


Interrupt ed Eccezioni

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson, versione 5: 4.9 e A.7

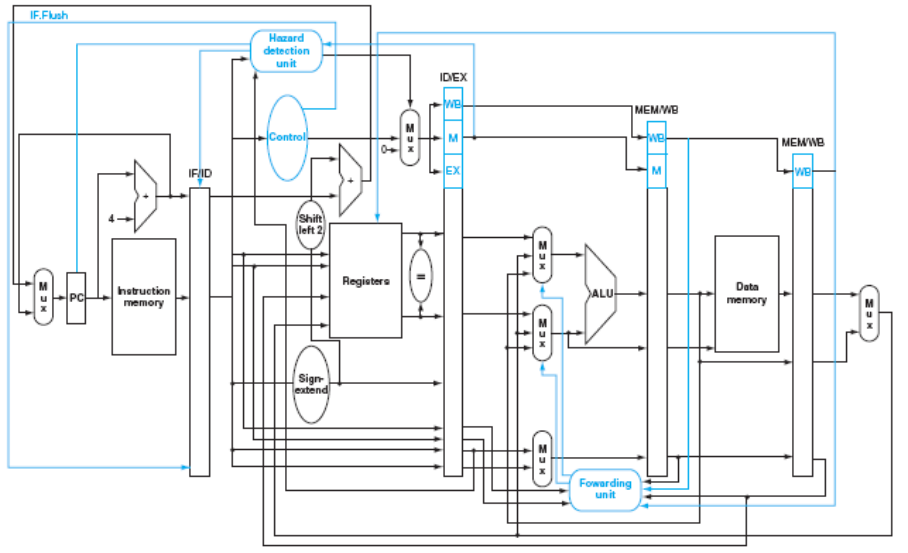

Sommario

Interrupt ed eccezioni

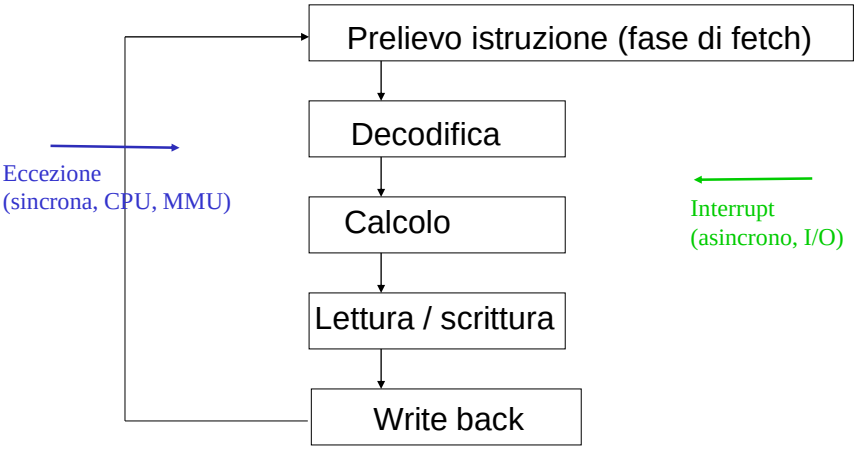
HW per la gestione delle eccezioni MIPS: modifica della CPU

SW per la gestione delle eccezioni MIPS: esempio di procedura di risposta

CPU con pipeline



Ciclo di esecuzione di un'istruzione



Hazard sul controllo: alterano il "normale" flusso delle istruzioni

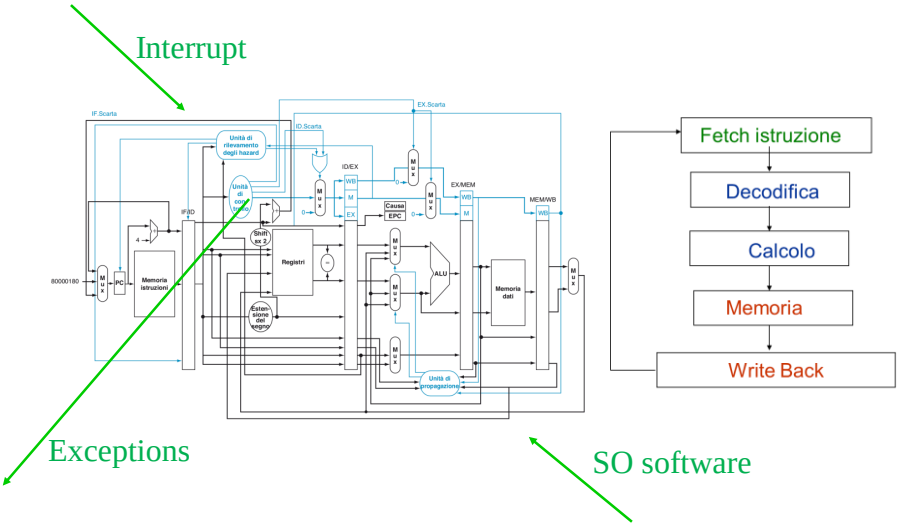
Eccezioni e Interrput

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

La situazione per la CPU



Tipo di risposta ad un'eccezione

E' software (Sistema Operativo)

Occorre il supporto dell'hardware

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

- Riconoscere che si è verificata un'eccezione / interrupt
- Garantire la corretta esecuzione del codice
- Gestire l'eccezione / interrupt
- Riprendere dal punto in cui l'esecuzione era stata interrotta

2 tipi di risposte

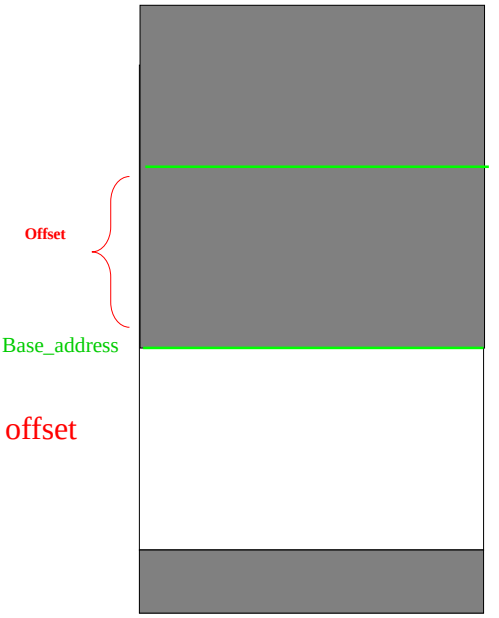
*Risposta vettorizzata: C*iascuna eccezione rimanda ad un indirizzo diverso del SO. Gli indirizzi sono spaziati equamente. Dall'eccezione si ricava l'indirizzo della prima istruzione di risposta e quindi implicitamente la causa (cf. Jump Address Table).
Interrupt # * offset + Base address -> indirizzo nel PC (prima istruzione del SO di gestione di quell'eccezione)

