Lezione E3

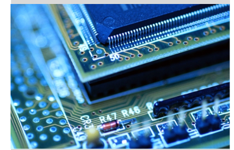
Architettura ARM

Sistemi operativi open-source, embedded e real-time

3 ottobre 2017

Marco Cesati

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

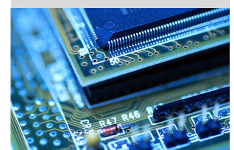


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Di cosa parliamo in questa lezione?

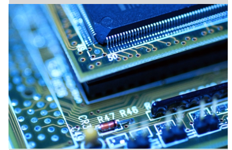
In questa lezione si fornisce una descrizione generale dei microprocessori ARM e della relativa architettura

1. Evoluzione degli ARM
2. L'architettura ARM
3. Istruzioni load e store
4. Modi di indirizzamento



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Utilizzo di microprocessori nei sistemi embedded



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

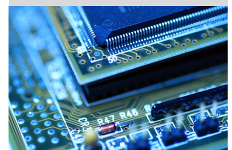
- Spesso i sistemi embedded di fascia alta utilizzano come elementi di calcolo microprocessori sofisticati
- Casi in cui l'uso di un microprocessore è giustificato:
  - sono richieste elevate capacità di calcolo
    - ad es., processamento di segnali audio e video
  - si utilizzano protocolli di comunicazione complessi
    - ad es., rete wireless IEEE 802.11
  - è richiesta memoria di grande capacità
    - ad es., mappe di un navigatore GPS

Cosa hanno in comune gli smartphone basati su iOS (Apple), Symbian (Nokia), Android (Google) e Windows CE (Microsoft)?

Utilizzano tutti lo stesso tipo di microprocessori: ARM

Origine dei microprocessori ARM

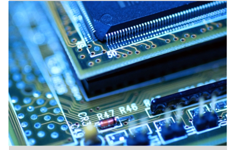
Architettura ARM
Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

- Negli anni '80 le società inglesi Acorn e British Broadcasting Corporation (BBC) crearono un nuovo calcolatore personale chiamato BBC Micro
- Grazie a questo accordo, Acorn ebbe le risorse per sviluppare la serie di calcolatori Acorn Archimedes
  - basati su un nuovo microprocessore RISC
  - commercializzati tra il 1987 ed il 1997
  - non ebbero grande successo

Fonte: www.old-computers.com



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Origine dei microprocessori ARM (2)

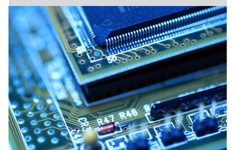
I calcolatori Archimedes utilizzavano il primo microprocessore RISC commerciale: ARM (Acorn Risc Machine)

- Commercializzato nel 1985
- Al contrario dei calcolatori Archimedes, ebbe subito un immediato successo
- Seguirono l'ARM2 (1985) e l'ARM3 (1989)
- L'Acorn non produceva fisicamente i chip
- La VLSI Technology aveva licenza per fabbricare e commercializzare i chip basati sulle specifiche di Acorn

Fonte: www.cpushack.com

Architettura ARM

Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

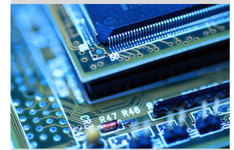
Evoluzione dei microprocessori ARM

- Il grande successo commerciale degli ARM imponeva di continuare a sviluppare ed estendere l'architettura, ma l'Acorn non aveva abbastanza risorse per farlo
- Nel 1990 venne creata la società Advanced RISC Machines Ltd. a cui partecipavano Acorn, VLSI Technologies e Apple Computer
- Il microprocessore fu ribattezzato Advanced Risc Machine
- Il primo prodotto fu il microprocessore ARM6 (1992), una versione migliorata dell'ARM3
- Da allora l'architettura ARM ha continuato ad evolversi con nuove funzionalità e migliori prestazioni

Diffusione dei microprocessori ARM

Nel mondo embedded gli ARM hanno un successo enorme e sempre crescente:

- 2005: Nel solo anno sono stati licenziati 1,6 miliardi di chip (di cui 1 miliardo nei telefoni cellulari)
- 2005: Il 98% di tutti i cellulari utilizza una CPU ARM
- 2008: a gennaio raggiunta la soglia di 10 miliardi di chip
- 2009: si stima che siano il 90% di tutte le CPU RISC a 32 bit
- 2011: a gennaio raggiunta la soglia di 15 miliardi di chip
- 2014: ha il 95% del mercato embedded e mobile
- 2017: raggiunta la soglia dei 100 miliardi di chip prodotti

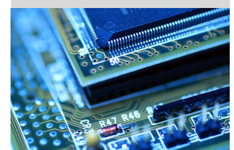


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

ARM Intellectual Property

- Uno dei componenti essenziali del successo di ARM consiste nel meccanismo di licenza
- La ARM Ltd. non produce chip ma vende licenze per l'utilizzo dell'IP (Intellectual Property) relativo al progetto di un microprocessore ARM
- La società licenziataria può
  - modificare il progetto per adattarlo ai propri scopi
  - produrre il chip microprocessore
  - integrare il microprocessore all'interno di un proprio sistema (SoC, System on Chip)
  - vendere il progetto modificato ad altre società
- Grazie a questo meccanismo i microprocessori ARM
  - si evolvono rapidamente
  - trovano impiego in molti ambiti e applicazioni

Architettura ARM
Marco Cesati

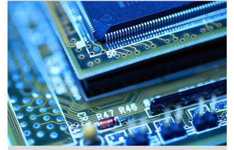


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Le famiglie di microprocessori ARM

Esistono molte famiglie di microprocessori ARM, ad esempio:

