

I/O con Interrupt

Argomento:

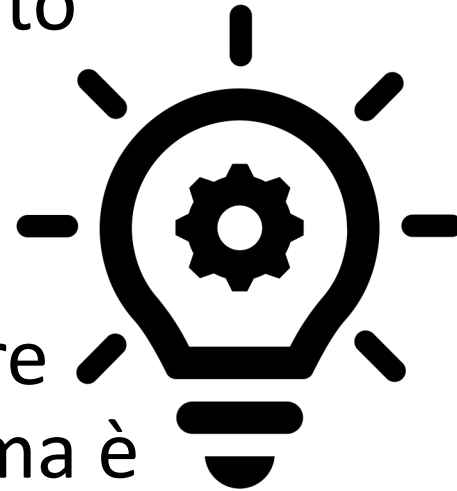
- Input/Output
- Interruzioni
- Gerarchia delle interruzioni

Materiale di studio

- Paragrafo 3.2
- Capitolo 7

Una nuova idea!

- Nella gestione a I/O programmato, il processore perde un sacco di cicli aspettando che il dispositivo sia pronto
- Nella gestione a interrupt (o interruzioni) il processore NON sta in attesa che il dispositivo di I/O sia pronto ma è il dispositivo che avverte il processore quando è pronto



I/O Programmato vs. Interruzioni



- Questo è un processore in busy waiting:
- Il processore è occupato nell'attesa che il dispositivo sia pronto per l'I/O
- Durante l'attesa il processore non può fare altro

I/O PROGRAMMATO
BUSY WAITING

I/O Programmato vs. Interruzioni

- Questo è un processore durante un evento di I/O gestito ad interruzioni
- Il processore deve sempre attendere che il dispositivo di I/O sia pronto
- Durante l'attesa però il processore può continuare a eseguire operazioni



GESTIONE
AD INTERRUZIONI

I/O ad Interruzioni: il concetto di base

- Il processore avverte il dispositivo di I/O che vuole effettuare una operazione di I/O
- Il processore torna a svolgere operazioni
- Il dispositivo, quando è pronto, avverte il processore
- Il processore quando riceve questo segnale:
 - smette di svolgere operazioni
 - esegue l'operazione di I/O
- Terminato l'I/O, il processore torna al programma che stava eseguendo

Diagramma

I/O PROGRAMMATO

Esecuzione	I/O	Dispositivo
Programma		I/O
lavoro		
lavoro		
vorrei dati		sto preparando
	wait	il
	wait	dato
	wait	richiesto
	wait	dato pronto
	ora	
	elaboro	
	il dato	
lavoro		
lavoro		
lavoro		
lavoro		
lavoro		
lavoro		

Il processore deve aspettare che il dato arrivi per proseguire a lavorare

I/O a INTERRUZIONI

Esecuzione	I/O	Dispositivo
Programma		I/O
lavoro		
lavoro		
vorrei dati		sto preparando
lavoro		il
lavoro		dato
lavoro		richiesto
lavoro		dato pronto
	ora	
	elaboro	
	il dato	
lavoro		
lavoro		
lavoro		
lavoro		
lavoro		
lavoro		

Il processore riesce ad eseguire più operazioni

Architettura degli Elaboratori

Il processore non deve stare in attesa, può continuare a lavorare

Il processore non sa quando dovrà smettere di eseguire il programma e dovrà elaborare il dato in I/O

interruzione

La gestione a interruzioni richiede alcuni nuovi accorgimenti

Gestione dell'I/O a Interruzioni

Cosa ci serve:

- Il modulo di I/O deve essere in grado di generare il segnale che avverte il processore che il dato è pronto
- Il processore deve essere in grado di interrompere quello che sta facendo per passare velocemente a leggere il dato pronto e poi ritornare a quelle che stava facendo, senza provocare danni

Gestione dell'I/O a Interruzioni

Cosa ci serve:

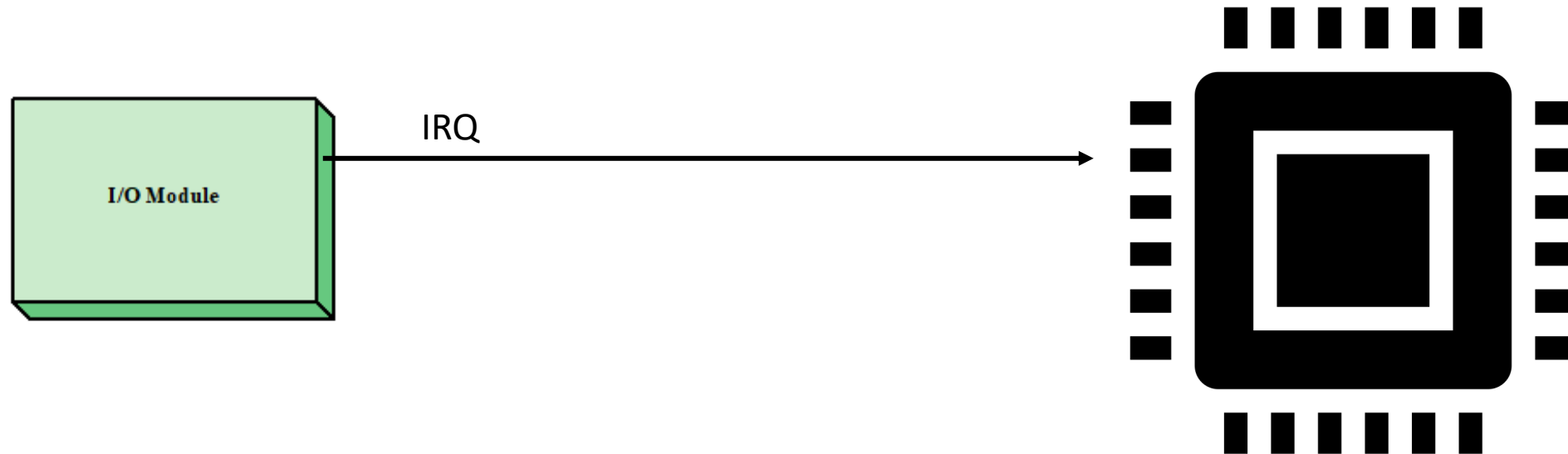
- Il modulo di I/O deve essere in grado di generare il segnale che avverte il processore che il dato è pronto (**IRQ**)
- Il processore deve
 1. interrompere quello che sta facendo (**Salvataggio Contesto**)
 2. passare velocemente a leggere il dato pronto (**RSI**)
 3. ritornare a quelle che stava facendo (**Ripristino Contesto**)

