Lezione E10

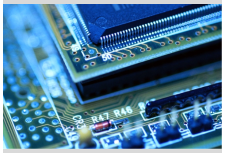
Il gestore delle interruzioni irqhandler.S

Sistemi operativi open-source, embedded e real-time

14 novembre 2017

Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata



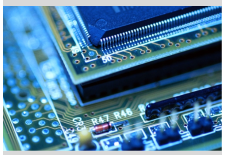
Di cosa parliamo in questa lezione?

In questa lezione analizziamo in dettaglio il programma contenuto nel file `irqhandler.S`, che realizza il salvataggio/recupero del contesto di esecuzione prima e dopo una interruzione

1. Considerazioni sullo scheduler non interrompibile
2. Meccanismo dello scheduler interrompibile
3. Analisi della funzione _irq_handler( )

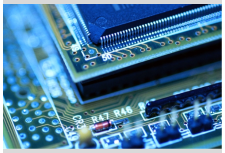
Il gestore delle interruzioni irqhandler.S

Marco Cesati



Esame dello scheduler non interrompibile

- Il ruolo della funzione `check_periodic_tasks()`:
  - Attivata periodicamente ad ogni tick di sistema
  - Rileva i rilasci periodici dei job
  - Aggiorna il campo `f->released` del task che contiene il numero di job pendenti (rilasciati e non ancora terminati)
- Il ruolo della funzione `run_periodic_tasks()`:
  - Cerca il task di priorità più alta avente job pendenti
  - Esegue il job prescelto in modo non interrompibile
  - Al termine del job ricomincia da capo

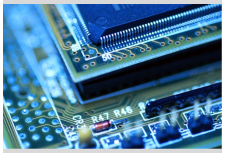


Esame dello scheduler non interrompibile (2)

Race condition: poiché `check_periodic_tasks()` è asincrona rispetto a `run_periodic_tasks()`, le modifiche ai campi `released` dei task possono avvenire in qualunque momento

- In particolare `run_periodic_tasks()` potrebbe scegliere erroneamente un job di priorità più bassa se un altro job viene rilasciato dopo che la funzione ha controllato il suo campo `released` nel ciclo `for`
- La variabile `globalreleases` serve a mitigare questa race condition: `run_periodic_tasks()` esegue un job solo se dopo una intera scansione del vettore di task nessun nuovo job viene rilasciato
- Esiste ancora una finestra temporale in cui è possibile avere una race condition (tra il controllo che `state == globalreleases` e l'esecuzione del job)
  - molto più piccola
  - tempi di blocco aggiuntivi trascurabili rispetto ai tempi di esecuzione dei job

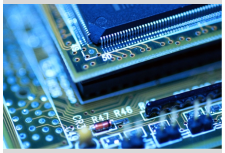
Il gestore delle interruzioni irqhandler.S
Marco Cesati



Schema della lezione
Scheduler non interrompibile
Scheduler interrompibile
Gestore dell'interruzione

Problematiche dello scheduler interrompibile

- Un job deve poter essere interrotto durante la sua esecuzione e sostituito con un altro job
- È necessario implementare il cambio di contesto, ossia
  - salvare le informazioni sullo stato di avanzamento del job interrotto
  - ripristinare le informazioni sullo stato di avanzamento del job che va in esecuzione
- Si deve gestire anche il caso particolare in cui un job viene rilasciato: è necessario creare le informazioni relative al suo primo cambio di contesto
- Si deve gestire anche il caso particolare in cui un job termina l'esecuzione: si deve sostituire il suo contesto con quello del miglior job eseguibile nel sistema



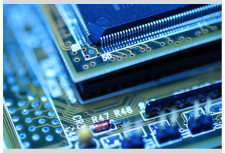
Problematiche dello scheduler interrompibile (2)

È necessario prevedere che il cambio di contesto possa avvenire

- in modalità IRQ

  (a seguito della gestione di una interruzione)
  - L'interruzione di un job è un evento che avviene in seguito ad una interruzione di tick, che avvia lo scheduler periodico, che riconosce che un job a priorità più alta è stato rilasciato
- in modalità SYSTEM

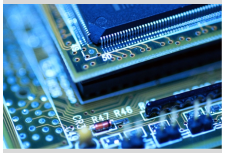
  (normale esecuzione del processore)
  - La terminazione di un job è un evento che avviene in modalità SYSTEM, ossia quando il processore *non* sta eseguendo il gestore di una interruzione



Problematiche dello scheduler interrompibile (3)

Per avere la massima flessibilità nel meccanismo di esecuzione dei job scegliamo di associare a ciascun task un proprio stack

