

# Istruzioni SIMD

L'architettura ARMv6 ha introdotto delle istruzioni aritmetiche che agiscono sui registri considerandoli come contenitori di strutture array definiti su word o halfword.

La sintassi generica di una istruzione aritmetica SIMD è la seguente:

`<MNEM>{<PreCond>} <Rd>, <Rn>, <Rm>`

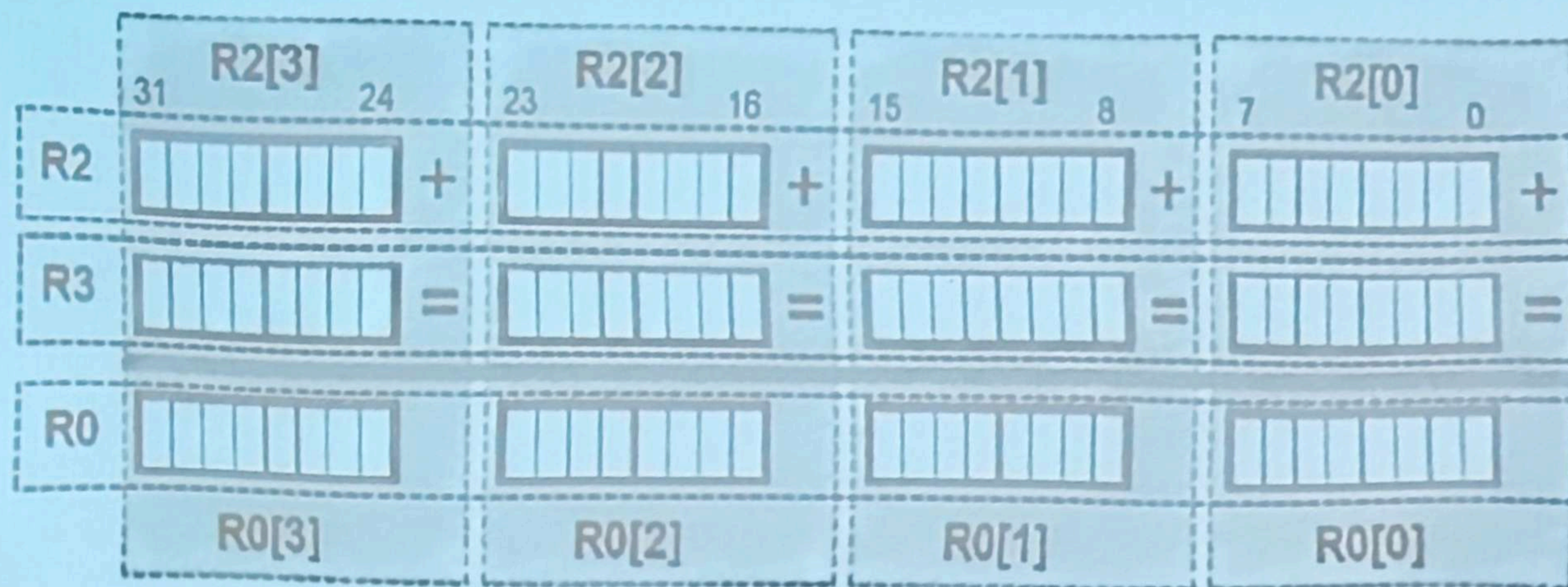
Il codice mnemonico <MNEM> si ottiene dalle combinazioni di:

- prefisso
  - U ➡ Unsigned
  - S ➡ Signed
  - Q ➡ Saturazione
  - H ➡ Risultati dimezzati per prevenire overflow
- tipo di istruzione / dimensione del elemento dell'array
  - 8 ➡ ADD8      SUB8
  - 16 ➡ ADD16    SUB16
  - X ➡ ADDSUBX   SUBADDX