Sistema di interruzione



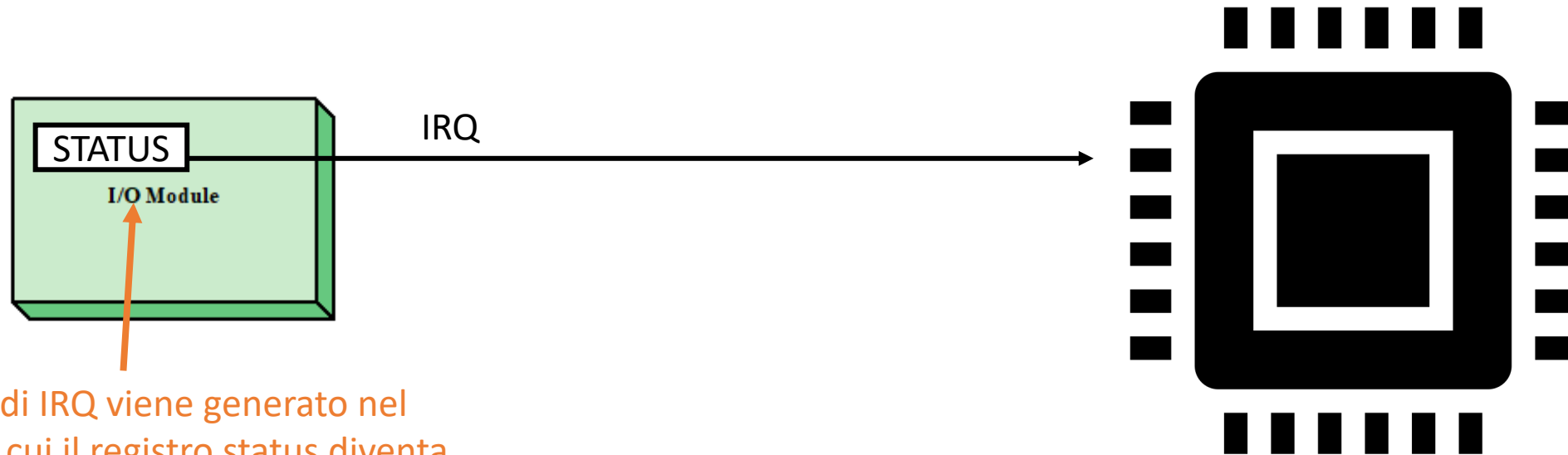
IRQ: Interrupt ReQuest

- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore



IRQ: Interrupt ReQuest

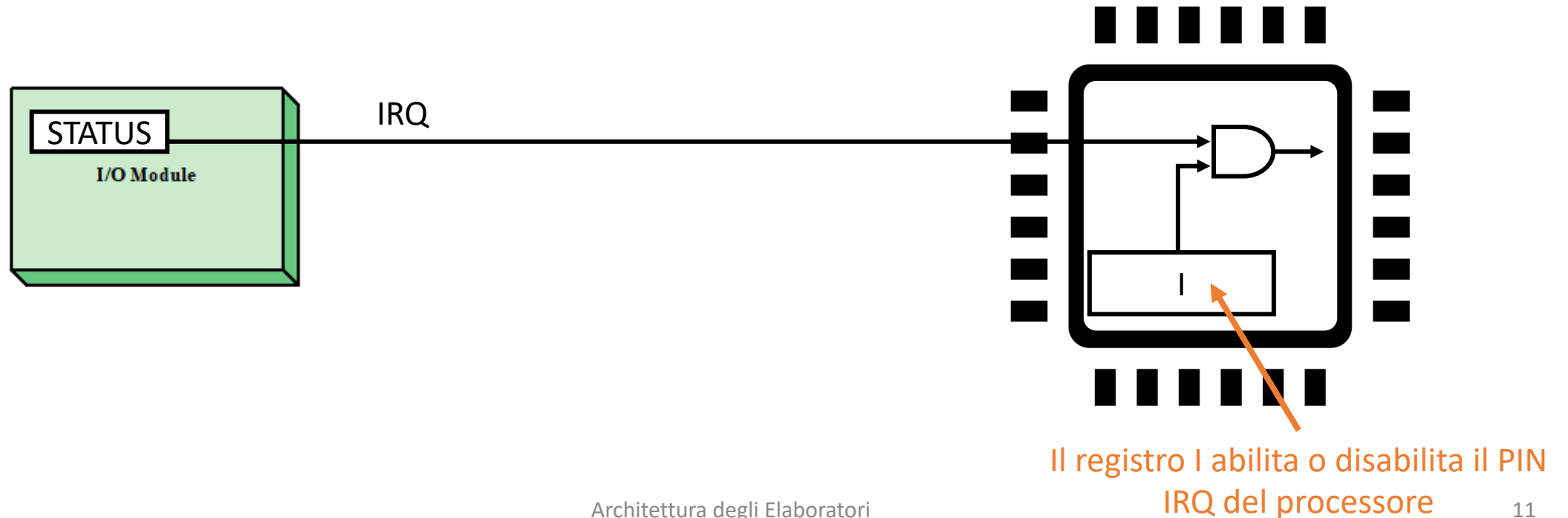
- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore



Il segnale di IRQ viene generato nel momento in cui il registro status diventa READY, segnalando che il dispositivo è pronto per l'I/O

