Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2017-2018

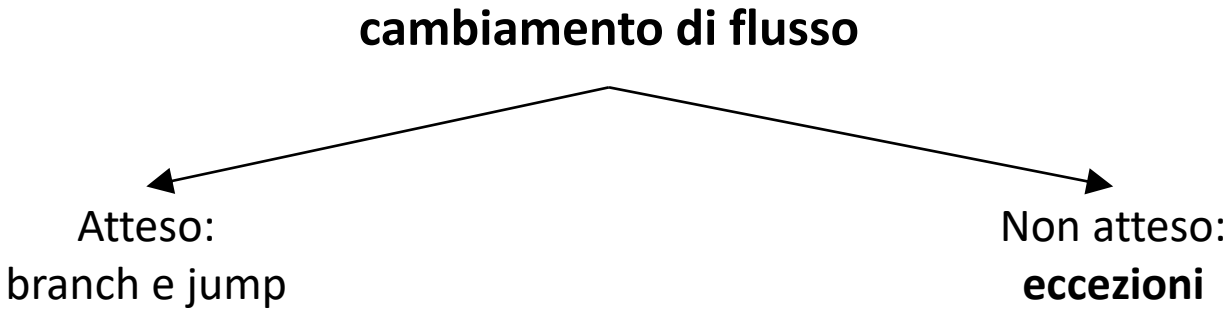
Eccezioni


Homepage del corso

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

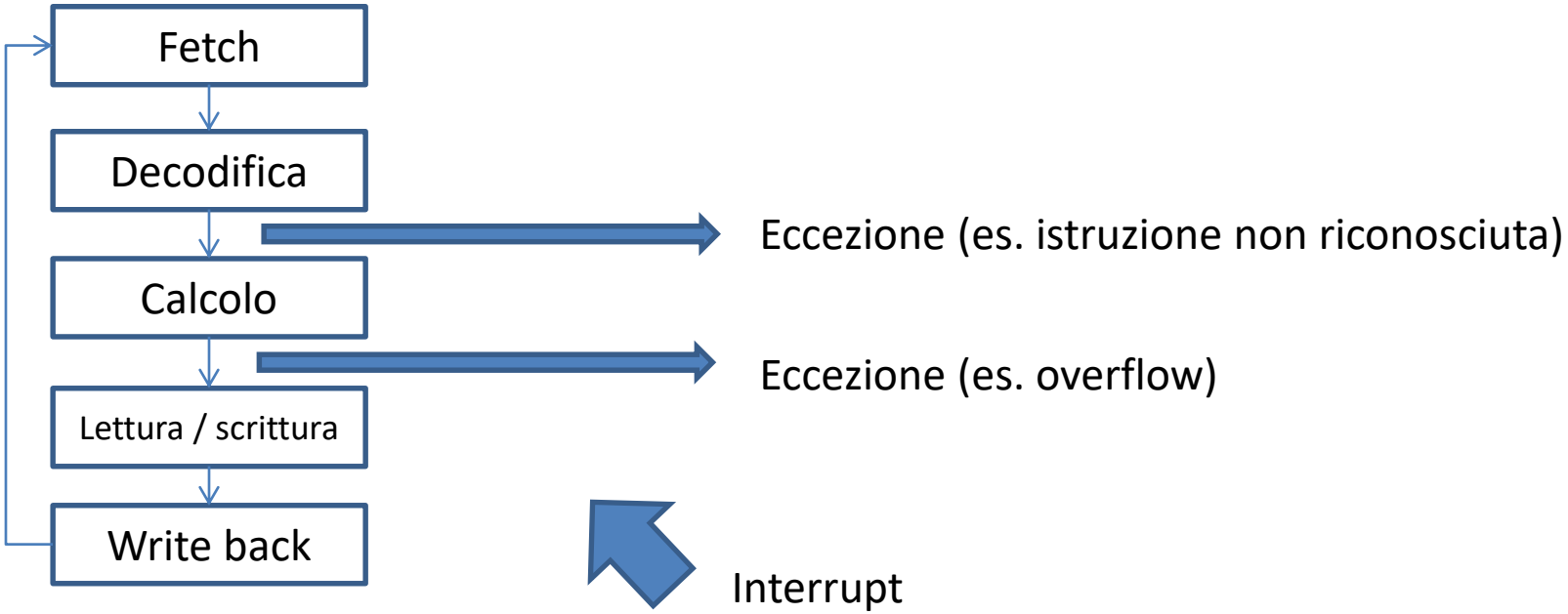
Turno B
Jacopo Essenziale
Dipartimento di Informatica
Via Celoria 20 - 20133 Milano (MI)
AISLab
jacopo.essenziale@unimi.it
+39 02.503.14010