Tramite registro: detto registro **causa**. Il SO ha un unico entry point per la gestione delle eccezioni (in MIPS 0x8000 0180 > 2Gbyte + 384 Byte).
La causa dell'eccezione viene memorizzata in MIPS nel registro **causa**.
Interrupt -> indirizzo nel PC (prima istruzione del SO di gestione di tutte le eccezioni)
Il SO decodifica la causa dell'eccezione analizzando il registro causa (le eccezioni sulla memoria rimandano all'indirizzo 0x800 000 = 2 Gbyte)

Interrupt vettorizzati

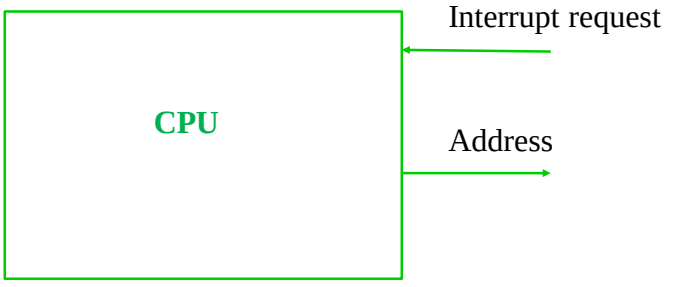
Offset = #_interrupt * space

Jump address table



Address_final = Base_address + offset

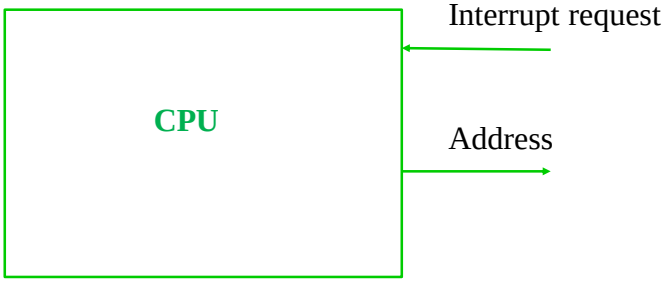
Intel: real mode (8088)



- IVT – Interrupt Vector Table (1KByte – 0x0000 a 0x0400, base address = 0)
- 256 interrupt diversi (primi 32 riservati a eccezioni del processore)
- A ciascuna eccezione viene associate un gruppo di **4 Byte** (CS:IP)

Salto a CS:IP (equivalente al PC nel MIPS). Il salto viene comandato dall'hardware**.**

Intel: protected mode



- IVT – Interrupt Vector Table (2KByte – 0x0000 a 0x800, base address contained in IDTR – Interrupt Descriptor Table Register)
- 256 interrupt diversi.
- A ciascun interrupt viene associate un blocco di **8 Byte** che rappresenta l'indirizzo di un segmento (LDT, GDT – Local, Global Description Table) + un offset di segmento.

Interrupt in Protected Mode

INT_NUM	Short Description
0x00	Division by zero
0x01	Single-step interrupt (see trap flag)
0x02	NMI
0x03	Breakpoint (callable by the special 1-byte instruction 0xCC, used by debuggers)
0x04	Overflow
0x05	Bounds
0x06	Invalid Opcode
0x07	Coprocessor not available
0x08	Double fault
0x09	Coprocessor Segment Overrun *(386 or earlier only)*
0x0A	Invalid Task State Segment
0x0B	Segment not present
0x0C	Stack Fault
0x0D	General protection fault
0x0E	Page fault (Virtual Memory)
0x0F	*reserved*
0x10	Math Fault
0x11	Alignment Check
0x12	Machine Check
0x13	SIMD Floating-Point Exception
0x14	Virtualization Exception
0x15	Control Protection Exception

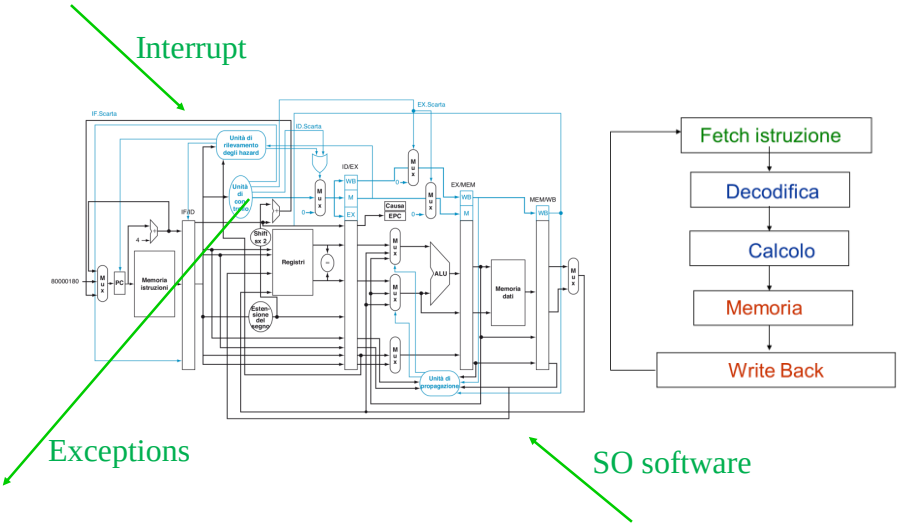
Sommario

Interrupt ed eccezioni

HW per la gestione delle eccezioni MIPS: modifica della CPU

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

La situazione per la CPU



I registri coinvolti

Nel MIPS un register file secondario, il coprocessore 0, salva le informazioni richieste dal SO per rispondere. Interfaccia tra SO e HW è rappresentata da questo register file.

Nome del registro	Numero del registro in coprocessore 0	Utilizzo
BadVAddr	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.
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.
Config	16	Configurazione della macchina

Insieme di registri a 32 bit denominato coprocessore 0.

Alcune eccezioni

Counter. Quando viene raggiunto il valore 0 di conteggio viene lanciato un interrupt HW.

Memory access error (lettura/scrittura memoria, trasferimento dati - Miss). L'indirizzo incriminato viene salvato nel registro BadVAddr e viene lanciata un'eccezione dalla MMU.

Nome del registro	Numero del registro in coprocessore 0	Utilizzo
BadVAddr	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.
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.
Config	16	Configurazione della macchina

Gestione HW di un'eccezione (mediante registro)

1) Identificazione (e.g. overflow, istruzione non valida)

2a) Salvataggio dello **stato** (dell'indirizzo (+4) dell'istruzione incriminate (**registro EPC**))

2b) Scrittura della causa dell'eccezione nel registro **causa**.

2c) Eliminazione delle istruzioni successive a quella che ha causato l'eccezione (**flush**).