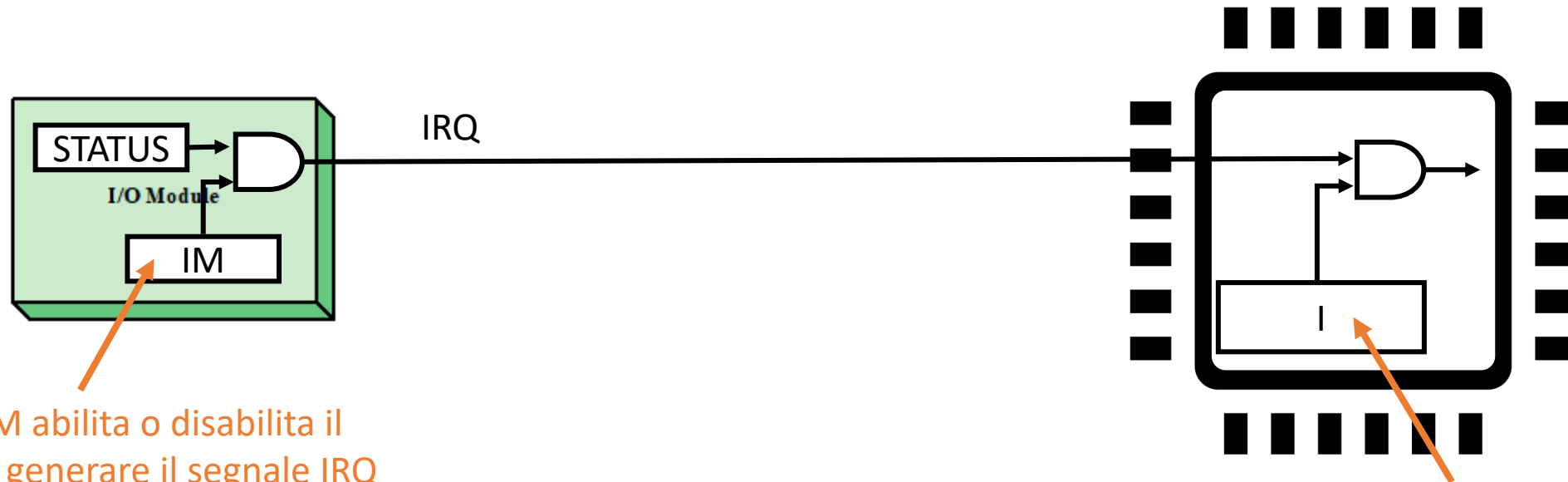
IRQ: Interrupt ReQuest

- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore



IRQ: Interrupt ReQuest

- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore

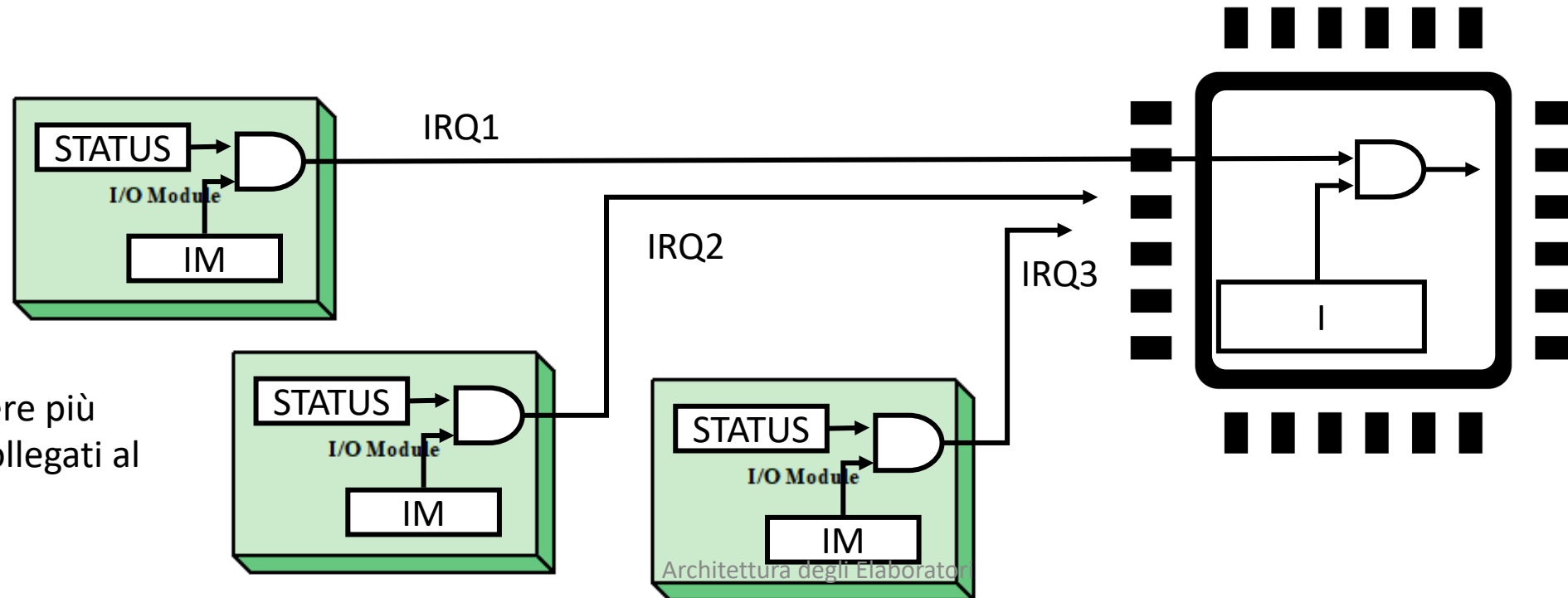


Il registro IM abilita o disabilita il modulo I/O a generare il segnale IRQ

Il registro I abilita o disabilita il PIN IRQ del processore

IRQ: Interrupt ReQuest

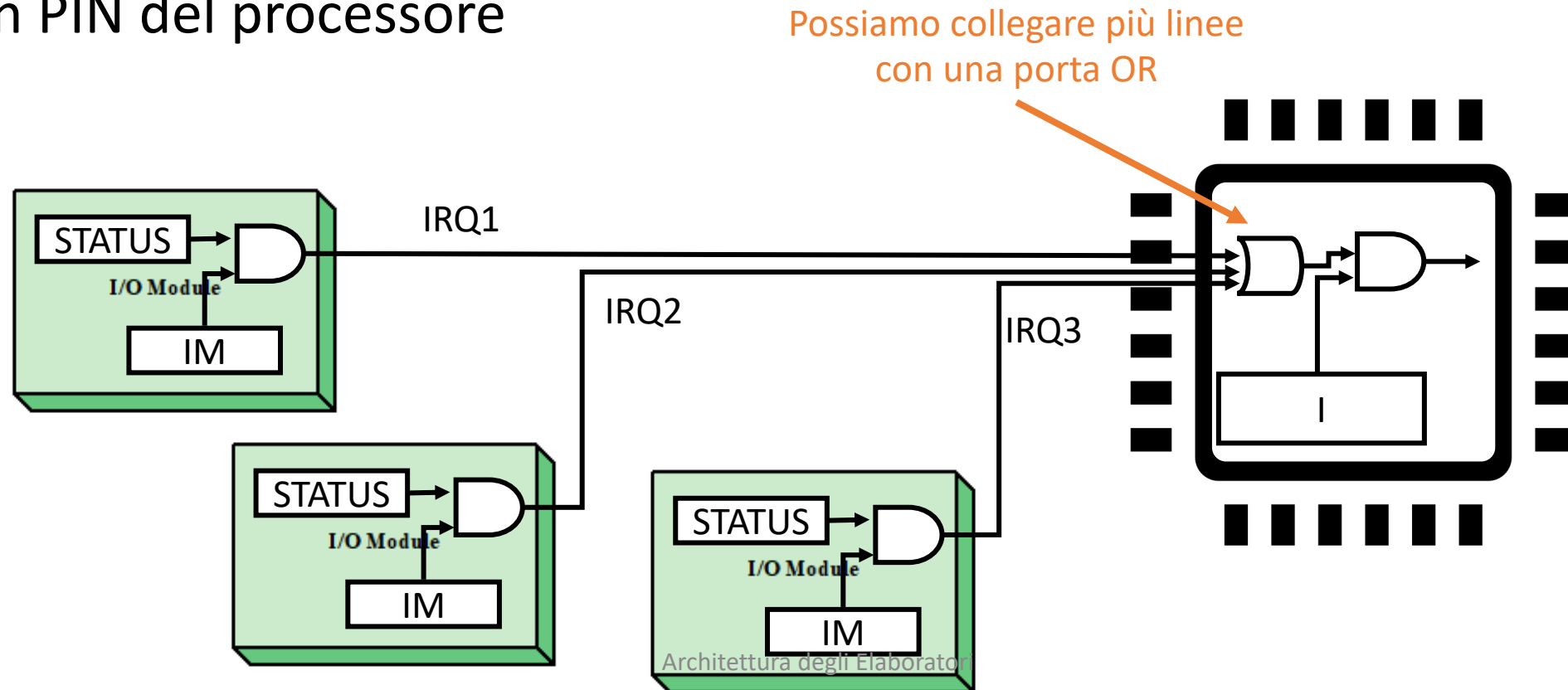
- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore



Possiamo avere più moduli I/O collegati al processore ...

IRQ: Interrupt ReQuest

- Per la gestione dell'IRQ dobbiamo aggiungere una nuova linea al nostro elaboratore: questa linea parte dal modulo di I/O e arriva ad un PIN del processore



Salvataggio del contesto

- Quando il processore «sente» sul suo PIN collegato alla linea IRQ che c'è una richiesta di interruzione deve interrompere quello che sta facendo e gestire l'I/O
- È necessario interrompersi «presto» perché i dati in I/O potrebbero altrimenti andare persi!
- Il programma in esecuzione viene interrotto «bruscamente», per eseguire le operazioni di I/O: la cosa è simile ad una chiamata a subroutine, ed è quindi necessario salvare i dati in uso e il PC, per poter tornare al programma una volta terminato l'I/O

Salvataggio del contesto

- C'è però un problema: NON SAPPIAMO quando avverrà questa chiamata a sub-routine!
- Tocca al processore salvare in automatico il contesto!
- Il processore salva:
 - Il registro PROGRAM COUNTER (PC)
 - Il registro con lo STATO del processore (CPSR)
 - ...e poi inizia la RSI

...e i dati del nostro programma??

→ Verranno salvati via software all'inizio della RSI

Ovviamente in tutto
questo il processore
NON DEVE ESSERE
INTERROTTO!

Salvataggio del contesto

Al verificarsi di una interruzione, l'Hardware deve:

1. commutare al valore 0 il registro I per impedire che si verifichino altre interruzioni
0 → I
2. provvedere a commutare almeno i registri CPSR e PC:
CPSR → push
PC → push
3. inserire, nel PC, l'indirizzo della RSI, la routine di servizio (collocata, ad esempio, a partire dall'indirizzo 0)
0 → PC



in questo modo, tutte le interruzioni attivano la stessa routine-preambolo, situata a partire da M[0]

La RSI: Routine di Servizio

- Lo scopo della RSI è quello di effettuare il prima possibile l'operazione di I/O sul dispositivo che si è dichiarato «ready»
 - La RSI è un programma software scritto appositamente per la gestione di un tipo di I/O: in un elaboratore con più dispositivi di I/O ci sono più RSI
 - È quindi necessario capire QUALE dispositivo di I/O ha generato l'IRQ ed eseguire la RSI corretta!
- ➡ Come facciamo a identificare il dispositivo I/O?
- ➡ E se abbiamo più IRQ contemporanei, quale gestiamo per primo?

Preambolo alla RSI

- **Prima** di eseguire la RSI che gestisce l'IRQ dobbiamo fare delle operazioni:
 1. Finire di salvare il contesto
 2. Capire chi ha generato l'IRQ
 3. Scegliere chi servire se abbiamo più dispositivi pronti
 4. Caricare nel PC l'indirizzo della RSI specifica che vogliamo eseguire
- A seconda dell'architettura, questi passi possono essere svolti via software o via hardware
 - Farlo via hardware è più veloce
 - Farlo via software non richiede dell'hardware dedicato (e quindi l'architettura del processore è più semplice)

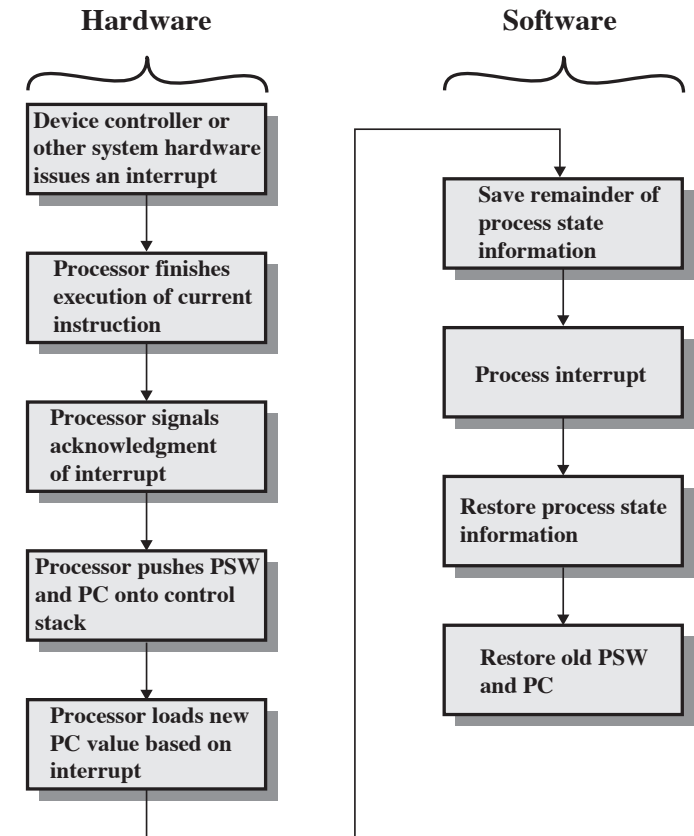


Figure 7.6 Simple Interrupt Processing

Preambolo: salvataggio del contesto

- Via hardware di base viene salvato solo PC e SR
- Ma per poter tornare poi al programma che era in esecuzione quando è arrivata l'interruzione, dobbiamo copiare anche gli altri dati in uso!

Via software

- Copiamo i valori dei registri del processore nello stack

Via hardware

- Una alternativa molto più efficiente è quello di avere più registri: quando arriva una interruzione, la RSI usa un nuovo set di registri (banked registers)

Preambolo: identificazione

Via software

- Andiamo a leggere il registro status di tutti i device per capire chi ha generato il segnale di IRQ (**software polling**)

Via hardware

- Una possibile alternativa è avere più linee di IRQ, una per device (ognuna collegata ad un pin diverso del processore): quando arriva un IRQ su un certo pin sappiamo già quale RSI chiamare
- ...ma vedremo altri modi di organizzare l'hardware per avere questo comportamento senza dover aggiungere fisicamente tantissime linee al processore

Prembolo: gestione di IRQ concorrenti

- Il processore è bombardato da eventi di I/O: è possibile che:
 - Quando andiamo a vedere chi ha chiamato l'IRQ scopriamo che ci sono più device da servire
 - Mentre stiamo eseguendo una RSI, arrivi un'altra richiesta di IRQ
- In questi casi dobbiamo anche scegliere chi servire prima: serve gestire una priorità!

