



Interrupt ed Eccezioni

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borgnese@dsi.unimi.it

Università degli Studi di Milano

Riferimento al Patterson: 4.9 e B.7

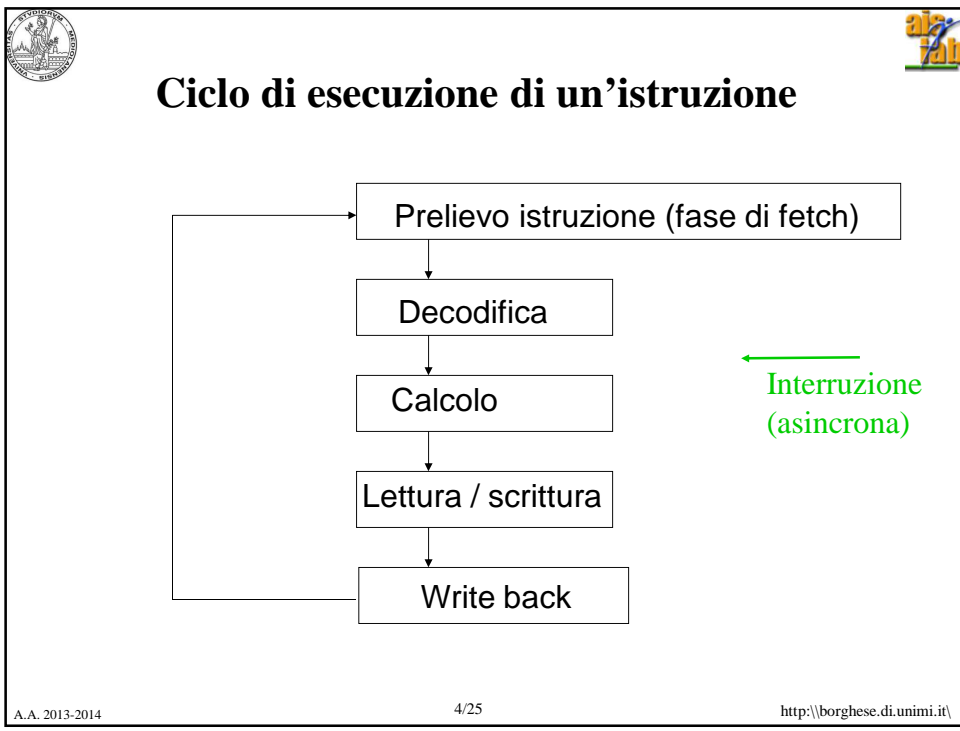
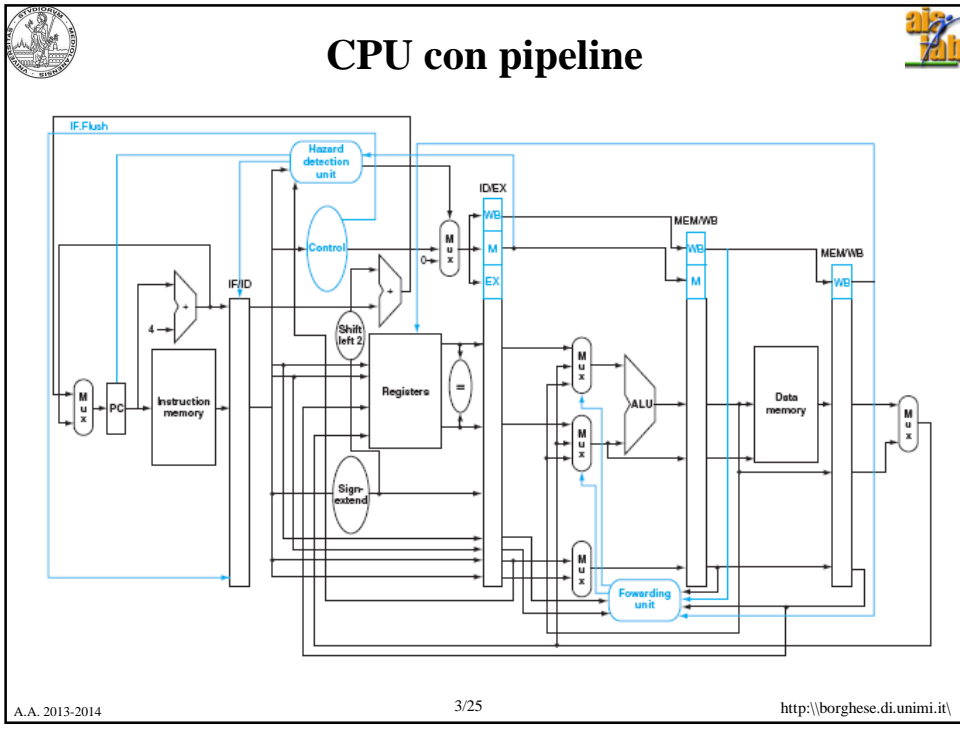