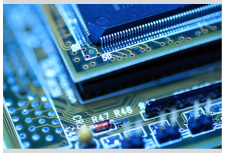
- Job dello stesso task non hanno bisogno di stack diversi
  - infatti job dello stesso task sono sempre eseguiti in modalità FIFO
- La procedura che effettua il cambio di contesto dovrà anche sostituire il puntatore contenuto nel registro `sp`
- Rimane valido il principio che non esistono stack specifici per la modalità d'esecuzione IRQ
  - il processore utilizza sempre e solo gli stack dei task utilizzati per la modalità SYSTEM (normale esecuzione)



Analisi del gestore d'interruzione di basso livello

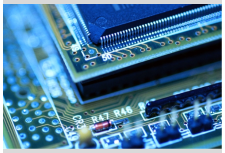
È necessario capire a fondo il funzionamento del gestore di interruzioni contenuto nel file `irqhandler.S`

- Per eseguire una procedura asincrona come il gestore di una interruzione si deve comunque salvare il "contesto d'esecuzione" della procedura interrotta
  - generalmente la procedura interrotta è il job di un task, ma può anche essere un altro gestore di interruzioni
- Si dovrà modificare il gestore di interruzioni per consentire, al termine di una interruzione, il ripristino di un job diverso da quello "corrente" al momento in cui l'interruzione si è verificata



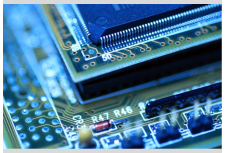
La funzione `_irq_handler()`: salvataggio del contesto

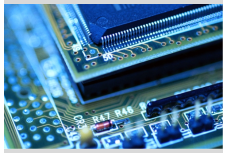
```
_irq_handler:                                          0
  mov     r13,r0                                       1
  sub     r0,lr,#4                                     2
  mov     lr,r1                                        3
  mrs     r1,spsr                                      4
  msr     cpsr_c,#(SYS_MODE|NO_IRQ)                    5
  stmfd   sp!,{r0,r1}                                  6
  stmfd   sp!,{r2-r3,r12,lr}                           7
  mov     r0,sp                                        8
  sub     sp,sp,#(2*4)                                 9
  msr     cpsr_c,#(IRQ_MODE|NO_IRQ)                    10
  stmfd   r0!,{r13,r14}                                11
  msr     cpsr_c,#(SYS_MODE|NO_IRQ)                    12
  ldr     r12,=_bsp_irq                                13
  mov     lr,pc                                        14
  bx      r12                                          15
```

La funzione `_irq_handler()`: recupero del contesto

```
  msr     cpsr_c,#(SYS_MODE|NO_INT)                    16
  mov     r0,sp                                        17
  add     sp,sp,#(8*4)                                 18
  msr     cpsr_c,#(IRQ_MODE|NO_INT)                    19
  mov     sp,r0                                        20
  ldr     r0,[sp,#(7*4)]                               21
  msr     spsr_cxsf,r0                                 22
  ldmfd   sp,{r0-r3,r12,lr}^                           23
  nop                                                  24
  ldr     lr,[sp,#(6*4)]                               25
  movs    pc,lr                                        26
```



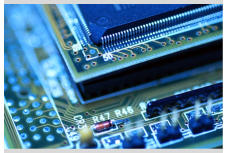
Stato prima e dopo l'interruzione

Prima dell'interruzione:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	(P)	Q	////
IRQ														?	?			?
STK	X	Y	Z															

Dopo l'interruzione:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														?	P_{+8}			Q
STK	X	Y	Z															



Istruzione 1: `mov r13,r0`

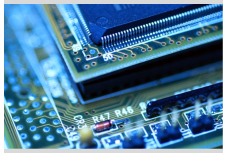
- L'interruzione si è appena verificata
- La CPU è in esecuzione in modalità IRQ
  - i registri `r13` (=`sp`), `r14` (=`lr`) e `spsr` sono differenti da quelli utilizzati in modalità SYSTEM
- Le interruzioni sono automaticamente disabilitate
- L'istruzione `mov r13,r0` salva il contenuto di `r0` in `r13`

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														A	P_{+8}			Q
STK	X	Y	Z															

Istruzione 2: `sub r0,lr,#4`

- Il registro `r14` (=`lr`) del modo IRQ contiene il valore del Program Counter al momento in cui l'interruzione è stata rilevata
- Sottrarre 4 da questo indirizzo ottiene l'indirizzo dell'istruzione seguente a quella appena terminata
  - Nell'architettura ARM il Program Counter è sempre pari all'indirizzo dell'istruzione che valuta `pc` più 8 byte
- L'indirizzo di ritorno dall'interruzione è scritto in `r0`