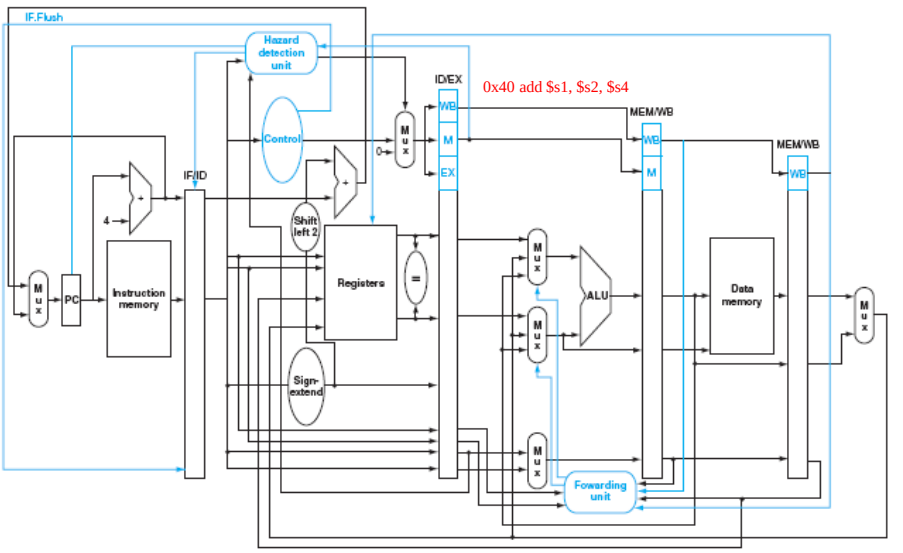
2d) Trasferimento del controllo (valore del PC) alla prima istruzione del programma di risposta alle eccezioni (***exception handler***): offerta di servizi, modifica operandi, terminazione del programma…).

…….

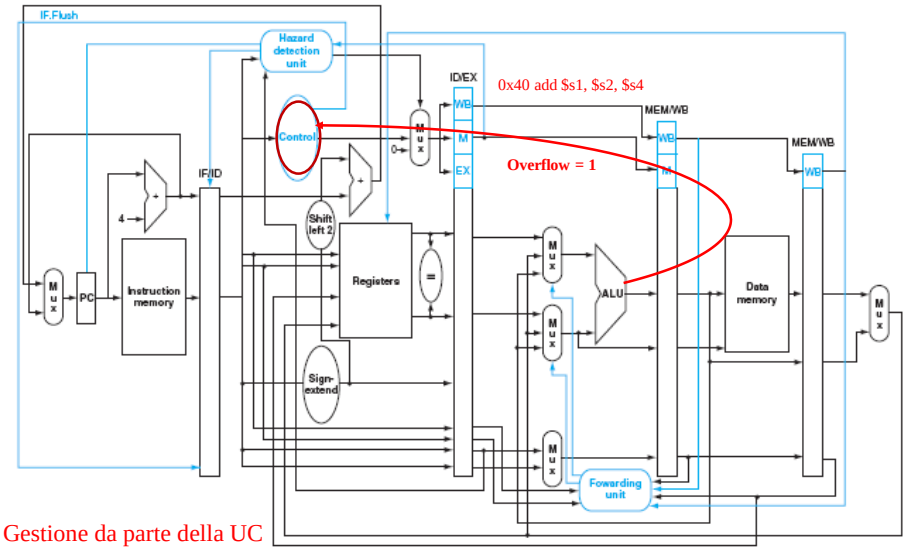
3a) Ripristino dello stato

3b) Ritorno all'istruzione che ha causato l'eccezione o all'istruzione successive.