Sommario

Interrupt ed eccezioni

HW per la gestione delle interruzioni: modifica della CPU multi-ciclo

SW per la gestione delle interruzioni: esempio di procedura di risposta





Eccezioni ed Interrupt

Alterano il funzionamento di un programma (funzionalmente equivalenti ad una jump).

Eccezioni. Generamente internamente al processore (e.g. overflow), modificano *immediatamente* il flusso di esecuzione di un'istruzione.

Interrupt. Generate esternamente al processore, asincrono (e.g. richiesta di attenzione da parte di una periferica). Viene *generalmente atteso il termine del ciclo di esecuzione di un'istruzione prima di servirlo*.

Tipo di evento	Provenienza	Terminologia MIPS
Richiesta di un dispositivo di I/O	Esterna	Interrupt
Chiamata al SO da parte di un programma	Interna	Eccezione
Overflow aritmetico	Interna	Eccezione
Uso di un'istruzione non definita	Interna	Eccezione
Malfunzionamento dell'hardware	Entrambe	Eccezione o Interruzione



Tipo di risposta ad un'eccezione

E' software (Sistema Operativo)

Vettorializzata: ciascuna eccezione rimanda ad un indirizzo diverso del SO. Gli indirizzi sono spazati equamente (8 parole). Dall'indirizzo si può ricavare la causa dell'eccezione (cf. Jump Allocation Table).

Tramite registro: detto registro **causa**. Il SO ha un unico entry point per la gestione delle eccezioni (in MIPS $0x80000180 > 2\text{Gbyte}$). La prima istruzione è di decodifica della causa dell'eccezione andando a leggere il registro causa.

Occorre un coordinamento tra
SW (Sistema Operativo) e
HW (struttura della CPU)



Interrupt multipli



Interrupt **accodati** (gestiti come FIFO).

Interrupt **annidati** (gestiti come LIFO).

Cosa suggerite di utilizzare per interrupt esterni?

Cosa suggerite di utilizzare per interrupt interni?

Come gestire le code di interruzioni? Il problema sono gli interrupt annidati.

La soluzione è quella di fermare l'esecuzione di interrupt accodati quando occorre servire interrupt che richiedono annidamento.

Meccanismi di gestione di interrupt annidati:

Maschere di interrupt. La maschera di interrupt è una sequenza di bit in cui ogni bit corrisponde ad un livello di interrupt. Gli interrupt di un certo livello possono essere serviti solo se il corrispondente bit della maschera vale 1. E' legata al **programma**. MIPS.

Piorità di interrupt. Ad ogni tipo di interrupt viene associata una priorità, una priorità è anche associata ai vari **stati del processore**.



Sommario



Interrupt ed eccezioni

HW per la gestione delle interruzioni: modifica della CPU multi-ciclo

SW per la gestione delle interruzioni: esempio di procedura di risposta



I registri del coprocessore 0

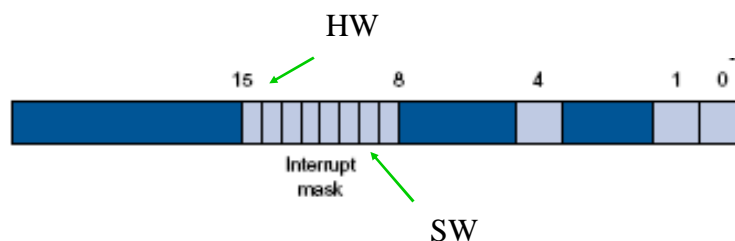
Nome del registro	Numero del registro in coprocessore 0	Utilizzo
Bad/Addr	8	Registro contenente l'indirizzo di memoria a cui si è fatto riferimento (cf. "page fault").
Count	9	Timer (MIPS: 10ms).
Compare	11	Valore da comparare con un timer. Genera un interrupt.
Status	12	Maschera delle interruzioni e bit di abilitazione. Stato dei diversi livelli di priorità (6 HW e 2 SW).
Cause	13	Tipo dell'interruzione e bit delle interruzioni pendenti
EPC	14	Registro contenente l'indirizzo dell'istruzione che ha causato l'interruzione.