- Sviluppate da ARM Ltd.:
  - ARM7 (1994)
  - ARM8 (1996)
  - ARM9 (1997)
  - ARM10 (2000)
  - ARM11 (2002)
  - SecurCore (2003)
  - Cortex-M (2004)
  - Cortex-A (2005)
  - Cortex-R (2011)
  - Cortex-A 64-bit (2012)

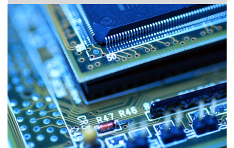


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Le famiglie di microprocessori ARM (2)

- Sviluppate da altre società licenziatarie:
  - StrongARM (1995): DEC, poi Intel
  - i.MX (2001): Freescale (ex Motorola)
  - XScale (2002): Intel e Marvell
  - OMAP SoC (2007): Texas Instruments
  - Tegra (2008), Denver (2014): NVIDIA
  - Snapdragon (2008): Qualcomm
  - Hummingbird (2009): Samsung
  - A4 (2010), A5 (2011), A5X, A6, A6X (2012), A7 (2013), A8 (2014): Apple
  - Nova e NovaThor (2011): ST-Ericsson
  - K12 (2016): AMD

Architettura ARM
Marco Cesati



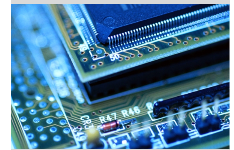
Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Le principali famiglie di microprocessori ARM a 32 bit

Famiglia	Novità	Cache (KB)	MIPS @ MHz
ARM1	pipeline 3 stadi	–	?
ARM2	MMU, GPU, I/O	–	7 @ 12
ARM3	cache	4	12 @ 25
ARM6	indirizzi 32 bit, FPU	4	28 @ 33
ARM7	integrato in SoC	8	60 @ 60
ARM8	pipeline 5 st., pred. salti	8	84 @ 72
ARM9	architettura Harvard	16+16	300 @ 300
ARM9E	istr. DSP migliorate	16+16	220 @ 200
ARM10E	pipeline 6 st.	32+32	500 @ 400
ARM11	pipeline 9 st.	variabile	740 @ 665
Cortex-A	pipeline supersc. 13 st.	variabile	2000 @ 1000
XScale	pipeline 7 st.	L1: 32+32 L2: 512	1000 @ 1250

Fonte: W. Stallings, Architettura ed organizzazione dei calcolatori, 8 ed., Pearson, 2010

Le caratteristiche possono variare a seconda del modello nella stessa famiglia



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

La famiglia Cortex

La famiglia Cortex contiene modelli ARM adatti agli usi più diversi

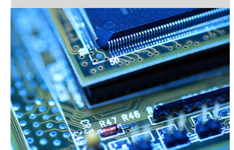
Sono stati definiti tre diversi profili applicativi:

- Cortex-A: profilo “Applications” per sistemi di uso generale
  - ad es., smartphone, TV digitali
- Cortex-R: profilo “Real-time” per sistemi real-time
  - ad es., impianti frenanti, dischi rigidi, switch di rete
- Cortex-M: profilo “Microcontroller” per sistemi embedded
  - ad es., sensori intelligenti, calcolatrici, pacemaker

Le famiglie di microprocessori ARM hanno essenzialmente lo stesso insieme di base di istruzioni macchina

Tuttavia programmi compilati specificatamente per un certo microprocessore non funzionano necessariamente con un microprocessore di un'altra famiglia

Architettura ARM
Marco Cesati

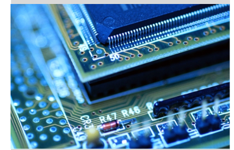


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Le versioni dell'architettura ARM

ISA	Famiglia
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM8, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale
ARMv6	ARM11, Cortex-M
ARMv7	Cortex-A, Cortex-M, Cortex-R
ARMv8	Cortex-A5x, A8, Denver (core a 64 bit)

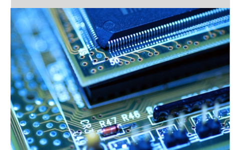
- ARMxTx: insieme alternativo di istruzioni Thumb a 16 bit
- ARMxDx: supporto per debug via JTAG
- ARMxMx: unità moltiplicazione più efficiente
- ARMxIx: supporto per debug con *EmbeddedICE*
- ARMxEx: supporto per DSP e multimedia (implica TDMI)
- ARMxJx: supporto per Java bytecode nativo (Jazelle)



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Caratteristiche principali dei microprocessori ARM a 32 bit

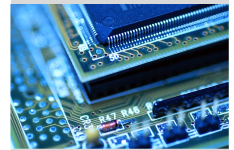
- Architettura a 32 bit
  - Dimensione dei registri e dei dati su cui operano le istruzioni
- Essenzialmente architettura RISC
  - Lunghezza fissa delle istruzioni macchina: 32 bit
  - Operazioni compiute sui registri, non in memoria
  - Istruzioni "load" e "store" per accedere alla memoria
- Memoria indirizzabile al singolo byte, con indirizzi da 32 bit
- Accesso alla memoria "allineato"
  - I dati in memoria sono lunghi 8 bit, 16 bit o 32 bit
  - L'indirizzo di un dato a 16 bit deve essere multiplo di 2
  - L'indirizzo di un dato a 32 bit deve essere multiplo di 4
- Memorizzazione di tipo "little-endian" o "big-endian"
  - A seconda dello stato di una linea di input del chip



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Caratteristiche notevoli dei microprocessori ARM a 32 bit