Eccezioni



- Eccezioni sincrone: dato segmento dati e segmento testo, si verificano sempre ad un tempo *t* determinato (overflow, salti a istruzioni non definite ...)
- Eccezioni asincrone (interrupt): avvengono ad un tempo *t* non determinato dal programma in esecuzione (I/O, errori hardware, ...)

Eccezioni



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

Eccezioni

- Le eccezioni vanno gestite attraverso la chiamata ad una procedura speciale: l'**exception handler**

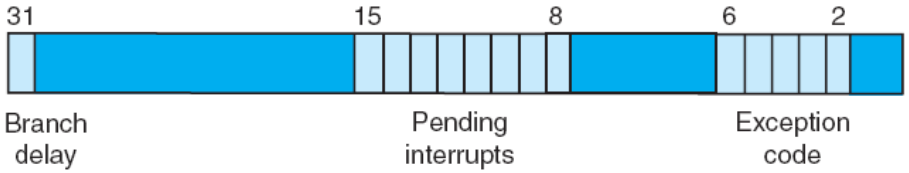
- Chiamata all'exception handler:
  - non prevede né passaggio di parametri, né valori di ritorno
  - deve preservare **interamente** lo stato del programma che ha causato l'eccezione (compresi i registri $t)
  - fa uso di informazioni per la gestione dell'eccezione che l'hardware ha lasciato in alcuni registri speciali

I registri per la gestione di eccezioni

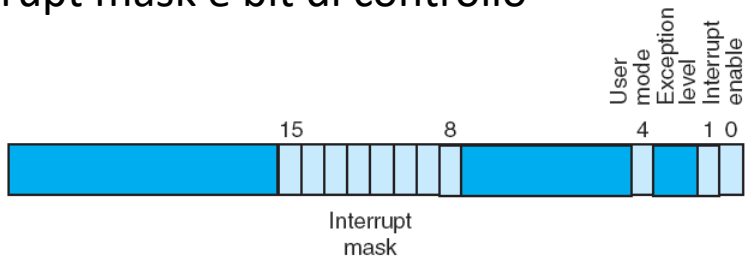
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (status)	12	65297
$13 (cause)	13	0
$14 (epc)	14	0

- Registro Cause: causa dell'eccezione specificata da un **Exception Code** (unsigned int, bit 2-6), indica anche gli interrupt che erano ancora da servire

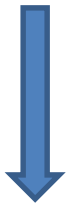


- Registro EPC: *exception program counter*, indirizzo (nel segmento testo) a cui sta la *faulting instruction*
- Registro BadVaddr: indirizzo errato nel caso di una *address exception*
- Registro Status: interrupt mask e bit di controllo



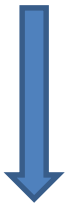
Esempi

```
.text
.globl main
main:
lui $t0, 0xffff
ori $t0, $t0, 0xffff
jr $t0
addi $v0, $zero, 10
syscall
```