Via software

- Il software polling già impone una gerarchia tra gli IRQ: è l'ordine con cui facciamo il polling, il primo device che troviamo con status=ready, viene servito
- Quando serviamo un device con una certa priorità X, possiamo impedire a tutti i dispositivi meno importanti di generare IRQ andando a mettere a 0 il loro registro IM

Via Hardware

- Questo stesso ragionamento si può realizzare in hardware ...

La Routine di Servizio

- La programmiamo noi
- Deve rimuovere il segnale IRQ!
- Opera in modo supervisore
- Può essere interrotta!
- Può avvenire molto dopo che abbiamo chiesto il dato

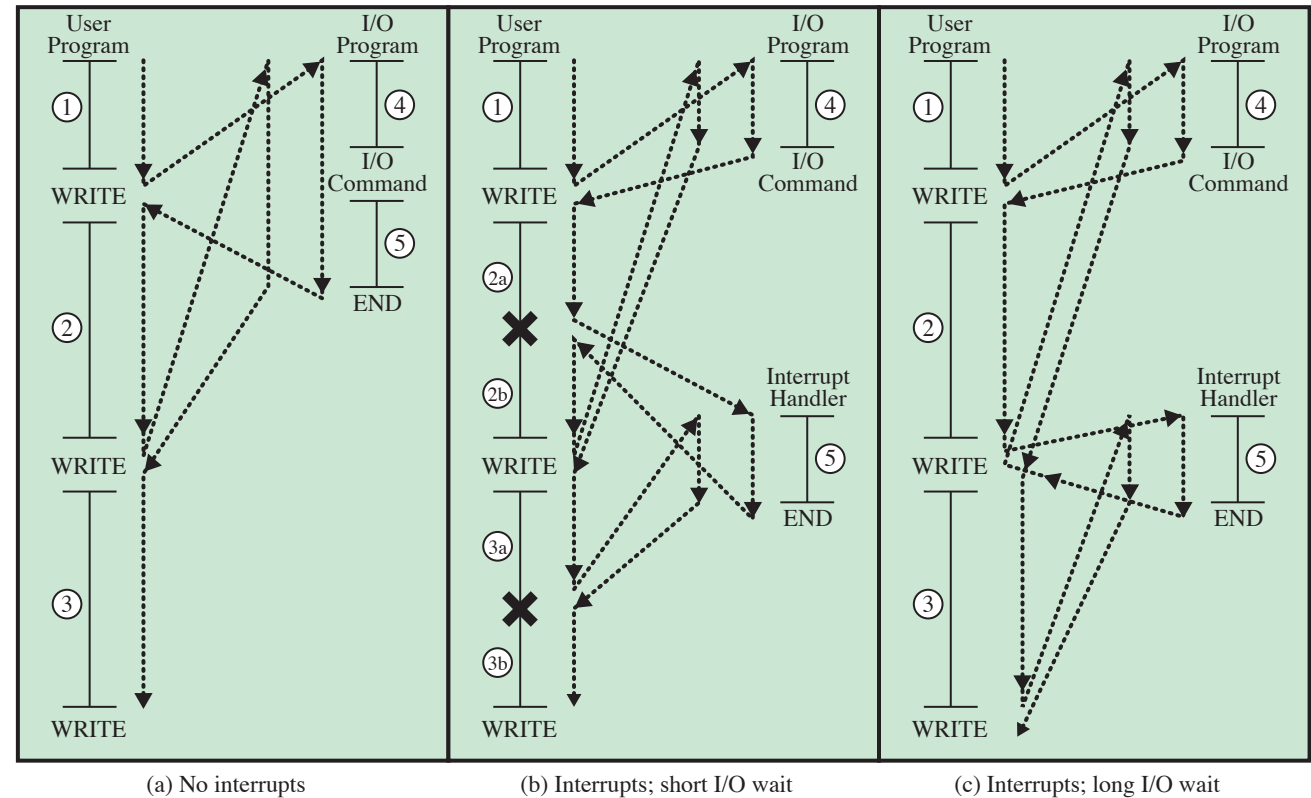


Figure 3.7 Program Flow of Control Without and With Interrupts

Ripristino del contesto

Al termine della interruzione, per ripristinare il contesto il processore deve:

1. provvedere a commutare almeno i registri SR e PC:

pop → **PC**

pop → **CPSR**

2. commutare al valore 1 il registro I per permettere che vengano rilevate altre interruzioni

1 → **I**

Queste operazioni permettono di riprendere l'esecuzione del programma che è stato interrotto dall'IRQ

Soluzioni hardware (più efficienti)

Salvataggio /
Ripristino contesto

- Banked Registers

Identificazione

- Interrupt Vector Number

Gestione priorità

- Masking
- Daisy Chain

Salvataggio del contesto

Via software

- Dobbiamo salvare nello stack tutti i registri che usiamo nella RSI (e poi ripristinarli)

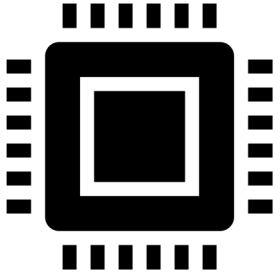
```
push {rX-rY}
```

```
str rX, [sp, #-4]!  
str r..., [sp, #-4]!  
str r..., [sp, #-4]!  
str r..., [sp, #-4]!  
str rY, [sp, #-4]!
```

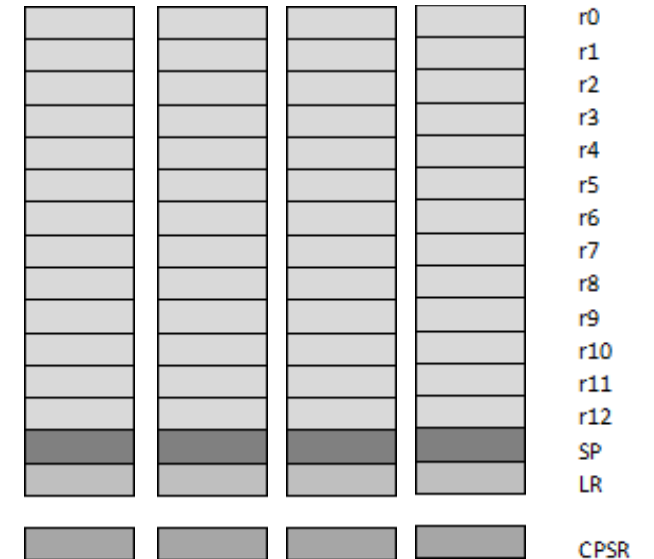
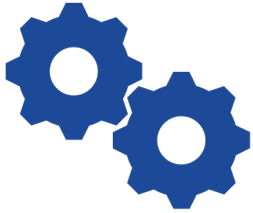
Via hardware

- Potremmo avere più set di registri: quando dobbiamo eseguire la RSI, usiamo un nuovo set dei registri
- La RSI scrive in questi nuovi registri, mentre i dati del programma interrotto restano non modificati nell'altro set
- Al termine della RSI, torniamo al set precedente

Banked Registers



Il processore sta eseguendo le istruzioni di un programma di un utente, che sta usando i registri

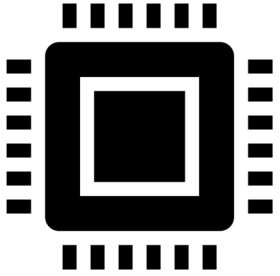


IRQ

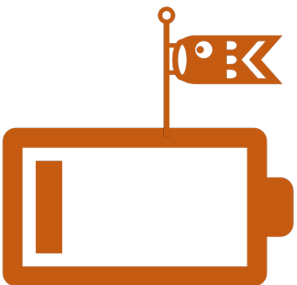
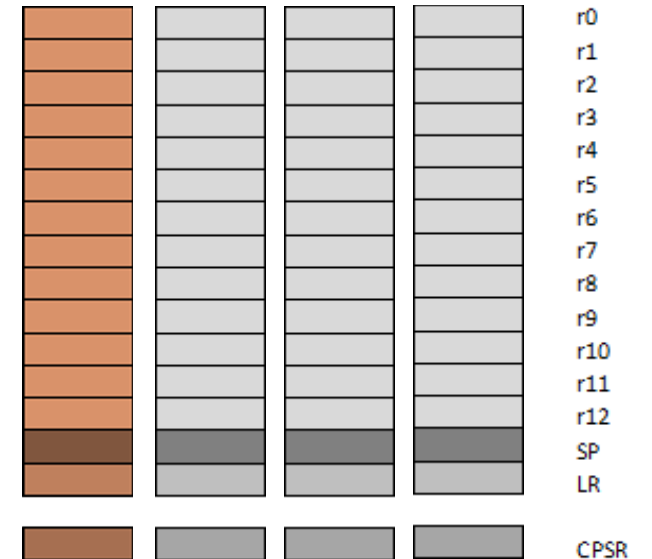
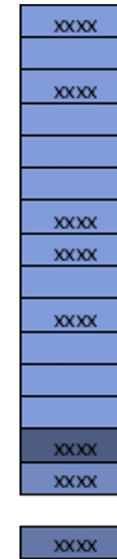
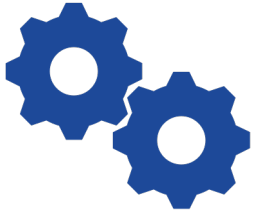