- Esecuzione condizionale delle istruzioni
  - Molte istruzioni macchina possono essere eseguite o meno a seconda del valore dei bit di stato
  - In molte altre architetture solo i salti sono condizionali
- Assenza di istruzioni esplicite per lo scorrimento dei bit
  - Tuttavia nelle istruzioni logiche, aritmetiche e copia è possibile eseguire lo scorrimento dei bit di un operando
- Supporto per diverse operazioni di moltiplicazione
  - Varianti ottimizzate per l'elaborazione dei segnali
- Può essere assente una istruzione macchina per la divisione
  - Deve essere realizzata da una procedura software ovvero affidata ad un coprocessore
- Schemi di indirizzamento tipici di una architettura CISC
  - Auto-incremento e auto-decremento degli indirizzi
  - Indirizzamento relativo al *program counter*
  - Una singola istruzione può trasferire dati tra un blocco di memoria ed un insieme di registri



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

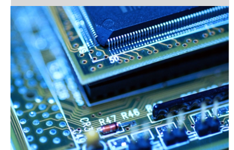
Il banco dei registri (ISA A32)

- L'ISA A32 di un ARM definisce 31 registri a 32 bit utilizzabili in ogni istruzione macchina
- Ad ogni istante sono visibili solo 16 registri chiamati

  `r0`, **`r1`**, **`r2`**, **`r3`**, **`r4`**, **`r5`**, **`r6`**, **`r7`**, . . . , **`r13`**, **`r14`**, **`r15`**
- **`r10`** (**`sl`**) talvolta contiene la dimensione dello stack
- **`r11`** (**`fp`**) è utilizzato spesso come *frame pointer*
- **`r12`** (**`ip`**) può essere utilizzato per l'invocazione di procedure
- **`r13`** (**`sp`**) è utilizzato spesso come *stack pointer*
- **`r14`** (**`lr`**) è il *link register*: può contenere l'indirizzo di ritorno di una funzione
- **`r15`** (**`pc`**) è il *program counter*: memorizza la posizione nel programma in esecuzione
  - **`r15`** contiene l'indirizzo della 2ª istruzione sotto a quella in esecuzione

Architettura ARM

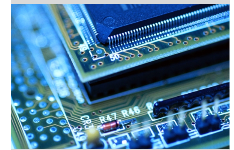
Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

I registri di stato

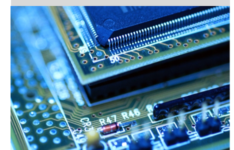
- Il registro di stato principale è chiamato **`cpsr`** (Current Program Status Register)
- In **`cpsr`** sono contenuti quattro bit di condizione:
  - N risultato negativo
  - Z risultato nullo (zero)
  - C si è verificato un riporto (carry)
  - V si è verificato un trabocco (overflow)
- In **`cpsr`** sono contenuti anche alcuni bit di controllo:
  - due bit per disabilitare le interruzioni
  - cinque bit che codificano il modo corrente del processore
  - due bit T e J che specificano la ISA utilizzata (ARM, Thumb, Jazelle)
- Ulteriori bit di condizione e di controllo sono definiti nelle architetture da ARMv5 in poi
- Esistono anche cinque registri **`spsr`** (Saved Program Status Register) utilizzati per preservare il valore in **`cpsr`** al verificarsi di una eccezione



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Architettura ARM a 64 bit (Aarch64)

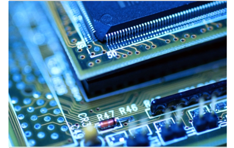
- Utilizza 31 registri general-purpose a 64 bit
- Registri **`pc`** e **`sp`** non più general-purpose
- Nuovo registro con valore 'zero'
- Codice operativo delle istruzioni di 32 bit, e per lo più identico alla ISA A32
- Operandi possono essere in genere da 32 o 64 bit, a scelta
- Indirizzi da 64 bit (attualmente 48 bit utilizzabili)
- Rimuove istruzioni load/store di lunghezza arbitraria
- La maggior parte delle istruzioni non sono più condizionali (solo salti e confronti)
- Nuova gestione delle eccezioni, con meno registri privati e meno modalità



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Tipi di istruzioni

- Istruzioni “load” e “store”
- Istruzioni aritmetiche
- Istruzioni di copia
- Istruzioni logiche
- Istruzioni di confronto
- Istruzioni di salto



Schema della lezione
Evoluzione degli ARM
L’architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni “load” e “store”

Poiché le architetture ARM sono RISC, due soli tipi di istruzione trasferiscono dati da e verso la memoria:

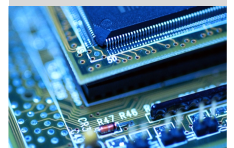
- L’istruzione di “load” (lettura dalla memoria) è **`ldr`**
- L’istruzione di “store” (scrittura in memoria) è **`str`**

È possibile indicare l’indirizzo della cella di memoria come parametro immediato (costante) di queste istruzioni?

Gli indirizzi sono di 32 bit, e le istruzioni hanno un formato fisso di 32 bit: in generale non è possibile!