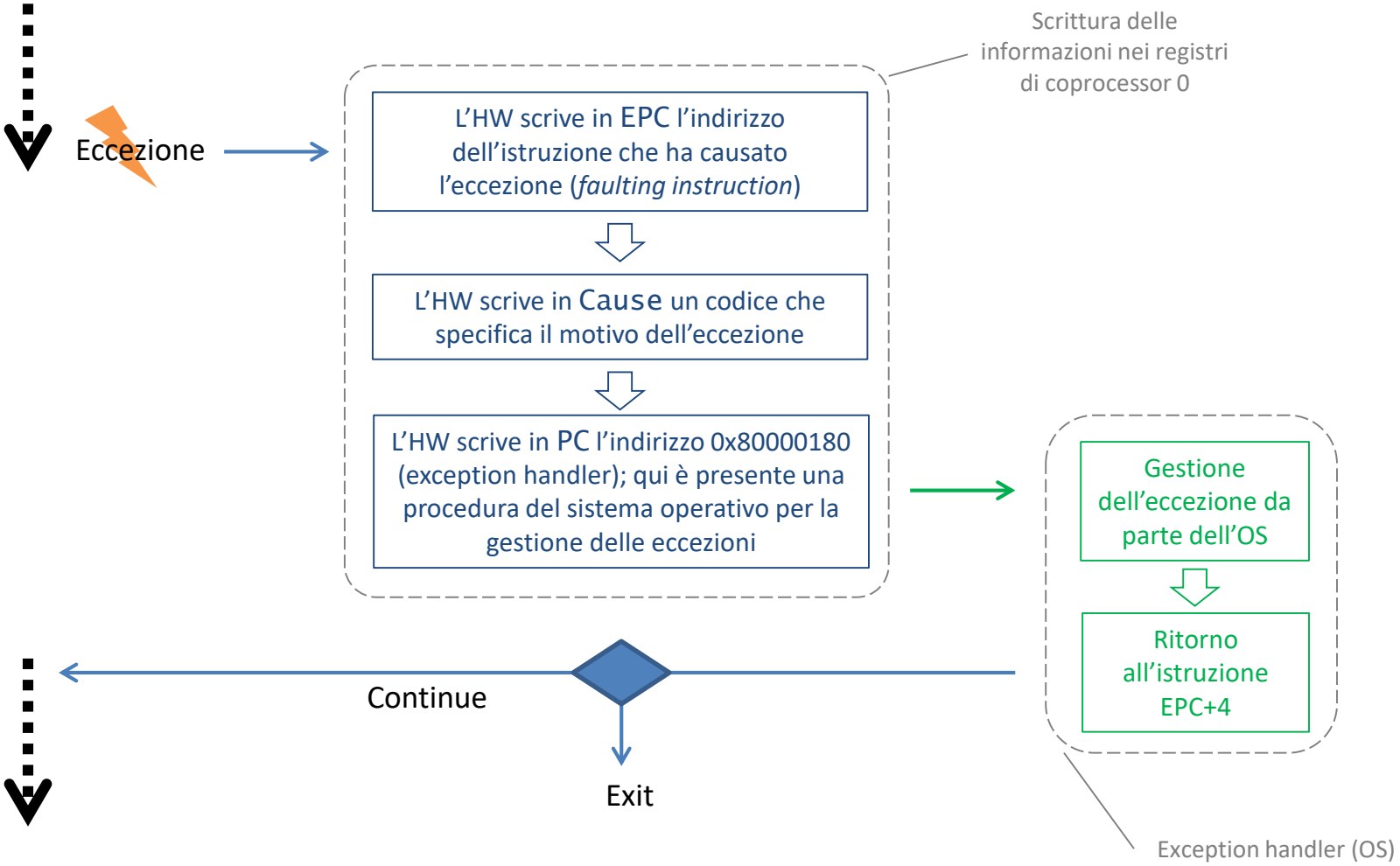
Bad Address in text read

```
.text
.globl main
main:
lui $t0, 0x8000
lui $t1, 0x8000
add $t2, $t0, $t1
addi $t3, $t2, -100
addi $v0, $zero, 10
syscall
```



Overflow

Gestione SW di un'eccezione



- Vediamo nel dettaglio come vengono svolte queste operazioni dal SW

Gestione SW di un'eccezione

- Come avviene la lettura e scrittura nei registri del coprocessore 0?

```
# copiare dal registro $13 del coprocessore 0 (Cause register)
al registro $t0

mfc0 $t0, $13

# copiare dal registro $t0 al registro $14 del coprocessore 0
(EPC)

mtc0 $14, $t0

# load word dall'indirizzo 100($t3) al registro $13 del
coprocessore 0

lwc0 $13, 100($t3)

# store word dal registro $13 del coprocessore 0 in memoria

swc0 $13, 50($t2)
```