Insieme di registri a 32 bit denominato coprocessore 0.
Molti gestiscono la paginazione della memoria.



Status register - I

Interrupt mask, memorizzata nei bit 8-15 dello **status register**. Sono infatti previsti 8 diversi livelli di interrupt (6 interrupt hw e 2 sw).
Il bit 8 della maschera di interrupt è relativo all'interrupt sw di livello 0, il bit 10 a quello hw di livello 2 e così via.
Un bit a 1 nella maschera di interrupt significa che gli interrupt a quel livello sono abilitati.
Vengono disabilitati ad esempio quando bit a priorità più elevata sono già in esecuzione (**mascheramento**).



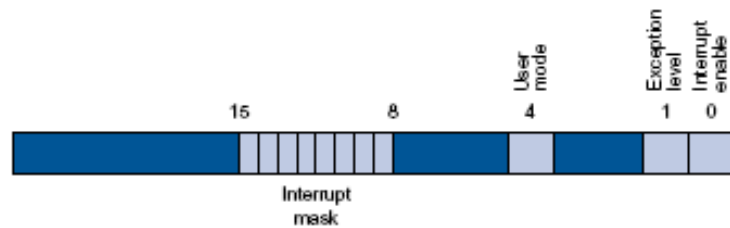


Status register - II

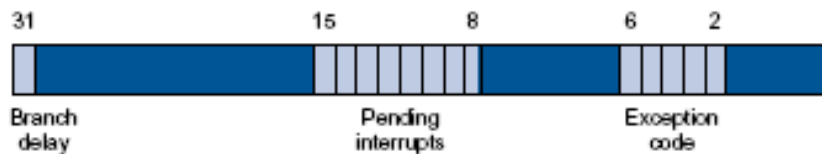
User Mode, abilita il Kernel mode (=0).

Exception level bit. Quando si verifica un'eccezione viene impostato ad uno, disabilitando così gli interrupt veri e propri.

Interrupt enable bit. E' set ad 1 quando le interruzioni sono consentite (la CPU "sente" gli interrupt).



Cause register



Interrupt accodati

Branch delay bit: è 1 se l'ultima eccezione si è verificata un "delay branch slot".

Pending interrupts. Diventano 1 quando un interrupt HW o SW viene richiesto ad un certo livello.

Exception code. Descrive la causa di un'eccezione mediante i seguenti codici (vale 0 nel caso di interrupt esterno, altrimenti codifica l'eccezione).



Codici inseriti nel registro causa



- 0 Interrupt esterno (non si tratta di un'eccezione)
- 4 Indirizzo errato in una load
- 5 Indirizzo errato in una store
- 6 Errore sul bus durante il caricamento di un'istruzione
- 7 Errore sul bus in fase di trasferimento dati
- 8 Eccezione generata da syscall
- 9 Eccezione generata da breakpoint
- 10 Eccezione generata da istruzione riservata
- 12 **Overflow aritmetico**
- 13 **Istruzione non valida.**

I codici inseriti nel registro causa sono relativi ad eccezioni (eventi che si verificano all'interno della CPU).



Strategie di gestione delle eccezioni



Le eccezioni vengono trattate come una forma di hazard sul controllo.

Nel caso si verifichi un'eccezione nella fase di calcolo (overflow ad esempio di add \$t2, \$t3, \$t4) occorre :

- fare il flush delle istruzioni già nella pipeline
- caricare l'indirizzo dell'entry point del programma di gestione delle eccezioni.

Flush delle istruzioni in:

- Fase di fetch
- Fase di decodifica
- Fase di calcolo della add (non deve scrivere il risultato = overflow)



Sommario



Interrupt ed eccezioni

La gestione degli interrupt mediante registro

Modifica della CPU multi-ciclo per la gestione delle eccezioni

Esempio di SW di risposta ad un'eccezione / interrupt.