L’indirizzo deve essere contenuto o derivabile dal contenuto di un registro generale

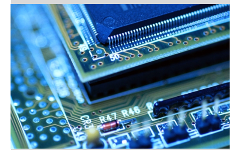
Ad esempio: **`str r2,[r4]`** scrive in memoria il valore a 32 bit contenuto in **`r2`** iniziando dall’indirizzo contenuto in **`r4`**



Schema della lezione
Evoluzione degli ARM
L’architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Trasferimento di interi a 8, 16 e 32 bit

- Le istruzioni `ldr` e `str` trasferiscono interi a 32 bit da/verso locazioni di memoria allineate a multipli di 4
- Le istruzioni `ldrb`, `ldrsb` e `strb` trasferiscono interi a 8 bit da/verso un singolo byte di memoria
  - `ldrb` memorizza il valore letto dalla memoria in un registro a 32 bit estendendolo con zeri
  - `ldrsb` memorizza il valore letto dalla memoria in un registro a 32 bit estendendolo con il segno
- Le istruzioni `ldrh`, `ldrsh` e `strh` trasferiscono interi a 16 bit da/verso locazioni di memoria allineate a multipli di 2
  - `ldrh` memorizza il valore letto dalla memoria in un registro a 32 bit estendendolo con zeri
  - `ldrsh` memorizza il valore letto dalla memoria in un registro a 32 bit estendendolo con il segno



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Modi di indirizzamento di base

- Pre-indexed mode: l'indirizzo effettivo è ottenuto sommando o sottraendo il contenuto di un registro base e di uno spiazzamento immediato o in registro

`ldr r1,[r2,#-100]`	$r1 \leftarrow [\,[r2] - 100\,]$
`ldr r3,[r5,r7]`	$r3 \leftarrow [\,[r5] + [r7]\,]$
`ldr r2,[r2,-r1]`	$r2 \leftarrow [\,[r2] - [r1]\,]$
`ldr r12,[r11]`	$r12 \leftarrow [\,[r11]\,]$

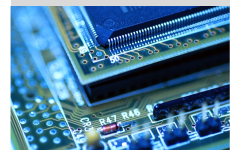
- Relative addressing mode: come il precedente, ma il registro base è `r15` (ossia il *program counter* `pc`)

  Ad esempio, le istruzioni assembler

```
DATO:   .word 100
        ldr r1, DATO
```

  sono tradotte in

```
.word 0x00000064
ldr r1,[pc,#-12]
```

Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Modi di indirizzamento con writeback

- Pre-indexed with writeback mode: come il modo pre-indexed, tranne che il registro base viene aggiornato con l'indirizzo effettivo

```
ldr r1,[r2,#-100]!
```

$r1 \leftarrow [\,[r2] - 100\,]$
$r2 \leftarrow [r2] - 100$

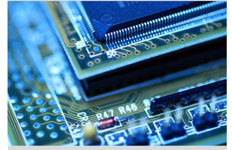
- Post-indexed: l'indirizzo effettivo è quello nel registro base; poi l'indirizzo base è aggiornato sommando o sottraendo uno spiazzamento immediato o in registro

```
ldr r1,[r2],#-100
```

$r1 \leftarrow [\,[r2]\,]$
$r2 \leftarrow [r2] - 100$

```
ldr r1,[r2],r3
```

$r1 \leftarrow [\,[r2]\,]$
$r2 \leftarrow [r2] + [r3]$



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Modi di indirizzamento con scorrimento

Nei modi di indirizzamento di base e con writeback lo spiazzamento può essere indicato con una costante immediata tra -4095 e $+4095$

Se invece lo spiazzamento è in un registro è possibile forzare uno scorrimento del suo valore verso sinistra o destra:

- Scorrimento verso sinistra:

```
ldr r1,[r2,r3,lsl #2]
```

$r1 \leftarrow [\,[r2] + 4 \times [r3]\,]$

```
ldr r1,[r2],-r3,lsl r4
```

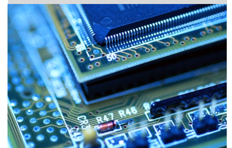
$r1 \leftarrow [\,[r2]\,]$
$r2 \leftarrow r2 - (r3 \ll (r4 \,\&\, 31))$

- Scorrimento <u>logico</u> verso destra:

```
ldr r1,[r2,r3,lsr #1]
```

$r1 \leftarrow [\,[r2] + [r3] \gg 1\,]$

`lsr #32` è ammesso, mentre `lsl #32` non è ammesso



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Caricamento di un valore da 32 bit

Perché in una architettura RISC è sempre problematico caricare in un registro un valore immediato?

Perché il valore da caricare ha la stessa dimensione dell'istruzione macchina

Nel linguaggio assembler ARM è possibile utilizzare una espressione simile a questa:

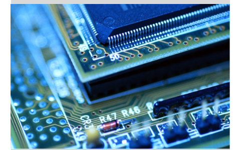
```
ldr r1,=0x12345678
```

In realtà questa viene tradotta dall'assemblatore così:

```
        ldr r1,Lcost
        :
Lcost:  .word 0x12345678
```

Non si ha lo stesso problema per caricare l'indirizzo di **`Lcost`** *?*

No: si usa l'indirizzamento relativo al **`pc`** ! `ldr r1,[pc,#offset]`



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni aritmetiche

La forma base delle istruzioni aritmetiche è

```
OP Rd,Rn,Rm
```

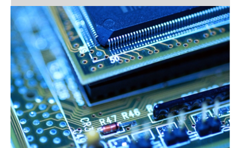
ove **`OP`** è il codice (nome) dell'operazione, **`Rd`** è il registro destinazione, **`Rn`** e **`Rm`** sono i registri operandi

Ad esempio:

- L'istruzione **`add r2,r1,r0`** somma il contenuto dei registri **`r1`** e **`r0`** e memorizza il risultato in **`r2`**
- L'istruzione **`sub r10,r4,#520`** sottrae al contenuto di **`r4`** il valore immediato 520 e memorizza il risultato in **`r10`**

È possibile utilizzare come ultimo operando valori immediati senza segno:

- La costante **`#0`** non può essere omessa: **`add r0,r1`** equivale a **`add r0,r0,r1`**
- L'assemblatore traduce automaticamente una istruzione come **`add r0,r1,#-15`** in **`sub r0,r1,#15`**

