


Le interruzioni

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

Università degli Studi di Milano

Riferimento al Patterson: 5.6 ed Appendice B.10
Appunti su I/O dei Proff. Bruschi e Rosti, Cap. 7


Sommario

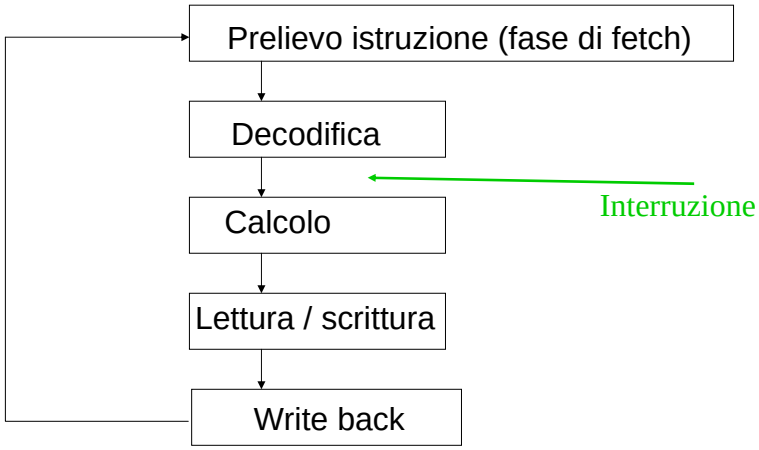
Interrupt ed eccezioni

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.

Ciclo di esecuzione di un'istruzione



Eccezioni ed Interrput

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

Eccezioni. Generamente internamente al processore (e.g. overflow), modificano 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 (SO)

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

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

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***.

Risposta ad interrupt ed eccezioni

`j <address>` incondizionato. Indirizzo della memoria di sistema. 80000180_{hex}

Questo indirizzo corrisponde a:

```
10 - 00  0000 0000 - 0000 0000 00 - 01 1000 0000
2Gbyte + 512 Byte + 256 Byte.
```

Occorre predisporre l'hardware in modo da gestire questo salto. I registri di coprocessore 0 vengono interrogati nel segmento di `.ktext` (kernel text).

Occorre anche predisporre l'hardware in modo da rendere disponibile l'indirizzo di ritorno (indirizzo dell'istruzione che ha generato l'eccezione).

```
.ktext  0x80000180
        ..........
        ..........
        ..........
```


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
Count	9	Timer
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.

Sommario

Interrupt ed eccezioni

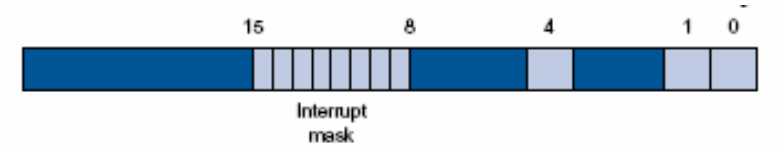
La gestione degli interrupt mediante registro

Modifica della CPU multi-ciclo per la gestione delle eccezioni

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

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).

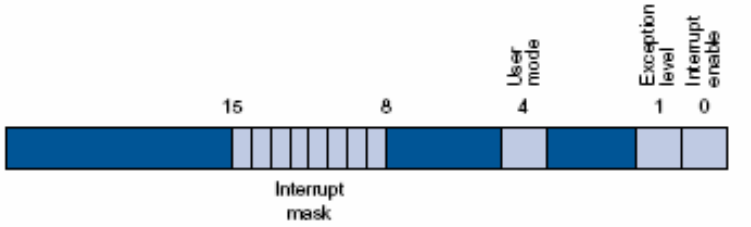


Status register - II

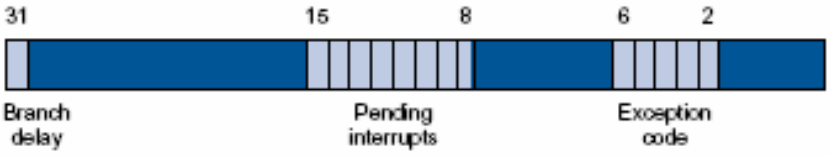
User Mode, abilita il Kernel mode (=0).