Gestione SW di un'eccezione

- L'exception handler non deve alterare lo stato corrente di registri e memoria
  - ha a disposizione due registri riservati $k0 e $k1
  - nel caso debba fare register spilling, non userà lo stack ma la apposita area .kdata

Nome	Numero	Utilizzo	Preservato durante le chiamate
$zero	0	costante zero	*Riservato MIPS*
$at	1	riservato per l'assemblatore	*Riservato Compiler*
$v0-$v1	2-3	valori di ritorno di una procedura	No
$a0-$a3	4-7	argomenti di una procedura	No
$t0-$t7	8-15	registri temporanei (non salvati)	No
$s0-$s7	16-23	registri salvati	Si
$t8-$t9	24-25	registri temporanei (non salvati)	No
$k0-$k1	26-27	gestione delle eccezioni	*Riservato OS*
$gp	28	puntatore alla global area (dati)	Si
$sp	29	stack pointer	Si
$s8	30	registro salvato (fp)	Si
$ra	31	indirizzo di ritorno	No

Gestione SW di un'eccezione

- Suddividiamo logicamente la gestione di un'eccezione nelle tre fasi operative di *prologo*, *corpo della procedura* ed *epilogo*

- *Prologo*: stampa le informazioni relative all'eccezione sollevata

- *Corpo*: valuta la possibilità di ripristinare il flusso di esecuzione e, nel caso di un interrupt, esegue una procedura specifica per la sua gestione

- *Epilogo*: ripristina lo stato del processore e dei registri, fa riprendere l'esecuzione dall'istruzione successiva a quella che ha causato l'eccezione

Exception Handler (preambolo)

- Definizione del segmento dati: vengono preallocate le stringhe contenenti i vari messaggi di errore (che dipendono dal tipo di eccezione) e due variabili locali (s1 e s2)

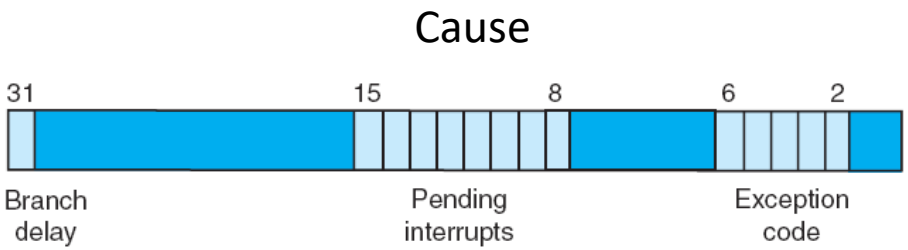
```
.kdata
__m1_:   .asciiz "  Exception "
__m2_:   .asciiz " occurred and ignored\n"
__e0_:   .asciiz "  [Interrupt] “

                    [...]

s1:      .word 0
s2:      .word 0
```

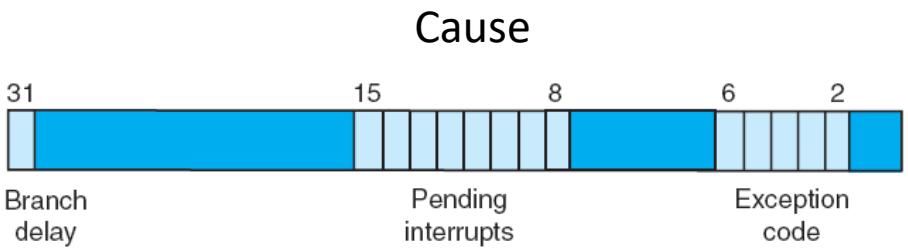
Exception Handler (prologo-1)

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



Exception Handler (prologo-1)