CPU con pipeline - Overflow

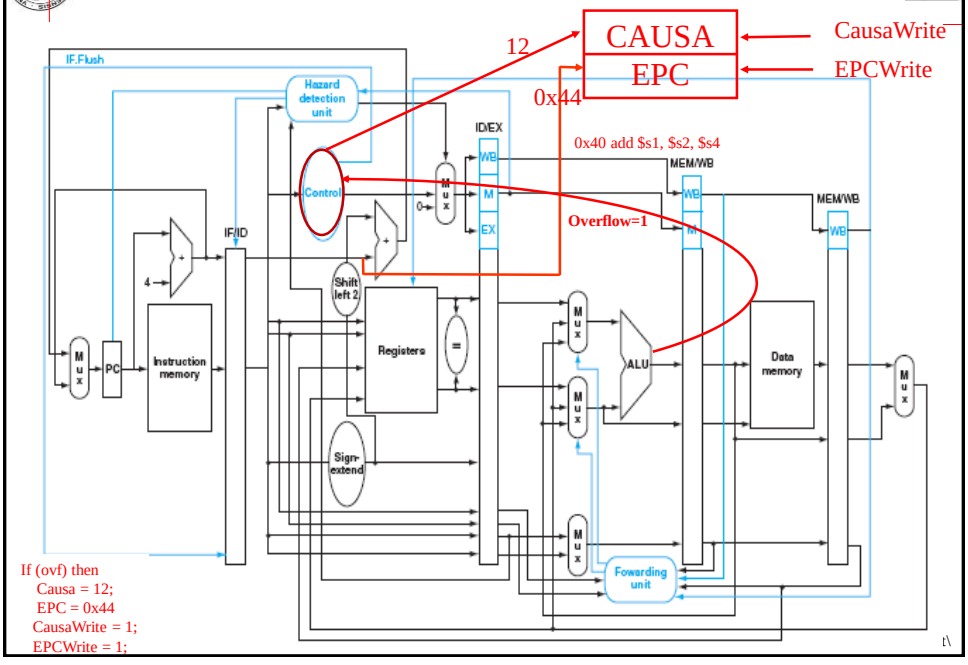


1. Identificazione eccezione Ovf



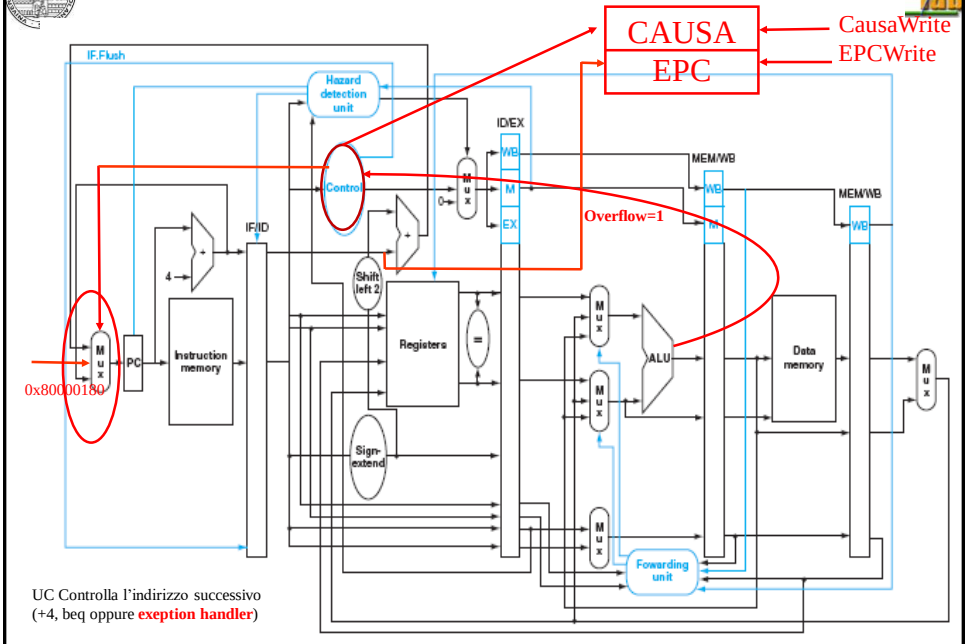
Gestione da parte della UC

2a. Salvataggio del PC e 2b. scrittura Causa

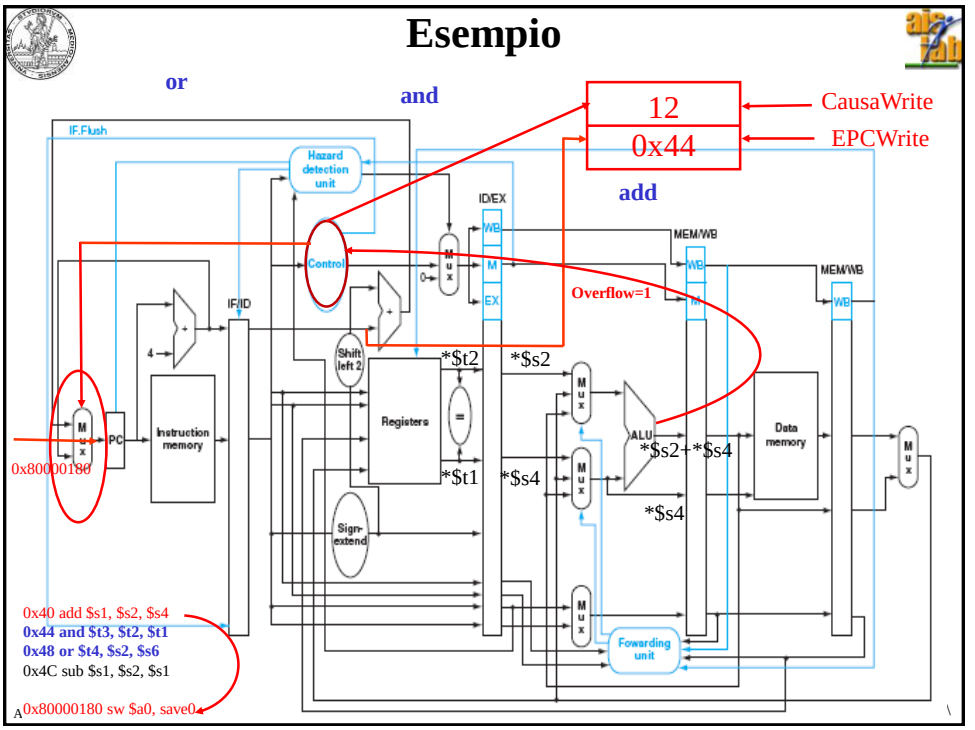


If (ovf) then
Causa = 12;
EPC = 0x44
CausaWrite = 1;
EPCWrite = 1;

2d. Trasferimento del controllo



Esempio



Strategie di gestione dell'eccezione Ovf

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

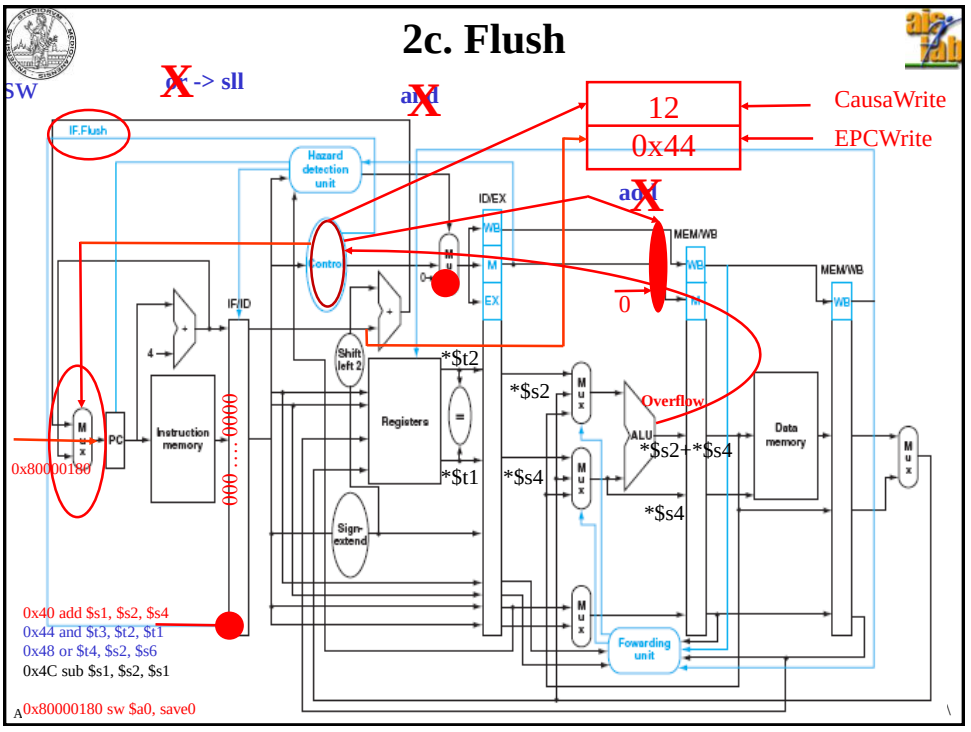
Nel caso si verifichi un'eccezione nella fase di calcolo (overflow ad esempio di add $s1, $s2, $s4) occorre :

a. fare il **flush** delle istruzioni già nella pipeline (compresa la add, occorre che queste istruzioni non modifichino lo stato della CPU).
b. caricare l'indirizzo dell'entry point del programma di gestione delle eccezioni.