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



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

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

Sommario

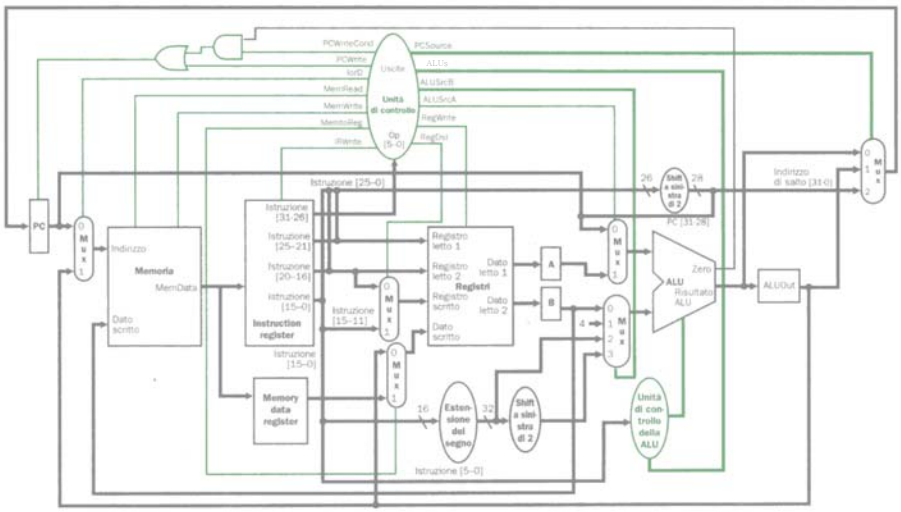
Interrupt ed eccezioni

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.

CPU multi-ciclo



Hardware addizionale

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

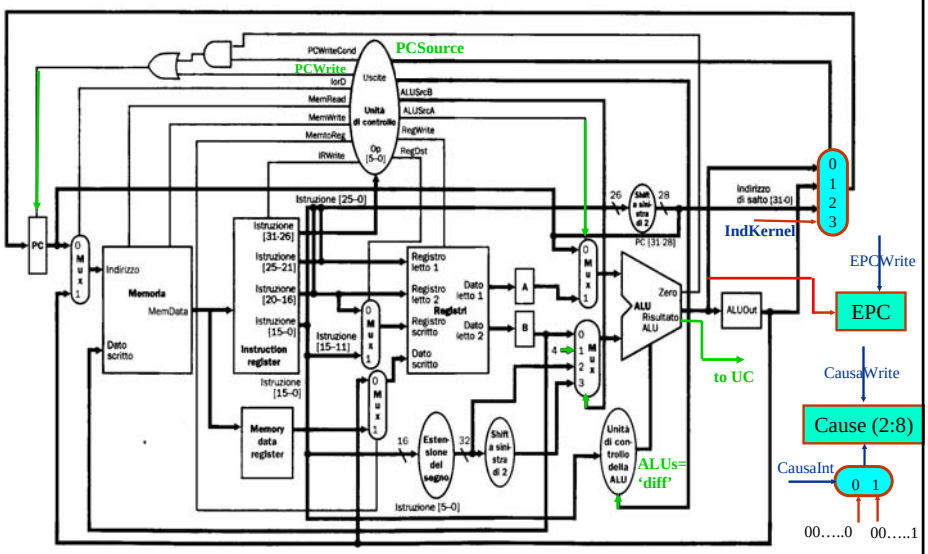
Registro causa: è un registro utilizzato per memorizzare la causa dell'eccezione; in MIPS sono 32 bit:
- bit 2 = 0 istruzione indefinita.
- bit 2 = 1 overflow aritmetico.

Segnali di controllo:
EPCWrite – scrittura nel registro EPC.
CausaWrite – scrittura nel registro Causa.
CausaInt – Dato per il registro Causa.

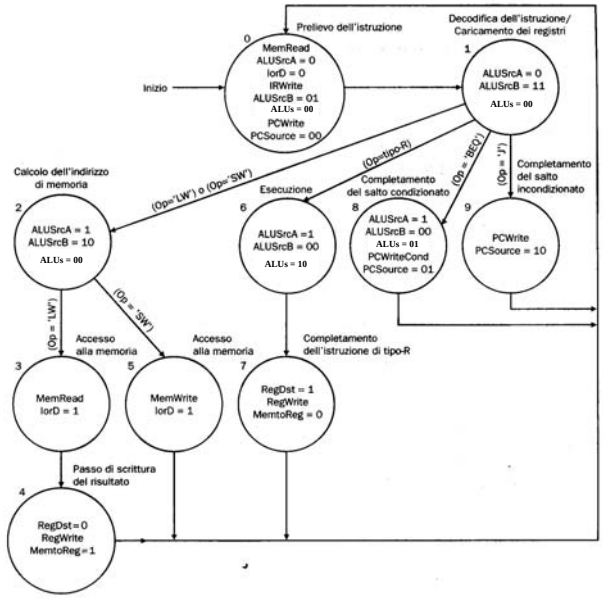
Set degli input della FSM modificato:
Aggiunta di Overflow.

Collegamenti HW per la risposta ad un'eccezione

Salvataggio del PC: PC → EPC; Selezione della causa; Segnali di controllo in verde.