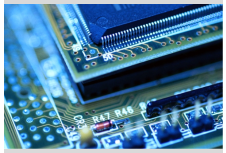
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	P+4	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														A	P+8			Q
STK	X	Y	Z															



Istruzione 3: `mov lr,r1`

- L'istruzione `mov lr,r1` salva il valore del registro `r1` nel registro `r14` (=`lr`) del modo IRQ

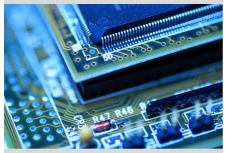
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	P+4	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														A	B			Q
STK	X	Y	Z															



Istruzione 4: `mrs r1,spsr`

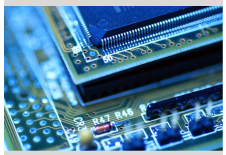
- L'istruzione `mrs r1,spsr` salva il valore del registro `spsr` del modo IRQ nel registro `r1`
  - Il registro `spsr` del modo IRQ contiene il valore del registro `cpsr` (Program Status Word) subito prima dell'interruzione

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	P+4	Q	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														A	B			Q
STK	X	Y	Z															



Istruzione 5: `msr cpsr_c,#(SYS_MODE|NO_IRQ)`

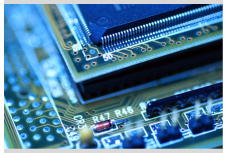
- L'istruzione modifica i bit del registro `cpsr` relativi al modo di esecuzione ed alla abilitazione degli IRQ e FIQ
- Si forza la CPU nel modo di esecuzione SYSTEM
  - I registri `r13` e `r14` tornano ad avere il valore "normale"
- Si continuano ad avere le IRQ disabilitate
- Le interruzioni veloci ad alta priorità FIQ sono abilitate
  - In questo progetto comunque non sono utilizzate



Istruzione 6: `stmfd sp!,{r0,r1}`

- L'istruzione effettua una operazione di "push" di un insieme di registri sullo stack
- Poiché è eseguita in modalità SYSTEM, lo stack è quello "corrente" al momento dell'interruzione
- Vengono salvati i valori che erano contenuti nei registri della modalità IRQ `spsr` e `pc`

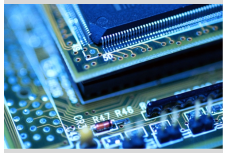
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	P+4	Q	C	D	E	F	G	H	I	J	K	L	M	N-8	O	#	*	////
IRQ														A	B			Q
STK	P+4	Q	X	Y	Z													



Istruzione 7: `stmfd sp!,{r2-r3,r12,lr}`

- Vengono salvati sullo stack il contenuto dei registri `r2`, `r3`, `r12` e `r14` (=`lr`)
- Questi sono tra i registri definiti *AAPCS-clobbered* che *non* devono essere preservati in una invocazione di funzione C
- Gli altri registri che *non* sono preservati in una invocazione di funzione C sono `r0` ed `r1`
  - per ora salvati nei registri `r13` e `r14` della modalita' IRQ

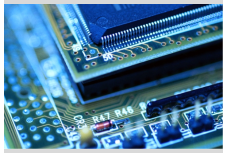
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	P+4	Q	C	D	E	F	G	H	I	J	K	L	M	N-24	O	#	*	////
IRQ														A	B			Q
STK	C	D	M	O	P+4	Q	X	Y	Z									



Istruzione 8: `mov r0,sp`

- Questa istruzione prepara il salvataggio del contenuto dei registri `r13` e `r14` della modalità IRQ sullo stack della modalità SYSTEM
  - Tali registri contengono i valori dei registri `r0` e `r1` al momento dell'interruzione
- L'istruzione copia `sp` della modalità SYSTEM in `r0` (il cui valore non è alterato cambiando modalità)

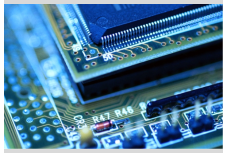
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-24	Q	C	D	E	F	G	H	I	J	K	L	M	N-24	O	#	*	////
IRQ														A	B			Q
STK	(C)	D	M	O	P+4	Q	X	Y	Z									



Istruzione 9: `sub sp,sp,#(2*4)`

- Estende lo stack della modalità SYSTEM di due posizioni

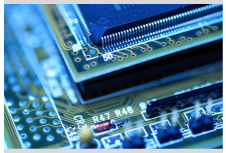
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-24	Q	C	D	E	F	G	H	I	J	K	L	M	N-32	O	#	*	////
IRQ														A	B			Q
STK	(?)	?	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 10: `msr cpsr_c,#(IRQ_MODE|NO_IRQ)`