Hardware addizionale



Registro EPC: è un registro a 32 bit utilizzato per memorizzare l'indirizzo dell'istruzione coinvolta.

Registro causa: è un registro utilizzato per memorizzare la causa dell'eccezione; in MIPS sono 32 bit:

- bit 2 = 0 istruzione indefinita.
- bit 2 = 1 overflow aritmetico.

Segnali di controllo:

CausaWrite – scrittura nel registro Causa.

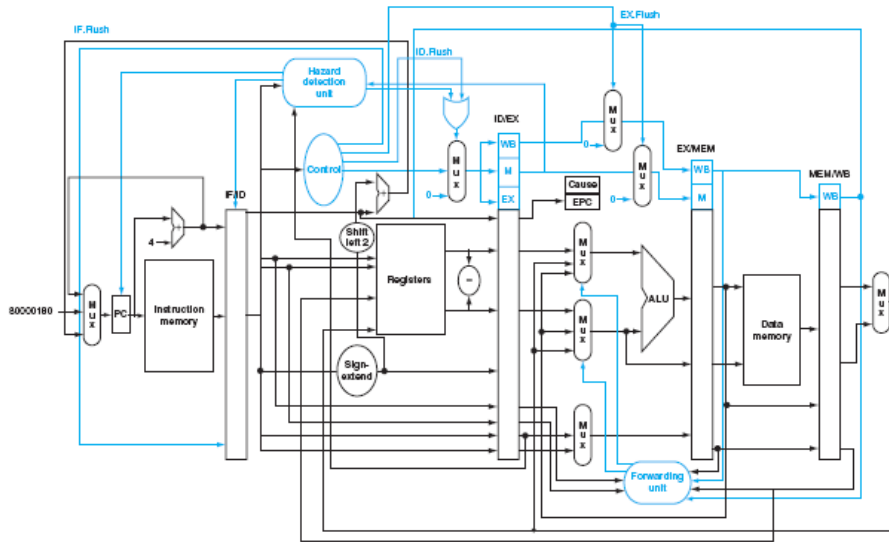
CausaInt – Dato per il registro Causa.

Modifiche ai registri di pipeline

Causa ed EPC appartengono al coprocessore 0



Modifiche alla pipeline



MIPS: Software conventions for Registers



0 zero constant 0	16 s0 callee saves ... (caller can clobber)
1 at reserved for assembler	23 s7
2 v0 expression evaluation &	24 t8 temporary (cont'd)
3 v1 function results	25 t9
4 a0 arguments	26 k0 reserved for OS kernel
5 a1	27 k1
6 a2	28 gp Pointer to global area
7 a3	29 sp Stack pointer
8 t0 temporary: caller	30 fp frame pointer



Prima e dopo l'eccezione

```

0x40  sub $11, $2, $4
0x44  and $12, $2, $5
0x48  or $13, $2, $6
0x4C  add $1, $2, $1
0x50  slt $15, $6, $7
0x54  lw $16, 50($7)
...

0x80000180  mfc0 $k0, $13          # Cause register
0x80000184  andi $a0 $a0, 0x3C    # Extract ExcCode Field
.....

.....          EPC -> PC

```

Cosa succede se si verifica un overflow nell'istruzione di add?