FSM - STG

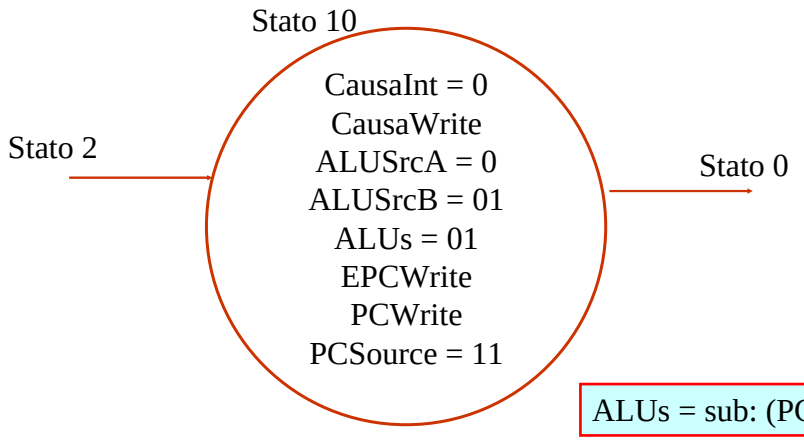


Modifiche alla FSM della CPU

Istruzione indefinita.
Non esiste, al passo 2 (decodifica), uno stato futuro valido.
Si aggiunge un nuovo stato futuro indicato con 'altro', è lo
Stato corrispondente al verificarsi dell'eccezione.

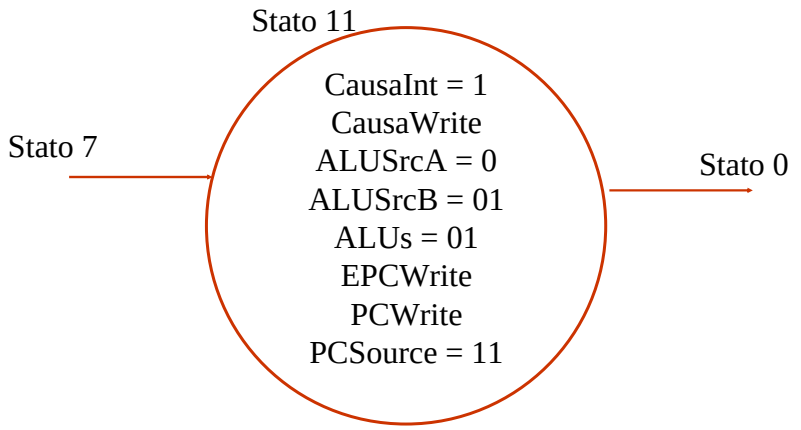
Overflow aritmetico.
Al passo 4 di esecuzione dell'operazione (stato di WriteBack), lo stato futuro è scelto in funzione del segnale di overflow (nuovo input alla FSM). Ho già eseguito la somma e sto scrivendo il risultato nel register file. In particolare, potremmo scegliere di gestire l'eccezione dopo la fase di calcolo; si preferisce completare la fase di WriteBack nell'ipotesi che il risultato dell'operazione di Overflow sia comunque valido.

Stato OpCode not valid

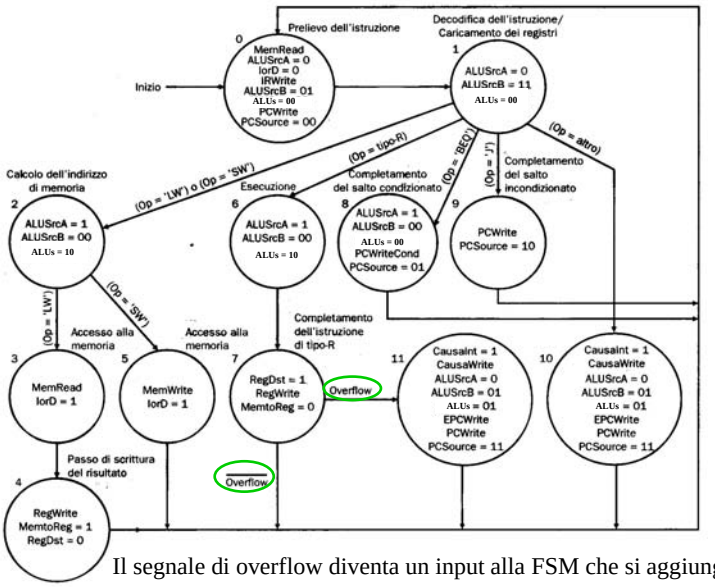


ALUs = sub: (PC + 4) - 4

Stato Overflow



FSM per la CPU multi-ciclo con gestione delle eccezioni



Il segnale di overflow diventa un input alla FSM che si aggiunge ad OpCode.