# Istruzioni SIMD: iniziamo con un esempio

UADD8 R0, R2, R3



$$R0[0] = R2[0] + R3[0]$$

$$R0[1] = R2[1] + R3[1]$$

$$R0[2] = R2[2] + R3[2]$$

$$R0[3] = R2[3] + R3[3]$$



MNEM	Semantica	Elemento
<b>ADD8</b>	$Rd[i] = Rn[i] + Rm[i], i \in [0, 1, 2, 3]$	byte
<b>SUB8</b>	$Rd[i] = Rn[i] - Rm[i], i \in [0, 1, 2, 3]$	byte
<b>ADD16</b>	$Rd[i] = Rn[i] + Rm[i], i \in [0, 1]$	halfword
<b>SUB16</b>	$Rd[i] = Rn[i] - Rm[i], i \in [0, 1]$	halfword
<b>ADDSUBX</b>	$Rm[0] \leftrightarrow Rm[1],$ $Rd[1] = Rn[1] + Rm[1],$ $Rd[0] = Rn[0] - Rm[0]$	halfword
<b>SUBADDX</b>	$Rm[0] \leftrightarrow Rm[1],$ $Rd[1] = Rn[1] - Rm[1],$ $Rd[0] = Rn[0] + Rm[0]$	halfword



# Principali Istruzioni SIMD

MNEM	Tipo	con Segno	Saturazione	Risultati/2
QADD8	ADD8	✓	✓	
SADD8		✓		
SHADD8		✓		✓
UADD8				
UHADD8				✓
UQADD8			✓	
QSUB8	SUB8	✓	✓	
SSUB8		✓		
SHSUB8		✓		✓
USUB8				
UHSUB8				✓
UQSUB8			✓	

Prefisso	Aritmetica	1st GPR
S	con segno, modulo $2^8$ o $2^{16}$	CEB-1
Q	con segno e con saturazione	
SH	con segno e risultati divaricati	
U	senza segno, modulo $2^8$ o $2^{16}$	CEB-1
UQ	senza segno e con saturazione	
UH	senza segno e risultati divaricati	



# Istruzioni SIMD su array di byte

MNEM	Descrizione	Semantica
USAD8	Unsign. Sum Abs. Diff.	$\langle Rd \rangle \leftarrow \sum_{i=0}^3  Rm[i] - Rs[i] $
USADA8	Unsign. Sum Abs. Diff. Acc	$\langle Rd \rangle \leftarrow \langle Rn \rangle + \sum_{i=0}^3  Rm[i] - Rs[i] $



# Flag GE (Greater than or Equal)

In ARMv6, le istruzioni SIMD impostano i flag GE della CPSR in base all'esito del risultato dei singoli byte o delle singole halfword che compongono la word.

Ogni registro Rx è in grado di memorizzare una word, che, a sua volta, è costituita da due halfword oppure da quattro byte:

31	30	....	24	23	22	....	16	15	14	....	8	7	6	....	0
		....				....				....				....	
Rx[24 ÷ 31]=B <sub>3</sub>				Rx[16 ÷ 23]=B <sub>2</sub>				Rx[8 ÷ 15]=B <sub>1</sub>				Rx[0 ÷ 7]=B <sub>0</sub>			
Rx[16 ÷ 31]=H <sub>1</sub>								Rx[ 0 ÷ 15]=H <sub>0</sub>							



# Flag GE

Nel testo H0 è sinonimo di B (Bottom), talvolta indicato con l'acronimo LSH (Least Significant Halfword), mentre H1 è sinonimo di T (Top), qualche volta indicato con la sigla MSH (Most Significant Halfword).

31	30	....	24	23	22	....	16	15	14	....	8	7	6	....	0
		....				....				....				....	
Rx[24 ÷ 31]=B <sub>3</sub>				Rx[16 ÷ 23]=B <sub>2</sub>				Rx[8 ÷ 15]=B <sub>1</sub>				Rx[0 ÷ 7]=B <sub>0</sub>			
Rx[16 ÷ 31]=H <sub>1</sub>								Rx[ 0 ÷ 15]=H <sub>0</sub>							

I quattro flag GE sono aggiornati in base al tipo di istruzione che agisce su word o byte:

Istruzione	risultato	GE			
		0	1	2	3
halfword	B	✓	✓		
	T			✓	✓
byte	B <sub>0</sub>	✓			
	B <sub>1</sub>		✓		
	B <sub>2</sub>			✓	
	B <sub>3</sub>				✓



## Istruzione di selezione dei byte

L'istruzione di selezione permette di analizzare i flag GE del registro di stato e scegliere tra due operandi quale byte copiare nel corrispondente byte nel registro destinazione.