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

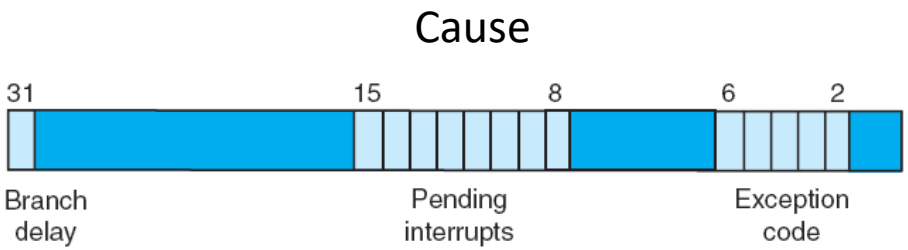



```
move $k1 $at          # Salvo $at (usato per pseudo istruzioni)
sw $v0 s1             # è necessario usare $v0 e $a0, quindi li salvo in memoria (no stack!)
sw $a0 s2

mfc0 $k0 $13          # copio Cause in $k0 (1)
srl $a0 $k0 2         # estraggo campo ExcCode (shift right logical) (2)
andi $a0 $a0 0x1f     # and con 00..0011111 (3)
```

Exception Handler (prologo-1)

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




```
move $k1 $at          # Salvo $at (usato per pseudo istruzioni)
sw $v0 s1             # è necessario usare $v0 e $a0, quindi li salvo in memoria (no stack!)
sw $a0 s2

mfc0 $k0 $13          # copio Cause in $k0 (1)
srl $a0 $k0 2         # estraggo campo ExcCode (shift right logical) (2)
andi $a0 $a0 0x1f     # and con 00..0011111 (3)
```

Esempio: Estrazione di ExcCode con Aritmetic Overflow (codice 12):

(1) Cause Register: 00000000000000000000000000 01100 00 *(in Spim vedremo 110000)*
(2) Shift a destra di 2 bit: 00 00000000000000000000000000 01100
(3) AND con 0x1F (11111): 00 00000000000000000000000000 01100

Exception Handler (prologo-2)

```
# Stampo informazioni sull'eccezione.

li $v0 4              # syscall 4 (print_str)
la $a0 __m1_
syscall

li $v0 1              # syscall 1 (print_int)
srl $a0 $k0 2                 # Extract ExcCode Field
andi $a0 $a0 0x1f
syscall

li $v0 4              # syscall 4 (print_str)
andi $a0 $k0 0x3c             # 3C=00111100
lw $a0 __excp($a0)
nop
syscall
```

Exception Handler (corpo-3)

- Prima cosa da fare: controllare se è possibile riprendere l'esecuzione, altrimenti eseguire exit

Se la causa **non** è *«bus error on instruction fetch»* (codice **6** (0x18), 00...00**110**00) allora $pc è valido, in caso contrario in $pc c'è un valore errato e sono necessari controlli aggiuntivi

```
bne $k0 0x18 ok_pc          # Bad PC exception requires special checks
nop
```

Exception Handler (corpo-3)

- Prima cosa da fare: controllare se è possibile riprendere l'esecuzione, altrimenti eseguire exit

Se la causa **non** è *«bus error on instruction fetch»* (codice **6** (0x18), 00...00**110**00) allora $pc è valido, in caso contrario in $pc c'è un valore errato e sono necessari controlli aggiuntivi

```
bne $k0 0x18 ok_pc          # Bad PC exception requires special checks
nop
```

Se è proprio un errore di fetch, controllo se in EPC ho un indirizzo valido (sintatticamente)

```
mfc0 $a0 $14  # copio EPC in $a0
andi $a0 $a0 0x3            # Is EPC word-aligned? (0x3=11)
beq $a0 0 ok_pc             # branch se gli ultimi due bit di EPC non sono 1 (divisibile per 4)
nop
```

Exception Handler (corpo-3)

- Prima cosa da fare: controllare se è possibile riprendere l'esecuzione, altrimenti eseguire exit

Se la causa **non** è *«bus error on instruction fetch»* (codice **6** (0x18), 00...00**11**00) allora $pc è valido, in caso contrario in $pc c'è un valore errato e sono necessari controlli aggiuntivi

```
bne $k0 0x18 ok_pc          # Bad PC exception requires special checks
nop
```

Se è proprio un errore di fetch, controllo se in EPC ho un indirizzo valido (sintatticamente)

```
mfc0 $a0 $14  # copio EPC in $a0
andi $a0 $a0 0x3            # Is EPC word-aligned (0x3=11)?
beq $a0 0 ok_pc             # branch se gli ultimi due bit di EPC non sono 1 (divisibile per 4)
nop
```