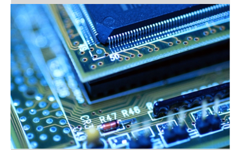


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni di somma e sottrazione

- **add** somma
  - `add r1,r2,r3` $r1 \leftarrow [r2] + [r3]$
- **adc** somma con riporto
  - `adc r1,r2,r3` $r1 \leftarrow [r2] + [r3] + [C]$
- **sub** sottrazione
  - `sub r1,r2,r3` $r1 \leftarrow [r2] - [r3]$
- **sbc** sottrazione con riporto
  - `sbc r1,r2,r3` $r1 \leftarrow [r2] - [r3] - [\neg C]$
- **rsb** sottrazione inversa (valore opposto di **sub**)
  - `rsb r1,r2,r3` $r1 \leftarrow [r3] - [r2]$
- **rsc** sottrazione inversa con riporto
  - `rsc r1,r2,r3` $r1 \leftarrow [r3] - [r2] - [\neg C]$

Nella sottrazione il flag C è 0 se c'è stato riporto, 1 altrimenti



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Valori immediati

Un valore immediato in una operazione aritmetica, logica o di copia è costruito nel seguente modo:

- L'istruzione codifica un valore a 8 bit tra 0 e 255
- Tale valore viene esteso senza segno a 32 bit
- Il valore a 32 bit viene ruotato verso destra di un numero pari di posizioni specificato nell'istruzione (tra 0 e 30)

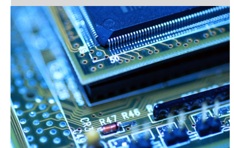
Esempi:

`add r1,r2,#1`	$r1 \leftarrow [r2] + 1$
`add r1,r2,#1,30`	$r1 \leftarrow [r2] + 4$
`add r1,r2,#1,2`	$r1 \leftarrow [r2] + 0x40000000$

L'assemblatore calcola automaticamente la codifica per una costante immediata a 32 bit

- La codifica migliore è quella con la rotazione più piccola
- Se non esiste codifica si ha un errore di sintassi

Architettura ARM
Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Rotazione e scorrimento

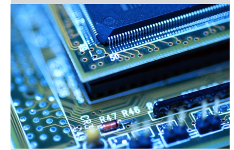
Se l'ultimo operando di una istruzione aritmetica è in un registro, allora è possibile:

- scorrerlo logicamente a destra (**lsr**) o sinistra (**lsl**)
- scorrerlo aritmeticamente a destra (**asr**) o sinistra (**asl**)
- ruotarlo verso destra (**ror**)
- ruotarlo verso destra con carry (**rrx**)

Esempi:

`add r1,r2,r3,lsl #4`	scorrimento a sx di 4 posiz.
`add r1,r2,r3,ror r4`	rotazione a dx di (**r4** & 31) posiz.
`add r1,r2,r3,rrx`	rotazione a dx di 1 posiz., entra C

Non è possibile utilizzare rotazioni e scorrimenti nelle istruzioni di moltiplicazione



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

I bit di condizione

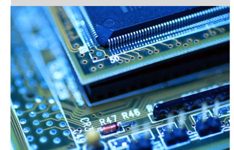
- Il formato delle istruzioni aritmetiche e logiche consente di specificare in un bit del codice operativo se si devono aggiornare i quattro codici di condizione Z, N, C e V
- Esempi:

`sub r0,r1`	non aggiornare i codici di condizione
`subs r0,r1`	aggiorna i codici di condizione

- Il bit di Carry (C) è particolarmente utile insieme a **adc** e **sbc** per operare su quantità maggiori di 32 bit:

`adds r0,r6,r8`	(sommano a 64 bit **r7**:**r6** e **r9**:**r8**,
`adc r1,r7,r9`	e memorizzano il risultato in **r1**:**r0**)

Architettura ARM
Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

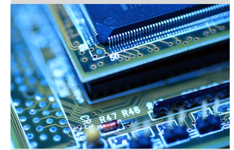
Istruzioni per la moltiplicazione

Esistono più di 20 istruzioni per la moltiplicazione tra interi:

- **mul Rd,Rm,Rs**
  - moltiplica i contenuti di **Rm** e **Rs**
  - memorizza i 32 bit inferiori del risultato in **Rd**
- **mla Rd,Rm,Rs,Rn**
  - moltiplica i contenuti di **Rm** e **Rs**, e somma il contenuto di **Rn**
  - memorizza i 32 bit inferiori del risultato in **Rd**
- **umull/smull Rdlo,Rdhi,Rm,Rs**
  - considera valori senza segno / con segno
  - moltiplica i contenuti di **Rm** e **Rs**
  - memorizza i 64 bit del risultato in **Rdlo** e **Rdhi**
- **umlal/smlal Rdlo,Rdhi,Rm,Rs**
- **umaal/smaal Rdlo,Rdhi,Rm,Rs**
- Altre istruzioni moltiplicano quantità di 8 e 16 bit, calcolano i 32 bit più significativi di un prodotto con segno, ecc.