**SEL{<PreCond>} <Rd>, <Rn>, <Rm>**

byte  $B_i$  di Rd = byte  $B_i$  di  $\begin{cases} Rn & \text{se } GE[i]=1 \\ Rm & \text{se } GE[i]=0 \end{cases}$  con  $i=0..3$ .

31 .. 28	27 26 ..... 22 21 20	19..16	15..12	11..8	7 .. 4	3..0
PreCond	0 1 1 0 1 0 0 0	Rn	Rd	SBO	1 0 1 1	Rm

La sigla SBO (Should Be One) indica che i bit all'interno della codifica dell'istruzione devono assumere il valore 1.



# Istruzioni di moltiplicazione con due operandi e risultato in word

$\langle \text{MNEM} \rangle \{ \langle \text{PreCond} \rangle \} \{ \langle \text{S} \rangle \} \langle \text{Rd} \rangle, \langle \text{Rm} \rangle, \langle \text{Rs} \rangle$

MNEM	Descrizione	Molt.	Semantica	Tronc.
MUL	Multiply	$32 \times 32$	$Rd \leftarrow Rm \cdot Rs$	$31 \div 0$
SMULxy	Sign. Mult. Long	$16 \times 16$	$Rd \leftarrow Rm[x] \cdot Rs[y]$	
SMULWy	Sign. Mult. Word	$32 \times 16$	$Rd \leftarrow Rm \cdot Rs[y]$	$47 \div 16$
SHUAD	Sign. Mult. Add Dual	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[B] + Rm[T] \cdot Rs[T]$	
SHUADX	Sign. Mult. Add Dual, eXch	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[T] + Rm[T] \cdot Rs[B]$	
SHUSD	Sign. Mult. Sub Dual	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[B] - Rm[T] \cdot Rs[T]$	
SHUSDX	Sign. Mult. Sub Dual, eXch	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[T] - Rm[T] \cdot Rs[B]$	
SMMUL	Sign. MSW Mult truncate	$32 \times 32$	$Rd \leftarrow Rm \cdot Rs$	$63 \div 32$
SMMULR	Sign. MSW Mult. Round	$32 \times 32$	$Rd \leftarrow Rm \cdot Rs$	$63 \div 32$



# Istruzioni di trasferimento dati

Le istruzioni di trasferimento dati si occupano di trasferire valori nei registri o tra registri.

Nel caso in cui si tratti di un trasferimento tra registri di uso generale la sintassi dell'istruzione è la seguente:

$\langle \text{MNEM} \rangle \{ \langle \text{PreCond} \rangle \} \{ \langle \text{S} \rangle \} \langle \text{Rd} \rangle, \text{OP}_2$

OPCODE	MNEM	Descrizione	Semantica
1101	MOV	Carica registro con $\text{OP}_2$	$\text{Rd} \leftarrow \text{OP}_2$
1111	MVN	Carica registro con l'inverso di $\text{OP}_2$	$\text{Rd} \leftarrow \overline{\text{OP}_2}$



## Istruzioni di accesso ai registri di stato

**MRS** (*Move to Register from Status register*) copia il valore del CPSR, o del SPSR nell'attuale modo di funzionamento del processore, all'interno dei registri di uso generale; la sintassi risulta:

MRS{<PreCond>} <Rd>, {CPSR|SPSR}

**MSR** (*Move to Status register from Register*) copia il contenuto di un registro o una costante in una o più ambiti del registro CPSR o del SPSR nell'attuale modo di funzionamento del processore:

MRS{<PreCond>} {CPSR|SPSR}{\_<ambiti>}, <Rm>  
MRS{<PreCond>} {CPSR|SPSR}{\_<ambiti>}, #<valore>

c=controllo, x=estensione, s=stato, f=flag



## Istruzioni di accesso ai registri di stato

```
MRS R0 , CPSR           // R0=CPSR (lettura stato)
BIC R0 , R0, #0x1F       // Ripulisce mode bit
ORR R0 , R0, #0x13       // Imposta modo Supervisor
MSR CPSR_c, R0           // CPSR=R0 (scrittura stato)
```

Nel caso si debba modificare esclusivamente un certo ambito del registro di stato (es. flag), si può restringere l'applicazione dell'istruzione utilizzando la sintassi:

```
MSR CPSR_F, #0xF0000000
```



# Istruzioni di branch

I processori ARM supportano un'istruzione di branch che in modo diretto permette di saltare in avanti o indietro fino a 32 MB.