Nel caso in cui anche EPC non sia valido, non mi resta che fare exit (l'esecuzione non può continuare dopo l'eccezione).

```
li $v0 10                                   # Exit on really bad PC
syscall

ok_pc:
…
```

Exception Handler (corpo-4)

Controllo se c'è stato un interrupt. In quel caso verrà gestito con del codice specifico.

```
ok_pc:
        li $v0 4                    # syscall 4 (print_str)
        la $a0 __m2_
        syscall

        srl $a0 $k0 2                       # Extract ExcCode Field
        andi $a0 $a0 0x1f
        bne $a0 0 ret                       # 0 means exception was an interrupt
        nop

# Interrupt-specific code
# Don't skip instruction at EPC since it has not executed.
```

Se non è un interrupt passo all'epilogo della gestione

Exception Handler (epilogo-5)

Calcolo l'indirizzo da cui riprendere l'esecuzione e lo copio in EPC

```
ret:
# Return from (non-interrupt) exception. Skip faulting instruction
# at EPC to avoid infinite loop.

mfc0 $k0 $14          # Bump EPC register
addiu $k0 $k0 4                  # Skip faulting instruction
mtc0 $k0 $14          # write back EPC
```

Exception Handler (epilogo-5)

Calcolo l'indirizzo da cui riprendere l'esecuzione e lo copio in EPC

```
ret:
# Return from (non-interrupt) exception. Skip faulting instruction
# at EPC to avoid infinite loop.

mfc0 $k0 $14          # Bump EPC register
addiu $k0 $k0 4                  # Skip faulting instruction
mtc0 $k0 $14          # write back EPC
```

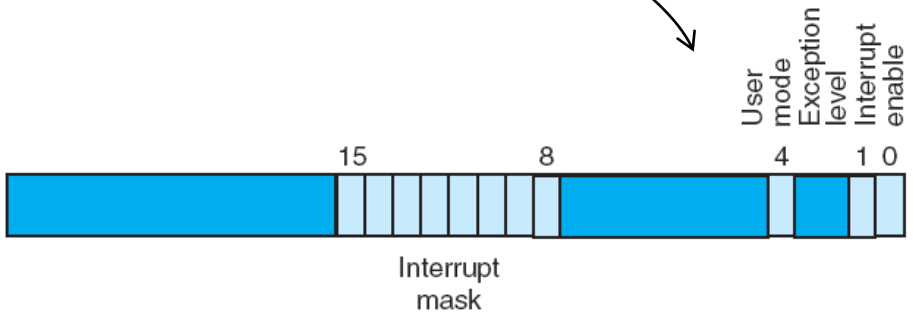
Ripristino i registri a lo stato del processore: $v0, $a0, $at, e i registri Cause e **Status**

```
# Restore registers and reset processor state

lw $v0 s1                # Restore other registers
lw $a0 s2
move $at $k1  # Restore $at

mtc0 $0 $13    # Clear Cause register

mfc0 $k0 $12  # Set Status register
ori  $k0 0x1     # Interrupts enabled
mtc0 $k0 $12
```



Exception Handler (epilogo-5)

Calcolo l'indirizzo da cui riprendere l'esecuzione e lo copio in EPC

```
ret:
# Return from (non-interrupt) exception. Skip faulting instruction
# at EPC to avoid infinite loop.

mfc0 $k0 $14          # Bump EPC register
addiu $k0 $k0 4                  # Skip faulting instruction
mtc0 $k0 $14          # write back EPC
```

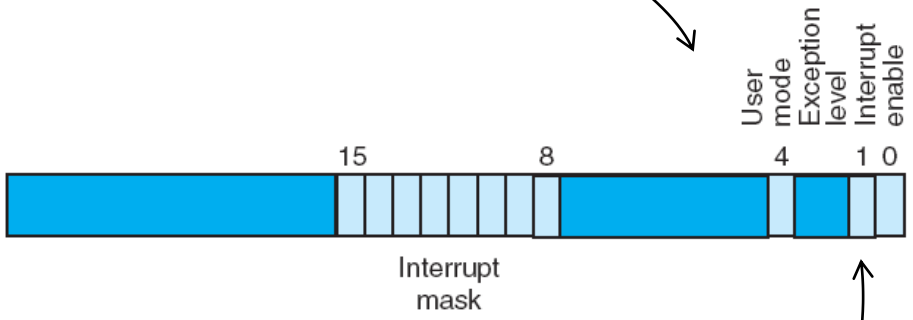
Ripristino i registri a lo stato del processore: $v0, $a0, $at, e i registri Cause e **Status**

```
# Restore registers and reset processor state

lw $v0 s1                # Restore other registers
lw $a0 s2
move $at $k1  # Restore $at

mtc0 $0 $13    # Clear Cause register

mfc0 $k0 $12  # Set Status register
ori  $k0 0x1     # Interrupts enabled
mtc0 $k0 $12
```