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.

Il coprocessore 0 - Istruzioni

Set di istruzioni che lavora sul coprocessore 0:
•lw (to coprocessor 0), sw (from coprocessor 0).
•move from coprocessor0, move to processor0:

lwc0 $<reg_c0> <offset>($reg) (e.g. lwc0 $13, 40($t0) # registro_cause <- mem)
swc0 $<reg_c0> <offset>($reg) (e.g. swc0 $13, 40($t0)) # registro_cause -> mem)

mfc0 $<reg>, $<reg_c0> pseudo-istruzione mfc0 $t0, $13
mtc0 $<reg_c0>, $<reg> pseudo-istruzione mtc0 $13, $t0

Il SO quando c'è un interrupt può interrogare l'architettura che gestisce gli interrupt andando ad analizzare i registri di coprocessore 0.

La prima istruzione di risposta ad un interrupt si trova all'indirizzo: 0x80000180:
***exception handler*.**

Risposta ad un interrupt

Le eccezioni vengono gestite dal SO.
1) Salvataggio dell'indirizzo dell'istruzione incriminata (PC-4) nell'EPC.
2) Trasferimento del controllo al SO (questo potrà eventualmente terminare il programma segnalando errore) jump all'indirizzo 0x80000180.

Cosa deve fare il SO?
1) Copia i registri del coprocessore 0: **status e cause**.
2) AND dei campi: **pending interrupts** (cause register) ed **interrupt mask** (status register) per individuare quali interrupt, tra quelli richiesti possono essere serviti.
3) Selezione dell'interrupt a priorità più elevata (per convenzione l'1 più a sx).
4) Modifica della maschera di interrupt per disabilitare gli interrupt più a dx (metto a 0 i bit più a dx di questo campo).
5) Salvare lo stato del processore.
6) Per lasciare interrupt di livello superiore, set **Interrupt enable** a 1 nel cause register.
7) Chiamare la routine appropriata di gestione dell'interrupt.

Al termine:
1) Ripristinare lo stato del processore.
2) Ritornare all'istruzione memorizzata in EPC.

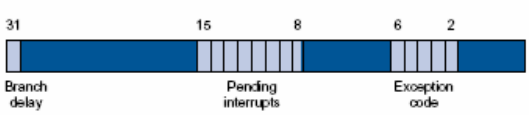
MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Esempio

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

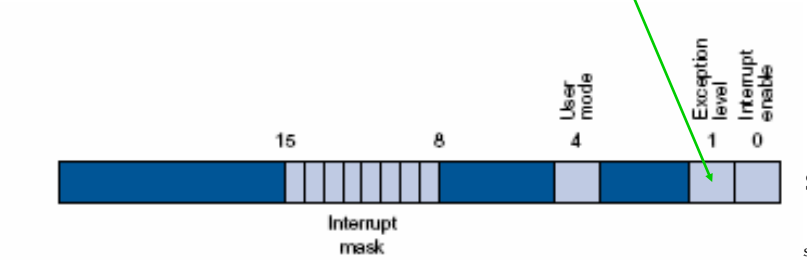
```
.ktext   0x80000180
         mov $k1, $at            # Save $at register in register file
         la $s0, save0
         sw $a0, 0($s0)          # Handler is not re-entrant and can't use
         sw $a1, 4($s0)          # stack to save $a0, $a1 (data required are
                                 # stored in kernel data segment)
                                 # Don't need to save $k0/$k1

         mfc0 $k0, $13           # Move Cause into $k0 (=$26) in register file
         srl $a0, $k0, 2         # Prepare ExcCode field at position 0
         andi $a0, $a0, 0x1f     # Extract ExcCode field (5 bits)
         beq $a0, $zero, done    # Branch if ExcCode=HW Interrupt (ExcCode=0)
         mov $a0, $k0            # Move Cause into $a0
         mfc0 $a1, $14           # Move EPC into $a1
         jal print_excp          # Print exception error message. Parameters
                                 # are in $a0, $a1

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



Cause register

Termine della procedura

```
done:   mtc0 $13, $0            # Clear Cause register
        mfc0 $k0, $12           # To fix Status register (bring status in k0)
        andi $k0, 0xfffd        # Clear EXL bit (Exception level, mask bit)
        ori $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)
        mov $at, $k1
        mfc0 $k0, $14           # Bump EPC into $k0
        addiu $k0, $k0, 4       # Do not reexecute faulting instruction:
                                # EPC = EPC + 4
        mtc0 $14, $k0           # Write EPC + 4 in EPC
        jr $k0                  # Return to EPC address (eret)
```



Status register

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.