Arriva una richiesta di IRQ

Banked Registers

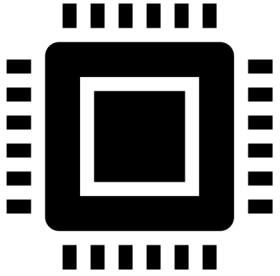


Il processore interrompe il programma utente e individua la RSI da chiamare, attiva un nuovo set di registri

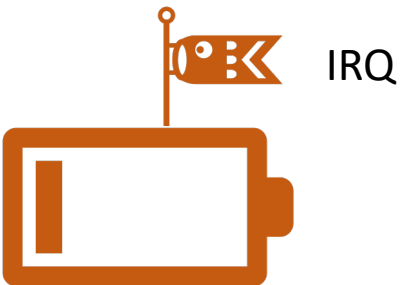
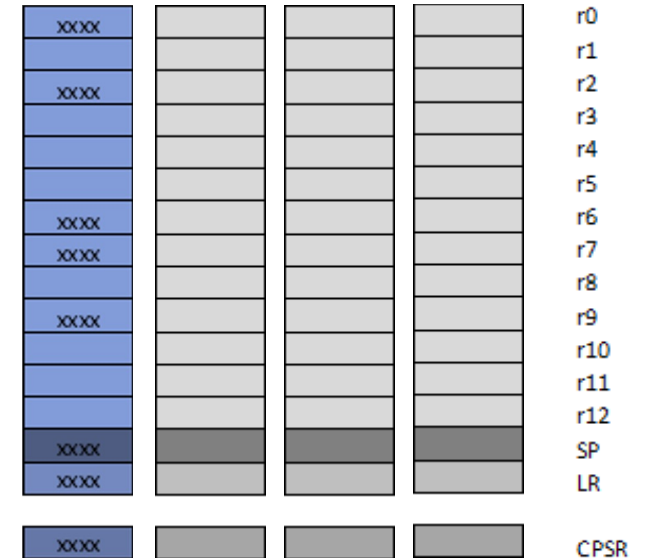
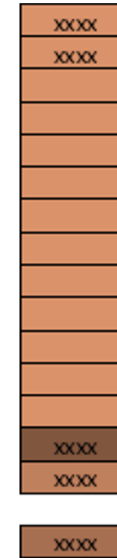
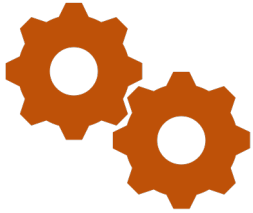


IRQ

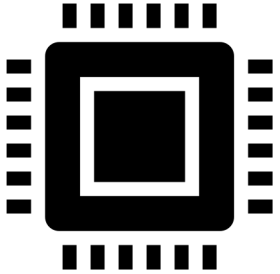
Banked Registers



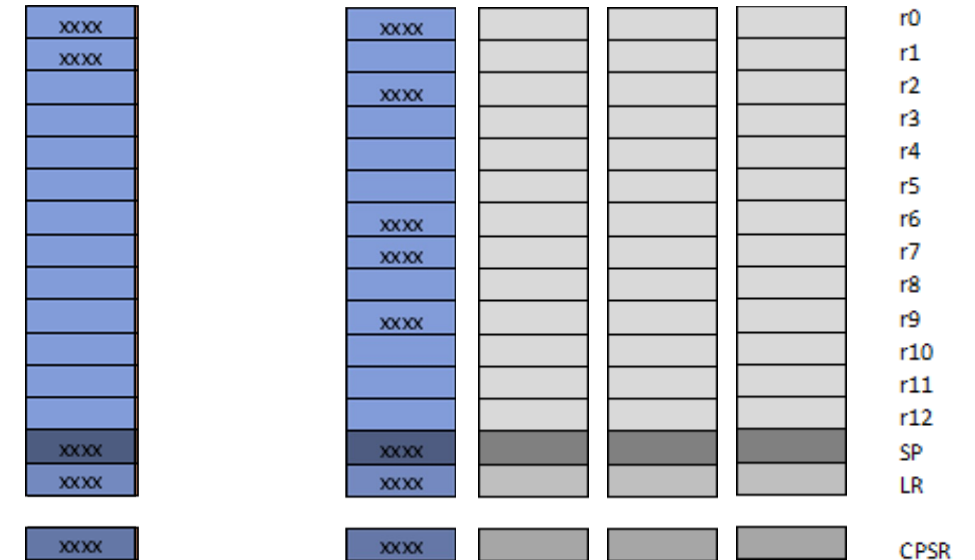
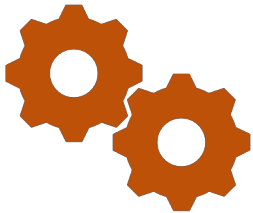
Il processore esegue la RSI, che rimuove la causa di IRQ



Banked Registers



Terminata la RSI, i registri vengono scambiati di nuovo, e il programma utente può tornare ad essere eseguito



Poiché una RSI può essere interrotta SOLO da IRQ più prioritari, ci basta aver un set di registri pari al numero di priorità

Identificazione

Via software

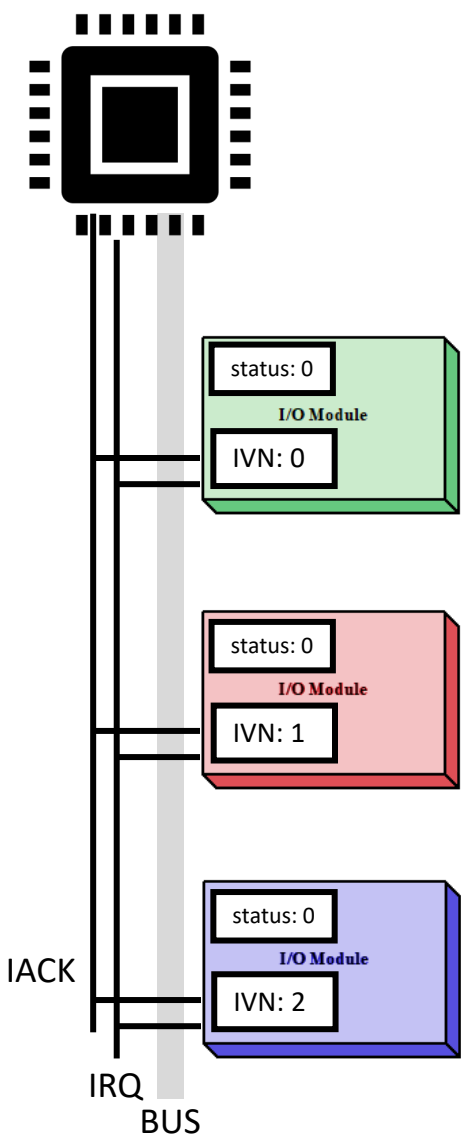
- Dobbiamo fare un poll software, controllando tutti i registri status dei device di I/O e poi fare un branch alla prima istruzione della RSI

```
ldr r0, =status_dev01
ldr r1, [r0]
cmp r1, #1
beq rsi_dev01
```

Via hardware

- Il processore, in risposta al segnale IRQ, risponde con un segnale IACK (Irq ACKnowledge)
- Il device invia sulla linea dati un suo codice identificativo
- Questo codice indica la riga da guardare in una tabella che contiene l'indirizzo di tutte le RSI

Interrupt Vettorizzati



Istruzioni della RSI del
dispositivo con codice IVN=0

Istruzioni della RSI del
dispositivo con codice IVN=1

Istruzioni della RSI del
dispositivo con codice IVN=2

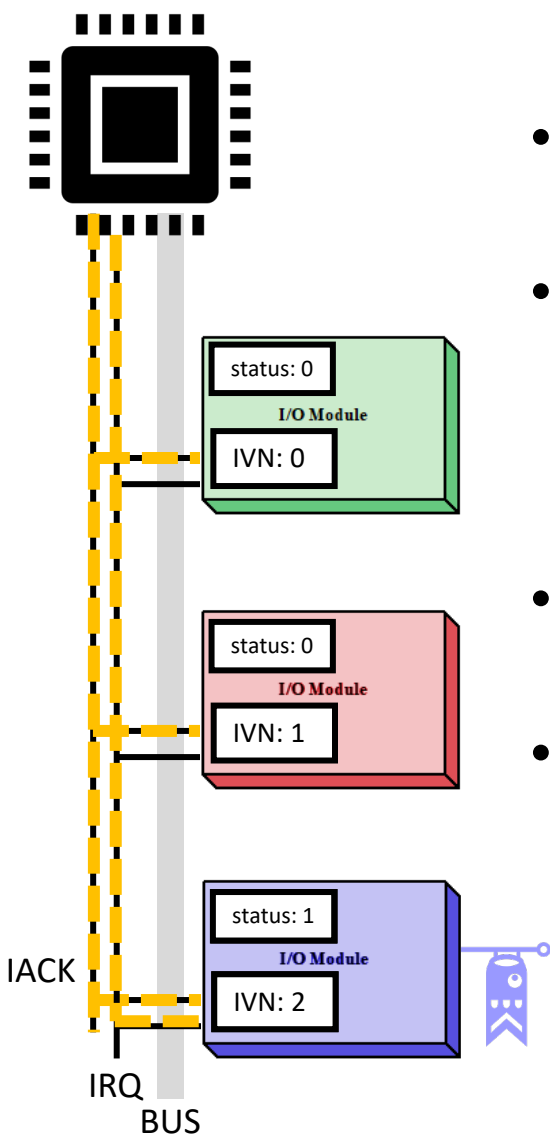
Interrupt
VectorTable

0x00
0x04
0x08
0x0C
0x10

Architettura degli Elaboratori

MEMORIA DELL'ELABORATORE (a 32 bit)