Esempio

Eccezione sulla add con ripetizione della add stessa

```

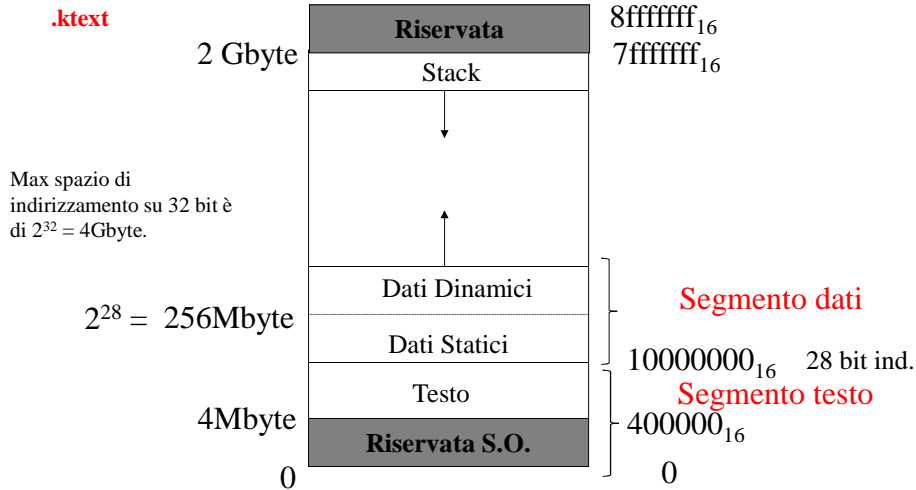
0x40  sub $11, $2, $4
0x44  and $12, $2, $5
0x48  or $13, $2, $6
0x4C  add $1, $2, $1
0x80000180  mfc0 $k0, $13          # Cause register
0x80000184  andi $a0 $a0, 0x3C    # Extract ExcCode Field
.....
Change the value of $2 or $1

.....          EPC = EPC-4
.....          EPC -> PC
0x4C  add $1, $2, $1
0x50  slt $15, $6, $7
0x54  lw $16, 50($7)
...

```



Organizzazione logica della memoria



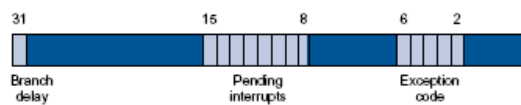
Esempio assembler



Controllo se è un'eccezione. Se è un'eccezione stampo un messaggio.

```
.ktext
0x80000180 la $s0, save0          # Handler is not re-entrant and can't use
sw $a0, 0($s0)      # stack to save $a0, $a1 (data required are
sw $a1, 4($s0)      # stored in kernel data segment)
# Don't need to save $k0/$k1
# First check
mfc0 $k0, $13      # Move Cause into $k0 (=26) in register file
andi $a0, $k0, 0x7C # Extract ExcCode field (5 bits) 0x7C = 124
beq $a0, $zero, done # Branch if ExcCode=HW Interrupt (ExcCode=0)
# Now display
mov $a0, $k0      # Move Cause into $a0
mfc0 $a1, $14     # Move EPC into $a1
jal print_excpc   # Print exception error message. Parameters
# are in $a0, $a1

.kdata
save0: .word 0
save1: .word 0
```



Cause register



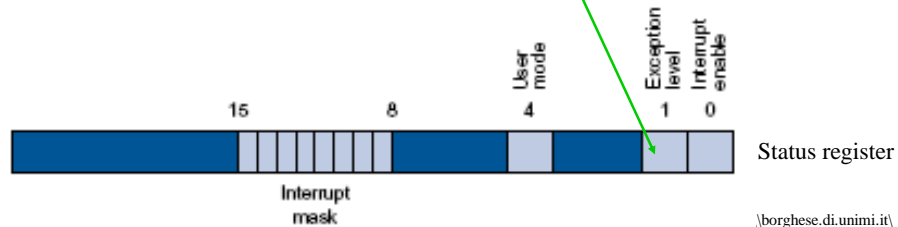
Termine della procedura

You may want to do something better than just displaying the exception

```
done: mtc0 $13, $0           # Clear Cause register
      mfc0 $k0, $12        # To fix Status register (bring status in k0)
      andi $k0, $k0, 0xfffd # Clear EXL bit (Exception level, mask bit)
      ori $k0, $k0, 0x1     # Enable interrupts in status register
      mtc0 $12, $k0        # Restore status register

      lw $a0, 0($s0)       # Restore previously saved registers
      lw $a1, 4($s1)

      # Prepare to return
      mfc0 $k0, $14        # Bump EPC (PC + 4) into $k0
      jr $k0               # Return to EPC address (eret)
```



Eccezioni multiple

Viene servita prima la prima eccezione.

Ruolo dell'HW. Fermare l'esecuzione e salvare le informazioni.

Ruolo del SW. Individuare l'eccezione e prendere i provvedimenti opportuni.

La gestione degli interrupt e delle eccezioni deve essere coordinata tra SW e HW



Sommario



Interrupt ed eccezioni

HW per la gestione delle interruzioni: modifica della CPU multi-ciclo

SW per la gestione delle interruzioni: esempio di procedura di risposta