Flush delle istruzioni in:

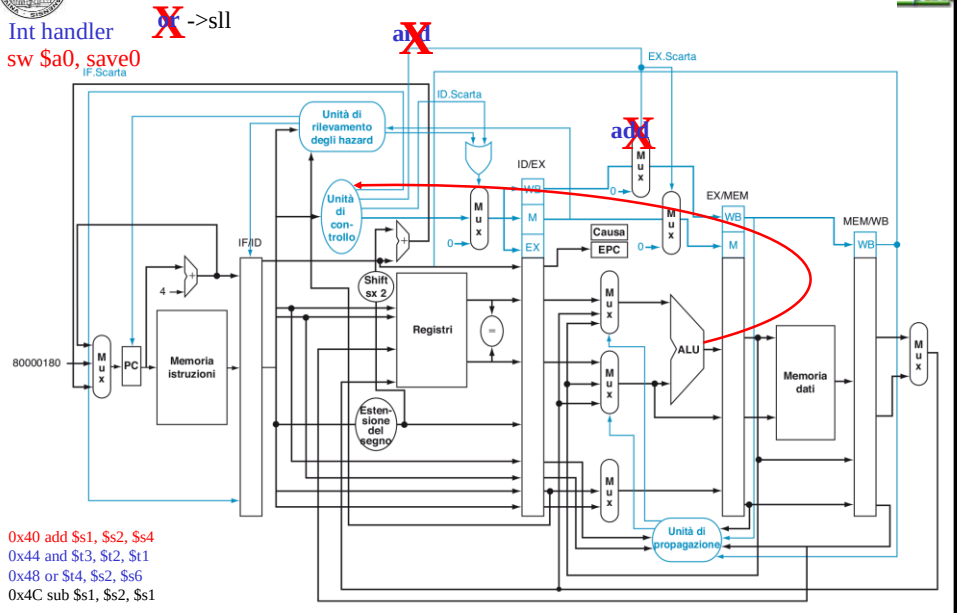
Fase di fetch
Fase di decodifica
Fase di calcolo della add (non deve scrivere il risultato = overflow)

2c. Flush



Schema finale CPU

Int handler
sw $a0, save0



0x40 add $s1, $s2, $s4
0x44 and $t3, $t2, $t1
0x48 or $t4, $s2, $s6
0x4C sub $s1, $s2, $s1

0x80000180 sw $a0, save0

Hardware addizionale

Gestione delle eccezioni di:

- Istruzione non valida (causa = 13; trap)
- Overflow (causa = 12)

Registro EPC ($14): è un registro a 32 bit utilizzato per memorizzare l'indirizzo dell'istruzione coinvolta (in coprocessore 0)

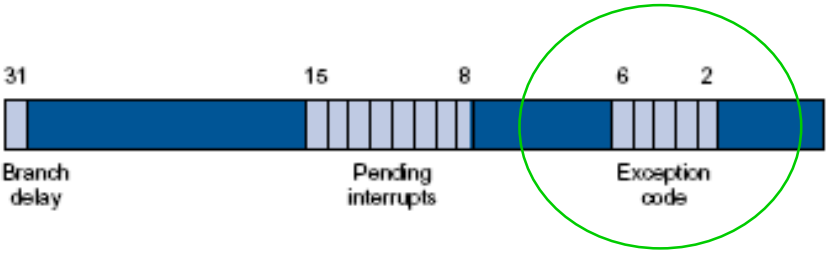
Registro causa ($13): è un registro utilizzato per memorizzare la causa dell'eccezione; in MIPS sono 32 bit. 5 bit servono per definire la causa dell'eccezione (in coprocessore 0):
- Registro causa = 12 -> istruzione indefinita.
- Registro causa = 13-> 1 overflow aritmetico.

Segnali di controllo (dalla UC):

CausaWrite – scrittura nel registro Causa.
CausaInt – Dato per il registro Causa.
EPCWrite – scrittura nel registro EPC.
PCSrc – Aggiunta di un terzo input al PC per la scelta dell'indirizzo istruzione risposta alle eccezioni.

Modifiche ai registri di pipeline e aggiunta di bus interni alla CPU

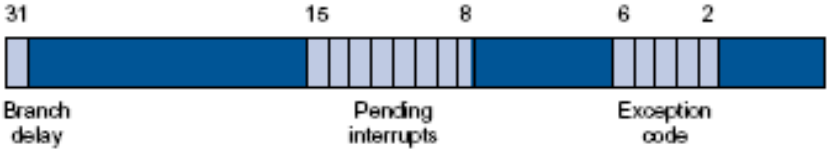
Cause register



interrupt

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

Cause register



Branch delay bit: è 1 se l'ultima eccezione si è verificata, se è legata a una branch, l'istruzione successiva si trova in un "branch delay slot".

L'istruzione successiva potrà essere o un'istruzione precedente alla beq o un' istruzione presente all'indirizzo di destinazione del salto.

Informazioni sugli interrupt

Interrupt multipli

Interrupt accodati (gestiti come FIFO) in una coda SW di interrupt (cf. semaforo)

Interrupt annidati (gestiti come LIFO) in una coda SW di interrupt (cf. traghetto)

Cosa suggerite di utilizzare per interrupt esterni?

Cosa suggerite di utilizzare per interrupt interni?

Come gestire la coda delle interruzioni?

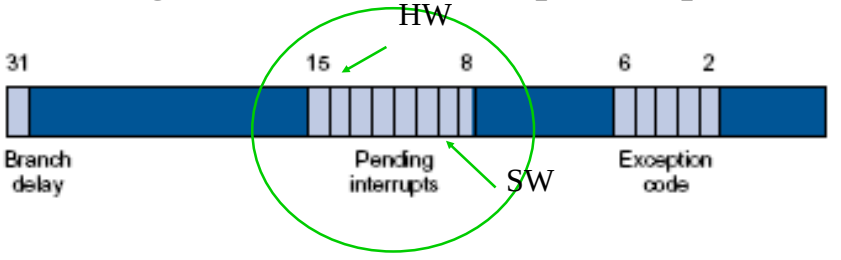
La soluzione è quella di fermare l'esecuzione di interrupt quando occorre servire un interrupt più importante (a priorità più elevata).

Meccanismi di gestione di interrupt multipli:

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.

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

Registro causa e interrupt multipli



Multiple exceptions. Possono esserci più eccezioni nello stesso ciclo di clock. Nel MIPS la **prima istruzione** in ordine temporale viene interrotta. Il codice dell'eccezione riguarda la prima istruzione.

Pending interrupts. Memorizzata nei bit 8-15. Sono 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 l'interrput è a quel livello di priorità.

Status register – maschera di interrupt

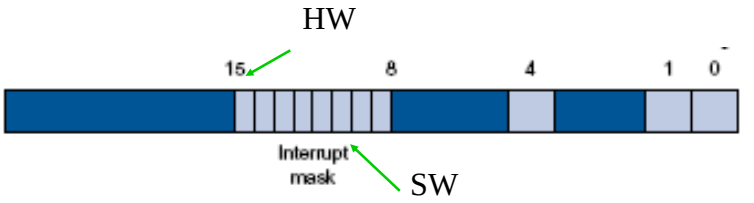
Registra **lo stato della CPU** nei confronti di interrupt / eccezioni

Un bit a 1 nella maschera di interrupt significa che gli interrupt a quel livello sono abilitati.