Interrupt Vettorizzati

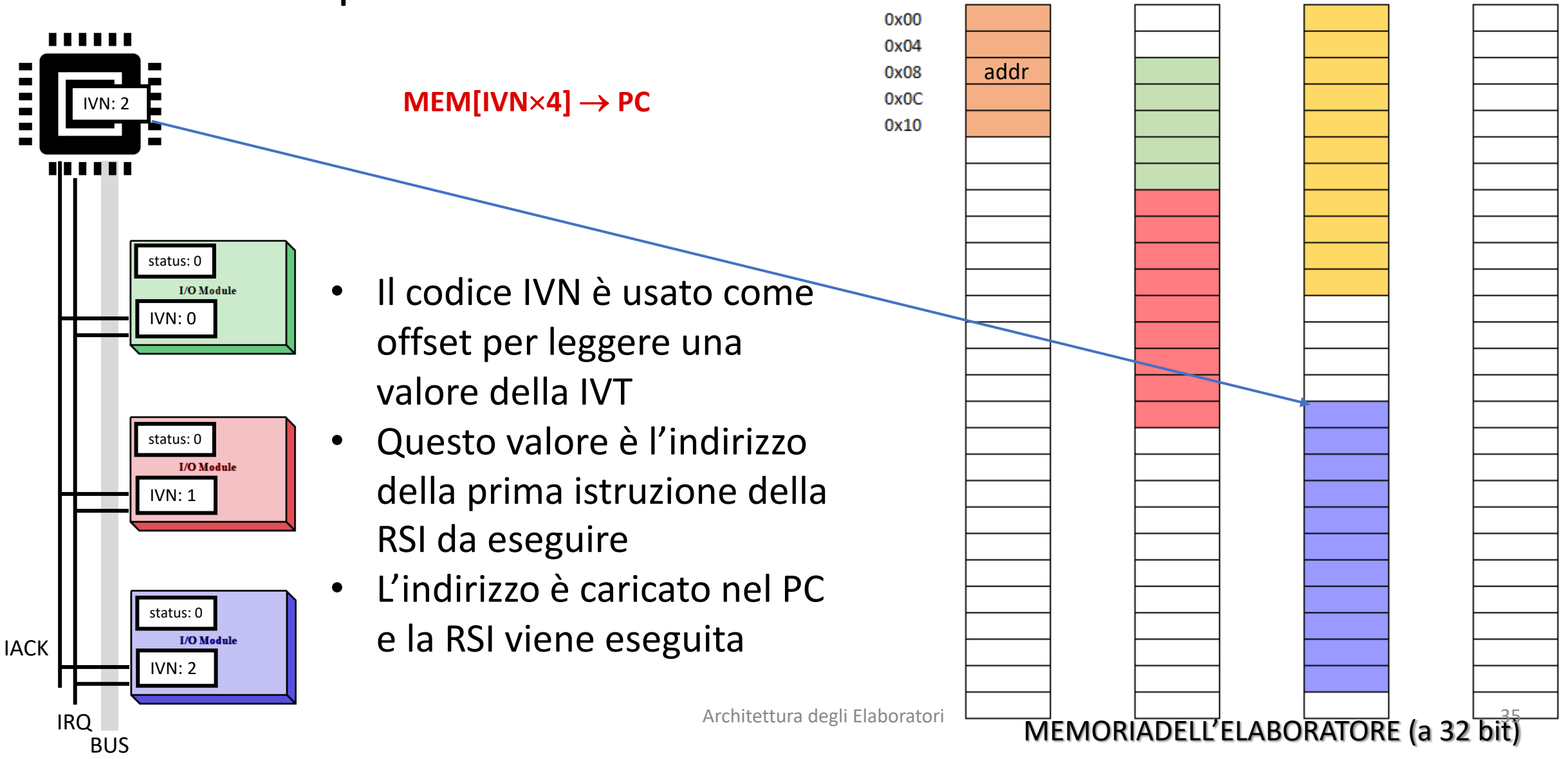


- Un device è pronto e genera un segnale IRQ
- Il processore quando è pronto a gestire l'interruzione genera un segnale IACQ
- Il segnale è inviato a tutti i dispositivi
- Solo il dispositivo READY risponde, inviando il suo codice sulla linea dati del bus

Architettura degli Elaboratori



Interrupt Vettorizzati



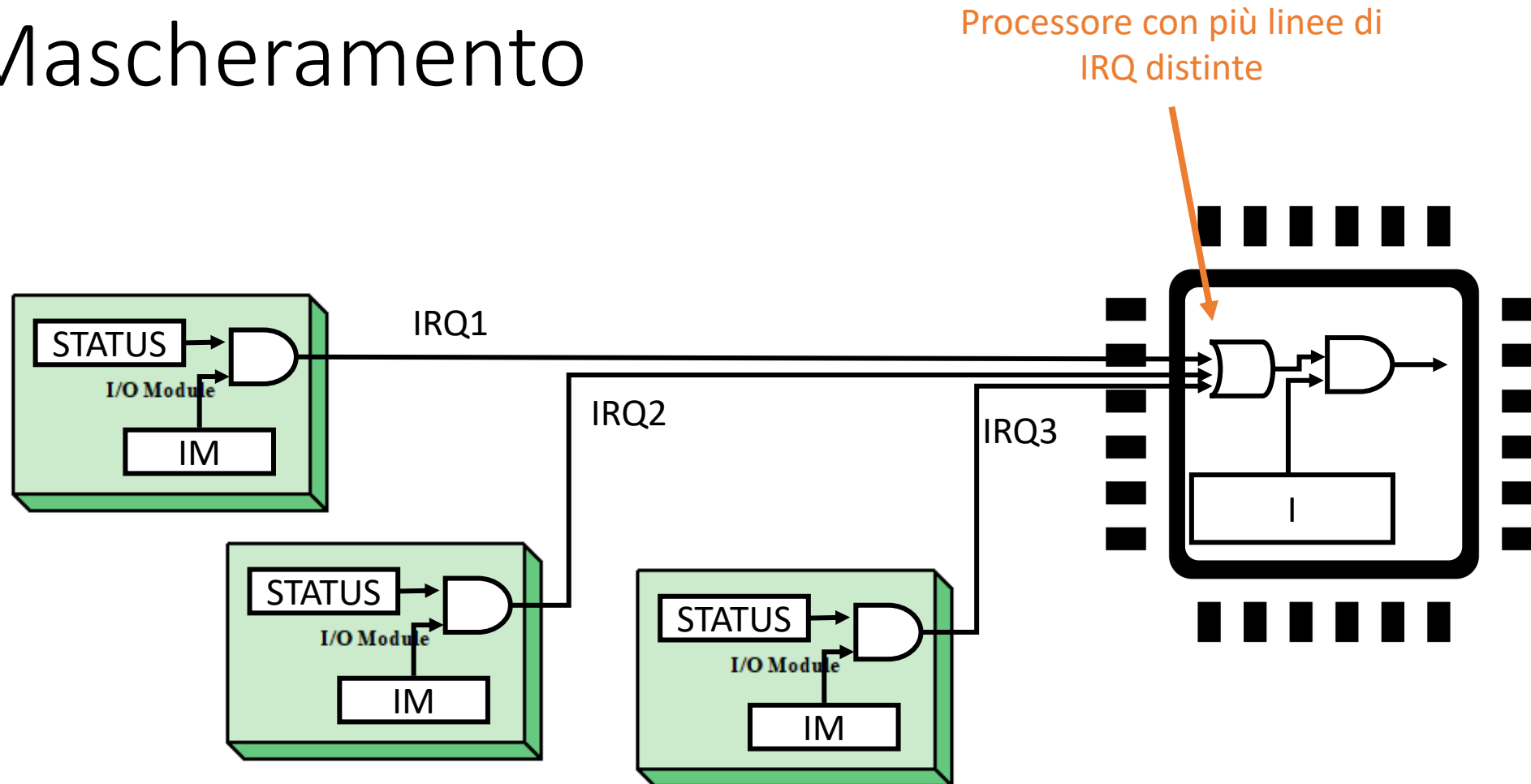
Gestione della priorità

- Un meccanismo di gestione della priorità è in grado, a fronte di più richieste di IRQ, di sceglierne una.
- E' possibile organizzare l'hardware in modo che in modo autonomo (senza eseguire istruzioni) scelga uno degli IRQ da servire e che mascheri interruzioni di priorità pari o inferiore mentre il processore sta eseguendo una RSI (questi IRQ rimarranno «pending» e verranno serviti dopo che la RSI è terminata)
- Solo gli IRQ con priorità maggiore di quella attualmente servita possono interromperla: quanto visto fin'ora permette di gestire senza problemi anche queste interruzioni di interruzioni (nested interrupt)