*Perché **mul** e **mla** non considerano il segno dei numeri?*

I 32 bit inferiori del prodotto sono identici nei due casi
Ad es., $(-1) \times (-3) =$ `0xfffffffc00000003`



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni di copia in registro

- **mov Rd,Rm** — copia il contenuto di **Rm** in **Rd**
- **mov Rd,#val** — copia il valore immediato in **Rd**
- **mvn Rd,Rm** — copia l'inverso bit a bit di **Rm** in **Rd**
- **mvn Rd,#val** — copia l'inverso bit a bit di **val** in **Rd**

È possibile applicare scorrimenti o rotazioni come per le istruzioni aritmetiche, ad esempio:

```
mov r2,r3,lsl #4
```

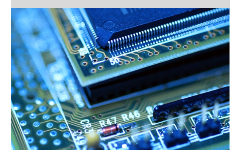
L'assemblatore traduce una pseudo-istruzione analoga a

```
mov r1,#-17
```

come

```
mvn r1,#16
```

Anche le istruzioni di copia possono impostare i codici condizione (ad es, **movs**)

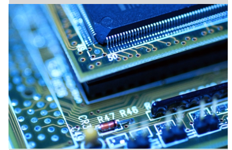


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni logiche

- `and Rd,Rm,Rn` AND logico bit a bit
  - `and r0,r1,r2` r0 ← [r1] & [r2]
- `eor Rd,Rm,Rn` XOR logico bit a bit
  - `eor r0,r1,r2,lsl #2` r0 ← [r1] ^ ([r2]<<2)
- `orr Rd,Rm,Rn` OR logico bit a bit
  - `orr r0,r1,#520` r0 ← [r1] | 0x208
- `bic Rd,Rm,Rn` Bit clear
  - `bic r0,r1,#0xff` r0 ← [r1] & 0xffffff00

Anche le istruzioni logiche possono impostare i codici condizione (ad es, `eors`)



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

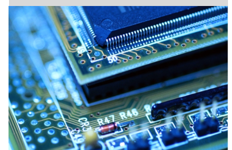
Istruzioni di confronto

- `tst Rn,Rs` Test
  - Effettua l'AND logico bit a bit, imposta i codici di condizione e scarta il risultato
- `teq Rn,Rs` Test Equivalence
  - Effettua l'XOR logico bit a bit, imposta i codici di condizione e scarta il risultato
- `cmp Rn,Rs` Compare
  - Effettua la sottrazione [Rn] − [Rs], imposta i codici di condizione e scarta il risultato
- `cmn Rn,Rs` Compare Negated
  - Effettua la somma [Rn] + [Rs], imposta i codici di condizione e scarta il risultato

Per l'operando `Rs` è possibile utilizzare valori immediati, scorrimenti o rotazioni

Architettura ARM

Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni di salto

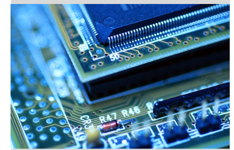
- Forma generale di una istruzione di salto (branch):

  `bcc locazione`

- `cc` codifica la condizione che deve essere verificata per effettuare il salto
- L'istruzione di salto incondizionato è `b` oppure `bal`
- `locazione` rappresenta l'istruzione alla quale saltare
  - In linguaggio assembler è una etichetta simbolica
  - Nel codice macchina rappresenta uno spiazzamento relativo alla posizione dell'istruzione di salto

Si può saltare senza utilizzare l'istruzione branch `bcc`*?* ***Sì!***

In Aarch32 è sufficiente scrivere un valore in `r15` (ossia `pc`)



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Calcolo dell'indirizzo destinazione di un salto

- Ogni istruzione di salto codifica un valore a 24 bit in complemento a due
- Si estende con segno tale valore a 32 bit
- Si moltiplica per quattro
- Il risultato è lo spiazzamento rispetto al valore di `pc`

Perché il valore viene moltiplicato per quattro?

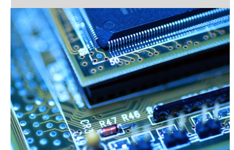
Le istruzioni di 32 bit sono allineate in memoria

Cosa fa una istruzione di salto il cui valore a 24 bit è -1?

Non ha alcun effetto sulla sequenza di istruzione eseguite

Architettura ARM

Marco Cesati



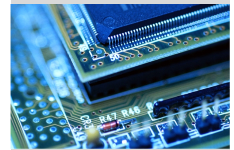
Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni di salto condizionato

L'esecuzione delle istruzioni di salto condizionato dipende dal valore di alcuni codici di condizione *cc*

cc	$cc = 1$		$cc = 0$	
Z	**beq**	"=", "= 0"	**bne**	"≠", "≠ 0"
C	**bcs**	carry	**bcc**	no carry
C	**bhs**	"$\geq_u$"	**blo**	"$<_u$"
N	**bmi**	"$<_s 0$"	**bpl**	"$\geq_s 0$"
V	**bvs**	overflow	**bvc**	no overflow
$\overline{C} \vee Z$	**bls**	"$\leq_u$"	**bhi**	"$>_u$"
$N \oplus V$	**blt**	"$<_s$"	**bge**	"$\geq_s$"
$Z \vee (N \oplus V)$	**ble**	"$\leq_s$"	**bgt**	"$>_s$"

- Gli operatori con "*u*" sono di confronto senza segno
- Gli operatori con "*s*" sono di confronto con segno



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Supporto all'invocazione di procedure

- Nei microprocessori ARM l'invocazione di una procedura è realizzato tramite l'istruzione **bl** (branch and link)
- **bl** è analoga ad una istruzione di salto (eventualmente condizionata)
- Prima di effettuare il salto, l'indirizzo dell'istruzione successiva a **bl** è salvato nel registro **lr** (o **r14**)

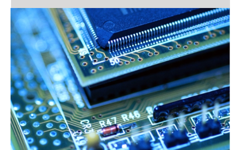
Non esiste l'istruzione "return": come si termina una procedura?

Ad esempio: **mov pc,lr**

In altre architetture l'istruzione "call" salva l'indirizzo di ritorno sullo stack. Che vantaggio ha il metodo usato da ARM?