Quando arriva un interrupt imposta a 1 il bit corrispondente del registro causa. Verrà servito quando l'equivalente bit della maschera di stato è impostato a 1.

Vengono disabilitati ad esempio quando bit a priorità più elevata va in esecuzione di (mascheramento).

Se l'interrupt non viene servito immediatamente, viene inserito nella coda degli interrupt secondo la sua priorità. Verrà estratto dalla coda secondo la loro priorità.



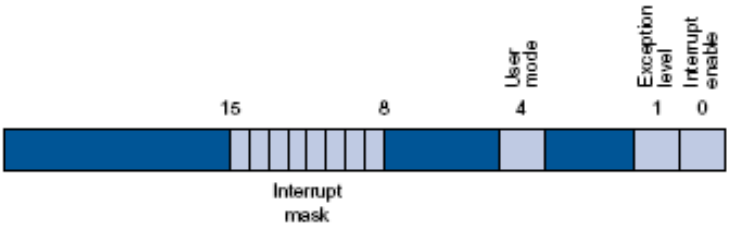
Status register - II

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

Exception level bit. Quando si verifica un'eccezione viene impostato a 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). Viene disabilitato per esempio quando si verifica un'eccezione.

I bit 0 e 1 abilitano gli interrupt esterni.



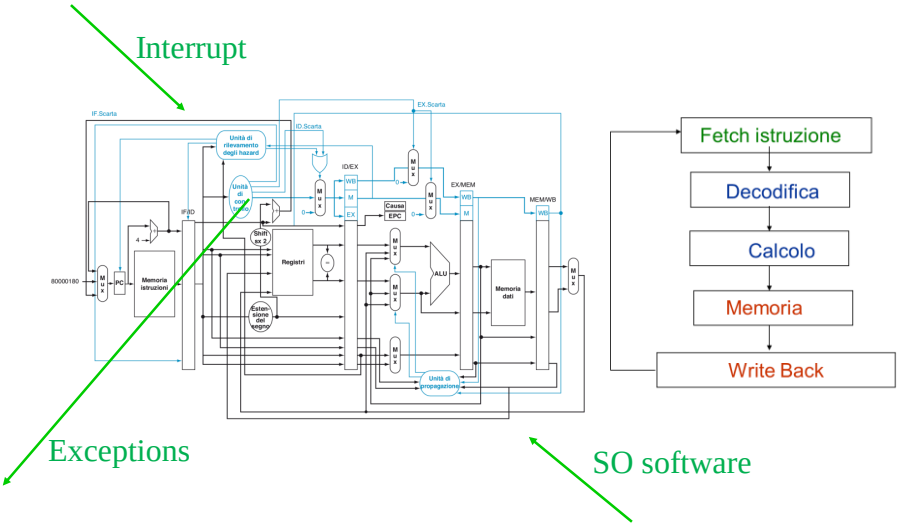
Sommario

Interrupt ed eccezioni

HW per la gestione delle eccezioni MIPS: modifica della CPU con pipeline

SW per la gestione delle eccezioni MIPS: esempio di procedura di risposta

La situazione per la CPU



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)

Come accedere ai registri del Coprocessore 0

Nome del registro	Numero del registro in coprocessore 0	Utilizzo
BadVAddr	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.
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.
Config	16	Configurazione della macchina

```
mfc0 $k0, $13        # copia il contenuto di "Causa" in $s0
mtc0 $k1, $14        # copia il contenuto di $k1 in "EPC"
```

La CPU non conosce nessuna semantica sui registri del coprocessore 0 come non la conosce sui registri del register file. *Lavora solo sui dati nel Register File principale.*

Come rispondere a un'eccezione

Ruolo dell'HW. Flush delle istruzioni e salvtaggio di alcune informazioni in coprocessore 0 o stack (salvataggio dello stato – tutto quello che serve per fare ripartire il codice correttamente da quel punto).

Ruolo del SW. Recuperare queste informazioni e prendere i provvedimenti opportuni.

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

Il SW, a seconda della situazione può eseguire alcune azioni oppure terminare il processo (o il sistema – schermata blu), cioè inserire nel PC la prima istruzione di un altro processo, eventualmente sospeso (e in coda) e non tornare al PC salvato.

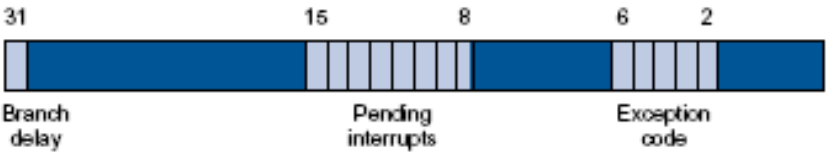
La sequenza logica di un "exception handler" (alcune operazioni svolte dall'HW altre devono essere svolte dal SW):

1. Leggere i registri del coprocessore 0
2. Salvare lo "stato": registri importanti + PC.
3. Rispondere all'eccezione
4. Ripristinare lo "stato".
5. Ripartire dall'istruzione successive a quella che ha generato l'eccezione.

2. Risposta all'eccezione (core)

Supponiamo che venga semplicemente stampata a schermo la causa dell'eccezione e poi il programma possa riprendere.

a. Leggo i registri causa ed EPC dal coprocessore 0
b. Estraggo la causa dell'eccezione dal registro causa
c. Se l'eccezione è un intrrupt (causa = 0), salto al codice di risposta agli interrupt (gestiti in ordine di priorita').
d. Gestisco l'eccezione (stampare la causa dell'eccezione e l'indirizzo (schermata blu))



2. Risposta all'eccezione (core) - I

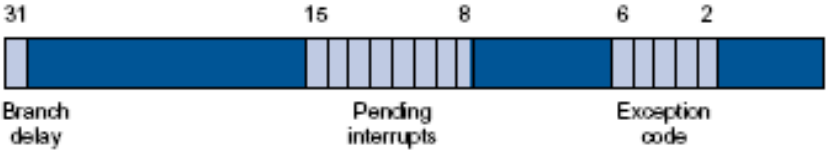
Supponiamo che venga semplicemente stampata a schermo la causa dell'eccezione e poi il programma possa riprendere.

a. Leggo i registri causa ed EPC dal coprocessore 0 e scrivo in `$k0` e `$k1`

```
mfco $k0, $13 # read from coprocessore 0 cause register
mfco $k1, $14 # read from coprocessore 0 EPC register
```

b. Estraggo la causa dell'eccezione dal registro causa

```
srl $a0, $k0, 2      # allineo l'Exception code di causa a 0
andi $a0, $a0, 0x1f # estraggo i 5 bit meno significativi 0:4
                     # dai bit di $a0 (maschera = 31 = 11111 = 0x1f)
                     # $a0 contiene ora la causa su 5 bit
```