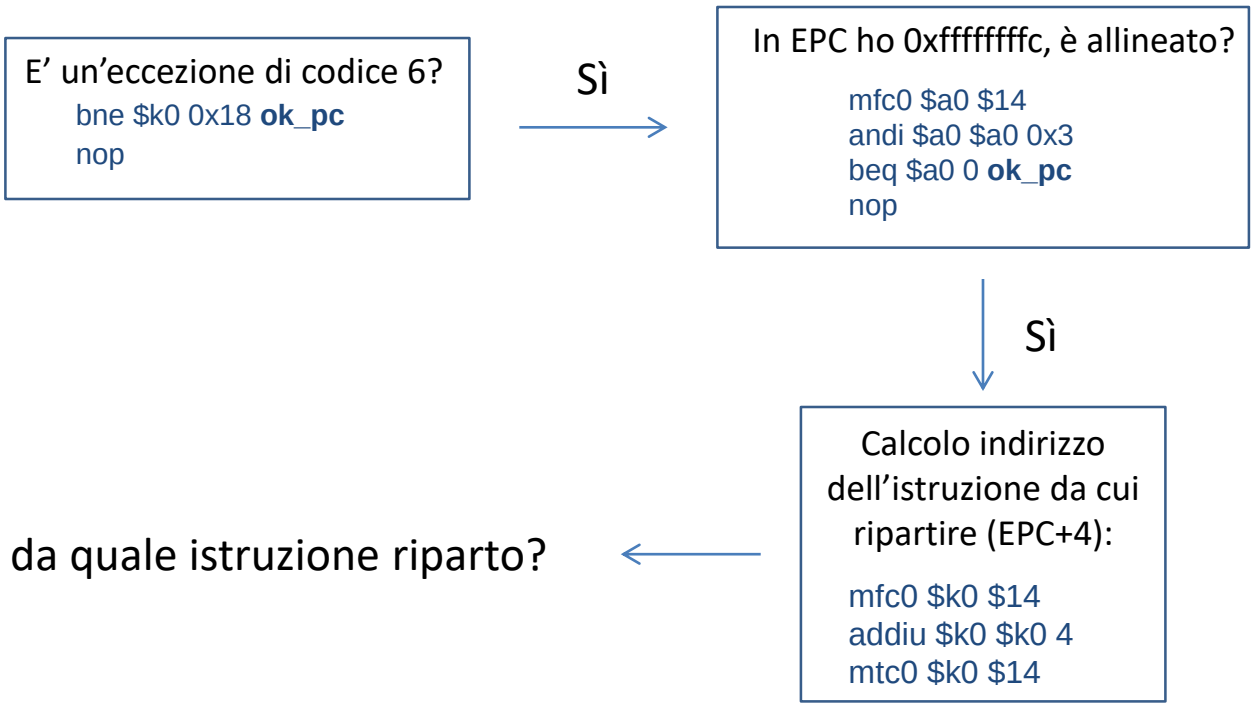



```
# Return from exception on MIPS32:

eret                     # setta exception level a 0
                         # ritorna a istruzione indirizzata da EPC
```

Esempio

- Cosa succede quando facciamo una jump a 0xffffffff (esempio nelle slides precedenti)?



Università degli Studi di Milano
Laboratorio di Architettura degli Elaboratori II
Corso di Laurea in Informatica, A.A. 2017-2018

Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Hanno contribuito alla realizzazione di queste slides:

- Iuri Frosio
- Jacopo Essenziale