Se la procedura non invoca altre sotto-procedure, si evitano due potenziali accessi alla memoria dinamica

D'altra parte la procedura deve salvare sullo stack o in altro registro il contenuto di **lr** prima di invocare altre procedure

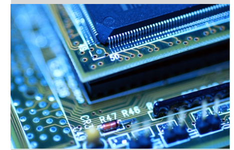


Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Application Binary Interface

- L'ABI (Application Binary Interface) definisce gli aspetti dell'ambiente di esecuzione che non sono determinati dall'architettura hardware
  - Come passare i parametri alle procedure
  - Il formato dei file eseguibili
  - Come si gestiscono le eccezioni
  - Come si collegano le librerie
  - Come si interfacciano i debugger
  - ...
- Nel mondo ARM sono state utilizzate diverse ABI
- La più recente (2005) e maggiormente adottata è

EABI: Embedded-Application Binary Interface



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

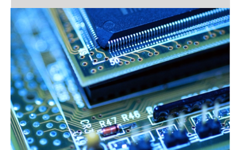
Passaggio dei parametri alle procedure

Ruolo speciale dei registri nell'invocazione delle procedure:

`pc/r15`		Program counter
`lr/r14`		Link register
`sp/r13`	p	Stack pointer
`ip/r12`		Registro di appoggio per il linker
`fp/r11`	p	Variabile locale
`sl/r10`	p	Variabile locale
`r9`	p?	Ruolo definito dalla piattaforma
`r8`	p	Variabile locale
`r7`	p	Variabile locale
`r6`	p	Variabile locale
`r5`	p	Variabile locale
`r4`	p	Variabile locale
`r3`		Argomento # 4
`r2`		Argomento # 3
`r1`		Argomento # 2, Risultato (32 bit superiori)
`r0`		Argomento # 1, Risultato (32 bit inferiori)

Ogni procedura deve preservare il contenuto dei registri "p"

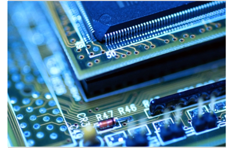
Architettura ARM
Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Uso dello stack per salvare e ripristinare i registri

- Poiché una procedura deve preservare il contenuto di alcuni registri, essa deve
  - salvarne il contenuto sullo stack all'inizio della procedura
  - ripristinare il contenuto dallo stack al termine
- È possibile farlo con una serie di istruzioni "push" e "pop":
  - "push" `lr`: `str lr,[sp,#-4]!`
  - "pop" `lr`: `ldr lr,[sp],#4`
- Nei microprocessori ARM la tipologia di stack non è pre-determinata
- EABI impone l'uso di stack *full descending*:
  - stack crescente per indirizzi decrescenti
  - `sp` puntante all'ultimo elemento inserito
- Tra i registri da preservare c'è anche `lr`, nel caso la procedura debba invocare altre sotto-procedure



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Trasferimento di blocchi di registri (Aarch32)

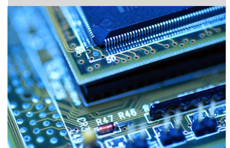
- Per facilitare le operazioni di salvataggio e ripristino dei registri sono definite le istruzioni `stmxx` e `ldmxx`
  - Le due lettere `xx` indicano il tipo di stack, ossia le operazioni di aggiornamento del registro puntatore
  - Nel caso di stack EABI *full descending* le lettere sono `fd`
- Esempi:
  - "Push": `stmfd sp!,{r4-r8,r10-r11,r14}`
  - "Pop": `ldmfd sp!,{r4-r8,r10-r11,r15}`

Che effetto particolare si ottiene utilizzando le due istruzioni sopra in una procedura?

L'istruzione `ldmfd` termina la procedura scrivendo in `pc` (`r15`) l'indirizzo contenuto in `lr` (`r14`)

Architettura ARM

Marco Cesati



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Modi di esecuzione (Aarch32)

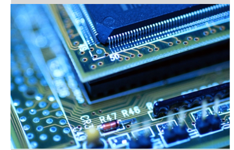
In ogni istante il processore ARM è in uno dei seguenti stati:

Modo	Esecuzione di	Registri privati
User	applicazioni normali	
System	processi privilegiati	
FIQ	interruzioni veloci	**r8** – **r14**, **spsr**
IRQ	interruzioni normali	**r13**, **r14**, **spsr**
Supervisor	codice del kernel	**r13**, **r14**, **spsr**
Abort	gestori di fault	**r13**, **r14**, **spsr**
Undefined	emulatori di coprocessori	**r13**, **r14**, **spsr**

In ogni modo di esecuzione sono comunque visibili:

- i 13 registri di uso generale (da **r0** a **r12**)
- **r13** ≡ **sp** (stack pointer), **r14** ≡ **lr** (link register)
- il contatore di programma **r15** ≡ **pc**
- un registro di stato (**cpsr**)

r8 . . . **r12** sono comuni a tutti i modi di esecuzione tranne FIQ: durante una interruzione veloce si possono utilizzare questi registri senza dover salvarne il contenuto in memoria



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Tipi di eccezione (Aarch32)

Nei microprocessori ARM sono definiti sette tipi di eccezioni:

Priorità	Nome	Indir. vettore	Modo
1	Reset	0x00000000	Supervisor
6	Undefined instruction	0x00000004	Undefined
7	Software interrupt	0x00000008	Supervisor
5	Prefetch abort	0x0000000C	Abort
2	Data abort	0x00000010	Abort
4	IRQ (interrupt)	0x00000018	IRQ
3	FIQ (fast interrupt)	0x0000001C	FIQ

Per ciascun tipo di interruzione è definito un *vettore*:

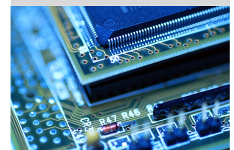
- il vettore è in una locazione prefissata della memoria
- il vettore codifica la [locazione della] prima istruzione che dovrà gestire l'eccezione

Perché è definita la priorità delle eccezioni?