2. Risposta all'eccezione (core) - II

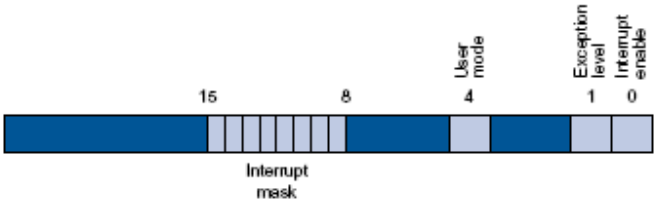
c. Se l'eccezione è un interrupt (causa = 0), passo a considerare i diversi interrupt

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)

```
beq $a0, $zero, dopo          # se causa = 0, nothing to do skip to interrupt
```

d. Imposto il registro di stato per gestire l'eccezione

```
mfco $k0, $12 # read from coprocessore 0 status register
addi $k0, $k0, 0xffff fffe  # azzero interrupt enable
ori $k0, $k0, 0x2           # imposto l'exception level
```



3. Risposta all'eccezione (core) - III

Risposta all'eccezione (viene stampata la causa dell'eccezione mediante `syscall`).

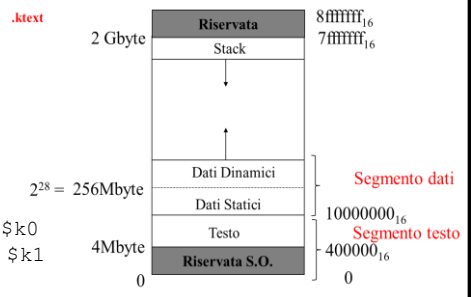
e. Stampare la causa dell'eccezione e l'indirizzo (schermata blu) o qualsiasi azione sia richiesta per risolvere l'eccezione.

```
move $a1, $k1                # move $k1 = EPC in $a1 for visualization
jal print_exception          # causa in $a0, EPC in $a1 (arguments di jal)
                             # print_exception utilizza una syscall
```


Risposta all'eccezione - Salvare lo stato

- In questo esempio lo stato è rappresentato dai registri `$a0, $a1` che vengono modificati dalla procedura di gestione dell'eccezione.
- L'exception handler deve risidere nella parte riservata al SO (**.ktext**) sopra i 2 GByte e non nel segmento testo (**.text**).
- PC viene salvato in EPC
- `$a0, $a1` (ed eventuali altri elementi dello stato) devono essere salvati in memoria kernel (**.kdata** e non .data) e non in stack o parte dati (.data), perchè l'eccezione può riguardare proprio l'accesso allo stack!

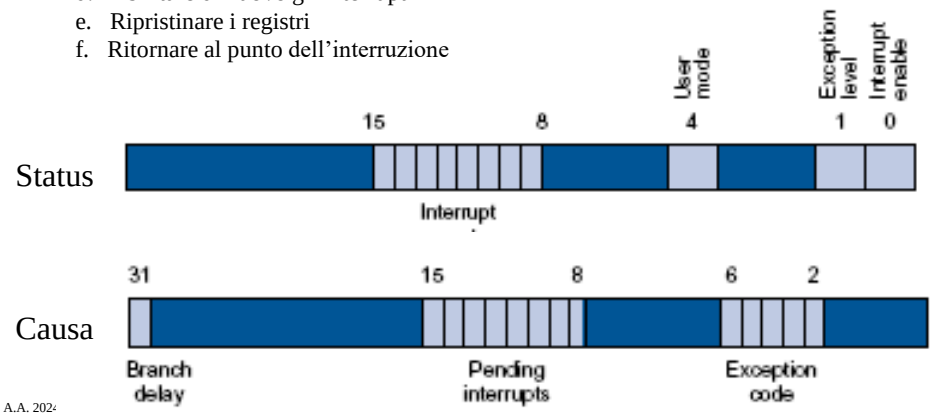
```
.ktext 0x80000180
       # Prima istruzione
0x8000 0180 sw $a0, save0
       # Seconda istruzione
0x8000 0184 sw $a1, save1

.kdata  # si trova sopra il .ktext
save0:  word 0    # Alloco spazio per $k0
save1: word 0     # Alloco spazio per $k1
```



Risposta all'eccezione - ripristinare lo stato

Dopo la parte "core", le operazioni da eseguire in sequenza sono:

a. Caricare l'indirizzo di ritorno (EPC-4) se l'istruzione deve essere rieseguita (e.g. eccezioni della memoria)
b. Cancellare il registro causa
c. Cancellare il flag di eccezione nel registro status
d. Abilitare di nuovo gli interrupt
e. Ripristinare i registri
f. Ritornare al punto dell'interruzione



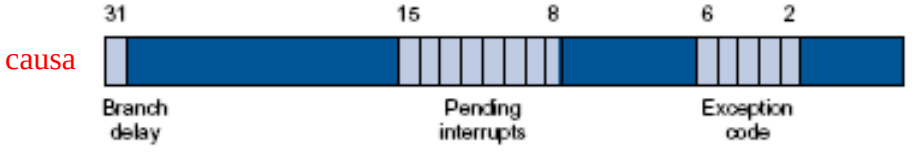
Ripristinare lo stato - I

a. Caricare l'indirizzo di ritorno (EPC-4) se l'istruzione deve essere rieseguita

```
done:                          # Si arriva a questo punto dopo avere
                               # risolto il problema sull'eccezione

        addi $k1, $k1, -4      # EPC is trova in $k1, calcolo PC-4
        mtc0 $k1, $14          # porta $k1, che contiene PC, in EPC
```

b. Azzerare il registro causa (ora contenuto in $a0)

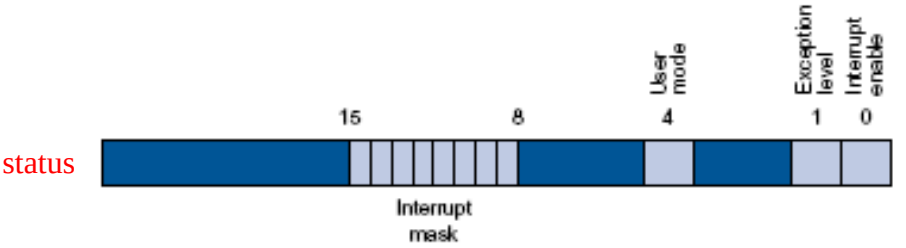
```
        addi $k0, $a0, 0xffff ffe0  # azzero i 5 LSB (exception code)
                                    # Maschera 11 .... 11100000
        sll $k0, $k0, 2             # Allineo di nuovo i bit di causa
        mtco $k0, $13               # aggiorna CAUSA
```