- Cambia la modalità del processore ad IRQ
- I registri `r13` e `r14` ora contengono i valori di `r0` e `r1` al momento dell'interruzione

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-24	Q	C	D	E	F	G	H	I	J	K	L	M	N-32	O	#	*	////
IRQ														A	B			Q
STK	?	?	C	D	M	O	P+4	Q	X	Y	Z							



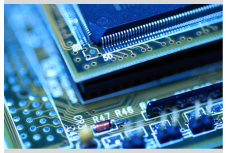
Istruzione 11: `stmfd r0!,{r13,r14}`

- Salva sullo stack il contenuto di `r13` e `r14`
- Questa "push" completa il salvataggio dei registri *AAPCS-clobbered* che non devono essere preservati in una invocazione di funzione C
- Sarà il gestore delle interruzioni ad alto livello, scritto in C, a salvare i registri `r4`–`r11` sullo stack in caso di loro utilizzo

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	Q	C	D	E	F	G	H	I	J	K	L	M	N-32	O	#	*	////
IRQ														A	B			Q
STK	A	B	C	D	M	O	P+4	Q	X	Y	Z							

Il gestore delle interruzioni irqhandler.S

Marco Cesati

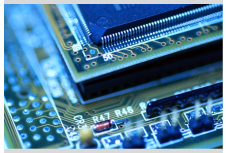


Schema della lezione
Scheduler non interrompibile
Scheduler interrompibile
Gestore dell'interruzione

Istruzione 12: `msr cpsr_c, #(SYS_MODE|NO_IRQ)`

- Torna in modalità SYSTEM
- Le interruzioni sono ancora disabilitate

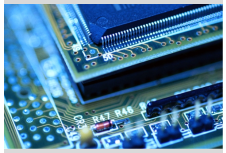
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	Q	C	D	E	F	G	H	I	J	K	L	M	N-32	O	#	*	////
IRQ														A	B			Q
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 13: `ldr r12,=_bsp_irq`

- Carica nel registro `r12` l'indirizzo Ω della funzione C `_bsp_irq()`
- Questa funzione è il gestore dell'interruzione di "medio livello", comune a tutte le interruzioni

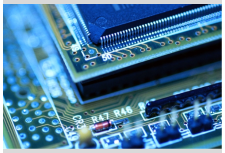
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	Q	C	D	E	F	G	H	I	J	K	L	Ω	N-32	O	#	*	////
IRQ														A	B			Q
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 14: `mov lr,pc`

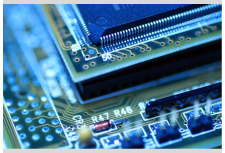
- Carica l'indirizzo di ritorno da `_bsp_irq()` nel link register `r14` (=`lr`)
- La CPU è in modalità SYSTEM, ma `r14` è tra i registri salvati sullo stack

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	Q	C	D	E	F	G	H	I	J	K	L	Ω	N-32	#	#	*	////
IRQ														A	B			Q
STK	Ⓐ	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 15: `bx r12`

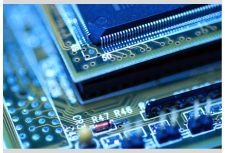
- Salta alla funzione `_bsp_irq()`
- La forma "`x`" dell'istruzione *branch* consente di saltare anche a procedure eventualmente compilate in formato compatto Thumb
- La funzione `_bsp_irq()`:
  - attiva il meccanismo di priorità delle interruzioni
  - legge l'indirizzo del gestore ad alto livello della interruzione
  - esegue il gestore ad alto livello della interruzione
  - abbassa il livello di priorità al valore precedente
  - termina l'esecuzione



Istruzione 16: `msr cpsr_c, #(SYS_MODE|NO_INT)`

- Questo è il punto di ritorno da `_bsp_irq()`
- Forza la modalità SYSTEM
- Disabilita sia gli IRQ che i FIQ

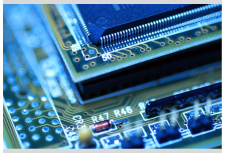
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	?	?	?	?	E	F	G	H	I	J	K	L	?	N-32	?	#	*	////
IRQ														?	?			?
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 17: `mov r0,sp`

- È necessario ripristinare il contenuto dei registri originale compiendo le operazioni fatte all'inizio in ordine inverso
- L'istruzione copia il puntatore alla cima dello stack in `r0`