Per arrivare fino a 4 GB possiamo caricare R15 o PC con il valore desiderato (assicurandosi di aver caricato la posizione originaria in R14 o LR).

`<MNEM>{<PreCond>} <Operando>`

**B** (branch) e **BL** (branch with link).

L'istruzione B carica nel PC (R15) l'indirizzo della prima istruzione della procedura che si desidera eseguire

L'istruzione BL è simile all'istruzione B, in più però carica nel registro LR (R14) l'indirizzo di ritorno della procedura, ovvero il valore del PC nel momento in cui viene eseguita l'istruzione BL.



# Istruzioni di branch

MNEM	Pre Cond	Oper.	Descrizione	Semantica
B	✓	label	<i>Branch</i>	$R15 \leftarrow \text{indirizzo label}$
BX	✓	$R_m$	<i>Branch, eXch. Thumb</i>	$R15 \leftarrow R_m$  Se $R_m[0]=0$ imposta modo ARM, altrimenti Thumb ( $R_m[0]=1$ )
BXJ	✓	$R_m$	<i>Branch, eXch. Java</i>	$R15 \leftarrow R_m$  Imposta modo Java se disponibile e abilitato altrimenti si comporta come BX
BL	✓	label	<i>Branch, Link</i>	$R14 \leftarrow \text{indirizzo istruz. succ.}$ $R15 \leftarrow \text{indirizzo label}$
BLX		label	<i>Branch, Link, eXch. Thumb</i>	$R14 \leftarrow \text{indirizzo istruz. succ.}$ $R15 \leftarrow \text{indirizzo label}$ Imposta modo Thumb
BLX	✓	$R_m$	<i>Branch, Link, eXch. Thumb</i>	$R14 \leftarrow \text{indirizzo istruz. succ.}$ $R15 \leftarrow R_m[31 \div 1]$  Se $R_m[0]=0$ imposta modo ARM, altrimenti Thumb ( $R_m[0]=1$ )



# Istruzioni di branch

MOV LR, PC

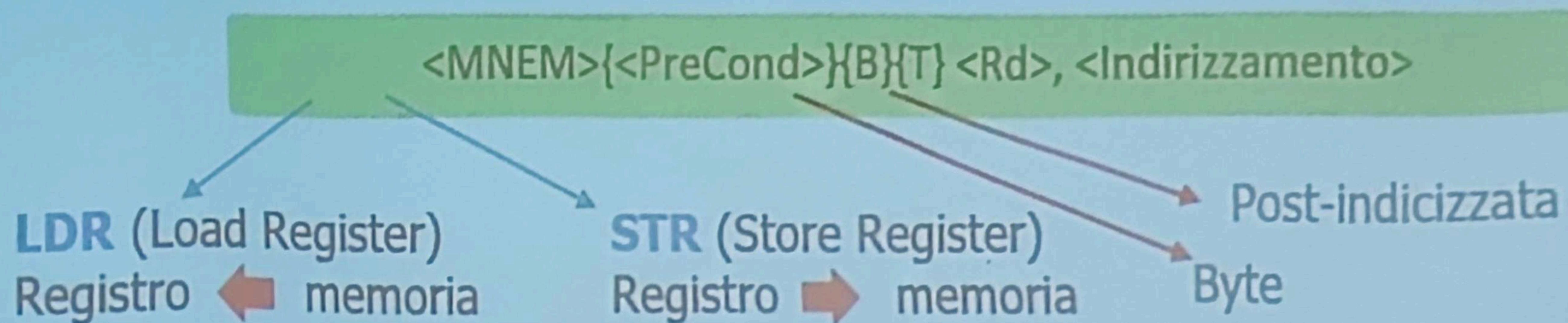
LDR PC, =indirizzo oltre 32MB

Di seguito alcuni esempi di istruzioni di branch incondizionato e condizionato al valore dei flag del registro di stato:

B label	// salta alla label senza condizioni
BCC label	// salta alla label se il flag C=0
BEQ label	// salta alla label se il flag Z=0