Ripristinare lo stato - II

c. Cancellare il flag di eccezione nel registro status
d. Abilitare di nuovo gli interrupt

```
mfco $k0, $12          # read status register from coprocessor 0
andi $k0, 0xfffd       # clear bit 1 (Exception level), preserve others
ori  $k0, $k0, 1       # set bit 0 (Enable interrupt)
mtco $k0, $12          # restore status register
```



Ripristinare lo stato - III

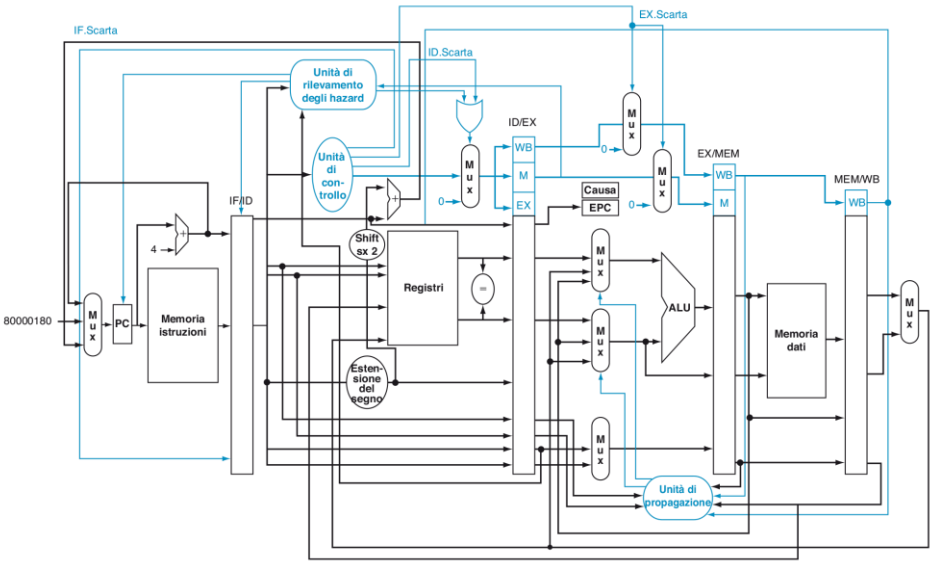
c. Ripristinare i registri $a0 e $a1 che erano stati utilizzati per la visualizzazione (`jal print_exception`)

```
lw $a0, save0
lw $a1, save1
```

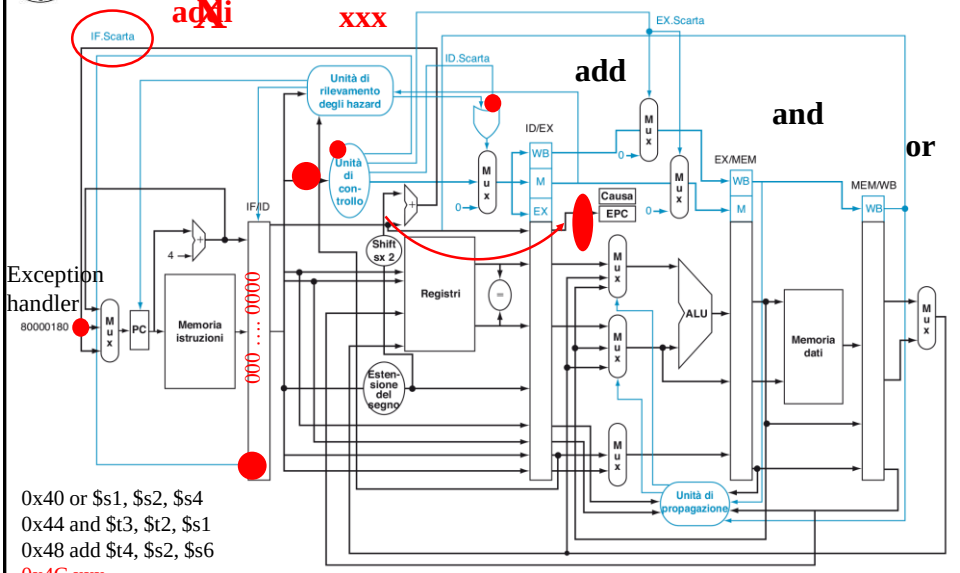
f. Ritornare al punto dell'interruzione

```
eret     #Exception return (EPC -> PC)
```

Schema finale CPU



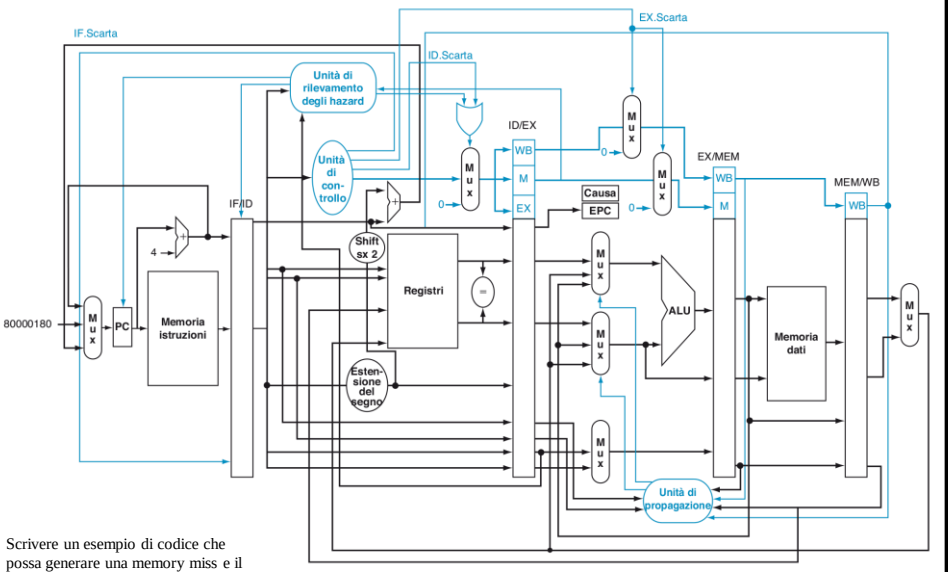
Istruzione non valida



Exception handler

0x40 or $s1, $s2, $s4
0x44 and $t3, $t2, $s1
0x48 add $t4, $s2, $s6
0x4C xxx
0x50 addi $s5, $s6, 20

Modificare la CPU per gestire una memory miss



Scrivere un esempio di codice che possa generare una memory miss e il contenuto della CPU.

Sommario

Interrupt ed eccezioni

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

SW per la gestione delle eccezioni MIPS: esempio di procedura di risposta