Mascheramento

- Se abbiamo più linee di IRQ che entrano nel processore, con delle semplici porte logiche possiamo fare in modo che un IRQ a priorità maggiore mascheri i segnali di IRQ di priorità minore
- Con un meccanismo analogo, possiamo anche bloccare l'invio del segnale IACK del processore sulle linee a priorità minore di quella attualmente da servire

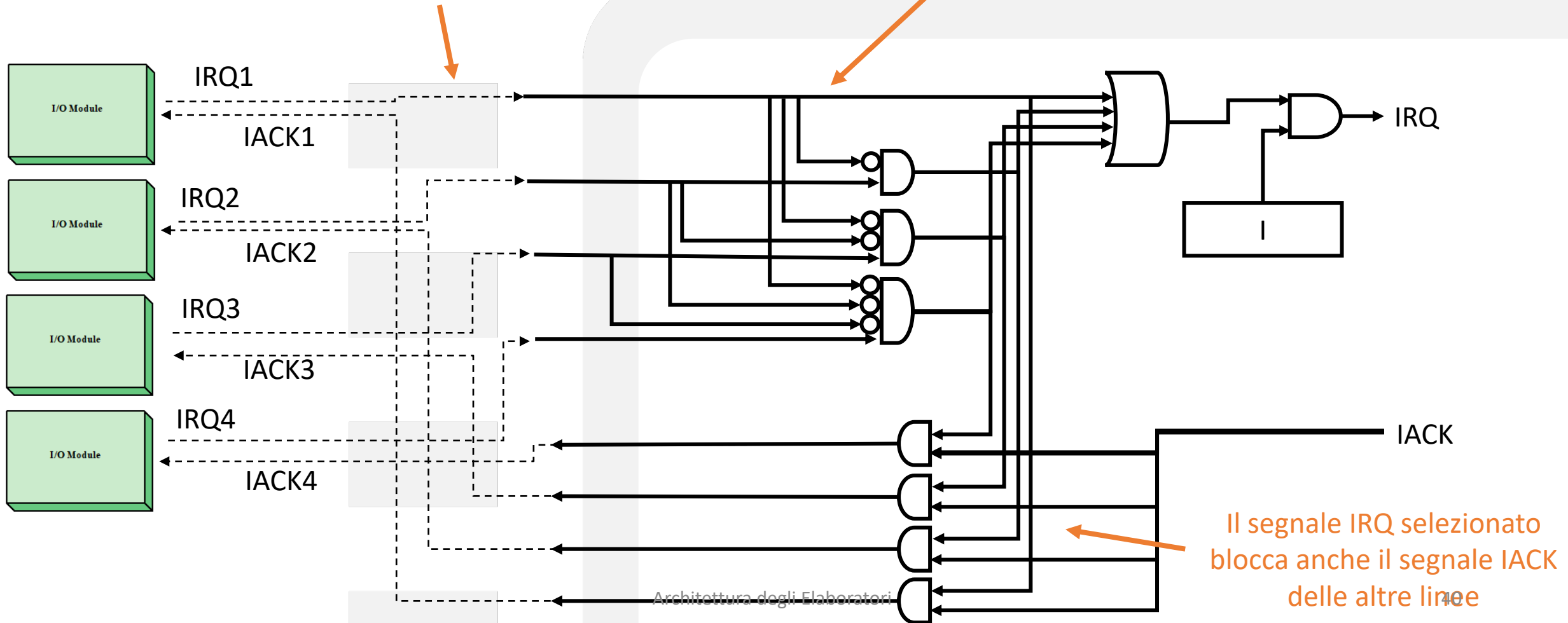
Mascheramento



Mascheramento

Ogni linea può essere un Wired-OR e gestire quindi più dispositivi

Il segnale IRQ di una linea blocca il segnale IRQ delle linee meno prioritarie

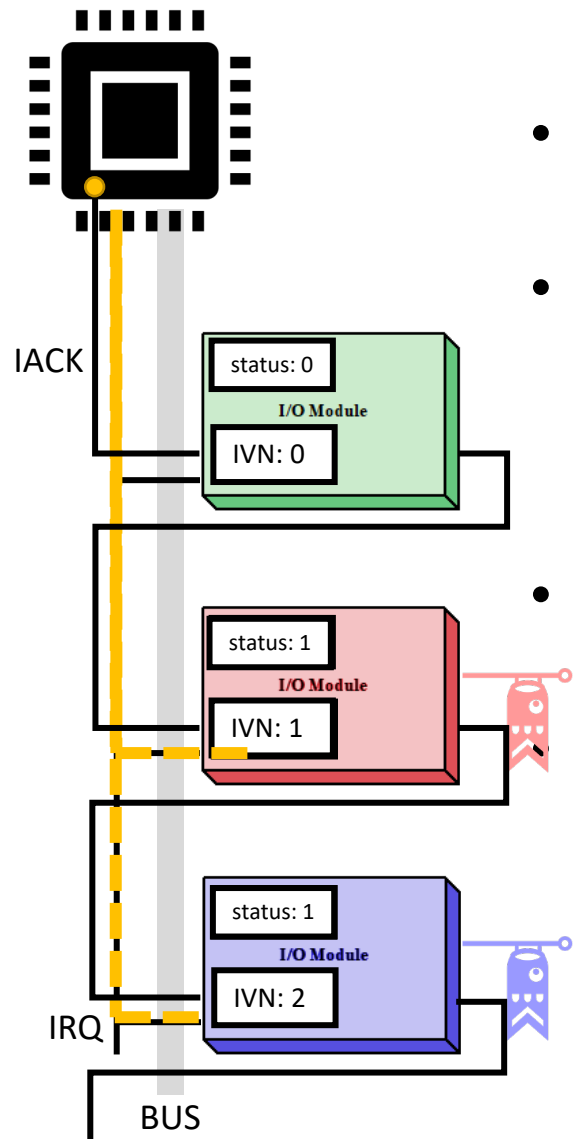


Daisy chain dei dispositivi

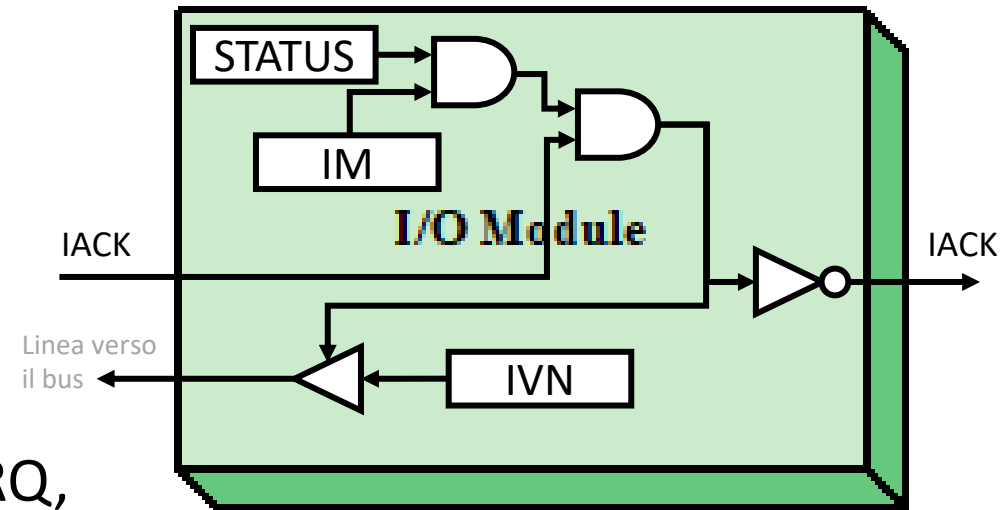
- I dispositivi sono collegati in «daisy chain»
- La linea del segnale IACK entra in ogni dispositivo e da questo prosegue verso quello successivo
- Il segnale di IACK viene generato dal processore e inviato lungo la linea: se il dispositivo non ha generato l'IRQ, passa il segnale; altrimenti ne blocca la propagazione (e invia al processore il suo IVN)
- Gli altri dispositivi restano in attesa (IRQ pending)
- La priorità è data dall'ordine con cui son collegati i dispositivi



Daisy Chain dei dispositivi



- Due device fanno richiesta di IRQ, inviando il segnale
- Quando il processore è pronto ad essere interrotto, invia il segnale di IACK, che passa attraverso un dispositivo alla volta
- Se il dispositivo non ha generato l'IRQ, passa il segnale al dispositivo dopo. Altrimenti il dispositivo ferma il segnale e risponde inviando l'IVN sulla linea dati



Tipi di interrupt

- Funziona così bene che viene usato non solo per gestire l'I/O

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

La gestione in ARM delle interruzioni

- ARM chiama le interruzioni «eccezioni»
- Gestisce 6 livelli di priorità, di cui 2 riservati agli eventi di I/O, distinti in un livello di IRQ e uno di FIQ (Fast Interrupt), che ha priorità maggiore
- La gestione delle interruzioni è resa efficace da:
 - Interrupt vettorizzati
 - Banked register
- Inoltre ha altri accorgimenti «furbi»:
 - Alcune istruzioni che occupano più di un ciclo possono essere spezzate (es STM, LDM) oppure interrotte e poi rifatte (es. DIV, UDIV)
 - La IVT non contiene indirizzi, ma una istruzione (tipicamente un branch); nel caso della FIQ, la riga è in realtà proprio la prima istruzione della RSI, che poi prosegue nelle righe successive alla IVT (così non è necessario fare il branch)
 - Se al termine di una RSI ha già un altro IRQ «pending», non fa il ripristino del contesto, ma lascia i dati nello stack e chiama la seconda RSI (tail-chaining)
 - Analogamente, se una IRQ prioritario arriva prima dell'inizio di una RSI, salta subito a questa senza salvare il contesto ma approfittando che l'aveva già fatto la RSI interrotta (late-arriving preemption)