# Istruzioni di Load e Store: word o unsigned byte modo 2 di indirizzamento





# Istruzioni di Load e Store: word o unsigned byte

Modo		Sintassi	Semantica
Offset	immediato	$[\langle R_n \rangle]$	$\text{indirizzo} \leftarrow R_n$
		$[\langle R_n \rangle, \# \{+ - \} \langle \text{offset}_{12} \rangle]$	$\text{indirizzo} \leftarrow R_n \pm \text{offset}_{12}$
	registro	$[\langle R_n \rangle, \{+ - \} \langle R_m \rangle]$	$\text{indirizzo} \leftarrow R_n \pm R_m$
	regis. scal.	$[\langle R_n \rangle, \{+ - \} \langle R_m \rangle, \langle \text{SHIFT} \rangle]$	$\text{indirizzo} \leftarrow R_n \pm R_m \langle \text{SHIFT} \rangle$
Pre-indiciz.	immediato	$[\langle R_n \rangle, \# \{+ - \} \langle \text{offset}_{12} \rangle]!$	$\text{indirizzo} \leftarrow R_n \pm \text{offset}_{12}$ $R_n \leftarrow \text{indirizzo}$
	registro	$[\langle R_n \rangle, \{+ - \} \langle R_m \rangle]!$	$\text{indirizzo} \leftarrow R_n \pm R_m$ $R_n \leftarrow \text{indirizzo}$
	registro scalato	$[\langle R_n \rangle, \{+ - \} \langle R_m \rangle, \langle \text{SHIFT} \rangle]!$	$\text{indirizzo} \leftarrow R_n \pm R_m \langle \text{SHIFT} \rangle$ $R_n \leftarrow \text{indirizzo}$
Post-indiciz.	immediato	$[\langle R_n \rangle], \# \{+ - \} \langle \text{offset}_{12} \rangle$	$\text{indirizzo} \leftarrow R_n$ $R_n \leftarrow R_n \pm \text{offset}_{12}$
	registro	$[\langle R_n \rangle], \{+ - \} \langle R_m \rangle$	$\text{indirizzo} \leftarrow R_n$ $R_n \leftarrow R_n \pm R_m$
	registro scalato	$[\langle R_n \rangle], \{+ - \} \langle R_m \rangle, \langle \text{SHIFT} \rangle$	$\text{indirizzo} \leftarrow R_n$ $R_n \leftarrow R_n \pm R_m \langle \text{SHIFT} \rangle$



# Istruzioni di Load e Store: esempi

// Offset immediato

LDR R0,[R1,#2]

// Offset a registro

STR R0,[R1,R2]

// Offset a registro scalato

LDR R0,[R1,R2,LSL #3]

// pre-indiciz. immediato

STR R0,[R1,#2]!

// Pre-indiciz. a registro

LDR R0,[R1,R2]!

// Pre-indiciz. a registro scalato

STR R0,[R1,R2,LSL #3]!

// Post-indiciz. immediato

LDR R0,[R1],#2

// Post-indiciz. a registro

STR R0,[R1],R2

// R0<-memoria[R1+2]

// R0->memoria[R1+R2]

// R0<-memoria[R1+R2\*8]

// R0->memoria[R1+2] , R1=R1+2

// R0<-memoria[R1+R2] , R1=R1+R2

// R0->memoria[R1+R2\*8] , R1=R1+R2\*8

// R0<-memoria[R1] , R1=R1+2

// R0->memoria[R1] , R1=R1+R2



## Istruzioni di Load e Store: esempi

// Post-indiciz. a reg. scalato

LDR R0,[R1],R2,LSL #3

//  $R0 \leftarrow \text{memoria}[R1]$ ,  $R1 = R1 + R2 * 8$

// ALTRI ESEMPI \*\*\*\*\*

LDREQB R2,[R5,#5]

// Se EQ allora  $R2 \leftarrow$  primo byte memoria[ $R5 + \#5$ ]

// azzerà altri 3 byte MSB di R2

// per invocare un sotto programma ROUTINE

STR R3, ROUTINE

//  $R3 \leftarrow \text{indirizzo}(\text{ROUTINE})$

MOV PC,R3

// Esegue la routine

...