Se due o più eccezioni si verificano contemporaneamente, queste vengono gestite rispettando la loro priorità

Architettura ARM

Marco Cesati



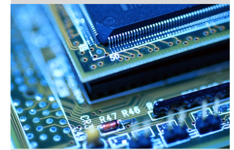
Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Gestione delle eccezioni

All'occorrenza di una eccezione la CPU effettua le seguenti operazioni:

- Il contenuto di `r15` (`pc`) è copiato nel registro `r14` (`lr`) relativo al modo dell'eccezione
- Il contenuto di `cpsr` è copiato nel registro `spsr` relativo al modo dell'eccezione
- I bit di `cpsr` sono modificati per forzare il processore nel modo corrispondente all'eccezione e, se necessario, per disabilitare FIQ e/o IRQ
- Il registro `pc` (`r15`) è caricato con l'indirizzo specificato dal vettore appropriato

Poiché il registro `r13` (`sp`) è privato per i vari modi di esecuzione (tranne che per User e System), il gestore dell'eccezione può facilmente utilizzare un proprio stack



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

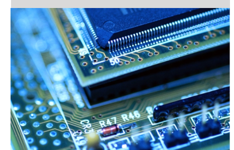
Terminazione della eccezione

I dettagli per concludere la gestione di una eccezione variano con il tipo di eccezione

- Il riconoscimento di eventi asincroni (FIQ, IRQ) è effettuato al termine dell'esecuzione di ciascuna istruzione
  - `pc` (e dunque `lr`) contiene l'indirizzo dell'istruzione conclusa $+8$
  - Si conclude la gestione tramite `subs pc,lr,#4`
- Quando l'eccezione è sincrona (attivata da una istruzione all'indirizzo $\mathcal{I}$)

Tipo di eccezione	`lr` =	Tipico ritorno
Undefined instruction	$\mathcal{I}+4$	`movs pc,lr`
Software interrupt	$\mathcal{I}+4$	`movs pc,lr`
Prefetch Abort	$\mathcal{I}+4$	`subs pc,lr,#4`
Data Abort	$\mathcal{I}+8$	`subs pc,lr,#8`

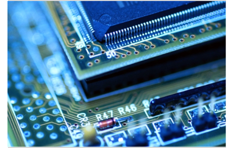
Quando il target di una istruzione è `pc`, la "`s`" finale significa: copia il contenuto di `spsr` (modalità dell'eccezione) in `cpsr`



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Forme speciali delle istruzioni `stm` e `ldm`

- `stmfd sp,{blocco registri}^`
  - Salva i valori dei registri della modalità User (a prescindere dalla modalità di esecuzione corrente)
  - `sp` non viene modificato
- `ldmfd sp[!],{blocco reg. senza pc}^`
  - Carica i valori nei registri della modalità User (a prescindere dalla modalità di esecuzione corrente)
- `ldmfd sp[!],{blocco registri,pc}^`
  - Carica i valori nei registri della modalità attiva
  - Inoltre copia il contenuto di `spsr` in `cpsr`
- Utili durante la gestione delle eccezioni



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Manipolazione dei bit di stato

- I gestori delle eccezioni debbono talvolta manipolare lo stato dei bit nel registro `cpsr`, ad esempio per disabilitare le interruzioni
- L'istruzione `mrs` copia il contenuto di `cpsr` o `spsr` in un registro GPR:

```
mrs r1,cpsr
```

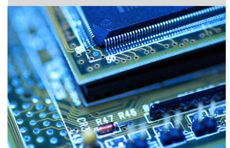
- L'istruzione `msr` copia il contenuto di un registro GPR in `cpsr` o `spsr`:

```
msr spsr,r0
```

- L'istruzione `cps` modifica direttamente la modalità d'esecuzione del processore

In modalità User `mrs`, `msr` e `cps` non funzionano: perché?

Se funzionassero qualunque processo (applicazione) potrebbe entrare in modalità privilegiata e scavalcare i meccanismi di protezione del sistema operativo



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb

Istruzioni Thumb

Le istruzioni da 32 bit possono essere un problema per sistemi embedded con ridotte capacità di memoria

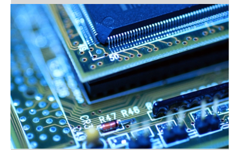
Molti microprocessori ARM dispongono di una ISA alternativa chiamata Thumb:

- Istruzioni codificate in 16 bit
- Solo 8 registri GPR: **r0**, **r1**, . . . , **r7** (oltre a **pc**, **lr**, **sp**)
- Molte istruzioni hanno solo un formato a due operandi in cui un registro è sia sorgente che destinazione
- Le uniche istruzioni condizionali sono i salti
- Registri, indirizzi e operandi sono sempre a 32 bit

È possibile mischiare procedure con ISA ARM e Thumb:
il bit T in **cpsr** indica l'ISA attiva

Si attiva la modalità Thumb con le istruzioni di salto **bx**, **blx** oppure avendo il bit T impostato in **spsr** e copiandolo in **cpsr**

Nell'arch. ARMv7: introdotta la ISA Thumb–2 che mescola istruzioni di lunghezza variabile: 16 bit (Thumb) e 32 bit (ARM)



Schema della lezione
Evoluzione degli ARM
L'architettura ARM
Istruzioni load e store
Modi di indirizzamento
Istruzioni aritmetiche
Istruzioni di copia
Istruzioni logiche
Istruzioni di confronto
Istruzioni di salto
Procedure
Eccezioni
ISA Thumb