece e' 12, stampo la stringa (print_string)
   la $a0, __e12_                      #che si trova in memoria all'indirizzo __e12__
   syscall
  j End_Excp_Hnd                         #vado alla fine dell'exception handler

End_Excp_Aritm:
   andi $a0, $k0, 0x1f                #metto in $a0 il codice dell'eccezione, così
```

Alla fine vado all'exception handler che ritornerà al PC corretto e reimposterà tutto i registri.

```
End_Excp_Hnd:
  mtc0 $0, $13             #azzero il registro Cause

  lw $v0, save0            #ripristino $v0
  lw $a0, save1            #ripristino $a0
.set noat
  move $at, $k1            #ripristino $at
.set at



  mfc0 $k0, $14            #copio in $k0 il contenuto del registro EPC ($14 del co
  addi $k0, $k0, 4         #incremento il valore di 4
  mtc0 $k0, $14            #copio in EPC il valore incrementato
  eret                  #ritorna alla istruzione specificata in EPC
```

Supponiamo ci fosse un interrupt al posto di un overflow:

```
li $v0, 4           #nel caso di interrupt generato da periferica, stampo la stringa
la $a0, __m3_       #e mettendo __m3_ in $a0
syscall
```

Non vediamo come viene gestitito l’interrupt però vediamo che il gestore delle eccezioni si “chiude” in modo diverso rispetto a quando c’è un’eccezione (non c’è l’incremento di 4).

```
IntXXXX:
   ...
EndIntXXXX:
#parte finale dell'exception handler (nel caso di Interruzione generata da perif
   mtc0 $0, $13        #metto a 0 il contenuto del registro Cause ($13 del copro

   lw $v0, save0      #ripristino $v0 caricandolo da memoria dove l'avevo salva
   lw  $a0, save1    #ripristino $v1 caricandolo da memoria dove l'avevo salvat
   .set noat
   move $at, $k1       #ripristino $at caricandolo dal registro $k1 dove l'avevo s
   .set at
   eret              #ritorna alla istruzione specificata in EPC
```

MODIFICA FSM

In caso si chieda di modificare la FSM per gestire un Overflow bisogna aggiungere un passaggio dopo la parte di execute delle istruzioni R-Type.

- ALUSrcA = 0   prendiamo il valore del PC
- ALUScrB = 01   selezioniamo il valore 4
- ALUOp = 01   sottrazione
- EPC WRITE = 1   salviamo il valore di PC-4 in EPC
- Int Cause = 1 scriviamo la causa dell’eccezione nel registro cause (1 è il valore dell’overflow)
- CAUSE WRITE = 1
- PC Source = 11 pc assume valore 8000180
- PC WRITE = 1   abilita la scrittura del PC

Supponiamo ora che l’istruzione lw $t2, 3($t0) abbia dato un’eccezione. In questo caso l’indirizzo non è valido. Noterai

che non cambia quasi un cazzo dall’overflow.

- ALUSrcA = 0
- ALUScrB = 01
- ALUOp = 01
- EPC WRITE = 1
- Int Cause = 0 (corrisponde all’eccezione istruzione non valida)
- CAUSE WRITE = 1
- PC Source = 11
- PC WRITE = 1
- BadAddrWrite = 1 dobbiamo aggiungere solo questa cosa

Quali registri generali deve preservare un gestore delle eccezioni MIPS?

Scegli un'alternativa:

- tutti i registri generali utilizzati, compreso $at, ma esclusi $k1 e $k0
- solo il registro $at
- solo i registri $at e $v0
- solo i registri $k1 e $k0
- i registri generali utilizzati e tutti i registri del Coprocessore 0
- solo i registri del Coprocessore 0
- non rispondo
- nessuna delle altre risposte e' corretta

14) Se il registro “cause” assume 0x0000 0014 quale eccezione si verifica?

Opzioni risposta multipla: 1) ades 2) interrupt exception 3) break exception
4) overflow 5) adel 6) IBE 7) DBE 8) syscall
9) nessuna eccezione 10) nessuna di quelle elencante

32) *Se il registro Cause assume il valore 0x00000100/0x00000200, quale eccezione si è verificata?*
Interrupt Exception.

33) *Se il registro Cause assume il valore 0x00000018, quale eccezione si è verificata?*
IBE (fetch illegale dalla memoria).

34) *Se il registro Cause assume il valore 0x00000000, quale eccezione si è verificata?*
Nessuna eccezione.

35) *Se il registro Cause assume il valore 0x00000020, quale eccezione si è verificata?*
Syscall Exception.

36) *Se il registro Cause assume il valore 0x00000014, quale eccezione si è verificata?*
AdES (scrittura non allineata in memoria).

37) *Se il registro Cause assume il valore 0x00000030, quale eccezione si è verificata?*
Overflow Exception.

Dato il seguente frammento di codice di un ExceptionHandler.s:

```
...
mfc0 $k0, $13
srl $a0, $k0, 2
andi $a0, $a0, 0x1f
riga mancante
...
is_irq:  #gestione  interrupt
```

Specificare quale delle seguenti istruzioni riconosce correttamente un'eccezione di tipo Interrupt:

riga1: bnez $a0, is_irq

riga2: bgtz $a0, is_irq

riga3: bltz $a0, is_irq

riga4: beqz $a0, is_irq ✓

In quale fase del ciclo di esecuzione di un'istruzione MIPS è rilevata una eccezione di tipo "non-istruzione non valida"? Quali azioni a livello hardware sono realizzate da MIPS per la sua gestione?

A livello hardware MIPS ha il compito di salvare il valore del PC-4 in EPC, del Cause Register, di alcuni registri (forse) e infine di saltare alla locazione di memoria dove si trova il gestore delle eccezioni. L’eccezione è rilevata nella decode, in quanto controlla l’opcode dell’istruzione che ha dato errore.