Gestione delle eccezioni in ARM

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7_fiq	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

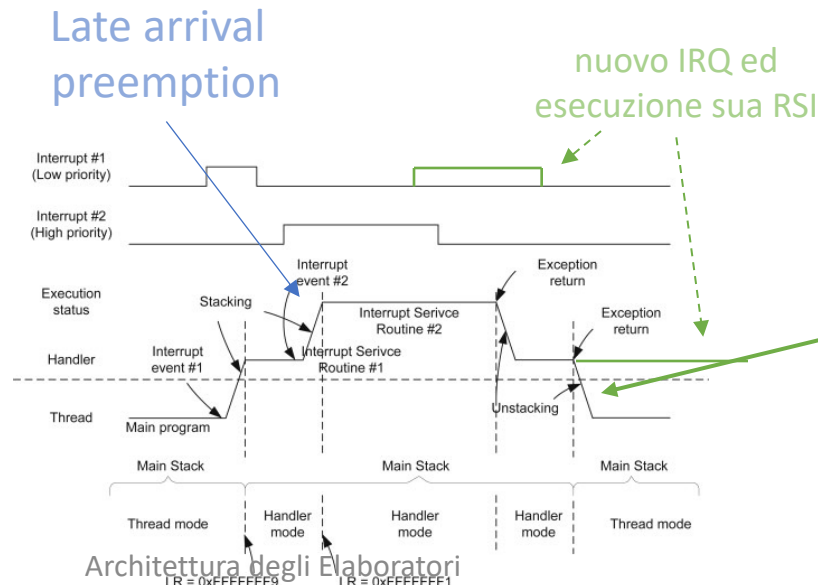
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Interruzioni su
eventi di I/O



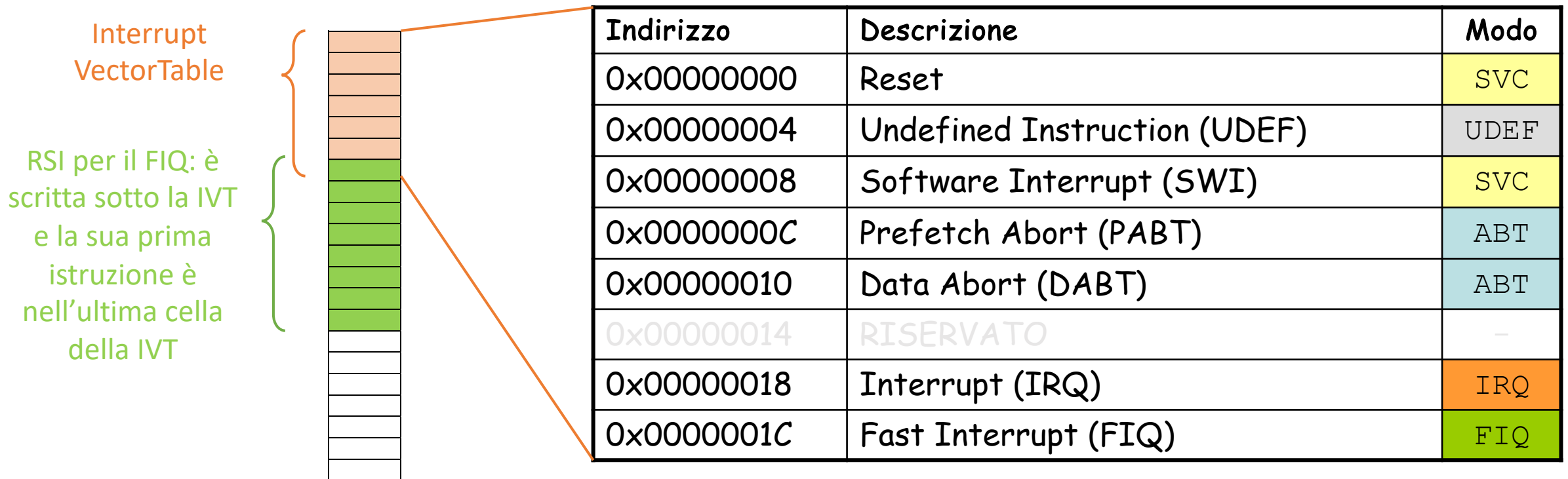
Eccezione	Priorità
Reset	1 (maggiore)
Data Abort	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined instruction	6 (minore)
SWI	6 (minore)

Banked Registers



Se sappiamo che c'è già un
altro IRQ da servire, qui
saltiamo l'unstacking e
passiamo alla esecuzione
della RSI (tail-chaining)

ARM: Interrupt Vector Table



Ricapitolando

- Processore chiede dati
- Device invia IRQ quando è pronto
- Processore (quando è interrompibile) sente IRQ e commuta il contesto
- In modalità supervisore vengono eseguite queste operazioni:
 1. Salvataggio del contesto
 2. Individuazione di chi ha generato IRQ
 3. Scelta del device con IRQ che ha maggiore priorità
 4. Recupero dell'indirizzo della sua RSI e sua esecuzione
 5. La RSI rimuove il segnale di IRQ su quel device
 6. Ripristino del contesto
- Ripresa dell'esecuzione del programma
- Nota: i passi da 1 a 4 si possono fare via SW o via HW (che è più veloce); durante l'esecuzione della RSI il processore è interrompibile da IRQ più prioritari

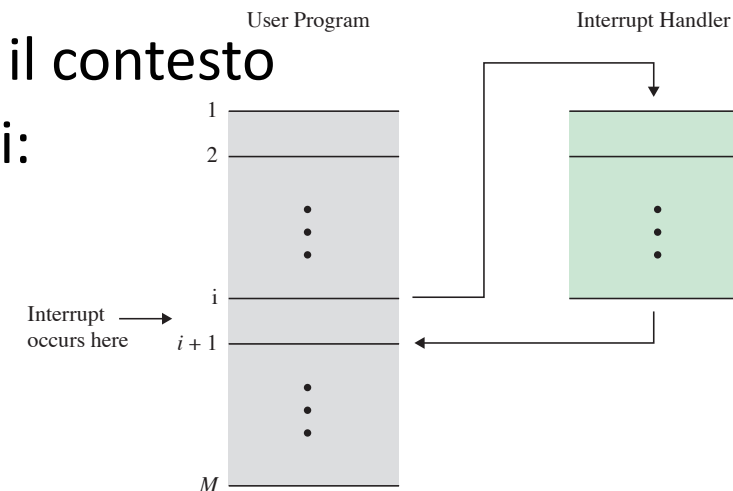


Figure 3.8 Transfer of Control via Interrupts

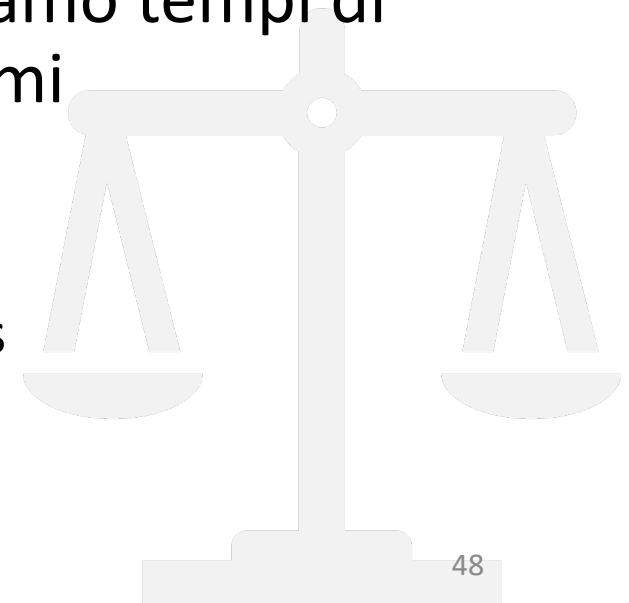
I/O a Interruzioni: pro e contro

CONTRO

- Sistema più complesso
- È necessario software dedicato
- È necessario hardware dedicato

PRO

- Efficiente: il processore non deve stare in busy waiting
- Usando HW nell'identificazione e gerarchia, abbiamo tempi di risposta rapidissimi
 - Hardware poll
 - Vectored IRQ
 - Banked registers



Ricapitolando

- Rispetto al I/O Programmato, a fronte di un sistema più complesso abbiamo una gestione degli eventi asincroni molto più efficiente
- Il processore non va in busy waiting
- Se usiamo dei componenti hardware, possiamo avere dei tempi di risposta a I/O paragonabili a quelli del busy waiting con dispositivo ready

*...la prossima volta vedremo come rendere
questo sistema ancora migliore, intanto
godiamoci le sue prestazioni!*