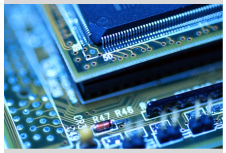
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	?	?	?	E	F	G	H	I	J	K	L	?	N-32	?	#	*	////
IRQ														?	?			?
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 18: `add sp,sp,#(8*4)`

- Aggiorna `sp` per togliere 8 elementi dallo stack
- In realtà il contenuto dello stack non è modificato

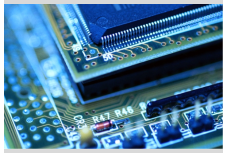
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	////
IRQ														?	?			?
STK	(A)	B	C	D	M	O	P+4	Q	(X)	Y	Z							



Istruzione 19: `msr cpsr_c, #(IRQ_MODE|NO_INT)`

- Forza la CPU in modalità IRQ
- IRQ e FIQ sono ancora disabilitati

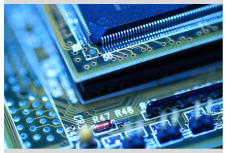
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	////
IRQ														?	?			?
STK	A	B	C	D	M	O	P+4	Q	(X)	Y	Z							



Istruzione 20: `mov sp,r0`

- Copia il puntatore alla cima dello stack della modalità SYSTEM nel registro `sp` relativo alla modalità IRQ

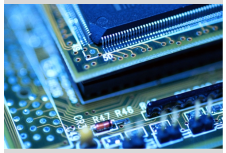
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	N-32	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	////
IRQ														N-32	?			?
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 21: `ldr r0,[sp,#(7*4)]`

- Carica in `r0` il valore che era stato salvato per il registro `spsr`

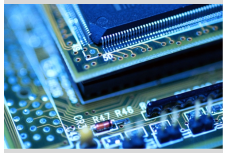
	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	Q	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	////
IRQ														N-32	?			?
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z							



Istruzione 22: `msr spsr_cxsf,r0`

- Ripristina il valore del registro `spsr` della modalità IRQ

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	Q	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	////
IRQ														N-32	?			Q
STK	A	B	C	D	M	O	P+4	Q	X	Y	Z							



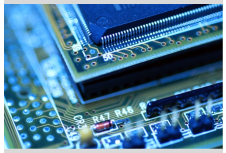
Istruzione 23: `ldmfd sp,{r0-r3,r12,lr}^`

- Compie l'operazione di "pop" per tutti i valori salvati sullo stack
- Il simbolo "^" alla fine dell'istruzione che il valore corrispondente a `r14` verrà ripristinato nel registro `r14` della modalità SYSTEM invece che in quello della modalità corrente (IRQ)
  - L'interpretazione del simbolo "^" sarebbe stata differente se il registro `pc` fosse stato incluso nell'elenco dei registri dell'istruzione

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														N-32	?			Q
STK	A	B	C	D	M	O	P+4	Q	X	Y	Z							

Il gestore delle interruzioni irqhandler.S

Marco Cesati



Schema della lezione

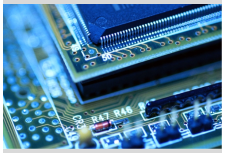
Scheduler non interrompibile

Scheduler interrompibile

Gestore dell'interruzione

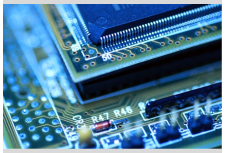
Istruzione 24: `nop`

- In alcune versioni dell'architettura ARM non è possibile accedere subito dopo l'operazione precedente ad un registro "privato"
- Il modo più semplice per evitare problemi è inserire una istruzione che non fa nulla (`nop`)



Istruzione 25: `ldr lr,[sp,#(6*4)]`

- Preleva dallo stack l'indirizzo di ritorno dal gestore delle interruzioni
- Ripristina l'indirizzo di ritorno nel registro `r14` (=`lr`) della modalità IRQ



	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	#	*	////
IRQ														N-32	P+4			Q
STK	A	B	C	D	M	O	P+4	Q	X	Y	Z							

Istruzione 26: `movs pc,lr`

- Salta all'indirizzo di ritorno in `lr`
- La "`s`" finale nell'istruzione significa che l'istruzione copierà anche il contenuto di `spsr` nel registro `cpsr`
- Di conseguenza la modalità di esecuzione della CPU tornerà ad essere quella corrente all'occorrenza dell'interruzione
- La gestione dell'interruzione è effettivamente conclusa

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp r13	lr r14	pc r15	cpsr	spsr
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P_{+4}	Q	////
IRQ														N_{-32}	P_{+4}			Q
STK	X	Y	Z