ROUTINE



# Istruzioni di Load e Store: byte, word o doubleword

## modo 3 indirizzamento

LDR {<PreCond>}{SB|H|SH} <Rd>, <Indirizzamento>


STR {<PreCond>}{H} <Rd>, <Indirizzamento>

Modo		Sintassi	Semantica	Suffisso	Descrizione
Offset	immediato	[<Rn>, # {+ -} <offset <sub>s</sub> >]	$indirizzo \leftarrow Rn \pm offset_s$	B	Byte senza segno
	registro	[<Rn>, {+ -} Rm]	$indirizzo \leftarrow Rn \pm Rm$	SB	Byte con segno
Pre-indiciz.	immediato	[<Rn>, # {+ -} <offset <sub>s</sub> >]!	$indirizzo \leftarrow Rn \pm offset_s$ $Rn \leftarrow indirizzo$	H	Halfword senza segno
	registro	[<Rn>, {+ -} Rm]!	$indirizzo \leftarrow Rn \pm Rm$ $Rn \leftarrow indirizzo$	SH	Halfword con segno
Post-indiciz.	immediato	[<Rn>], # {+ -} <offset <sub>s</sub> >	$indirizzo \leftarrow Rn$ $Rn \leftarrow Rn \pm offset_s$	<non indicato>	Word
	registro	[<Rn>], {+ -} Rm	$indirizzo \leftarrow Rn$ $Rn \leftarrow Rn \pm Rm$		



# Istruzioni di Load e Store su registri multipli modo 4 indirizzamento

$\langle \text{MNEM} \rangle \{ \langle \text{PreCond} \rangle \} \{ \langle \text{Modo agg} \rangle \} \langle \text{Rn} \rangle \{ \text{l} \}, \langle \text{registri} \rangle \{ \wedge \}$



Modo Aggiornamento		Memoria		Calcolo di Rn
Sintassi	Descrizione	Ind. Inizio	Ind. Fine	PreCond · W
IA	<i>Increment After</i> post-incremento	Rn	$Rn+4 \cdot \text{NumReg}-4$	$Rn \leftarrow Rn+4 \cdot \text{NumReg}$
IB	<i>Increment Before</i> pre-incremento	$Rn+4$	$Rn+4 \cdot \text{NumReg}$	$Rn \leftarrow Rn+4 \cdot \text{NumReg}$
DA	<i>Decrement After</i> post-decremento	$Rn-4 \cdot \text{NumReg}+4$	Rn	$Rn \leftarrow Rn-4 \cdot \text{NumReg}$
DB	<i>Decrement Before</i> pre-decremento	$Rn-4 \cdot \text{NumReg}$	$Rn-4$	$Rn \leftarrow Rn-4 \cdot \text{NumReg}$



# Return From Exception

La RFE permette di caricare la coppia di registri PC e CPSR a partire dall'indirizzo contenuto in  $\langle Rn \rangle$  e nella successiva word

$RFE\langle \text{Modo Agg} \rangle \langle Rn \rangle \{!\}$

Modo Aggiornamento		Memoria		Calcolo di $Rn$ PreCond · W
Sintassi	Descrizione	Ind. Inizio	Ind. Fine	
IA	<i>Increment After</i> post-incremento	$Rn$	$Rn+4 \cdot NumReg-4$	$Rn \leftarrow Rn+4 \cdot NumReg$
IB	<i>Increment Before</i> pre-incremento	$Rn+4$	$Rn+4 \cdot NumReg$	$Rn \leftarrow Rn+4 \cdot NumReg$
DA	<i>Decrement After</i> post-decremento	$Rn-4 \cdot NumReg+4$	$Rn$	$Rn \leftarrow Rn-4 \cdot NumReg$
DB	<i>Decrement Before</i> pre-decremento	$Rn-4 \cdot NumReg$	$Rn-4$	$Rn \leftarrow Rn-4 \cdot NumReg$



## Istruzioni verso i coprocessori: elaborazione dati

Coprocessor Data Processing (**CDP**) consente di richiedere ad un coprocessore tra quelli disponibili (P0..P15) di eseguire una istruzione tra quelle disponibili nel suo Set Instruction.

`CDP{<PreCond>} <Pd>, <OPCODE_1>, <CRd>, <CRn>, <CRm> {,<OPCODE_2>}`

`CDP2 <Pd>, <OPCODE_1>, <CRd>, <CRn>, <CRm> {,<OPCODE_2>}`

### Esempio

`CDP P1, 3, C3, C2, C1, 4 // Coprocessore P1 OPCODE_1=3 OPCODE2=4`